- Master's thesis -

CONSTRUCTION AND ANALYSIS OF TRUST GRAPHS USING HTTP TRACES

Freie Universität Berlin

Malvin Thiel

Department of Mathematics and Computer Science

Prof. Dr. Ina Schieferdecker

Dr. Edzard Höfig

Abstract

Users of the World Wide Web often aren't aware of the masses of private data they are revealing while browsing the Web. For the visualisation of privacy violations, a browser plug-in that enables transparent surfing has been developed by Tobias Fielitz, an associate at Freie Universität Berlin. The possibilities that the data of this plug-in provides can be intensified by the collection of any plug-in data at a central place. Beside returning thus gain knowledge to single instances of the plug-in, whatsoever analysis are feasible using the obtained data set. The exploration of possible analyses is part of this thesis.

The central data server is developed by using the Google Web Toolkit (GWT) whereas the data itself is stores as RDF triples using a Virtuoso server. HTTP traces collected by the transparency plug-in are sent to the data server using RESTful web services. Exemplary analyses are created to demonstrate the possibilities of the created trust graph. To enable simple adding and removal, analyses are constructed as modules inside an evaluation framework. One such analysis is the calculation of a trust score that's integrated into the transparency plug-in using the REST web service.

The trust score that is pictured by the transparency plug-in informs users about the privacy rating of websites while browsing them. Contrary to existing trust ratings, the trust score is calculated using objective parameters only. However, those HTTP traces merely include a part of possible privacy related information. Created exemplary analyses are capable of pointing out some interesting findings about the Web. Anyhow, the underlying trust graph needs protection from intentional falsifications of HTTP traces.

Statement of Authorship

With this statement I, Malvin Thiel declare, that I have independently completed this master's thesis entitled with "Construction and Analysis of Trust Graphs using HTTP traces".

The thoughts taken directly or indirectly from external sources are properly marked as such. This thesis was not previously submitted to another academic institution and has also not yet been published.

Ahnatal, March 27, 2013

_____

Malvin Thiel

Acknowledgements

I would like to thank many people who helped me completing this thesis. First of all my parents Irmgard and Jürgen Thiel for their support through my whole studies. Without their understanding of the importance of this academic grade, this thesis would not have been possible.

Then Dr. Edzard Höfig, my supervisor, for his valuable support, his helpful suggestions and encouragement for this project. The Software Engineering research group at Freie Universität Berlin for the helpful ideas and motivation they gave me. Prof. Dr. Ina Schieferdecker for making this thesis possible. Tobias Fielitz for his excellent support about the transparency plug-in.

I also have to thank Richard Hancock for his help of getting a custom built version of the Jena framework to run on the Google App Engine. Gerrit Rindermann for bouncing ideas with me. Stefan Macke for his LaTeX template.

Contents

List of Figures

List of Tables

List of Listings

1. Introduction

1.1. Background

Back in the days, when the Internet was only a few years old (1994) and the populace wasn't using it, there was nothing like a policy but only some sort of network etiquette [Cra94] which was meant to "protect" users. In the following few years the size of the World Wide Web grew enormously. Along with the process of growth, privacy issues came up. Motivated from the concerns of online user data collection, several organisations have launched user-empowerment approaches to online privacy. One such approach was the idea of posting privacy policies on any specific website. The user was able to read them and decide based on that privacy policy whether he wants any further interactions. The problem was, and still today is that they rarely get read [Cra02, p. 3]. The WC3's platform for Privacy Preferences Project (P3P) developed a system that empowers the user to automatically establish connections only to websites that match the previously defined privacy criteria of the user [Cra98]. Even though P3P might sound like an overall solution, only 10% of the Web are supporting P3P in 2003 [BCK03] and not more than 15% in 2009 [RDM09]. Today it seems like we are going back to policies based on Internet etiquette like the "Do Not Track" HTTP header which does nothing than kindly ask for not getting tracked [Sch11]. On the other hand tools and browser extensions got developed to prohibit any interaction - and therefore any possibility to get tracked - to untrusted websites or third party widgets (see section 3.4 Existing trustworthiness ratings).

1.2. Motivation

One big reason why browser extensions to prohibit connections to websites that might lower the users online privacy got developed is that users often have no idea about this problem. When they do, they mostly have no idea which websites are potentially dangerous and which are not. A good example is Facebook's WOT rating (Web of Trust; see section 3.4 Existing trustworthiness ratings). It is well known that the usage of Facebook is a big privacy issue for the users [WXG11] [Lee11]. However, it got an excellent rating from users in all categories (including Trustworthiness and Privacy) [WOT13]. That is very deflating since the WOT rating is especially designed to warn users of such sites. Still this fact isn't much of a surprise since most web users haven't got much expertise about privacy topics which makes their rating very subjective.

Other extensions, also with the aim to improve user privacy display any third party widget of the visited website to the user. Ghostery (see section 3.1.3 Techniques against tracking) is one of them. The problem is the relevance of different widgets, because not all have the same impact on the users privacy. An experienced user might permit the one and forbid another, but the big mass can't distinguish between the bulk of widgets out there. Indeed, much more information is applicable when browsing websites including third party widgets than just the visible to the user. Often it's not even possible to see what exactly is included when browsing websites other than with special developer tools and knowledge.

A browser extension that aims to display possible threats of third party widgets just got developed by Tobias Fielitz, an associate at Freie Universität Berlin [Fie12]. Based on the data this extension collects, analyses can be made on the inclusion and privacy violations of third party widgets and its integrating websites. Especially when this data gets collected from all extensions that are installed in the browsers of many users at one central place. The possibilities that data is coming with can be used to provide users with additional information about websites, but also save

them from getting tracked. This thesis is about to discover and analyse the possibilities that arise with the collection of that data.

1.3. Approach

For the construction of the trust graph a data server that is capable of storing arbitrary data collected by the transparency plug-ins of many users is needed. By demand, this data server should be using the Google App Engine combined with any RDF storage (see chapter 2 Background Information for the final runtime environment). After those HTTP traces are altogether stored into the trust graph, an evaluation of the graph designates possible analyses. A couple of exemplary analyses which are mainly based on SPARQL queries demonstrate the opportunities of the trust graph. A framework where analyses can be embedded into, enables a simple way of adding, modifying and removing them. The calculation of a trust score enhances the awareness of privacy to users of the transparency plug-in. The transparency plug-in is modified for the retrieval and display of this trust score towards any browsed website.

2. Background Information

This chapter explains and describes technical background knowledge about technologies whose familiarity is necessary for the comprehension of this thesis. A reader that is confident about the technologies described may skip this chapter but consult it when necessary.

Furthermore, information about the technology selection process is covered in section 2.5.4 RDF frameworks.

2.1. Hypertext Transfer Protocol

The world's web browsers, servers, even related web applications talk to each other through the Hypertext Transfer Protocol (HTTP). HTTP is based on a question-answer model where a client is always asking a server for something that the server responses. To provide reliability, the HTTP does not come with a bunch of details about how this is realised, it just makes use of the Transmission Control Protocol (TCP) for any connection made [GTS+02, p. 11].

A common HTTP connection sequence could look like figure 2.1.

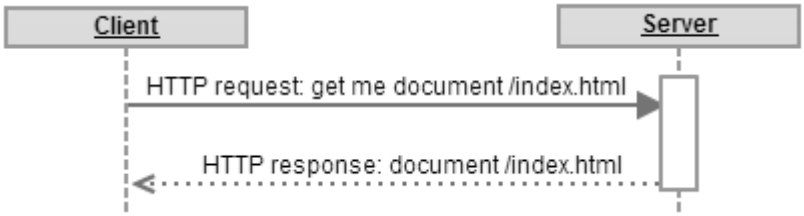

Figure 2.1.: Simple HTTP communication

A client is making a HTTP GET request to a server for a document called "index.html". When the server has this specified document and wants to return it to the client, a positive HTTP response including the

desired document is sent. Every HTTP request must specify its goal. The most common case would probably be the retrieval of a document. But there are several other methods like deleting or creating documents. To determine whether the server is returning a positive or a negative response to the client, status codes are sent within each HTTP response. Numerous codes exist to describe possible response situations in detail (e.g. 200 - OK, 404 - Not Found, 500 - Internal Server Error). All possible status codes and their exact meaning are defined along with every existing HTTP method in RFC 2616 [F+99].

In addition to these status codes every response and request can also have numerous header variables to provide the server with more detailed information about a request than specified in the request method. Those variables are also used to provide the client with details about the transmitted document.

The whole transmission of an simple HTTP request and response to "`http://spiegel.de/index.html`" could look like listing 2.1.

```
1  # Requets #
2  GET /index.html HTTP/1.1
3  Host: www.spiegel.de
4  User-Agent: Mozilla/5.0 Gecko/20100101 Firefox/15.0.1
5  Accept: text/html,application/xhtml+xml,application/xml;q=0.9
6  Accept-Language: de-de,de;q=0.8,en-us;q=0.5,en;q=0.3
7  Accept-Encoding: gzip, deflate
8  Connection: keep-alive
9
10 # Response #
11 HTTP/1.0 200 OK
12 Date: Tue, 25 Sep 2012 13:00:53 GMT
13 Server: Apache-Coyote/1.1
14 X-Powered-By: Servlet 2.4; JBoss-4.0.3SP1/Tomcat-5.5
15 Cache-Control: max-age=120
```

```
16 Expires: Tue, 25 Sep 2012 13:02:54 GMT
17 X-Host: lnxp-2863
18 x-robots-tag: index, follow, noarchive
19 Content-Type: text/html;charset=ISO-8859-1
20 Vary: Accept-Encoding
21 Content-Encoding: gzip
22 X-Cache: MISS from lnxp-3950.srv.mediaways.net
23 X-Cache-Lookup: HIT from lnxp-3950.srv.mediaways.net:100
24 Via: 1.1 www.spiegel.de, 1.0 lnxp-3950.srv.mediaways.net
25 Connection: close
26 [... content]
```

Listing 2.1: HTTP stub of `http://spiegel.de/index.html`

The listing has two parts, the request and the response. In the first line of the request, the clients goal is specified: GET document /index.html as well as the used protocol version (HTTP 1.1). Numerous HTTP header variables are sent to provide information about the client what could help to provide the best suited response possible. Example: the header field "Accept-Language" shows the server that the client would like the requested resource only in one of the specified languages. The "User-Agent" information can be used to provide responses optimized for specific user agent (e.g. web browser or smart phones).

The other part of the listing is the HTTP response of the server. The first line indicates the protocol version (HTTP 1.0), the status code and a status message which describes the status code (200 - OK). After a couple of HTTP headers the actual document is transmitted which is not shown in the example. Anyhow, the response headers can tell many things about the server and the requested document; for example:

- **Server:** The server is using the software Apache-Coyote in version 1.1.

- **X-Powered-By:** Apache-Coyote is running a JBoss Application Server with an Tomcat servlet container which tells us the requested page is probably generated with Java.
- **Cache-Control, Expires:** Tells how long this document is valid and whether it could be cached.
- **Content-Type:** The document type is HTML and is encoded with the charset ISO-8859-1.
- **Content-Encoding:** The transmitted document is encoded with gzip.

There are much more standardised header fields which are not used by this response, but applications are also free to invent their own home-brewed headers. All of those can be classified into the following five categories [GTS+02, p. 51]:

1. **General headers** - Can appear within the request and response message
2. **Request headers** - Provide more information about the request
3. **Response headers** - Provide more information about the response
4. **Entity headers** - Describe body size and contents, or the resource itself
5. **Extension headers** - New headers that are not defined in the specification

2.2. Google App Engine

Charles Severance author of the Book "Using Google App Engine" described the GAE with the following few words:

> Google's App Engine opens Google's production infrastructure to any person in the world at no charge. [Sev09, p. 6]

Actually the GAE provides a runtime environment inside the Google's data centers (the Google cloud). It lets the developer deploy their application on it without telling much details about the insides but that it will run reliable with high performance. Once deployed in the GAE it is impossible to tell exactly where the application is running. The Google Cloud decides which instance in what Google data store is used to execute a request [Sev09, p. 11]. That means a request to an application "example.appspot.com" from the eastern United States might get one numeric IP address, and in south Africa, a totally different numeric IP address.

Popular applications might run in a number of different data stores at the same time. Other non popular ones are probably not running anywhere most of the time. Anyhow, the application itself has no idea if or where it is running. Which is fact is a very nice feature about the GAE since it hides all those details completely from the developer by promising reliability at the same time.

The Google App Engine provides several possible runtime environments for applications (i.e. Java, Go, Python). The Java runtime environment is probably the most used one. But also any other programming language that compilse to, or run in the Java Virtual Machine can be used, such as PHP (using Quercus), Ruby (using JRuby) and others [San09, p. 3].

All programs running in the GAE sandbox have their access to the environment managed by the Java Datastore API (including Java Persistence API and Java Data Objects) [Goo12c]. Since some time ago, this API controls the whole persistency within the App Engine. Normal file access like on conventional JRE's is not allowed in the cloud. Now, the App Engine provides the developer with two more storage methods: The Google Cloud SQL and the Google Cloud Storage. The Cloud SQL which is still pretty new provides the App Engine application with a relational SQL database which is based on the MySQL database system. The Cloud Storage which is still experimental and therefore under heavy develop-

ment provides a storage service for objects and files up to the huge size of terabytes.

However, the restrictions of the file access is not the only thing an App Engine developer has to live with. The App Engine restricts the JRE (or Java standard library) to a limited set of classes, the JRE Class White List (`https://developers.google.com/appengine/docs/java/jrewhitelist`).

The GAE was set to be the chosen runtime environment for this project by my supervisor, Dr. Edzard Höfig. Therefore tests were made to get some hello world programs to run. After this first step, an RDF framework needed to run there as well. The open source Semantic Web framework for Java, Jena, was chosen to be the most suited one. After some trouble making it runnable on the GAE with the precompiled version and with some excellent tips from Richard Hancock about a custom built version, this problem was solved.

The next step was the data storage. The following options were available [Goo12c]:

1. The **App Engine Datastore** provides a NoSQL schemaless object datastore, with a query engine and atomic transactions.
2. **Google Cloud SQL** provides a relational SQL database service, based on the MySQL relational database management system.
3. **Google Cloud Storage** provides a storage service for objects and files up to terabytes in size.
4. The **Memcache Java API** provides high performance distributed in-memory data cache.
5. An **external** data store provider that is accessed through the GAE.

The Memcache is not suitable since it isn't a permanent storage method. A problem with the other internal stored methods is the restriction of the Java programming language by the GAE. Actually it is not possible

to use java.lang.Thread to create new threads on the GAE [San09, p. 89]. Threads are necessary to run an RDF Framework on it.

Another restriction that the GAE comes with, are connections.

> App Engine only supports web requests via HTTP or HTTPS, and email and XMPP messages via the services. It does not support other kinds of network connections. For instance, a client cannot connect to an App Engine application via FTP. [San09, p. 13]

That implies that it is not possible to create a custom socket connection. This limitation together with the constraint about the threads makes it nearly impossible to get any RDF framework to run neither to connect the GAE to an external one. Using the SPARQL protocol via HTTP would have been an option, but the RDF frameworks also make use of threads. A custom implementation or a complex and time-consuming adaption of the framework code would have been necessary which was neglect in comparison of the advantages the GAE comes with.

Therefore an alternative needed to be chosen. But a suitable runtime environment needed to fulfil a couple of requirements:

- A possible programming language had to be Java
- The chosen RDF framework Jena needed to be runnable on it
- It must be free for at least academic projects

The next section contains further information about the chosen runtime environment.

2.3. Google Web Toolkit

The Google Web Toolkit (GWT) got unveiled to the unsuspecting public in 2006 at the annual JavaOne conference in San Francisco. The main purpose of GWT is to solve the problem of direct Asynchronous Javascript

and XML (Ajax) development which normally is very complicated and tough to debug. The second speciality about it, is the fact that the whole Web application (client and server part) is written in Java and gets compiled to HTML and Javascript code. That hides all browser incompatibilities from the software developer which are always problematic when developing Web applications that should be runnable on multiple browser-platforms. It also allows the programmer to work in a familiar Java development environment [Bur06]. New versions of GWT also allow complete debugging of the application which is very useful when developing Ajax applications. The fact that far more developers know the Java programming language than Javascript and that the GWT development and debugging is possible with famous integrated development environments like Eclipse or NetBeans it became very famous over the last years. Even though a Java developer doesn't know how to create Web applications, GWT gives them the opportunity to create Web applications very similar to Swing applications (visual components), setting up event handlers, debugging, and so forth - all within their familiar IDE. GWT also provides the developer with an abstracted version of the Document Object Model (DOM) that hides all differences between browsers behind easy to extend object-oriented user interface patterns.

But the usage of GWT also comes along with some disadvantages. Since GWT compiles Java code into Javascript that runs in the clients browser it is not hard to imagine that this Java code might not be as powerful as native Java code can be. That's why Google introduced some restrictions that apply on the GWT classes. GWT's Java Runtime Environment (JRE) Emulation Reference [Goo12b] starts with the following consideration:

> Google Web Toolkit includes a library that emulates a subset of the Java runtime library. [...] GWT supports only a small subset of the classes available in the Java 2 Standard and Enterprise Edition libraries, as these libraries are quite large and rely on functionality that is unavailable within web browsers.

It doesn't seems that Google tries to change these compatibility issues in the future since right below that last information they give nothing but the following hint for not relying on unsupported classes.

> You will save yourself a lot of frustration if you make sure that you use only translatable classes in your client-side code from the very beginning. To help you identify problems early, your code is checked against the JRE emulation library whenever you run in development mode. As a result, most uses of unsupported libraries will be caught the first time you attempt to run your application. So, run early and often.

Unfortunately those not supported classes are not only super rarely used ones. For example the Java 2 Standard class **Hashtable** is an often used but not supported class of the GWT runtime environment. Not only the fact that several classes are not supported, also the whole Java type system has changes with the browser restrictions GWT comes with. Of course GWT's Java is basically the same than the normal Java, but there are a few differences a developer should be aware of when using it. Specially tricky data type gadgetry might be a very bad idea since the primitive type system comes with some caveats. Integer overflows are behaving pretty different in GWT than in the Java 2 Standard Edition. The 64-bit integer long is completely emulated by a pair of 32-bit integers which can result in heavy performance impacts due to the underlying emulation. Operations on float are the same than on double and result in higher precision. Anyhow developers should be aware of these restrictions and changes from the Java Standard Edition. See the GWT Coding Basics - Compatibility with the Java Language and Libraries for further details [Goo12a].

These restrictions might turn out to be some sort of trouble maker during the development, but in comparison to the GAE its at least possible to use Jena, the chosen RDF framework on it (see section 2.5.4 RDF frameworks). Also, GWT does not only fulfil all requirements of a possible runtime environment it also provides a very comfortable way of creating

Web applications compared to other similar technologies like Java Server Pages. Therefore the chosen development platform became GWT.

2.4. Metadata

Whatever is returned when de-referencing an URI has several names. Since many things on the World Wide Web are often human readable documents it is often referred as a document. Formally it is called a resource which can be everything addressed by an URI (see RFC 3986). These resources can be described using Metadata.

In this thesis the term Metadata is used in the context of the Semantic Web movement towards a perspective of a Federated Knowledge Base [MS03] where it can be understood as the following:

> Metadata is machine understandable information about web resources or other things [BL97]

The main statement in that sentence is the fact that it is machine understandable. Often it is only referred as machine readable which makes a big difference between understanding and reading. The whole World Wide Web, which is probably one of the biggest knowledge Database humans every created, is almost completely machine readable. The common Web format HTML is of course machine readable. The difference between readable, when a Web-Browser parses and displays a website and the understanding when a search engine knows what that specific website is actually about, is enormous.

The format of how Metadata is stored can vary depending on the specific type of content to describe. The place where to store Metadata can also be very different since it can be stored either external or internal. When stored internally the syntax of the Metadata has to fit into the document's syntax. Also when stored externally a link between the resource described and the Metadata needs to be established. An example of how to store Metadata is shown in section 2.5 Resource Description Framework.

2.5. Resource Description Framework

One of the main goals of Metadata is the retrieval of information. A typical example for this usage are the information systems in libraries where the librarian can tell where to find a book depending on the Metadata stored in those systems (conventionally card files; today computer aided information systems). How Metadata can be used to retrieve information out of such systems is defined in vocabularies. They can tell computers where the information about Author and Title is found and stored in Meta-information systems.

The Resource Description Framework (RDF) is such a system for relocating data. As the name implies it is a system for describing resources. These resources are described with Metadata and the vocabularies mentioned before. The structure of RDF is build on the following rules [Bra01]:

1. Resources can be anything as long as they can be identified with an URI - therefore pretty much everything from Web pages to specific elements on them.

2. Named properties which are defined in resources. An example of such a property would be Author.

3. Statements are the combination of a resource, a property and a value. Also known as the subject, predicate and object of statements. A simple example of such a statement would be: The **Author** of **urn:nbn:de:kobv:83-opus-30657** is **Edzard Höfig**.

Apart from these rules, RDF is designed upon a few criteria.
First of all it is **independent**. Since properties are nothing more than resources, everyone can invent their own. Because RDF doesn't come with a default set of properties it is also necessary that everyone creates their own properties. The fact that those created properties are resources itself described with an URI enables everyone else to use of them.
The format of RDF is **interchangeable** and not based on a specific

representation. The most common one is probably XML, but others like Notation 3 are existing.
By keeping in mind that RDF is often used to create a semantic Web and the fact that the Web itself is a gigantic database, it is very handy that RDF is designed to be highly **scalable** with its simple format of three-part records [Bra01].

2.5.1. RDF Graph

RDF can be used to represent information as a graph. This graph contains objects along with connections between them. Figure 2.2 shows two graphs where the first one contains ages of persons and the second one the best friend relationship.

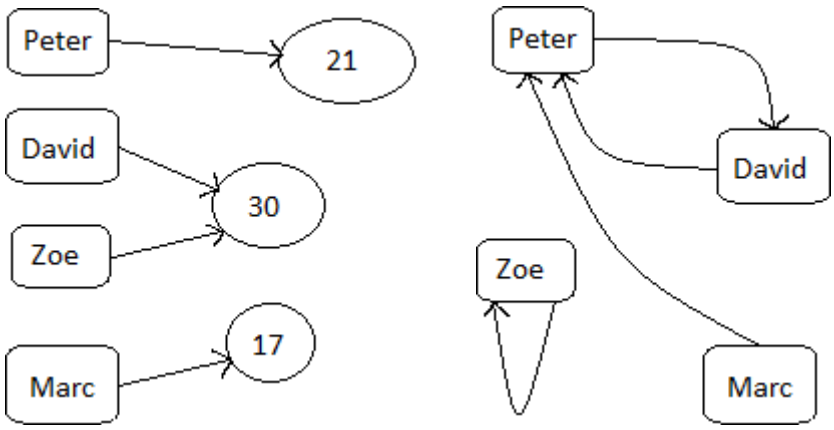

Figure 2.2.: RDF graphs 1 and 2: Person ages and friends

The information on the two previously shown graphs is combined into figure 2.3 using named edges.

This graph can then be read as "Zoe is 30" or "Marc's best friend is Peter". Each edge is like the predicate of a triple connecting the subject and object with each other so that the whole graph represents a set of statements. RDF graphs are not necessarily complete graphs, in fact rarely they are. Therefore every information on that graph could be left out.

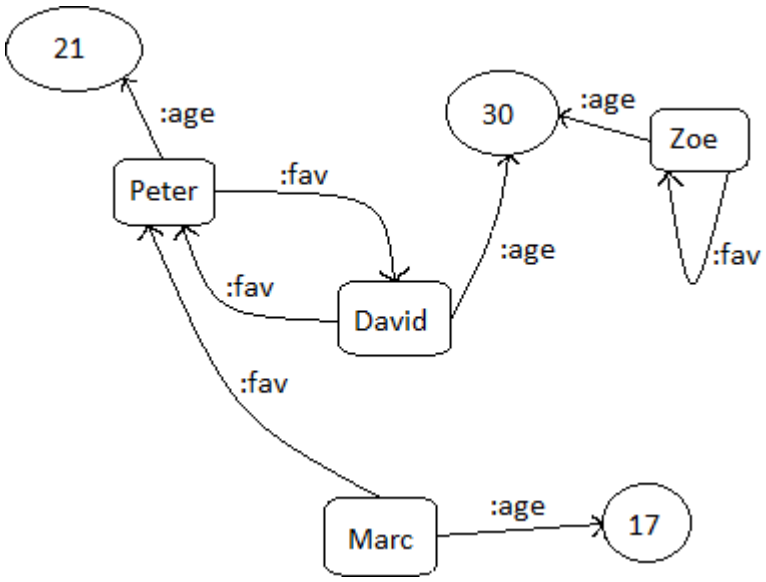

Figure 2.3.: RDF graph 3 (1 and 2 combined): Person ages and friends

David's best friend or Zoe's age aren't included in the graph. Those are problems the Web comes with, but RDF is capable of handling them.

RDF is based upon such graphs, but as mentioned before the final format is not defined by RDF itself. To exchange RDF data with end points, a serialisation needs to take place so that generated triples contain the mentioned subjects, predicates and objects. The predicates of figure 2.3 aren't containing complete URI's but only names like ":age" or ":fav". These names are similar to types in XML where the complete type can be abbreviated by using namespaces. Before serialising the RDF graph those types need to be substituted with the complete URI of the predicates [Gro09]. For this example the properties are defined in "http://example.com/definitions" which is used as the prefix example. The substitution then is shown in table 2.1.

2.5.2. RDF-XML

RDF can be displayed in different ways using XML. The W3C RDF/XML Syntax Specification from February 2004 recommends several techniques

Subject	Predicate	Object
"Peter"	example:fav	"David"
"Peter"	example:age	"21"
"David"	example:fav	"Peter"
"David"	example:age	"30"
"Marc"	example:fav	"Peter"
"Marc"	example:age	"17"
"Zoe"	example:fav	"Zoe"
"Zoe"	example:age	"30"

Table 2.1.: RDF graph 1 and 2 combined and serialised to triples

to do so. Two examples should demonstrate the RDF-XML persistence. One is written with XML entities the other one with XML attributes. The first way is shown in listing 2.2.

```
1  <rdf:RDF
2      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3      xmlns:example="http://example.com/definitions#">
4
5      <example:Person>
6          <example:name>Peter</example:name>
7          <example:age>21</example:age>
8          <example:fav>David</example>
9      </example:Person>
10 </rdf:RDF>
```

Listing 2.2: RDF as XML entities

The information shown in this example is actually equal with three triples of table 2.1. Alternatively the XML can be reduced to display only one single entity (i.e. the name). That other possibility, to write RDF as XML attributes is shown in listing 2.3.

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:example="http://example.com/definitions#">

    <rdf:Description
        example:name="Peter"
        example:age="21"
        example:fav="David"
    </rdf:Description>
</rdf:RDF>
```

Listing 2.3: RDF as XML attributes

This XML listing contains the same information as the other XML representation but stores the data as attributes in the RDF predefined tag description. Anyhow, all possible RDF-XML persistence methods lack of performance but come with high redundancy. Other exchange formats for huge amounts of RDF triples are getting specified. One example is the ongoing doctoral thesis of Javier D. Fernández which tries to address efficient formats for publication, exchange and consumption of RDF on a large scale. The main issue is the format RDF is stored; Fernández is using an binary serialization format for RDF called Header-Dictionary-Triples (HDT) in combination with compressed rich-functional structures which take part of efficient HDT representation. Right now the HDT format has been accepted as a W3C Member Submission [Fer12].

2.5.3. RDF-Schema

The Resource Description Framework Schema (RDFS) is used to describe the syntax of RDF models. This is of particular importance when interchanging data and can be compared with XML Schema. Anyhow, RDFS is made to specify the syntax only. For the enrichment of semantic, a common known vocabulary has to be defined (e.g. Dublin Core [Mes07,

p. 4-5]). Another method would be the definition of the syntax together with the semantic by creating an ontology (i.e. using OWL [MKR04]).

The following listing is an example of how RDFS can be used to syntactically describe the RDF graph shown in figure 2.3. In other words, it can be treated as the "example"-namespace definition in listing 2.3.

```
1  <?xml version="1.0"?>
2  <rdf:RDF
3  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
5
6  <rdfs:Class rdf:ID="Person" />
7
8  <rdf:Property rdf:ID="name">
9      <rdfs:domain rdf:resource="#Person"/>
10     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema
           #string"/>
11 </rdf:Property>
12
13 <rdf:Property rdf:ID="age">
14     <rdfs:domain rdf:resource="#Person"/>
15     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema
           #int"/>
16 </rdf:Property>
17
18 <rdf:Property rdf:ID="fav">
19     <rdfs:domain rdf:resource="#Person"/>
20     <rdfs:range rdf:resource="#Person"/>
21 </rdf:Property>
22
23 </rdf:RDF>
```

Listing 2.4: RDF-Schema describing figure 2.3

2.5.4. RDF frameworks

Apart from these RDF exchange formats are RDF stores (also known as triple stores) that keep the RDF data in a high-performance way to access, manipulate and find triples. The ease of exchangeability is not goal of these stores (comparable with the storage of SQL databases and the backups of them). RDF stores are either used as a middleware or stand-alone systems which provide access to other applications or users through APIs or particular query languages. The persistence of the RDF data is often adapted from relational databases or graph storage methods.

All common RDF store middlewares that are written and accessible with the Java programming language are shown in table 2.2.

Name	**Year**	**Storage**	**SPARQL**	**Licence**
OntoBroker [Ont12a]	N/A	Extern	Yes	Commercial
Jena [CDD+04]	2000	Extern	Yes	Free (Apache)
Sesame [BH01]	2001	Extern	Yes	Free (GNU)
Kowari [WGA05]	2005	Intern	Partly	Free (Mozilla)
YARS [HD05]	2005	Intern	No	Free (BSD)
SwiftOWLIM [Ont12b]	2005	Intern	Yes	Commercial
RDFBroker [SK06]	2006	Extern	No	Free (GNU)
OpenAnzo [The12]	2007	Extern	Yes	Free (Eclipse)
Bigdata [SYS12]	2008	Intern	Yes	Free (GPL)
Oracle 11g [Ora12]	2009	Intern	Yes	Commercial
Smart-M3 [HLBT10]	2010	Intern	Partly	Free (BSD)
CumulusRDF [LH11]	2011	Intern	Partly	Free (GNU)
TripleCloud [GKG11]	2011	Extern	Yes	N/A
Stardog [Cla12]	2011	Intern	Yes	Commercial

Table 2.2.: Triple store middlewares useable with Java

The table is ordered by the publishing year. The storage column shows if the RDF is stored internally or if an external storage provider can be used or is needed. Apart from that is displayed whether the middleware supports a SPARQL interface as well as if its freeware or not. External storage is often the case when the framework itself makes use of an re-

lational database system which does not provide SPARQL support. But also to provide high exchangeability through the data layer below.

Two of the oldest, free of cost and promising looking frameworks with an active developer community were looked up in detail: Jena and Sesame.

2.5.4.1. Jena

Jena is one of the leading Semantic Web toolkits for Java developers that was first released in 2000. Jena2, (which is just named Jena in the rest of this document) which is the newest vesion of Jena was introduced in August 2003 with an revised internal architecture and many new features. Due to the fact that the main concept of the Semantic Web recommendation is the RDF graph as a universal data structure which is simply a set of triples, Jena similarly has the graph as its core interface where other components are built around [CDD+04].

Jena was built to fulfil two key architectural goals:

1. Multiple as well as flexible presentations of RDF graphs to the application developer which allows graph data to be accessed and manipulated through high-level interfaces.
2. A very simple and minimalistic view of the RDF graph to the developer who wishes to manipulate data triples.

A complete architecture overview is shown in figure 2.4.

One pretty useful design issue about Jena is the Model API which abstracts the whole graph data from the underlying data store. Therefore Jena can be used either with one of Jena's provided data store mechanisms, completely in-memory or with any external data store that is compatible with Jena's Model API (store API options in the figure).

Jena provides two ways of accessing the data store. Either by direct Java invocation or via the Fuseki HTTP server. The direct Java invocation access is provided by the Jena RDF API. Using the HTTP server to

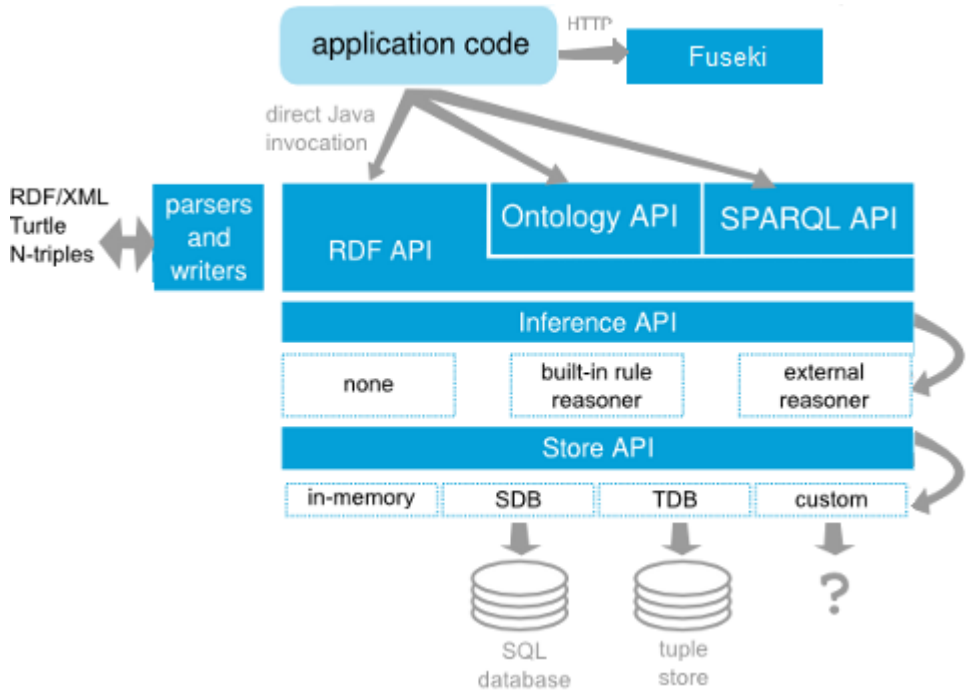

Figure 2.4.: Jena architecture overview [Jen12]

access or alter data in the RDF store works via an Representational State Transfer (REST) style API which makes use of HTTP methods like PUT, POST or DELETE.

By using the RDF API from direct Java invocation different parsers are provided by Jena to handle incoming and outgoing streams of RDF data in formats like RDF/XML or N3.

2.5.4.2. Sesame

Sesame was created to provide an architecture for efficient storage as well as excessive querying of large quantities of RDF data. It was developed by Aidministrator Nederland b.v. as part of the European IST project On-To-Knowledge [BH01].

Sesame is a Web-based architecture that allows persistent storage of RDF data together with RDF schema information. The online querying of that

information together with an overview of the Sesame's architecture is shown in figure 2.5.

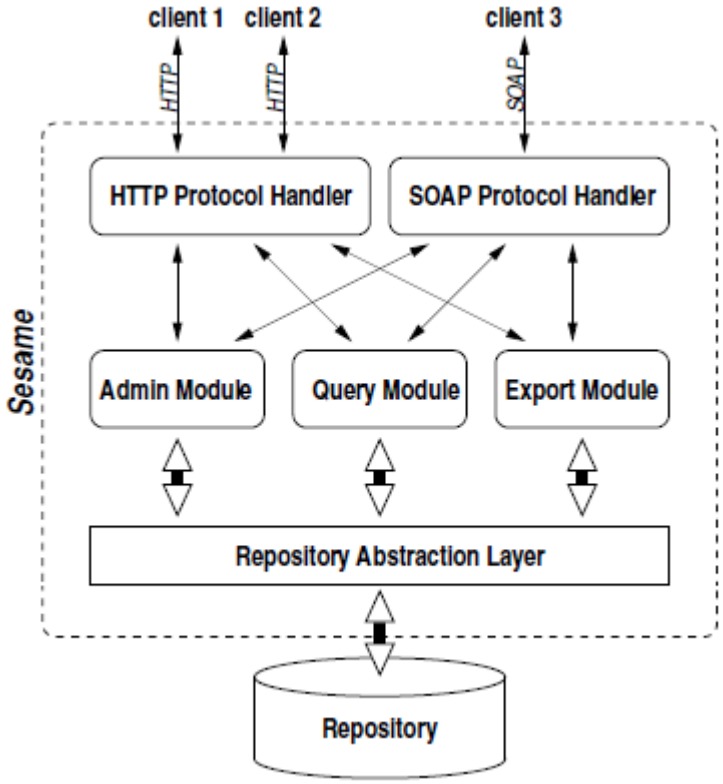

Figure 2.5.: Sesame architecture overview [BH01]

The underlying Repository Abstraction Layer (RAL) provides Sesame with the same possibility as Jenas's Model API - an exchangeable scalable repository for the data storage. Thus Sesame can make use of any kind of data store for which an implementation of the RAL (such as Relational database management systems) exist. Sesame's functional modules are designed to be clients of the RAL. Originally Sesame came with three of those modules [BH01]:

- **The RQL query module.** A module for RQL query evaluation posed by users.
- **The RDF administration module.** An interface for incremental uploading and deletion of RDF data and schema information.

- **The RDF export module.** This module allows the extraction of schema information together with data in various RDF formats.

Sesame supports different ways of access to the modules which finally provide access to the RDF stores. Sesame supports an HTTP REST interface for accessing data which might be the preferred method in a Web context. It can also be used via Java's Remote Method Invocation (RMI) method or with other Web requests methods. For example from a .NET platform like C# via the Simple Object Access Protocol (SOAP).

2.5.4.3. Selecting a framework

As shown, quite a lot RDF Frameworks are existing today. Table 2.2 shows all well known frameworks that are useable within the Java programming language. A first selection was made by dropping all commercial ones which reduces the list of OntoBroker, SwiftOWLIM, Oracle and Stardog. The CumulusRDF and TripleCloud projects which were both introduced in 2011 ([LH11], [GKG11]) as mostly academical research results/proposals that are not of practical relevance for the moment since no implementation is available. Another criteria of the chosen framework was fully SPARQL support. The SPARQL is emerging as the de-facto standard for RDF querying and is also a W3C recommendation ([Wor08]). Since YARS only supports N3QL which is classified being not longer the state of the art, it got dropped as a framework candidate. The same applies to RDFBroker which only supports specially developed Java querys. Smart-M3/Kowari only support some parts of SPARQL. Since the Google App Engine doesn't allow direct file access it was important that the selected framework supports external RDF stores as the underlying infrastructure to be runnable on the GAE. Bigdata's storage method is managed internally and is therefore not suited for this project. OpenAnzo itself is runnable on the GAE, but none of its supported database systems is.

Both of the remaining frameworks (Jena and Sesame) seemed to be pretty solid and reliable in their development state and community activity. Benchmarks such as the Berlin SPARQL Benchmark ([BS09]) won't help in this decision since they only compare some of Jena's internal storage methods with Sesame's internal storage method and not about the frameworks itself. Anyway, small performance issues are not of big importance in this mainly academic work. Therefore my supervisors software engineering working group has been asked for the better framework and Jena was the collaborated answer.

The selection of the data store itself is anyhow not as important as the selection of the RDF framework and the runtime environment. Pretty much the only requirement was the opportunity to be runnable with the selected RDF framework. Apart from that it could run on the same server, be completely externally or even hosted by third parties.

Since the requirements to the data store are quite low, a lot of possibilities are available. My supervisor Edzard Höfig made the decision to Virtuoso.

2.5.5. SPARQL

SPARQL is a recursive acronym for SPARQL Protocol and RDF Query Language. SPARQL is only one of many existing query languages for RDF but its the recommended language for RDF querying of the W3C. Also nowadays, most triple stores support SPARQL (see also table 2.2) endpoints [HFB+11]. Even if SPARQL is a query language as well as a protocol, only the language is explained here (the protocol describes how a client talks with a SPARQL endpoint).

The query language is basically pretty similar to SQL. A small example in listing 2.5 shows a simple select statement in SPARQL.

```
SELECT ?name
WHERE {
    ?name <http://example.com/definitions#fav> "Peter"
}
```

Listing 2.5: SPARQL select: everyone who is best friend with Peter

Note that in contrast to SQL there is no table specified where to select from. That is not necessary because the whole query is performed on the graph which was selected by connection instantiation. Terms delimited by "<>" are parsed as URIs whereas everything delimited by double quotes("") are literals. Integers or floating points can be written without any enclosing. Variables start with an "?" (alternatively with "$") and have global scope in the whole query. Since URIs can be quite long, SPARQL provides an abbreviation mechanism (PREFIX). See the W3C SPARQL specification [Wor08] for more details.

The first version of SPARQL didn't come with a query construct to insert data. Therefore most SPARQL frameworks provide other mechanisms to insert data - mostly unconditional and straight forward (i.e. graph.add(new Triple(s, p, o));). Anyhow, with the latest version of the SPARQL specification [GPP13] it is now possible to use an "INSERT" statement for the insertion of triples into specific graphs. See listing 2.6.

```
PREFIX example: <http://example.com/definitions#>
INSERT INTO GRAPH <graph1> {
    "David" example:fav "Peter"
}
```

Listing 2.6: SPARQL insert: Davids favourite friend

The mentioned abbreviation mechanism is used with the keyword "PREFIX" at the beginning of the query. The prefix statement can be repeated

(also to replace an existing prefix) and is directly in effect after its written.

However, the SPARQL is still in an active development state and the release of new versions that support new functionality is very probable.

2.6. Transparency plug-in

The data this work is using and analysing is collected by a browser plug-in [Fie12] that got developed in the diploma thesis of Tobias Fielitz at Freie Universität Berlin in October 2012. The plug-in collects and virtualises all data traffic that is generated by the web browser whenever the user is navigating to any website. Possible data to analyse can be pretty much everything that is transferred by browsing websites (see chapter 2.1 Hypertext Transfer Protocol for details). Anyhow the plug-in displays only a very small number of that data to the user. Mostly what other websites are connected with the current one, what companies are involved, the country the websites are hosted in and the cryptography level of the transfers. A request to the website `http://www.spiegel.de/` could produce the visualisation as shown in figure 2.6.

The overview shows the following information, marked with a small red number next to it.

1. Icon of the plug-in extension
2. Number of requests
3. Icon of the website
4. Information about a third party website
5. Information about the requested website
6. Name of the company
7. Domain

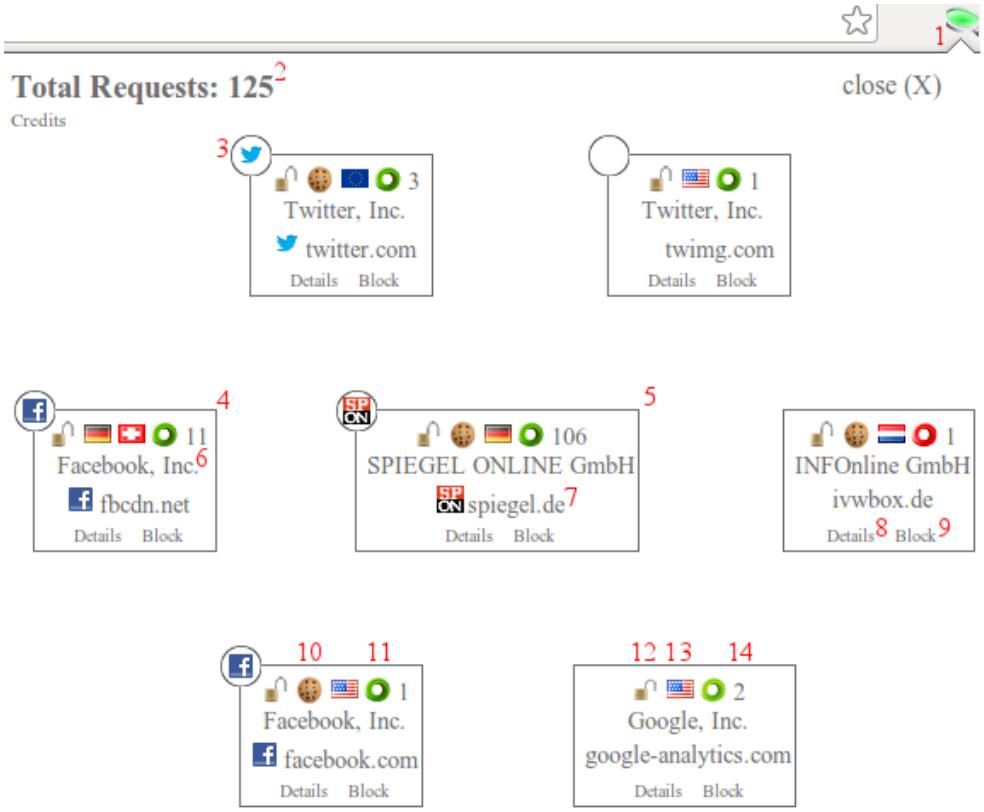

Figure 2.6.: Transparency plug-in

8. Link to show all requests of the domain
9. Link to block everything from this specific company
10. Shows whenever cookies are set or not
11. Web of Trust Rating
12. Shows whenever the communication is secured
13. The flag of the country the website is hosted in
14. Number of requests from that domain

All those information and more are sent frequently and anonymised for data storage to the trust graph explained later on.

3. Analysis of existing literature

This chapter discusses past and current researches about selected important topics regarding this thesis. The general issues of user privacy, third party trackers, analyses of tracking behaviour and its outcomes in form of trustworthiness ratings aren't new. Therefore numerous research is done with the expansion and the increasing popularity of the World Wide Web.

3.1. Privacy concerns of third party widgets

Whenever a user is browsing to a specific website he isn't afraid of the fact that this website knows what he is watching on there - of course not. What this user doesn't know is that most commercial websites include several third party widgets where each of them tracks an estimation of more than 20% of a user's browsing behavior [RKW12]. Since most websites include trackers and some trackers are very popular, a small number of companies are able to track users movement across almost all of the popular Web sites [KW09].

Privacy issues in computer systems are nothing but new. Already in the 70s Denning et al. described "white-collar criminals" as men who learned stealing only with the help of a computer by taking much less risks than conventional thieves [DD79]. Even though concerns about user privacy have risen dramatically with the increasing dependency on the Internet within the last years nothing very effective happened to protect the user privacy (see section 3.1.3 Techniques against tracking). One reason why this process isn't making much of a progress is of course the lobby of the big companies who want to track users, but not to disregard is the fact that even when the average internet user has heard about those privacy issues he doesn't care enough about it [Kri01, p. 164]. This discrepancy

between expressed privacy concerns and actual disclosure behavior is also known as the privacy paradox [OG11]. It seems that every user has his own personal definition of privacy which might also differ from time to time. One definition of privacy taken from literature is the following.

> Privacy is the interest that individuals have in sustaining a "personal space", free from interference by other people and organisations. [Cla06]

By having third party widgets included into websites it's pretty obvious that other people and organisations can interfere users personal space. The process of interference is enabled through third party tracking. A very good explanation of how this process works is given in the following citation.

> Third-party web tracking refers to the practice by which an entity (the tracker), other than the website directly visited by the user (the site), tracks or assists in tracking the user's visit to the site. For instance, if a user visits cnn.com, a third-party tracker like doubleclick.net - embedded by cnn.com to provide, for example, targeted advertising - can log the user's visit to cnn.com. For most types of third-party tracking, the tracker will be able to link the user's visit to cnn.com with the user's visit to other sites on which the tracker is also embedded. [RKW12]

Many different tracking mechanisms exist today. Often, new ways of user tracking are more likely a side effect that comes across with new technologies to improve the user experience. Users often have to sacrifice or at least decrease their user experience for browsing the web with a little more privacy. Anyhow, the problem behind such tracking methods aren't new. Lots of different methods to prevent trackers from doing their job got introduce and developed. Some of them are shown in section 3.1.3 together with an evaluation of their success. Collected information about specific problems and negative aspects are listed combined with current mechanisms of tracking and the way how trackers work nowadays.

3.1.1. Methods of user tracking

Tracking has become more and more important within the last years. Studies have proven that the percentage of websites which analyse their user has increased from 5% in 2006 to about 40% in 2010 [IP11]. To understand the importance of tracking, the ambiguous term needs to be explained first. Tracking methods can be divided into two groups. The technical way of tracking a user and the part to embed third party trackers into a website.

Most methods have one thing in common, they try to link every user together with some sort of unique identifier to be able to follow him through the Web. Other than the methods in the following list are techniques to re-identify a user with for example the comparison of the IP address. The problem with such techniques is that the IP-address changes from time to time. Therefore most ways of tracking a user over a long time are based on storing some unique information on the users computer.

The following list shows and explains the technical tracking mechanisms.

- **HTTP cookie**
  The oldest and most traditional way of tracking users is done by the help of cookies. The cookie standardisation process began in April 1995. Originally cookies weren't invented and used to track people, but rather to maintain some state into the otherwise stateless HTTP (i.e. to build stateful web applications like shopping baskets). As a part of HTTP's response a server may send arbitrary information in a "Set-Cookie" response header (see chapter 2.1 Hypertext Transfer Protocol) which will be stored in the browser and sent back to the server when navigating there once more. [Kri01, pp. 151-153].

- **Local Shared Objects (Flash cookies)**
  Flash cookies are not like the name might divine similar to HTTP cookies. Of course they enable the opportunity to store information on the client side. But in contrast to conventional cookies, flash

cookies are neither stored in browser nor does the browser itself ever get to know about them. The flash runtime environment handles everything regarding to them [SCM+09]. In other words, even when the browser deletes everything that was stored or even when any kind of privacy mode is activated, flash cookies are still there and still completely functioning.

- **Silverlight Isolated Storage**
  Silverlight is the competitor product to flash and comes with pretty much the same method to store information.

- **Google Gears**
  Gears was an open source project by Google to allow web sites the storage of data so that it can be used offline. Gears is not longer in an active developed state.

- **IE-userData**
  Internet Explorer's userData enables authors to specify an object to persist on the client during the current and later sessions. Anyhow, like Google Gears its no longer actively maintained [Lib12].

- **DOM storage (HTML5 cookies)**
  The web storage is an W3C specification from 2009 which is similar to HTTP cookies, for storing structured data on the client side [Hic11].

- **Browser fingerprinting**
  Browser fingerprinting is different to all other methods shown before since it doesn't stores anything on the client site to identify a unique user. A algorithm creates an almost unique fingerprint only out of the information included in every HTTP request a browser sends to a web server. That is possible since current browsers include masses on information about the users system into the HTTP request with the aim to provide certain kinds of debuggability, which in current browsers is weighted heavily against privacy [Eck10].

- **Others**
  Several other techniques are existing to track users. Their difference to the first ones which were officially developed to store data on the users computer is, that these methods make use of implementation gaps or ill-conceived techniques used in the Web. The evercookie [Kam10] is a cookie implementation that combines several different tracking techniques together:
  - Storing cookies in HTTP ETags
  - window.name caching
  - Storing cookies in RGB values of auto-generated, force-cached PNGs using HTML5 Canvas tag to read pixels (cookies) back out
  - Storing cookies in Web History
  - Storing cookies in Web cache

Besides from the technical way of tracking is the embedding part. Most trackers are probably included in a way that isn't visible to the user. That could be an included **Javascript** running in background making all necessary requests to tracking companies. Another way is the **web bug** (well known under several other names like tracking pixel, 1x1 gif or web beacon) which is often nothing more than a small image (i.e. 1x1 pixel) which is included into the tracked website. The inclusion of such a pixel could look like that, when browsing the website http://example-website.de.

```
<img src="http://tracking-company.com/example-website/pixel.gif"
width="1" height="1" alt="" />
```

An inexperienced web user might find nothing problematic with that because its only an image but a versed one knows the threat behind it. At the point the image gets displayed in the browser the tracking has already happened. With the HTTP request to the image, all necessary information to track the user is transferred. The clue behind it, is that

the "image" is nothing more than a complex script that returns an image when finished. Since every web resource (images or HTML pages) are loaded via the same HTTP requests, the same information about the user is transferred to websites as well as to simple "images".

Advertising like **banner ads** is used for tracking. Often inconsiderable looking but absolutely able to do nothing less than what tracking pixel are able to do. It doesn't matter whether the ad is an image, a flash image or just text included via JavaScript.

Social plug-ins which are often even accepted with the user's pleasure are pretty well masked trackers [KPKM12]. Nowadays, a very common one is the social plug-in of facebook. This can be a button, a comment box and all sorts of things (like Twitter and Google+ [CKB12]). In the end they are always included via a small Javascript code that has every information needed to track the user request to the website the like button is included in.

Even though the used techniques and the involved privacy issues might be the same when having two different trackers included, the subjective stance can differ completely. Most internet users aren't aware of the fact that social plug-ins or advertising can track them the same way than Javascript which is specifically designed to track users. So even when users know about the tracking issues they might not want to abstain something.

3.1.2. Privacy threats

A negative issues of tracking is the exploitation of user data against the users will. Even when some users may not see any issues sharing specific parts of their personal information, others feel much less comfortable by doing that. Anyhow, in order to initiate the collection of data, most services are enforced by law to have the user agreed with their privacy policy. Unfortunately in most cases the users are not able to use the service at all whenever they decline. On the other hand, many websites don't even

ask their visitors whether they agree to their privacy policy or not. Quite often, in such cases personal information is extracted without the users consent or knowledge. Understandably, some users feel uncomfortable in such a situation. At the time when a legal framework exists that requires a privacy policy, it is a clear legal violation of the privacy of a web site visitor. Thus the definition of privacy depends on individuals as well as the legal framework in effect [HKS12].

Sometimes it's frightening how much user information trackers are able to collect. With the help of data mining techniques, only the knowledge of a users browsing history and browsing behaviour can tell much information about him. This process is referred to Web usage mining which is further explained in section 3.2.2. Obviously might the obtained information not be necessarily freed from errors since the involved data mining process can generate some. Anyway, often enough the forecast is true. Maybe the most usage of the gathered information is the creation of personalised ads. Popular web sites such as Yahoo! or the Microsoft Network (MSN) automatically provide information addressed to users interests. When it comes to recommendation systems, the probably most popular one is the online marketplace amazon.com where the system analyses past purchases together with other information available to customise recommendations for the user. Commercial search engines have associations with commercial marketing companies to create advertisements that are specific to the user and his recent activities on the Web [EV03, p. 8].

Those referred impacts might actually not even harm the user in first place, they can be seen as helpful and advantageous. Probably nobody actually wants to view ads which are neither affecting nor interesting someone. Another advantage could be the improved user experience while browsing the web. Finding information could be much easier and quicker when your browsing behaviour is known and understood. Well, that was one perception. Is it still wanted when someone else is using the same computer sharing the same product recommendations? Couldn't it be

very private what someone is searching online? The following parts taken from an article of the Forbes magazine will explain: [Hil12]

> An angry man went into a Target outside of Minneapolis, demanding to talk to a manager: "My daughter got this in the mail!" he said. "She's still in high school, and you're sending her coupons for baby clothes and cribs? Are you trying to encourage her to get pregnant?" The manager didn't have any idea what the man was talking about. He looked at the mailer. Sure enough, it was addressed to the man's daughter and contained advertisements for maternity clothing, nursery furniture and pictures of smiling infants. The manager apologized and then called a few days later to apologize again. On the phone, though, the father was somewhat abashed. "I had a talk with my daughter," he said. "It turns out there's been some activities in my house I haven't been completely aware of. She's due in August. I owe you an apology."

The title of this article is "How Target Figured Out A Teen Girl Was Pregnant Before Her Father Did". This is one small example to demonstrate the power and the risk of tracking.

3.1.3. Techniques against tracking

Since tracking and its privacy threads aren't something new, several methods have been introduced to protect the user from unwanted privacy violations. Those methods can be classified into five different groups which are explained in the following.

1. **Plug-ins**
   A popular way of improving online privacy is the usage of browser plug-ins and extensions. Lots of them got developed within the last years, many with the aim to block advertisers which is some kind of improvement, but won't help much against tracking. Others are

specifically designed to display or even block any connection to trackers or anything that could be one.

The **Ghostery** [Gho13] extension is such one. It advertises with the ability to be able to detect over 1200 trackers as well as giving the user the ability to get notified of ad networks, behavioural data providers, web publishers and other companies interested in the users activity. After knowing about them, the software allows a user to block trackers of selected companies. After visiting the website cnn.com, Ghostery found the following invisible inclusions:

- ChartBeat
- DoubleClick DART
- Facebook Social Plug-ins
- NetRatings SiteCensus
- Optimizely
- ScoreCard Research Beacon
- Twitter Button

The problem here is that far too much know-how is needed to be able to decide which one to block and which not.

2. **Disable possible causes**
   Another way would be the deactivation of all possible privacy violating causes. This would include
   - JavaScript,
   - the Adobe Flash Player,
   - Microsoft Silverlight and
   - cookies.

Problematic is that some of the described tracking mechanisms explained in section 3.1.1 would still work. Therefore this solution does nothing more than decreasing the browsing experience.

3. **Delete private data**
   A popular way to cut off trackers is the deletion of all private data saved by the browser. However, as described before, so called flash cookies aren't in control of the browser and won't be deleted then. Special tools for that aim got developed. BetterPrivacy and Glary Utilities Pro are two of those which are able to delete even flash cookies [SCM+09].

4. **Anti-fingerprinting technologies**
   Obviously a powerfully and easy way of reducing the own fingerprint would be the decrease of data that's included. Paradoxically, anti-fingerprinting technologies can be self-defeating when they are not used by enough people [Eck10].

5. **Privacy arrangements**
   A very interesting way to achieve privacy are privacy arrangements. One of them is the HTTP header directive **DoNotTrack** which can be sent together with a request from a web browser to tell the web server that the user don't want to be tracked. Unfortunately, not many of the today's websites are considering that. A study [May12] has shown that from 64 companies, only the half of the actual tracking cookies in place where left dropped after opting out. Even more disconcerting is the outcome of that study. All companies that participate in the self-regulatory Network Advertising Initiative do only pledge to opt out behavioural ad targeting, not tracking.

   Another privacy arrangements is the W3C proposed recommendation **Platform for Privacy Preferences** (P3P). The standard suggests an infrastructure for the privacy of data interchange where websites are able to express their privacy practices in a standardised format which can automatically be read by user agents such as browsers. This simplifies the process of reading policies for the users

> dramatically. Key information about what data gets collected by a website can automatically be compared with the users preferences to disclosure tolerances. This thought has the same vulnerability as DoNotTrack since it must rely on the arrangement without the chance to check it [EV03, p. 7].

One problem lots of those techniques have is the combination of tracking and improved user experience. A user that wants to be able to use facebook's like button won't care of him being tracked by facebook.

3.2. Analysis of HTTP traces

Many analyses of HTTP traces haven been made in the past. The ways of doing so distinguish each other just like the ambitions of each other. The following sections organise existing HTTP analysis by their superordinate targets into five different groups.

3.2.1. Personalised advertising

One big goal of many analysis made in the past aim to a personalise the user's experience of the Web. Which is often used as a synonym for custom-tailored advertising. As mentioned in section 3.1.2, advertising is a big deal of the personalised Web. In contrast to advertising exists the concept of usability improvements which are explained in section 3.2.2. The personalisation of ad recommendations has gain particular interest within the last years, since most internet business models rely heavily on advertising. Anyhow, user profiling is a challenging process. A typical scenario is the following: the user consumes content items (i.e. ads, articles, annotated videos). The textual data of those items are analysed in order to extract its semantic information based on domain ontologies. Improvements can be made by enriching this data with terms in an offline process. This information is then translated into a set of user

preferences which are gathered in an automated semantic user profiling procedure. The calculated user preferences are then matched with the supplied semantically annotated ads to determine the recommendation degree of every single ad. The match confidence degree is then used to rank recommended ads [TMKD09]. The described process architecture is depicted in figure 3.1.

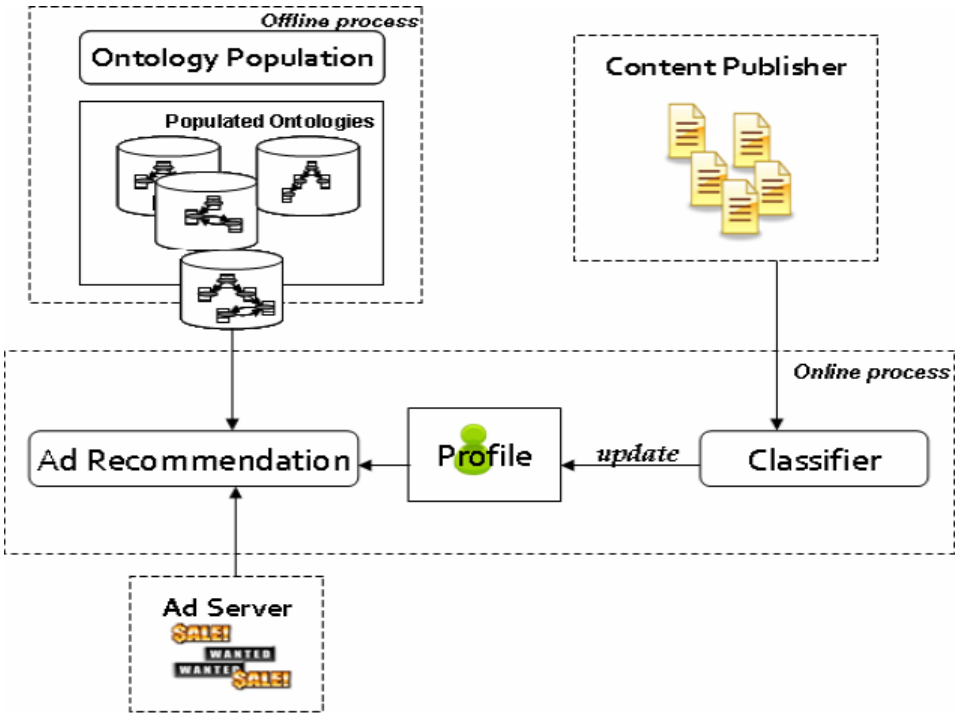

Figure 3.1.: Overview of an Ad Recommendation System [TMKD09]

Such personalised advertisement can be found on many major online markets such as amazon.com or ebay.de. When signed into those platforms it takes a couple of clicks on some products to feed the recommendation algorithms. Afterwards, many similar products to the viewed ones will be recommended to you in the future [EV03, p. 8]. Though, personalised recommendation systems don't necessarily need a user to be logged into their system to create user profiles and generate recommendations for them. Google's search feature is a good example which is explained in the following section.

3.2.2. Usability improvements

Personalising the Web in another way than using the gathered information to improve advertisement is the usability enhancement. Of course, personalised ads can be seen as an improved user experience too. However, the process of information gathering for customising a website to the needs of each specific user or sets of users is referred to Web usage mining which can be regarded as a part of the creation of user profiles. Both are integrated in the Web personalisation process which tries to enhance the users Web experience [EV03]. The Web usage mining process can be divided into three phases [CMS99]:

1. **Data preparation/preprocessing**
   Data cleaning, user identification, session identification and transaction identification is done here. The output gets stored into a database or a data warehouse.

2. **Knowledge Discovery**
   Specialised algorithm are needed (data mining, clustering and classification) for the detection of interesting patterns.

3. **Pattern Analysis**

A famous company makes use of personal data to improve search results. Google's search engine provides every user with a personalised search so that the most relevant results possible will be returned [HK09]. Google explains this feature with the following example:

> For example, since I always search for [recipes] and often click on results from epicurious.com, Google might rank epicurious.com higher on the results page the next time I look for recipes. Other times, when I'm looking for news about Cornell University's sports teams, I search for [big red]. Because I frequently click on www.cornellbigred.com, Google might show me this result first, instead of the Big Red soda company or others [HK09].

Previously Google offered this feature only to signed-in users which makes it pretty obviously that some sort of tracking is involved (see section 3.1.1 Methods of user tracking).

Another way of making use of HTTP traces is the error detection in existing Web applications. Thought, this technique has nothing to do with personalisation, it's a way to improve user experience. Either in a way that an established website won't be offline in the near future due to security problems or so that existing error will be fixed before they restrict users in their doings. Arachne, a prototype for asynchronous policy evaluation is a system to detect various problems in Web applications. Its sensors are the eyes of the system. Such a sensor could parse an access log file where each entry represents a step taken by the web server in response to a request (i.e. HTTP traces). Simplified, those entries then get checked against previously defined policies. Whenever a request won't match, a potential inconsistency is discovered [BK08].

Usability improvements can also be made by analysing the user navigation path of websites. The Google Analytics tool can be used to track and analyse navigation path patterns. The website AfterTheInjury.org made a study about their visitors browsing behaviour with the aim of achieving usability enhancements. The gain knowledge about the top entry pages can provide help for further content optimisations. The same for observation of navigation path patterns which helps people to find out the reason users visit their sites. In general, but also in particular for every entry page. This can help to obtain a more efficient assignment of the content priorities. Either to extend efforts or maybe even delete content or pages [YWT+10].

3.2.3. Pattern recognition on the Web

Different to the Web usage mining process which tries to personalise each users Web experience is the general pattern recognition of the Web. For example, this could be done to gain further knowledge of the existing

Web, making general improvements or answering any kind of academic questions.

A study that investigated on the typical usage of keyword searches across the Web and their names have been made. They tried to find out whether a requested URL (taken from HTTP traces) is containing a search, as well as if they do, the search field and keywords. Its apparently easy to recognise "q" as the query field in "`https://www.google.com/search?q=berlinale`". However it's a challenge when trying to automate this task in the long tail of the Web. Such an analysis performed at sites all across the Web - not just single site like Google could be useful for marketing, building domain specific web directories, discovering competitors or potential collaborators, positioning products and future development [FKR10].

Another study is characterising the organisational use of Web based services. Traces of HTTP activities from a large enterprise and from a large university environment have been analysed to identify and characterise Web-based services. As a first step unique service instances have been identified. They used the HTTP header to identify unique host names (e.g. www.google.com), domains (e.g. goggle.com), brands (e.g. google.com, google.de, and google.co.uk is considered to be the same brand: google), and service instances (e.g. the brands youtube and google are determined to run as a single service instance). They used the User-Agent header field to determine what generates the traffic. To do so, they divided each activity into one of those four categories: browsing, applications, updates and other. Hence they could analyse changes of the Web usage during the past decades. Furthermore, many other things got evaluated and examined. For instance consumer identification, HTTP method and response code examination, the effectiveness of caching or heavy tails [GAC+11].

3.2.4. Market analyses

Documents from the Web are often used for the identification and aggregation of relevant market statements. Even though, strategic business decision making is highly complex and requires expertise about economics, politics and technological developments, important tasks such as market forecasting which relies on identifying and aggregating relevant information from the Web may be automated. Analysts who interpret relevant data may get a reasonable idea about things like the future market volume, competitors and market shares [WPS10].

A problem about current analyses of market trends is that no single platform exists which provides a market analysis tool based on combined trends of the entire web. Tools that analyse search engines or social media websites only provide specific traffic trends based on their available data. A generic overview of the whole web would help to understand the current market and user interest about a specific product or topic. Research investigations are made to combine the available trends from multiple search engines and social media websites to provide realistic overall Web trends [WB10].

3.3. Trust among the Web

Trust among the WWW can have many facets. Depending on the context, trust can refer to completely different definitions. Those differences can, of course, be summarised into one general term: trust. Anyhow, to get a comprehensible idea about trust it is necessary to inspect all of its facets. Probably one of the most often way of understanding trust is the websites **content reliability**. With the possibility of everyone to create websites and publishing content, the substance of the content is questionable. One reason sources like Wikipedia are treat with caution. Often it's fundamental for users to estimate reliability among websites (i.e. when searching for medical information). Another criteria is the trust

degree in the context of **online market trustworthiness**. Customers want to make sure that a specific online shop is trustworthy before entering credit card details or the initiation of money transfer. This directly leads to the third facet: **data privacy**. One wants to be sure that private data (i.e. credit card details) are safely stored from any sort of third party access. Another big issue comes along with trust when its about downloads. Unfortunately, nowadays downloads are far away from being the only possibility of infecting users computer systems. Even though directly downloaded **malicious software** might be one of the easiest way of spreading viruses, but browser exploits or browser extension exploits (i.e. adobe flash player, java runtime environment) are often even more dangerous due to their unpredictable character. Therefore, users want to know about a websites trust before even entering it. A pretty new way of seeing trust in websites is the **legal protection for children** and young persons due to inappropriate age-dependent content. The mentioned trust characteristics can be summarised into five concerns.

1. Content reliability
2. E-commerce
3. Data privacy
4. Computer security
5. Youth protection

The following section demonstrates several existing tools and techniques that are build to protect those listed trust concerns. Every technique is associated with one or many characteristics.

3.4. Existing trustworthiness ratings

From the wide range of tools and techniques that have been developed for the measurement and evaluation of trust among the Web, a selection of them is described and presented in the following.

- **Web of Trust** (E-commerce, Data privacy, Computer security, Youth protection)
  The Web of Trust (WOT) is a rating platform for websites which is making use of swarm intelligence mechanisms. Every user who has installed the WOT plug-in can rate websites from poor to excellent on the four different characteristics: trustworthiness, vendor reliability, privacy and child safety. Other users browsing the same website see those ratings and therefore will be warned of untrustworthy sites [WOT13]. Unfortunately, the ratings of most users are obviously not very accurate. Since normal users can't tell whether a website is untrustworthy, ratings are often based on the particular degree of respect. The following two examples demonstrate this disadvantage. It is well known that the usage of Facebook is a big privacy issue for the users [WXG11] [Lee11]. However, it got an excellent rating from users in all categories (including Trustworthiness and Privacy) [WOT13]. Wikipedia, the collaboratively edited, Internet encyclopedia is well known to be imprecise and incomplete. Anyhow, it is widely spread, popular and used, therefore its not very stunning that its average WOT rating is excellent on any category.

  Similar product: **Web Security Guard** [Cra13]

- **McAfee's SiteAdvisor** (Computer security)
  The SiteAdvisor performs extensive and frequent heuristics-based evaluations of websites to measure their level of trustworthiness. Those evaluations include tests for phishing, infected downloads, spam, drive-by-downloads, e-commerce vulnerabilities, browser exploits, popups, etc. Based on the findings, every website is grouped into one of those risk categories: safe, caution, risky, and untested. Each category is associated with a color to give users a quick overview about the site's reliability (see figure 3.2). This rating is other than the WOT rating not influenced by subjective user ratings, but has the same issues than every anti virus software. Yet unknown risks won't be discovered by any heuristics.

Figure 3.2.: McAfee's SiteAdvisor risk groups [McA13]

Similar product: **Norton's Safe Web** [Nor13]

- **DoNotTrackMe** (Data privacy)
  DoNotTrackMe is similar to the Ghostery plug-in [Gho13]. Instead of informing the user about possible privacy threads it directly blocks anything that's on their predefined list. DoNotTrackMe is not a traditional trustworthiness rating. It can be seen as one when taking the number of found and blocked trackers as a score (see figure 3.3), where a low value demonstrates trust. Anyhow, DoNotTrackme doesn't specify any categorisation of this value.

Figure 3.3.: The DoNotTrackMe icon shows the number of tracking attempts made [Abi13]

Similar product: **Ghostery** [Gho13]

- **TrustGauge** (E-commerce)
  TrustGauge is described to be a quick and easy way to determine whether a visited website is trustworthy or not. It is part of the BrowserAccelerator toolbar and helps consumers to determine the trustworthiness of any site appearing in their browser window. Each website is assigned to a trust score value from 0 to 10, where 10

is the best. The score calculation is based on the factors displayed in table 3.1 which are then compared with the classes displayed in figure 3.4 [Tru09]. Since the score mechanism is mostly based on

Website Content Points	#	**Website Feature Points**	#
Email address or feedback form	2	Secure Billing Pages	2
Postal address (not a PO box)	2	Top 100 overall traffic	45
Brick & Mortar to visit	1	Top 1.000 overall traffic	35
Phone number available	1	Top 10.000 traffic	10
Toll free phone number	1	Top 100.000 traffic	5
A person answers the phone	1	Top 1.000.000+ traffic	1
Privacy statement page	2	Top 10 business category	5

Table 3.1.: TrustGauge factors for TrustScore determination [Tru09]

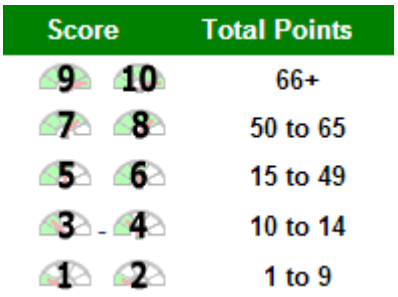

Figure 3.4.: TrustGauge classes for TrustScore categorisation [Tru09]

the website's traffic and contact possibilities, small and unknown pages will always get low scores even when trustworthy.

Similar product: **TrustScore** [Tru13]

- **PageTrust** (General trustworthiness)
  The PageTrust algorithm is a derivation of the PageRank algorithm. Other than the traditional PageRank, which was intended to order web pages by their importance [PBMW99], the PageTrust algorithm is able to take negative links into account and converges to a trust value for each page. This algorithm decreases trust values of nodes that receive negative links, while trying to ignore negative link of attackers who wants to decrease trust values of competing

nodes. The key element of the algorithm is a distrust matrix that spreads distrust along pages that trust distrusted pages [dKvD08]. Many algorithms are derived from the original PageRank. All trying to add some sort of distrust into the computed value. Those calculations which are using link graphs are called link analysis.

Other PageRank derivated algorithms: **TrustRank** [GGMP04], **A novel approach to propagating distrust** [BCK+10], **Dirichlet PageRank** [CTX11].

- **BrowseRank** (General trustworthiness)
  The BrowseRank algorithm computes page importancy based on a user browsing graph. This browsing graph contains information about the pages users visit, the links they clicked on and data about how long they spend on a specific page. This approach lets users implicit vote the page importance by providing the algorithm with his browsing history. Other than PageRank derivated algorithms, the BrowseRank interprets links and hence weights them. Anyhow, the calculation of this rank needs detailed browsing information on a large scale [LGL+08]. Special browser extensions could collect this data, but also the use of Javascript. The transparency plug-in (see section 2.6 Transparency plug-in) wouldn't collect enough information for a calculation.

- **Privacy International ranking** (General trustworthiness)
  Privacy International (PI) is registered as a charity in the UK. Their chosen mission is the defence for the right to privacy across the world, the fight against unlawful surveillance and other intrusions into private life by governments and corporations. Their knowledge and findings are, for example, used to advise the Council of Europe, the European Parliament, the Organisation for Economic Cooperation and Development and the UN Refugee Agency. The Global Surveillance Monitor is a program of PI that released the global study "Privacy and Human Rights". This study ranks countries based on their privacy policies and has become a global bench-

mark, used by international organisations, regulators and politicians to advance privacy protections in their own countries. The overall rating of a specific country includes the sub-values of constitutional protection, statutory protection, privacy enforcement, Identity Cards and Biometrics, data-sharing, visual surveillance, communication interception, communication data retention, government access to data, workplace monitoring, surveillance of medical, financial, and movement, border and trans-border issues, leadership and democratic safeguards. The mean value of all those scores yields the final value. The expressiveness of the coming out value ranges from 1.1 to 5.0 whereas a high value intends a less invasive policy [Int07]:

4.1 - 5.0 Consistently upholds human rights standards

3.6 - 4.0 Significant protections and safeguards

3.1 - 3.5 Adequate safeguards against abuse

2.6 - 3.0 Some safeguards but weakened protections

2.1 - 2.5 Systemic failure to uphold safeguards

1.6 - 2.0 Extensive surveillance societies

1.1 - 1.5 Endemic surveillance societies

Nevertheless, many indicators are far away from internet privacy and hence disturbing factors that falsify the values when used for internet ratings only. Basically the values constitutional protection (CP), statutory protection (SP), privacy enforcement (PE), data-sharing (DS), communication interception (CI), communication data retention (DR) and government access to data (GA) are merely interesting when talking about the internet privacy. Therefore an updated version of the Privacy Ranking has been calculated. This version is based only on the previously defined indicators and is shown in table A.1.

This rating can be used to rate a websites trustworthiness based on their country belonging. However, this rating can hardly be used directly by users, since they normally have no clue in what country a specific website is hosted in. Still, this data could be used as an input for other privacy rating algorithms.

3.5. Storage methods for HTTP traces

Several methods to log HTTP traces exist today. Probably the easiest way would be the storage of the whole network traffic. Many well known formats are developed by network analysers and packet sniffer. Anyhow, this thesis aims only for the storage of HTTP traces.

One method of storing HTTP metadata is the web server's log files. However, log files only store little information compared to the data available. Log files often look like the **Common Logfile Format** which is defined by the W3C [htt95] (note that an extended, but very similar version exists [HBB96]). The format stores seven entries for every HTTP request, whereas each entry represents the following information according to [htt95].

1. *remotehost*: Remote hostname (or IP number if DNS hostname is not available, or if DNSLookup is Off.
2. *rfc931*: The remote logname of the user.
3. *authuser*: The username as which the user has authenticated himself.
4. *date*: Date and time of the request.
5. *request*: The request line exactly as it came from the client.
6. *status*: The HTTP status code returned to the client.
7. *bytes*: The content-length of the document transferred.

Several log files similar to this one exist. A list can be found in [IBM04].

The logfile format is not very suitable for the storage of whole HTTP traces since they contain a varying number of data fields. Even the fields itself aren't known due to extensions (see section 2.1 Hypertext Transfer Protocol). A graph based method for the storage of the whole data that arises during a HTTP connection is proposed by the W3C. The draft **HTTP Vocabulary in RDF** is a RDF schema definition that provides a representation of the HTTP vocabulary. The terms defined in that vocabulary represent the core HTTP specification defined by RFC 2616, but also additional headers registered by the Internet Assigned Numbers Authority [KVA11]. RDF collections are used to store arbitrary and variable header fields. The following UML diagram (figure 3.5) shows the main part of the architecture.

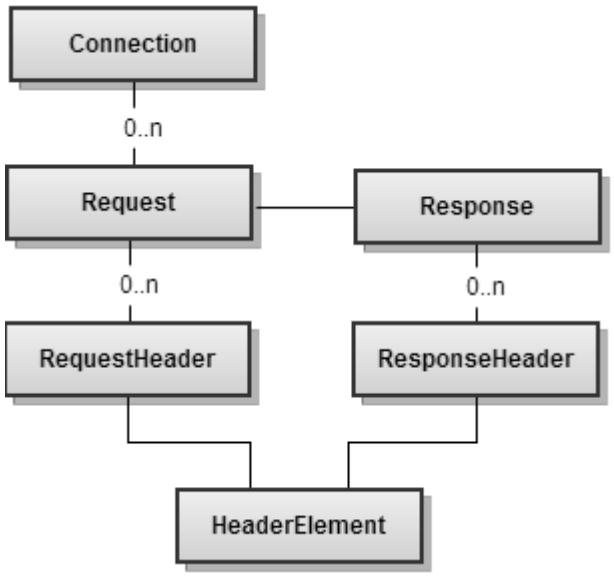

Figure 3.5.: HTTP Vocabulary in RDF: Simplified UML diagram

4. The trust graph

This chapter describes the creation and the analysis of the trust graph. The graph itself is basically the store of trust related information gathered from HTTP traces. The following sections document every step that is done during the creation of the graph, but also from the collection and the separation of incoming data. Furthermore, a trust score is calculated using the trust graph together with some exemplary analyses that demonstrate the possibilities the trust graph is offering.

4.1. Data collection and preparation

This section presents the data the transparency plug-in made available. A discussion about the removal and the addition of specific information is made to develop the trust graph's data basis.

4.1.1. Collectable data

The transparency plug-in delivers numerous information in form of HTTP requests and responses. When browsing a website, normally many more than one HTTP request needs to be made. After fetching the requested HTML (or XHTML, etc.) document, the browser requests every embedded source (e.g. images, Javascripts). Such a request graph could look like the one shown in figure 4.1. The user browsed the website `http://funsporting.de/` which is the middle element of the graph. All other nodes are embedded elements of the page which are not necessarily from the same domain. Especially when they aren't, a tracker might be in use.

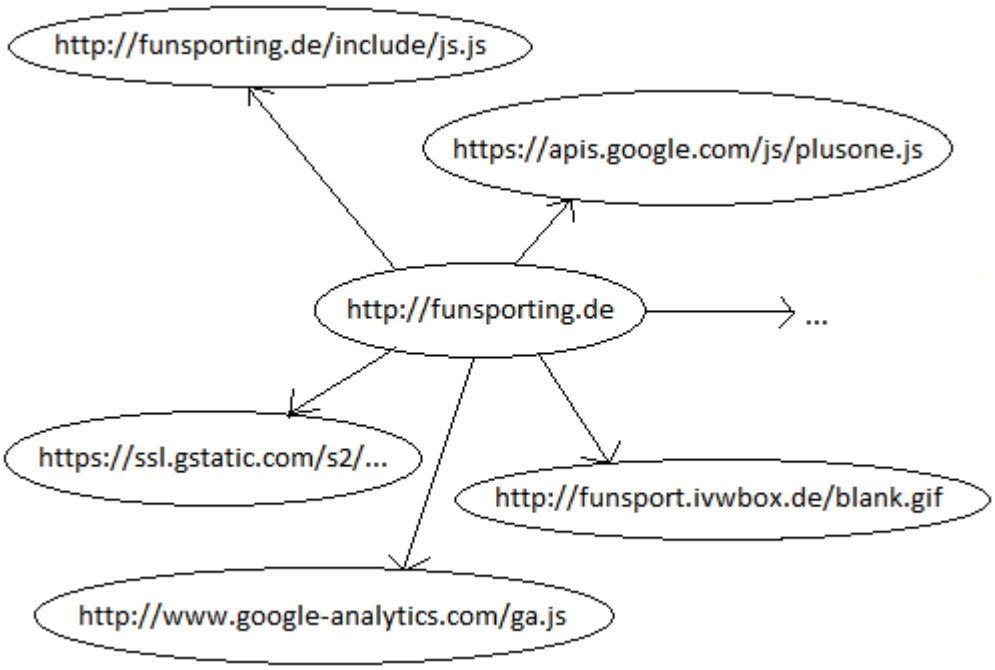

Figure 4.1.: HTTP request graph

Every node on this graph is the representation of a HTTP request and response. Those HTTP traces are delivered by the transparency plug-in via JSON files. A snippet of this data is shown in listing 4.1.

```
{  "url":"http://www.funsporting.de/",
   "root":"funsporting.de",
   "authorities":[{
      "domain":"google.com",
      "ip":"173.194.69.138",
      "requests":[{
         "ip":"173.194.69.138",
         "method":"GET",
         "url":"https://apis.google.com/js/plusone.js",
         "domain":"google.com",
          [...]
```

```
12        }, {
13            "ip":"173.194.69.138",
14            "url":"https://apis.google.com/_/scs/apps-stat[...]",
15             [...]
16        },  [...]
17 ]}]}
```

Listing 4.1: HTTP traces provided by the transparency plug-in

All information received by every single transfer (request mixed with response) of one embedded source (e.g. Google plus) is listed in table 4.1, whereas HTTP headers originate only from the response. Consider that some of the fields listed aren't part of any HTTP request or response, but browser internal information. Also might one HTTP request or response transfer a couple of different header fields than another (see section 2.1 Hypertext Transfer Protocol or [GTS+02, p. 67-73]).

Before separating out uninteresting fields, which is done in section 4.1.2 a few questions about the completeness of the data need to be asked.

- **Are other plug-ins able to falsify the collected data?**
  By having other plug-ins installed simultaneously with the transparency extension, the risk might be given that those other plug-ins block HTTP traffic which isn't registered by the transparency extension. In fact, plug-ins like Adblock are capable of suppressing HTTP requests. Whenever that happens, the transparency plug-in won't register those requests [Fie12, p. 58-59]. This a possible source of falsification.

- **How are Ajax requests registered?**
  Ajax requests are different to other embedded resources because they can be bound to user inputs. A small HTML file (listing 4.2 can be used to test user depended requests.

Field	Example value
frameId	0
fromCache	true
tabId	1970
requestId	78886
parentFrameId	-1
ip	173.194.69.138
method	GET
statusCode	200
statusLine	HTTP/1.1 200 OK
timeStamp	1362078183891.64
type	script
url	https://apis.google.com/js/plusone.js
domain	google.com
hostname	apis.google.com
scheme	https
country_code	us
header → status	200 OK
header → version	HTTP/1.1
header → content-encoding	gzip
header → content-type	application/javascript; charset=utf-8
header → date	Thu, 28 Feb 2013 19:02:48 GMT
header → etag	aea6d0e4f974eca55224c8a0d793f26a
header → expires	Thu, 28 Feb 2013 19:02:48 GMT
header → set-cookie	REMOVED BY EXTENSION

Table 4.1.: Request+response information gathered by the plug-in

```
<script>
function performRequest() {
    var req = new XMLHttpRequest();

    req.onreadystatechange = function() {
        if (req.readyState == 4) {
            if (req.status == 200) {
                alert (req.responseText);
    }   }   }
```

```
10
11      req.open('GET', 'ajaxTest.htm?new=1', true);
12      req.send(null);
13  }
14  </script>
15  <a href="#" onclick="performRequest();">ajax</a>
```

Listing 4.2: User depended Ajax test

By browsing this document all requests transmitted by the plug-in where analysed. It was found that even depended Ajax requests are handled the same way as any other embedded resource. This has to be kept in mind either way, because it could affect the analyses. Anyhow, Ajax shouldn't take much affect in tracking because of the same origin policy which restricts Ajax requests to other pages than the same site (same protocol, hostname, and port number) [KSTW07].

- **Are cached objects/documents registered?**
  The question if cached documents are registered by the plug-in is very important, since it could falsify any analysis. Even though the field "fromCache" almost answers this questions already, it wasn't quite sure how it works. A HTML file which includes an cacheable Javascript file (defined by header information) is used to test caching. After loading the HTML page once, the network packet analyser Wireshark [Fou13] has been used while reloading the site to capture any HTTP traffic. This experiment has shown how the "fromCache" field changed from false to true as well as no further traffic to the cached Javascript file is done. That proves the fact that even cached files are noticed from the plug-in and therefore caching won't falsify nothing. Anyhow, this result correlates with the statement of [Fie12, p. 47].

4.1.2. Data separation

The next step is the data separation. Some of the collected data might not be interesting and can therefore be sorted out. The following list shows dropped items together with their respective reasons.

- **tabId, requestId**
  Those fields contain browser internal information only, which can't be used or connected to HTTP relevant data.
- **parentFrameId, frameId**
  The parentFrameId is also browser internal information, but it could be connected to the frameId and therefore be used to follow a call graph of the frames. The problem is that the information about the parent frame and the child frame could be sent at different points in time. Therefore the receiving part would have to remember and to distinguish every single client, because the frame id's are only unique on one specific browser.
- **statusLine, status**
  Both redundant since header → version and statusCode contains the same information.
- **header → etag**
  Unfortunately, the entity tag is not standardised. It can be described as:

  > Entity tags are arbitrary labels (quoted strings) attached to the document. They might contain a serial number or version name for the document, or a checksum or other fingerprint of the document content. [GTS+02, p. 180]

  Because of the unpredictable and therefore unusable content the entity tag is dropped.
- **timeStamp**
  The timestamp value represents the time the request was sent. Unfortunately, there is no such value about the time the response is

actually arriving the user. Therefore, the only value to compare with would be the header → date, which is the server time. Due to potentially unsynchronised clocks, this comparison would be useless. Only the two values header → date and header → expires can be used, because they are both representations of the server time.

4.1.3. Data enhancement

In contrast of removing unusable data the possibility is also given to add data or either combine existing data to obtain new information. In addition to the data shown in table 4.1 the following fields have been added for every request/response connection pair.

- **Request reference**
  The reference of a request is the URL of the page that caused the browser to perform that request. Without this information it would be impossible to keep track of the websites that actually include trackers. Anyhow, this information is given from the tree structure within the JSON trace file provided by the transparency plug-in. Of course there won't be any reference available for the website which was actually typed in by the user. For such cases, the reference will be pointed out with null.

- **Server location (IP2Country)**
  IP2Country databases are a well known way to determine the server location of an IP address. Some providers even claim the accuracy of being able to track IP addresses down to city level. That is, however, very questionable even though the country-level geolocation accuracy performs quite well [PUK+11]. Since IP address and the corresponding countries change from time to time, it would be to late to evaluated the respective country while analysing the data. The used geolocation databse is [Web13].

- **Tracker**
  A pattern database can be used to determine whether a specific

Web resource is involved in user tracking. The Ghostery extension (see section 3.1.3 Techniques against tracking) uses about 1200 patterns to detect trackers which can easily be extracted from the browser extension. The patterns are stored in a JSON file which is basically nothing more than an array of pattern whereas each is connected with some information about the tracker. Listing 4.3 demonstrates the setup of that list.

```
{
    "bugs":[
        {
            "type":"ad",
            "aid":"2",
            "cid":"145",
            "pattern":"static\\.scribefire\\.com\\/ads\\.js",
            "name":"ScribeFire QuickAds",
            "id":"33",
            " affiliation ":""
        },
         [...]
    ]
}
```

Listing 4.3: Tracker detection patterns

- **Company**
  It would be useful having knowledge about the company a website is operated by. Especially, whensoever one company is well known to be untrustworthy, it would be nice to know about the other websites this company is owning. Unfortunately, the logical way of reading a website's WHOIS data, which should contain information about the operating company, is not in a readable format [Fie12, p. 44]. However, it was found that the knowledge of a websites operating company is that important so that the existing transparency

plug-in was modified for the collaborated information retrieval of this information. Unluckily, collaborated editing is pretty much a synonym for faulty and unreliable data which has to take account of when analysing. See chapter 5 for further information about the browser extension customisations.

- **Entered**
  It can be very useful to have every request connected with a timestamp when the request was stored in the graph. This timestamp is in fact pretty much the same than the dropped timestamp in the previous section, but attention: the dropped timestamp encloses the time of the client which may not be a true value. Therefore this data field is added by the time the HTTP trace is stored in the graph - all based on the same clock.

After the removal and the enhancement of the usable data, the final information that will be stored as RDF triples is summarised in table 4.2 along with their corresponding data types.

4.2. Construction of the trust graph

After determining the data that needs to be stored, an information architecture that is based on the data described in section 4.1 has to be defined. Existing storage methods described in section 3.5 are not capable of storing the necessary data. Log files won't allow the storage of varying data fields. The "HTTP Vocabulary in RDF" makes massive use of RDF collections which are poorly supported by the RDF framework Virtuoso [Sof09]. Also, the data structure would produce a lot of overhead due to all created classes (i.e. Connection Class, Message Class, Request Class, MessageHeader Class and so on) which actually aren't necessary because the data the transparency plug-in delivers is not ambiguous when everything is stored together (request and response).

Field	Type
parentRequest	reference
ip	string
method	string
url	string
domain	string
hostname	string
scheme	string
fromCache	boolean
statusCode	number
countryCode	string
contentEncoding	string
contentType	string
expires	timestamp
date	timestamp
setCookie	boolean
entered	timestamp
tracker	string
serverLocation	string

Table 4.2.: Extended request+response information

The following Entity-relationship model diagram (figure 4.2) demonstrates an easy and mostly overhead-free approach of storing the data as a RDF graph. Every data from a request and the corresponding response

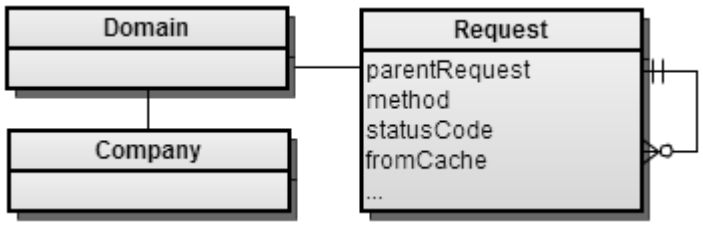

Figure 4.2.: Entity-relationship model of the trust graph

is joined into one entity named response. This entity contains every field from table 4.2 as an attribute. This simple construct contains only one RDF triple for the definition of a new request, but every other triple can directly be used to store the incoming data. With the amount of data that needs to be stored, much overhead would unnecessarily enlarge the

created graph as well as extend and complicate analyses. The proposed simplified graph will be used to build the trust graph. A RDF schema of this format is defined in section A.1. The schema, available at the imaginary URL http://trustgraph.org/schema will be used to describe RDF in this document.

4.3. Analysing the trust graph

The trust graph can be used to create multifarious analyses. The available data provides enough information for analyses of multiple areas such as the ones shown in section 3.2. Some quintessential ones are picked and explained within this section to demonstrate the capabilities of the trust graph. The roots of any analysis starts with SPARQL queries. Sometimes, nothing more than one query might be needed just as further processing through programming languages is required in other analyses. One complex usage example of the trust graph is provided by the calculation of a trust score. This is done in section 4.4. In addition to that, each of the following sub sections contains an exemplary analysis.

The data set on which each analysis is based on, is gathered by browsing the German top 50 websites (based on unique visitors per month [Gmb10]) together with America's top 50 [AI13]. The used browser sends a German identification (Accept-Language: de-DE) which might redirect multilingual websites to their German part. This data is, of course, not a reliable base to gather expressive answers, however, this thesis is about the possibilities the trust graph is providing and not about precise results.

4.3.1. Tracking probabilities and market shares

This analysis measures the probability of getting tracked when browsing the web. It breaks down the different trackers and probabilities of them.

The following SPARQL query returns a list of trackers and their usage number along all direct requests.

```
PREFIX tg: <http://trustgraph.org/schema#>

SELECT COUNT(*) AS ?count ?company WHERE {
  {
    SELECT DISTINCT ?parentRequest ?company WHERE {
      ?parentRequest tg:isA 'Request' .
      ?childRequest tg:isA 'Request' .
      ?childRequest tg:parentRequest ?parentRequest .
      ?childRequest tg:tracker ?trackingCompany .
      ?trackingCompany tg:companyName ?company
    }
  }
}
GROUP BY ?company
```

Listing 4.4: SPARQL: Tracker probability and market shares

This list of absolute trackers is then relativised and shortened to only trackers that reach a minimum of 10 percent of all websites. The outcome is shown in figure 4.3, where the probability of a tracker to appear on a website is shown on the y-axis and the name of that tracker on the x-axis. Due to the mentioned data set the results might not be representative for the whole web, the diagram still shows the tendency of online tracking: two tracking widgets are capable to track more than 50% of users browsing behaviour. However, the actual percentage for one tracking company might be much higher since the statistic is based on trackers and not on companies. Obviously, Google is a good example, because it appears three times in the list: Google Tag Manager, Google Affiliate Network and Google Website Optimizer. But caution, those three tracking values can't simply be added together, since one website could include more than one of them at the same time!

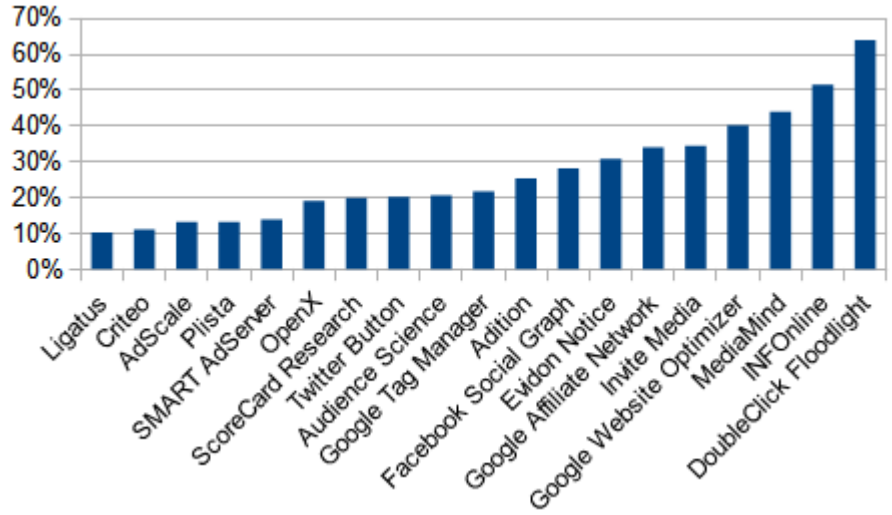

Figure 4.3.: Tracker probability and market shares

4.3.2. Third party content distribution by country

A question that might also come along with political background is the distribution of third party content along the Web. Third party content refers to anything that's included by a website that is not hosted on the same domain than the website itself. The probabilities that a website hosted in one country receives content from another is evaluated by the following SPARQL query.

```
1  PREFIX tg: <http://trustgraph.org/schema#>
2  PREFIX fn: <http://www.w3.org/2005/xpath-functions#>
3
4  SELECT ?childCountry COUNT(*) AS ?count WHERE {
5    ?childRequest tg:parentRequest ?parentRequest .
6
7    ?childRequest tg:domain ?childDomain .
8    ?parentRequest tg:domain ?parentDomain .
9    ?parentRequest tg:serverLocation '%s' .
10   ?childRequest tg:serverLocation ?childCountry .
11
```

```
12   FILTER(fn:not(?childDomain = ?parentDomain))
13 }
14 GROUP BY ?childCountry
```

Listing 4.5: SPARQL: Third party content distribution by country

This analysis is done for three countries: Germany, the United States of America and China. The probabilities for Germany and the USA are illustrated in figure 4.4 whereas only countries with a probability greater than one percent are shown. For the calculation of the Chinese websites,

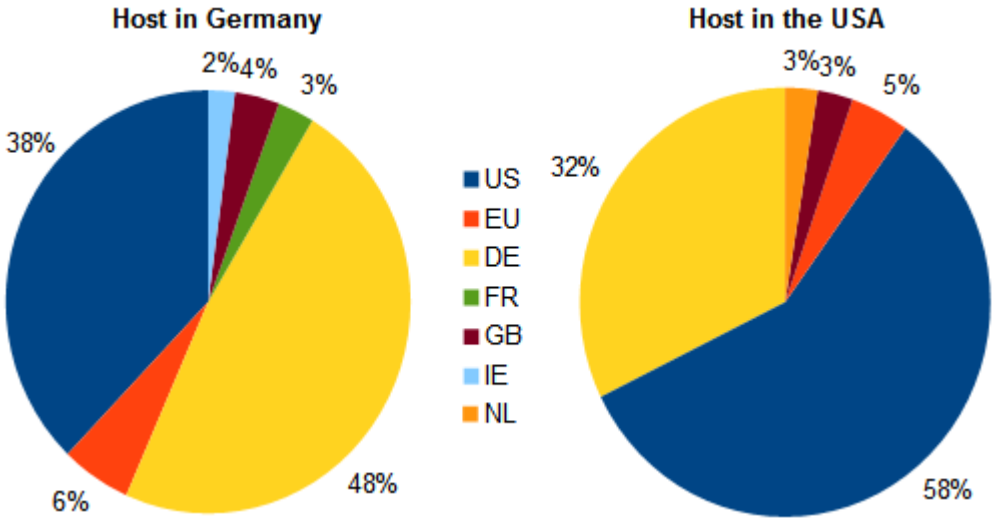

Figure 4.4.: Third party content distribution by country

a completely new data set is chosen. The HTTP traces generated by browsing the Chinese top 20 websites [www13] is used. Some of those websites are actually hosted in Taiwan or Hong Kong. Those ones hosted in China, include 74% of its third party contents from other websites hosted in China, 9% from the USA and 7% from Germany. Especially the China example demonstrates the international interaction between content distribution. Why else would Chinese websites include that much content from Germany? However, probably the biggest reason is language and location optimized advertising.

4.3.3. Media caching

Caching across the Web can significantly improve the browsing speed. This analysis evaluates the usage of cache specifiers of included media whereas media can be understood as images, CSS and flash objects. The ratio of objects that could have been cached, objects that are cached and objects that are specifically not cachable is evaluated. Objects that are recognised to be trackers are completely ignored. The following SPARQL query is capable to retrieve those values.

```
1  PREFIX tg: <http://trustgraph.org/schema#>
2  PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
3
4  SELECT COUNT(*) WHERE {
5    ?childRequest tg:parentRequest ?parentRequest .
6    ?childRequest tg:contentType ?contentType .
7
8    # Option 1: All
9
10   # Option 2: Cached
11   #?childRequest tg:date ?date .
12   #?childRequest tg:expires ?expires  .
13   #FILTER(xsd:dateTime(?expires) > xsd:dateTime(?date))
14
15   # Option 3: Not cacheable
16   #?childRequest tg:date ?date .
17   #?childRequest tg:expires ?expires  .
18   #FILTER(xsd:dateTime(?expires) <= xsd:dateTime(?date))
19
20   FILTER(
21     REGEX(?contentType, 'text/css.*') OR
22     REGEX(?contentType, 'image.*') OR
23     ?contentType = 'application/x-shockwave-flash'
```

```
24    )
25
26    FILTER NOT EXISTS {
27      ?childRequest tg:tracker  ?tracker
28    }
29  }
```

Listing 4.6: SPARQL: Media caching

The in-line comments explain which part needs to be uncommented for the calculation of the different values. Figure 4.5 demonstrates the distribution of media objects that are either cached, not cachable or nothing of the both. Those 9% of objects are either missing the date/expire field or

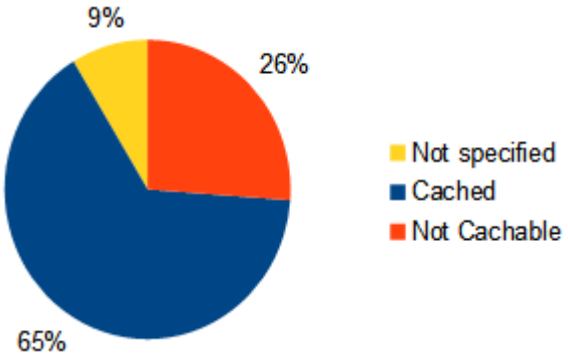

Figure 4.5.: Distribution of media caching

the value is not well formatted. This might be almost 10% of unnecessary traffic. However, it is hard to tell why those objects are not containing any cache specifiers. When unintentionally, a lot of traffic could have been spared.

4.3.4. Content inclusions by country

This time, the usage of included elements is analysed. Elements refer to anything that is included via an URL into a website (CSS, images, Javascript). The average number of inclusions per website is calculated

for every country. At the same time, the number of trackers are counted separated (a tracker is also an element!). This is done by putting the results of the following three SPARQL queries together.

```
PREFIX tg: <http://trustgraph.org/schema#>

#visited websites by country
SELECT COUNT(DISTINCT ?parentRequest) AS ?count ?
    serverLocation WHERE {
  ?childRequest tg:parentRequest ?parentRequest .
  ?parentRequest tg:serverLocation ?serverLocation

}
GROUP BY ?serverLocation

#content inclusions by country
SELECT COUNT(?childRequest) AS ?count ?serverLocation WHERE
    {
  ?childRequest tg:parentRequest ?parentRequest .
  ?parentRequest tg:serverLocation ?serverLocation
}
GROUP BY ?serverLocation

#content inclusions detected as trackers by country
SELECT COUNT(?childRequest) AS ?count ?serverLocation WHERE
    {
  ?childRequest tg:parentRequest ?parentRequest .
  ?parentRequest tg:serverLocation ?serverLocation .
  ?childRequest tg:tracker ?tracker
}
GROUP BY ?serverLocation
```

Listing 4.7: SPARQL: Content inclusions by country

Instead of using client side logic to put the results of those three queries together, a query using CONSTRUCT as a sub-query could have generated the same results with only one query. Unfortunately, the latest W3C SPARQL Recommendation [Wor08] doesn't allow that. However, extensions to the existing recommendation have been suggested [AG11] [Ves12] and the W3C SPARQL working group has discussed this feature already [wg08], so that it might appear in a further version.

Anyway, the results are illustrated in figure 4.6. The numbers of elements

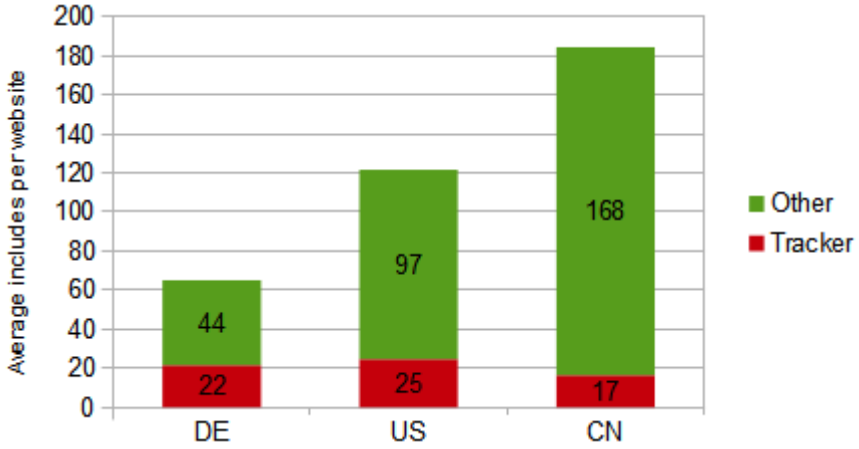

Figure 4.6.: Content inclusions by country

consists of the number of trackers and the number of other elements. The quantities shown are absolute values which reflect the average elements included by websites per country. The data set for China (see section 4.3.2 Third party content distribution by country) has also been used for this analysis. Then, other countries than Germany, the USA and China have been dropped.

Whereas the number of trackers is approximately similar, the total number of included elements is significantly different between the three countries. Even though this is an interesting fact, the interpretation of it is not part of this thesis.

4.4. Calculation of a trust score

One practical application of a trust graph analysis is the generation of a trust score. This score is used to provide users of the transparency plug-in with trustworthiness information related to the currently viewed page. The score rates websites on an objective scale (compare with WOT in section 3.4 Existing trustworthiness ratings) based on previous HTTP connections which got initiated by loading all contents of a specific site. A website is considered to contain a top- and second-level-domain. Addresses including more than two subdomains, but sharing the same top- and second-level-domain are treat as the same website (i.e. sub.example.com equals example.com). This score is able to warn users that are about to browse a specific website based on the information gain when other users were browsing that site. This can protect the privacy of many users by making use of information shared by others. However, it seems like no other trust score has been yet calculated using only the data the trust graph provides. Any significance of the score needs to be evaluated.

4.4.1. Data parameters

The trust score uses, of course, the data that is provided by the trust graph. This information needs, however, a semantic interpretation. Especially when talking about implications of concatenated data. Also graph external data might be used as an incoming parameter. Still, the algorithm will be based on the provided data. The following list contains a rule set that represents the basic ideas of the trust score.

- The number of third party trackers a website is including lowers the trust degree of that website (trackers are identified using the tracker detection patterns).
- Any third party include that is not identified as a tracker, but still sending a cookie to the users computer is considered to be a privacy

risk. That is because cookies are the biggest contributor when its about tracking (see section 3.1.1 Methods of user tracking).

- Previous factors are relativised by the internet privacy related part provided by the Privacy International ranking (see section 3.4 Existing trustworthiness ratings), whereas the websites country is allocated through the mentioned IP2Country database (see section 4.1.3 Data enhancement).
- Websites that are operated through the same company will share the trust score in some degree, since it can be assumed that one company handles all websites with approximately the same privacy policies.

Those parameters will be used for the calculation of the trust score value that is available for every website for which data is present within the trust graph. The detailed algorithm that considers and weights every single rule is explained in the next section.

4.4.2. The algorithm

The trust score algorithm outputs an value that expresses the privacy protection of a specific domain in relation to other domains. Therefore the trust score is not a fix value, but rather changes anytime new information is available (stored into the trust graph). The score can be interpreted as the factor of how trustworthy a website is compared to others. For example, the website example.com obtains a score of 2. Then, example.com is two times less trustworthy compared to the average. Trustworthiness is, however, defined as a product of the data parameters defined in the previous section. The following algorithm, written in pseudo code for readability reasons, demonstrates the calculation of the trust score. Anyway, the Java implementation can be found in section A.4 Sourcecode. The algorithm is split in two functions: *calculateTrustScore* and *calculateTrustScoreForDomain* which are explained as follows.

Algorithm 1 calculateTrustScore(*domain*)

```
company ← getCompany(domain)
if company ≠ null then
  domains ← getDomainsOfCompany(company)
  score ← 0
  count ← 0
  for each domain ∈ domains do
    tmpScore ← calculateTrustScoreForDomain(domain);
    if tmpScore > 0 then
      score ← score + tmpScore
      count ← count + 1
    end if
  end for
  return score/count;
else
  return calculateTrustScoreforDomain(domain)
end if
```

This first algorithm is used to perform the trust score calculation over all domains of a specific company or when not existent over a specific domain. That implies that the trust score of one domain of a company is the same as any other domain of that company. The next algorithm actually calculates a score of one domain without taking other domain of the same company into account.

Algorithm 2 calculateTrustScoreForDomain(*domain*)

```
if not isInformationAvailableforDomain(domain) then
  return 0
else
  country ← getDomainCountry(domain)
  if not set PI[country] then
    return 0
  end if
  trackerRatio ← AVGTrackersDomain(domain)/AVGTrackersAll()
  PIRatio ← PI[country]/AVG(PI)
  cookieRatio ← AVGCookiesDomain(domain)/AVGCookiesAll()
  return PIRatio × trackerRatio × cookieRatio
end if
```

Both procedures make use of functions that aren't defined here, but can be found at section A.3 SPARQL queries. Those functions actually refer to SPARQL queries performed on the trust graph.

4.4.3. Evaluation towards existing ratings

A big advantage of the trust score is the objectivity of the calculated rating. Many reports [Rep10] [NT12] [Max12] proved the problem subjective likert-scale based rating system are facing (i.e. WOT). The calculation needs less personalised data compared to the BrowseRank algorithm. Still, transferred data of the trust score might become a privacy issue when methods for users to remain anonymous are not operating. But this will be less detailed information than the data gathered by the BrowseRank algorithm. Another advantage is the spare data that is actually necessary to calculate the trust score of a specific website. A few HTTP traces collected from previous users is enough for a calculation. In contrast to that is PageTrust, an extension to the PageRank algorithm witch needs information about all websites that are linking to the website of interest.

One unfortunate issue about using the trust score technique is, that information which is available through the TrustGauge or the TrustScore isn't available at all. However, the TrustScore is based on subjective data only. The TrustGauge's score is calculated by using objective parameters, nevertheless, the method is based on a website's site notice. Whereas only the presence of this data implies trustworthiness, which isn't much of meaningful information.

Expert knowledge about current viruses and browser exploits that is provided through McAfee's SiteAdvisor can't be evaluated from the given data the trust score is receiving.

5. Design and Implementation

In consequence to the previous commitment of using the GWT (see section 2.3 Google Web Toolkit) and the Virtuoso RDF Triple Store (section 2.5.4 RDF frameworks), some basic architectural structure is already given which needs to be considered. Basically, the architecture design is made to cover the following two areas: data analyses and a browser interface. Whereas the browser interface handles: store and retrieve of company to domain information, retrieval of a domains trust score and the submission of any performed HTTP traces. This interface is only used for communications between the transparency plug-in and the data server (machine to machine communication). A REST based web service is easy to handle and to debug and will therefore be used for this area. Other ways of communication, e.g. handling Java Remote Procedure Calls (RPC) with the transparency plug-in are, anyhow, problematic due to the language restriction of the browser plug-ins (Javascript). On the other hand, the analyses may require user interactions which are perfectly fulfilled by GWT. However, the system must be build in a way that it's simple to either integrate new analyses, but also modify existing ones. Therefore, analyses should be coupled loose into the system, but also remain highly cohesive itself. An interface based plug-in system will be used.

Before describing the components within the next sections, an overview diagram is shown in figure 5.1. This demonstrates the involved components and gives an overview of the architecture and possible interactions between the components.

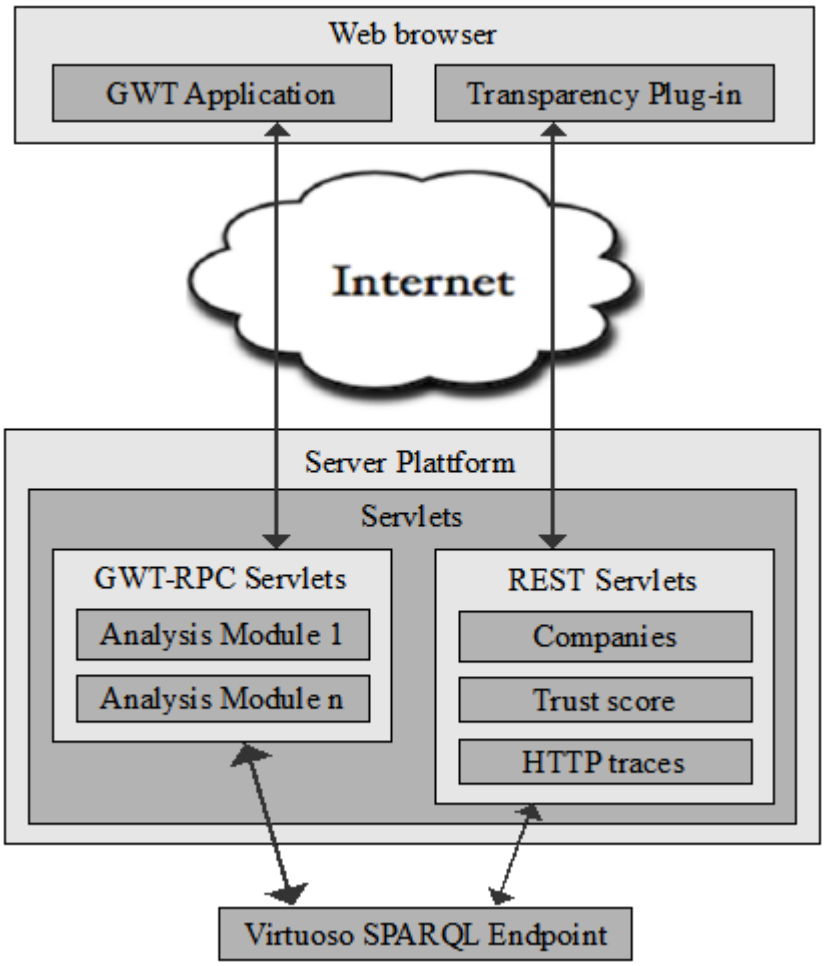

Figure 5.1.: Architecture design overview

5.1. RESTful web services

The RESTful web services are made for any communication between the transparency plug-in and the data server. The structure of every REST service is already determined by the GWT Servlet mapping. This mapping joins an URL or a set of URLs to an entry point (i.e. a method). The following code demonstrates the mapping of such a REST web services.

```
<servlet>
    <servlet-name>RESTCompanyInterface</servlet-name>
    <servlet-class>ExternalCompanyService</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>RESTCompanyInterface</servlet-name>
    <url-pattern>/company/*</url-pattern>
</servlet-mapping>
```

Listing 5.1: GWT: Web service URL to entry point mapping

This listing creates a web service called *RESTCompanyInterface* which is mapped to the entry point class *ExternalCompanyService* which reacts upon URLs that start with "/company/". Equivalent code maps all created web services (see table 5.1). Every entry point (i.e. REST-

Web service	**URL**	**Entry point**
RESTCompanyInterface	/company/*	ExternalCompanyService
TrustScoreInterface	/trustscore/*	ExternalTrustScoreService
RESTLinkInterface	/link	ExternalLinkService

Table 5.1.: GWT: Web service URL to entry point mapping

CompanyInterface) is actually nothing more than a class derived from *HttpServlet*. Such a class can overwrite a set of methods that are able to handle incoming HTTP requests such as *doGet*, *doPost*, *doPut*, etc. The word after "do" stand for the handled HTTP method. Naturally, the insertion of a company would make use of either the PUT or the POST method whereas the retrieval of a company uses the GET method. Depending on the result of the company retrieval process, the returning value can differ. REST web services are capable of returning multiple values at the same time. First of all comes the HTTP status code that is returned to the caller. When no company is found, the returning status code will be 404 (defined in HTTP as Not Found). When found, the

code 200 (OK) is returned together with the name of that company in the HTTP message body. That's the basic principal of how REST based web services are working. The sequence diagram in figure 5.2 demonstrates the company retrieval process. All of the three web services work

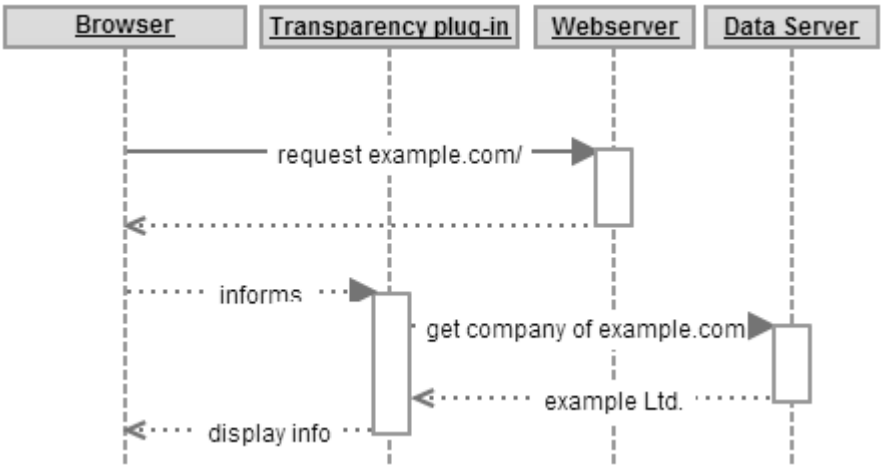

Figure 5.2.: Sequence diagram of the company retrieval process

pretty much the same way. Relevant SPARQL queries that are used to perform those operations are explained in chapter 4 or else in A.4 Sourcecode. The remaining section is used to describe the insertion part of new HTTP traces.

The HTTP traces are transferred to the data server by the transparency plug-in in regular intervals. As explained before, the data exchange is done with a RESTful POST request where the trace data is covered inside the HTTP body. The actual data is encoded in the JSON format an it's buildup is described in section 4.2. This JSON data is then parsed to extract every single request a website is containing. All extracted information is then stored in the trust graph as RDF triples connected with the main request of the website. A demonstration is made with the example HTTP traces gathered from a request to `http://funsporting.de/` which is shown in listing 4.1. The inserted triples would be:

```
@prefix tg: <http://trustgraph.org/schema#> .

_:a tg:isA              'Request' .
_:a tg:url              'http://funsporting.de/' .

_:b tg:isA              'Request' .
_:b tg:parentRequest _:a .
_:b tg:url              'https://apis.google.com/js/plusone.js' .
_:b tg:domain           'google.com' .
_:b tg:ip               '173.194.69.138' .
_:b tg:method           'GET' .

_:c tg:isA              'Request' .
_:c tg:parentRequest _:a .
_:c tg:url              'https://apis .google.com/_/scs/apps-stat[...]'
_:c tg:ip               '173.194.69.138' .
```

Listing 5.2: Turtle RDF extraced from HTTP traces

This is an shortened example trace and a real HTTP trace would, of course, be more complete and would contain more information.

5.2. Analysis modules

All analyses are mapped to an URL as the REST web services are. In contrast, this mapping is done to only one URL which is a container that handles all different kinds of all analyses. This container is build-up as described in [Thi10, p. 28]. Basically a menu that lists all possible analyses and a content part which is freely filled by the specific analysis. Figure 5.3 shows the container together with the performed media caching analysis. Each analysis module contains any necessary code that it needed for the generation and display of the results. Every module is

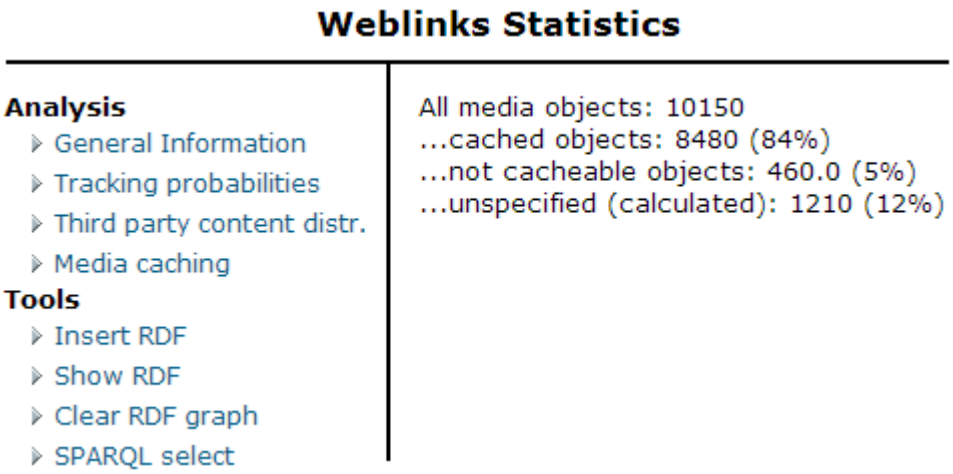

Figure 5.3.: Media caching analysis screenshot

derived from the interface class *IModule* to fulfil the previously described plug-in ability of the system. Figure 5.4 shows the UML class diagram of that interface. All links that an module registers is visible within the cat-

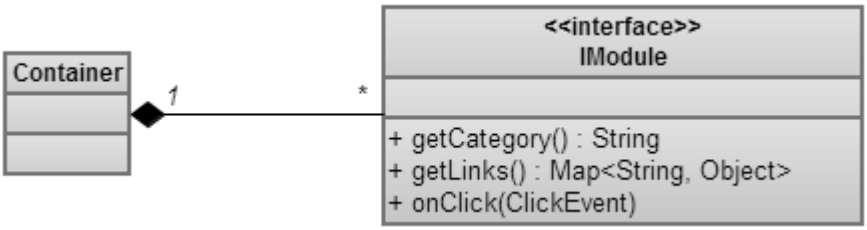

Figure 5.4.: UML diagram of IModule

egory it specifies. Therefore multiple modules can add links to the same category. With the use of this interface, all existing modules could be added to the container by making use of reflection. This practice would be very useful, since no code changes need to be made when adding or removing an analysis. Unfortunately, reflection is not allowed on the client side of GWT applications, therefore a registration of the modules at the container is necessary. This makes the insertion and removal of modules not completely trivial, because after any change a recompilation of the container code is inevitable. However, only one line of code needs to be added for the registration of new modules.

The execution of an analysis is predefined by GWT: a server side part and a client site (the users web browser) part. The client side party which is generated with GWT's Java to Javascript compiler. Since the Javascript code is running on the client's machine, any access data to the SPARQL endpoint would be readable by everyone using the system. That is where the GWT RPC is used which is shown in figure 5.1. Every SPARQL query is therefore not directly performed from the client machine, an indirect way using RPC to the data server is used. The sequence diagram in figure 5.5 demonstrates the flow of such an operation.

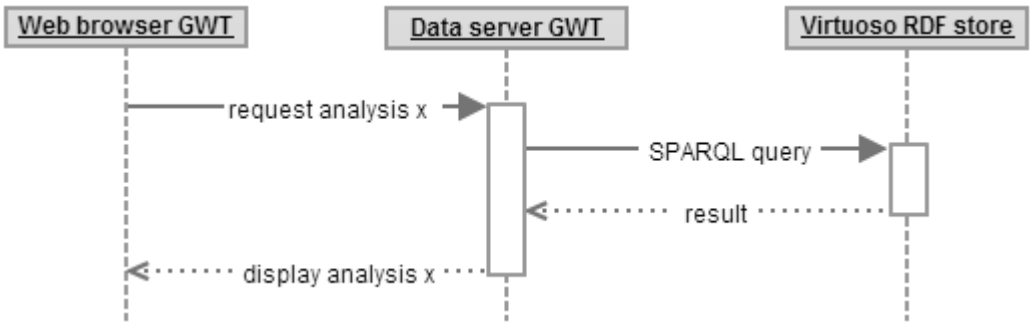

Figure 5.5.: Sequence diagram of an analysis

5.3. Transparency plug-in modifications

The transparency plug-in is modified to provide some additional functionality: retrieval, insertion and display of companies and the retrieval and display of the trust score. To add these features, the existing model and view of the Model-View-Controller architecture [Fie12, p. 36] needs to be extended with corresponding code. The retrieval and insertion part is done by making use of the previously described REST web services, which are operated by Ajax requests. Such a requests is basically performed with the following Javascript code which retrieves the trust score of the domain "cnn.com".

```
var req = new XMLHttpRequest();
req.open("GET", "http://data-server.org/trustscore/cnn.com", true);

req.onreadystatechange = function() {
    if (req.readyState == 4 && req.status == 200) {
        set_tooltip(req.responseText);
    }
}
req.send();
```

Listing 5.3: Javascript code for trust score retrieval

This asynchronous Ajax request is not delaying the website speed while loading the company or trust score data. The insertion part is pretty similar. The basic difference is the change of the HTTP method from "GET" to "PUT" and the payload that's added to the HTTP body (i.e. the company name). For the presentation of the trust score, the Chrome API function *chrome.browserAction.setBadgeText()* is chosen to display the score directly on the plug-in icon which is placed on the browser's frontend. The result is shown in figure 5.6.

Figure 5.6.: Trust score visible on browser plug-in icon

All changes within the transparency plug-in are either made inside the *background.js* or the *tools.js*.

6. Conclusion

6.1. Results

A system that stores HTTP traces collected by the transparency plug-in has been developed. The traces data itself is stored into a trust graph using RDF triples. The connection that is needed for the data transfer between the transparency plug-in and the data server is implemented via REST web services. Those web services are used to store HTTP traces into the trust graph, to store and retrieve company information about domains and to retrieve a trust score that is computable for any website. A possible falsification of those HTTP traces is not given, when compared with robots which could obtain special robot optimized content. The trust score is calculated based on the trust graph information gained by previous HTTP traces sent by many plug-in users. Thus, a plug-in user benefits from the information gained by others. This trust score is, in contrast to existing trustworthiness ratings, based on objective parameters only. Besides the trust score, the trust graph can be used for any further analyses. A couple of exemplary ones have been implemented and evaluated in section 4.3 to demonstrate the possibilities of the graph. Those analyses are build as modules for a framework that manages a simple adding, modification and removal. Therefore, only very few changes are necessary at the data server code when adding a new analysis to it.

6.2. Critical review

Even though the trust score is a helpful privacy indicator of websites, the data that is used for the calculation is only based on the HTTP traces that are gathered by users of the transparency plug-in. Certainly, privacy related issues are not only present in HTTP traces. Several other

characteristics listed in section 3.4 are needed for an all-inclusive trust score. Therefore, some harmful websites might actually obtain a positive trust score. Another problem is present in the calculation of the score: the tracker recognition. While an increasing number of trackers lowers the trust, the different trackers are not evaluated against each other.

The storage of company related information about domains, but also the HTTP traces itself are not subject of any verification. Therefore, deliberate spoofing with the aim of changing a trust score or whatsoever can easily be done.

Another aspect is the privacy issue that might come up with the storage of the HTTP traces itself. Even though cookie values are not transferred by the transparency plug-in, URLs often contain properties that might enable the identification of unique users. Due to the various possibilities of transporting user identification attributes within URLs, it's probably impossible to securely filter them out.

6.3. Further Work

A very next step before collecting data from a mass of users has to be the assurance of their privacy. Also, the protection from data spoofing is a very necessary task to ensure before putting credit into the trust score or any analysis when random users are able to insert their HTTP traces into the trust graph. Unfortunately, this activity might be difficult to fulfil, since any source code of transparency plug-in is visible to every user. Stress test should be done to test the stability and the run-time of the system when huge amounts of requests get stored in it. The analysis modules could be equipped with methods that help diagram creation. The final visualisation of current analysis shown in 4.3 is done with external software. The transparency plug-in should be modified in a way that it's possible to stop connections from even getting initialised on websites with very poor trust scores. This would unfortunately reduce the

browsing speed significantly, since the trust score would be needed to be fetched before any HTTP request.

Glossary

AJAX Asynchronous JavaScript and XML

API Application programming interface

CSS Cascading Style Sheets

DOM Document Object Model

GAE Google App Engine

GWT The Google Web Toolkit

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

IDE Integrated development environment

JRE Java Runtime Environment

JSON JavaScript Object Notation

OWL Web Ontology Language

P3P Platform for Privacy Preferences Project

P3P Platform for Privacy Preferences

RAL Repository Abstraction Layer

RDFS Resource Description Framework Schema

REST Representational State Transfer

RFC Request for Comments

RMI Remote Method Invocation

RPC Remote Procedure Calls

SOAP Simple Object Access Protocol

SPARQL SPARQL Protocol and RDF Query Language

SQL Structured Query Language

TCP Transmission Control Protocol

UML Unified Modeling Language

URI Uniform Resource Identifier

W3C World Wide Web Consortium

WOT Web of Trust

XML Extensible Markup Language

Bibliography

[Abi13] inc. Abine, *Dntme frequently asked questions*, Available from: <http://www.abine.com/dntp/faq.php>, 2013, Accessed Februar 26, 2013.

[AG11] Renzo Angles and Claudio Gutierrez, *Subqueries in sparql*, Proceedings of the 5th Alberto Mendelzon International Workshop on Foundations of Data Management, Santiago, Chile, May 9-12, 2011, CEUR Workshop Proceedings, vol. 749, CEUR-WS.org, 2011.

[AI13] Inc. Alexa Internet, *Top sites in united states*, Available from: <http://www.alexa.com/topsites/countries/US>, 2013, Accessed March 19, 2013.

[BCK03] Simon Byers, Lorrie Faith Cranor, and David Kormann, *Automated analysis of p3p-enabled web sites*, Proceedings of the 5th international conference on Electronic commerce (New York, NY, USA), ICEC '03, ACM, 2003, pp. 326–338.

[BCK+10] Christian Borgs, Jennifer Chayes, Adam Tauman Kalai, Azarakhsh Malekian, and Moshe Tennenholtz, *A novel approach to propagating distrust*, Proceedings of the 6th international conference on Internet and network economics (Berlin, Heidelberg), WINE'10, Springer-Verlag, 2010, pp. 87–105.

[BH01] Jeen Broekstra and Frank Van Harmelen, *Sesame: An architecture for storing and querying rdf data and schema information*, Writer **2342** (2001), 1–16.

[BK08] Matthew Burnside and Angelos D. Keromytis, *Asynchronous policy evaluation and enforcement*, Proceedings of the 2nd ACM workshop on Computer security architectures (New York, NY, USA), CSAW '08, ACM, 2008, pp. 45–50.

[BL97] Tim Berners-Lee, *Axioms of web architecture: Metadata*, Available from: <http://www.w3.org/DesignIssues/Metadata.html>, January 1997, Accessed September 10, 2012.

[Bra01] Tim Bray, *What is rdf*, Available from: <http://www.xml.com/pub/a/2001/01/24/rdf1.html>, January 2001, Accessed September 11, 2012.

[BS09] Christian Bizer and Andreas Schultz, *The berlin sparql benchmark*, International Journal On Semantic Web and Information Systems **5** (2009), no. 2, 1–24.

[Bur06] Ed Burnette, *Google web toolkit*, The Pragmatic Bookshelf, 11 2006.

[CDD+04] Jeremy J. Carroll, Ian Dickinson, Chris Dollin, Dave Reynolds, Andy Seaborne, and Kevin Wilkinson, *Jena: implementing the semantic web recommendations*, Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters (New York, NY, USA), WWW Alt. '04, ACM, 2004, pp. 74–83.

[CKB12] Abdelberi Chaabane, Mohamed Ali Kaafar, and Roksana Boreli, *Big friend is watching you: analyzing online social networks tracking capabilities*, Proceedings of the 2012 ACM workshop on Workshop on online social networks (New York, NY, USA), WOSN '12, ACM, 2012, pp. 7–12.

[Cla06] Roger Clarke, *What's privacy?*, Available from: <http://www.rogerclarke.com/DV/Privacy.html>, 2006, Accessed Februar 14, 2013.

[Cla12] Clark & Parsia, *Stardog: The rdf database*, Available from: <http://stardog.com/>, June 2012, Accessed June 28, 2012.

[CMS99] Robert Cooley, Bamshad Mobasher, and Jaideep Srivastava, *Data preparation for mining world wide web browsing patterns*, Knowl. Inf. Syst. **1** (1999), no. 1, 5–32.

[Cra94] Lorrie Faith Cranor, *The road less traveled: stop and smell the policy*, Crossroads **1** (1994), no. 1, 3–4.

[Cra98] Lorrie Faith Cranor, *Laws, self-regulation, and p3p: Will w3c s privacy platform help make the web safe for privacy?*, Computer Networks **30** (1998), no. 1-7, 751–753.

[Cra02] L. Cranor, *Web privacy with p3p*, Oreilly Series, O'Reilly Media, Incorporated, 2002.

[Cra13] Crawler, *Internet browser security - get safe, secure browsing - web security guard*, Available from: <`http://www.websecurityguard.com`>, 2013, Accessed Februar 26, 2013.

[CTX11] Fan Chung, Alexander Tsiatas, and Wensong Xu, *Dirichlet pagerank and trust-based ranking algorithms*, Proceedings of the 8th international conference on Algorithms and models for the web graph (Berlin, Heidelberg), WAW'11, Springer-Verlag, 2011, pp. 103–114.

[DD79] Dorothy E. Denning and Peter J. Denning, *Data security*, ACM Comput. Surv. **11** (1979), no. 3, 227–249.

[dKvD08] C. de Kerchove and P. van Dooren, *The PageTrust Algorithm: How to rank web pages when negative links are allowed?*, SIAM: Data Mining Proceedings, 2008, pp. 346+.

[Eck10] Peter Eckersley, *How unique is your web browser?*, Proceedings of the 10th international conference on Privacy enhancing technologies (Berlin, Heidelberg), PETS'10, Springer-Verlag, 2010, pp. 1–18.

[EV03] Magdalini Eirinaki and Michalis Vazirgiannis, *Web mining for web personalization*, ACM Trans. Internet Technol. **3** (2003), no. 1, 1–27.

[F+99] R. Fielding et al., *Hypertext transfer protocol – http/1.1*, Available from: <http://www.ietf.org/rfc/rfc2616.txt>, 1999, Accessed March 27, 2013.

[Fer12] Javier D. Fernández, *Binary rdf for scalable publishing, exchanging and consumption in the web of data*, Proceedings of the 21st international conference companion on World Wide Web (New York, NY, USA), WWW '12 Companion, ACM, 2012, pp. 133–138.

[Fie12] Tobias Fielitz, *Entwicklung eines browser plug-ins für transparentes surfen*, Master's thesis, Freie Universität Berlin, 2012.

[FKR10] George Forman, Evan Kirshenbaum, and Shyamsundar Rajaram, *A novel traffic analysis for identifying search fields in the long tail of web sites*, Proceedings of the 19th international conference on World wide web (New York, NY, USA), WWW '10, ACM, 2010, pp. 361–370.

[Fou13] Wireshark Foundation, *Wireshark go deep*, Available from: <http://www.wireshark.org/>, 2013, Accessed March 01, 2013.

[GAC+11] Phillipa Gill, Martin Arlitt, Niklas Carlsson, Anirban Mahanti, and Carey Williamson, *Characterizing organizational use of web-based services: Methodology, challenges, observations, and insights*, ACM Trans. Web **5** (2011), no. 4, 19:1–19:23.

[GGMP04] Zoltán Gyöngyi, Hector Garcia-Molina, and Jan Pedersen, *Combating web spam with trustrank*, Proceedings of the Thirtieth international conference on Very large data bases - Volume 30, VLDB '04, VLDB Endowment, 2004, pp. 576–587.

[Gho13] Ghostery, *Ghostery - about*, Available from: <http://www.ghostery.com/>, 2013, Accessed Februar 16, 2013.

[GKG11] Christophe Gueret, Spyros Kotoulas, and Paul Groth, *Triplecloud: An infrastructure for exploratory querying over web-scale rdf data*, Proceedings of the 2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology - Volume 03 (Washington, DC, USA), WI-IAT '11, IEEE Computer Society, 2011, pp. 245–248.

[Gmb10] CHIP Xonio Online GmbH, *Die 50 beliebtesten websites der deutschen*, Available from: <http://www.chip.de/bildergalerie/Die-50-beliebtesten-Websites-der-Deutschen-Stand-Mai-2010-Galerie_43740946.html>, 2010, Accessed March 19, 2013.

[Goo12a] Google, *Coding basics - compatibility with the java language and libraries*, Available from: <https://developers.google.com/web-toolkit/doc/latest/DevGuideCodingBasicsCompatibility>, 2012, Accessed March 27, 2013.

[Goo12b] ———, *Jre emulation reference*, Available from: <https://developers.google.com/web-toolkit/doc/latest/RefJreEmulation>, 2012, Accessed March 27, 2013.

[Goo12c] Google Developers, *Google app engine: Storing data*, Available from: <https://developers.google.com/appengine/docs/java/datastore/>, 2012, Accessed September 27, 2012.

[GPP13] Paul Gearon, Alexandre Passant, and Axel Polleres, *Sparql 1.1 update*, Available from: <http://www.w3.org/TR/sparql11-update/>, 2013, Accessed March 23, 2013.

[Gro09] Michael Grobe, *Rdf, jena, sparql and the semantic web*, Proceedings of the 37th annual ACM SIGUCCS fall conference (New York, NY, USA), SIGUCCS '09, ACM, 2009, pp. 131–138.

[GTS+02] David Gourley, Brian Totty, Marjorie Sayer, Anshu Aggarwal, and Sailu Reddy, *Http: The definitive guide*, 1st ed., O'Reilly Media, Inc., 2002.

[HBB96] Phillip M. Hallam-Baker and Brian Behlendorf, *Extended log file format*, Available from: <`http://www.w3.org/TR/WD-logfile`>, 1996, Accessed March 07, 2013.

[HD05] Andreas Harth and Stefan Decker, *Optimized index structures for querying rdf from the web*, Proceedings of the Third Latin American Web Congress (Washington, DC, USA), LA-WEB '05, IEEE Computer Society, 2005, p. 71.

[HFB+11] J. Hebeler, M. Fisher, R. Blace, A. Perez-Lopez, and M. Dean, *Semantic web programming*, ch. 6, Wiley, 2011.

[Hic11] Ian Hickson, *Web storage*, Available from: <`http://www.w3.org/TR/webstorage/`>, 2011, Accessed Februar 15, 2013.

[Hil12] Kashmir Hill, *How target figured out a teen girl was pregnant before her father did*, Available from: <`http://www.forbes.com/sites/kashmirhill/2012/02/16/how-target-figured-out-a-teen-girl-was-pregnant-before-her-father-did/`>, 2012, Accessed Februar 16, 2013.

[HK09] Bryan Horling and Matthew Kulick, *Personalized search for everyone*, Available from: <`http://googleblog.blogspot.de/2009/12/personalized-search-for-everyone.html`>, 2009, Accessed Februar 17, 2013.

[HKS12] Hans Hofinger, Alexander Kiening, and Peter Schoo, *When browsing leaves footprints: automatically detect privacy violations*, Proceedings of the First Workshop on Measurement, Privacy, and Mobility (New York, NY, USA), MPM '12, ACM, 2012, pp. 9:1–9:6.

[HLBT10] Jukka Honkola, Hannu Laine, Ronald Brown, and Olli Tyrkko, *Smart-m3 information sharing platform*, Computers and Communications, IEEE Symposium on **0** (2010), 1041–1046.

[htt95] httpd@w3.org, *Logging control in w3c httpd*, Available from: <http://www.w3.org/Daemon/User/Config/Logging.html>, 1995, Accessed March 07, 2013.

[IBM04] IBM, *Log file formats*, Available from: <http://publib.boulder.ibm.com/tividd/td/ITWSA/ITWSA_info45/en_US/HTML/guide/c-logs.html>, 2004, Accessed March 07, 2013.

[Int07] Privacy International, *Global surveillance monitor*, Available from: <https://www.privacyinternational.org/projects/global-surveillance-monitor>, 2007, Accessed March 13, 2013.

[IP11] Sunghwan Ihm and Vivek S. Pai, *Towards understanding modern web traffic*, Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems (New York, NY, USA), SIGMETRICS '11, ACM, 2011, pp. 143–144.

[Jen12] Apache Jena, *Jena architecture overview*, Available from: <http://jena.apache.org/about_jena/architecture.html>, 2012, Accessed September 14, 2012.

[Kam10] Samy Kamkar, *evercookie - virtually irrevocable persistent cookies*, Available from: <http://samy.pl/evercookie/>, 2010, Accessed Februar 15, 2013.

[KPKM12] Georgios Kontaxis, Michalis Polychronakis, Angelos D. Keromytis, and Evangelos P. Markatos, *Privacy-preserving social plugins*, Proceedings of the 21st USENIX conference on Security symposium (Berkeley, CA, USA), Security'12, USENIX Association, 2012, pp. 30–30.

[Kri01] David M. Kristol, *Http cookies: Standards, privacy, and politics*, ACM Trans. Internet Technol. **1** (2001), no. 2, 151–198.

[KSTW07] Chris Karlof, Umesh Shankar, J. D. Tygar, and David Wagner, *Dynamic pharming attacks and locked same-origin policies for web browsers*, CCS '07: Proceedings of the 14th ACM conference on Computer and communications security (New York, NY, USA), ACM, 2007, pp. 58–71.

[KVA11] Johannes Koch, Carlos A Velasco, and Philip Ackermann, *Http vocabulary in rdf 1.0*, Available from: <`http://www.w3.org/TR/HTTP-in-RDF10/`>, 2011, Accessed March 07, 2013.

[KW09] Balachander Krishnamurthy and Craig Wills, *Privacy diffusion on the web: a longitudinal perspective*, Proceedings of the 18th international conference on World wide web (New York, NY, USA), WWW '09, ACM, 2009, pp. 541–550.

[Lee11] Ronald Leenes, *Who needs facebook or google+ anyway: privacy and sociality in social network sites*, Proceedings of the 7th ACM workshop on Digital identity management (New York, NY, USA), DIM '11, ACM, 2011, pp. 31–32.

[LGL+08] Yuting Liu, Bin Gao, Tie-Yan Liu, Ying Zhang, Zhiming Ma, Shuyuan He, and Hang Li, *Browserank: letting web users vote for page importance*, Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval (New York, NY, USA), SIGIR '08, ACM, 2008, pp. 451–458.

[LH11] Günter Ladwig and Andreas Harth, *Cumulusrdf: Linked data management on nested key-value stores*, Proceedings of the 7th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2011) at the 10th International Semantic Web Conference (ISWC2011), October 2011.

[Lib12] MSDN Library, *Introduction to persistence*, Available from: <`http://msdn.microsoft.com/en-us/library/`

ms533007(v=vs.85).aspx>, 2012, Accessed Februar 15, 2013.

[Max12] Maximus, *Wot - web of trust: Die große volksverarsche eines unseriösen mozilla-ad-on-anbieters*, Available from: <http://www.kriegsberichterstattung.com/id/1423/wot-web-of-trust-die-grose-volksverarsche-eines-unseriosen-mozilla-ad-on-anbieters/>, 2012, Accessed March 12, 2013.

[May12] Jonathan Mayer, *Tracking the trackers: Early results*, Available from: <http://cyberlaw.stanford.edu/node/6694>, 2012, Accessed Februar 16, 2013.

[McA13] McAfee, Inc., *Mcafee siteadvisor-software - site-sicherheitsbewertungen und sichere suche*, Available from: <http://www.siteadvisor.com/download/windows.html>, 2013, Accessed Februar 26, 2013.

[Mes07] Oliver Messner, *Entwurf und umsetzung eines architekturkonzepts zur generierung und nutzung von metadaten in der unternehmenssoftware minos*, Master's thesis, Hochschule Karlsruhe, 2007.

[MKR04] Erica Meena, Ashwani Kumar, and Laurent Romary, *An extensible framework for efficient document management using rdf and owl*, Proceeedings of the Workshop on NLP and XML (NLPXML-2004): RDF/RDFS and OWL in Language Technology (Stroudsburg, PA, USA), NLPXML '04, Association for Computational Linguistics, 2004, pp. 51–58.

[MS03] Catherine C. Marshall and Frank M. Shipman, *Which semantic web?*, Proceedings of the fourteenth ACM conference on Hypertext and hypermedia (New York, NY, USA), HYPERTEXT '03, ACM, 2003, pp. 57–66.

[Nor13] Norton, *Is this website safe | website security | norton safe web*, Available from: <http://safeweb.norton.com/>, 2013, Accessed Februar 26, 2013.

[NSD+01] Natalya F. Noy, Michael Sintek, Stefan Decker, Monica Crubézy, Ray W. Fergerson, and Mark A. Musen, *Creating semantic web contents with protégé-2000*, IEEE Intelligent Systems **16** (2001), no. 2, 60–71.

[NT12] Netz-Trends.de, *Dubios: Wot web of trust - mozilla firefox ad-on wenig glaubhaft*, Available from: <http://www.netz-trends.de/id/1615/Dubios-WOT-Web-of-Trust---Mozilla-Firefox-ad-on-wenig-glaubhaft/>, 2012, Accessed March 12, 2013.

[OG11] Marie Caroline Oetzel and Tijana Gonja, *The online privacy paradox: a social representations perspective*, CHI '11 Extended Abstracts on Human Factors in Computing Systems (New York, NY, USA), CHI EA '11, ACM, 2011, pp. 2107–2112.

[Ont12a] Ontoprise, *Triplestore professional licenses*, Available from: <http://www.smwplus.com/index.php/Buy_TripleStore_Professional>, June 2012, Accessed June 25, 2012.

[Ont12b] Ontotext AD, *Owlim-se fact sheet*, Available from: <http://owlim.ontotext.com/display/OWLIMv50/OWLIM-SE+Fact+Sheet>, June 2012, Accessed June 27, 2012.

[Ora12] Oracle Corporation, *Oracle database enterprise edition*, Available from: <https://shop.oracle.com/pls/ostore/f?p=700:6:0::::P6_LPI:4509382199341805719938>, June 2012, Accessed June 25, 2012.

[PBMW99] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd, *The pagerank citation ranking: Bringing order to the web.*, Technical Report 1999-66, Stanford InfoLab, November 1999.

[PUK+11] Ingmar Poese, Steve Uhlig, Mohamed Ali Kaafar, Benoit Donnet, and Bamba Gueye, *Ip geolocation databases: unreliable?*, SIGCOMM Comput. Commun. Rev. **41** (2011), no. 2, 53–56.

[RDM09] Ian Reay, Scott Dick, and James Miller, *A large-scale empirical study of p3p privacy policies: Stated actions vs. legal obligations*, ACM Trans. Web **3** (2009), no. 2, 6:1–6:34.

[Rep10] Ripoff Report, *Complaint review: Web of trust*, Available from: <`http://www.ripoffreport.com/internet-services/web-of-trust/web-of-trust-my-web-of-trust-c34ad.htm`>, 2010, Accessed March 12, 2013.

[RKW12] Franziska Roesner, Tadayoshi Kohno, and David Wetherall, *Detecting and defending against third-party tracking on the web*, Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation (Berkeley, CA, USA), NSDI'12, USENIX Association, 2012, pp. 12–12.

[San09] Dan Sanderson, *Programming google app engine: Build and run scalable web apps on google's infrastructure*, 1st ed., O'Reilly Media, Inc., 2009.

[Sch11] Holger Schmidt, *"do not track": Der große streit um den kleinen befehl*, FOCUS (2011), 1.

[SCM+09] A. Soltani, S. Canty, Q. Mayo, L. Thomas, and C. J. Hoofnagle, *Flash cookies and privacy*, Social Science Research Network, Working Paper Series (2009), 1.

[Sev09] Charles Severance, *Using google app engine*, 1st ed., O'Reilly Media, Inc., 2009.

[SK06] Michael Sintek and Malte Kiesel, *Rdfbroker: a signature-based high-performance rdf store*, Proceedings of the 3rd European conference on The Semantic Web: research and applications (Berlin, Heidelberg), ESWC'06, Springer-Verlag, 2006, pp. 363–377.

[Sof09] OpenLink Software, *9.32. transitivity in sql*, Available from: <http://docs.openlinksw.com/virtuoso/transitivityinsQL.html>, 2009, Accessed March 08, 2013.

[SYS12] SYSTAP, *Bigdata®*, Available from: <http://www.systap.com/bigdata.htm>, June 2012, Accessed June 28, 2012.

[The12] The Open Anzo project, *Introduction*, Available from: <http://www.openanzo.org/projects/openanzo/wiki>, June 2012, Accessed June 27, 2012.

[Thi10] Malvin Thiel, *The development of a web application framework forcontent management systems*, Master's thesis, Dundalk Institute of Technology, 2010.

[TMKD09] Dorothea Tsatsou, Fotis Menemenis, Ioannis Kompatsiaris, and Paul C. Davis, *A semantic framework for personalized ad recommendation based on advanced textual analysis*, Proceedings of the third ACM conference on Recommender systems (New York, NY, USA), RecSys '09, ACM, 2009, pp. 217–220.

[Tru09] TrustGauge.com, *Trustgauge*, Available from: <http://www.trustgauge.com/>, 2009, Accessed Februar 26, 2013.

[Tru13] TrustYou, *The trustscore*, Available from: <http://www.trustyou.com/products_the_trust_score_en.html>, 2013, Accessed March 11, 2013.

[Ves12] Rob Vesse, *Extending sparql with construct sub-queries*, Available from: <http://yarcdata.com/blog/?p=143>, 2012, Accessed March 21, 2013.

[WB10] Abdul Wahid and Boyan Bontchev, *Platform for extraction, visualization and analysis of search trends*, Proceedings of the 8th International Conference on Frontiers of Information Technology (New York, NY, USA), FIT '10, ACM, 2010, pp. 13:1–13:6.

[Web13] Webnet77.com, *Ip to country database (ipv4 and ipv6)*, Available from: <http://software77.net/geo-ip/>, 2013, Accessed March 03, 2013.

[wg08] W3C SPARQL working group, *Chatlog 2009-05-07*, Available from: <http://www.w3.org/2009/sparql/wiki/Chatlog_2009-05-07>, 2008, Accessed March 21, 2013.

[WGA05] David Wood, Paul Gearon, and Tom Adams, *Kowari: A platform for semantic web storage and analysis*, XTech 2005, May 2005.

[Wor08] World Wide Web Consortium, *Sparql query language for rdf*, Available from: <http://www.w3.org/TR/rdf-sparql-query/>, 2008, Accessed September 26, 2012.

[WOT13] WOT Services, *Wot reputation scorecard facebook.com*, Available from: <https://www.mywot.com/en/scorecard/facebook.com>, 2013, Accessed January 23, 2013.

[WPS10] Henning Wachsmuth, Peter Prettenhofer, and Benno Stein, *Efficient statement identification for automatic market forecasting*, Proceedings of the 23rd International Conference on Computational Linguistics (Stroudsburg, PA, USA), COLING '10, Association for Computational Linguistics, 2010, pp. 1128–1136.

[www13] www.chinesetop100.com, *The top 20 of the global chinese website*, Available from: <http://www.chinesetop100.com/>, 2013, Accessed March 20, 2013.

[WXG11] Na Wang, Heng Xu, and Jens Grossklags, *Third-party apps on facebook: privacy and the illusion of control*, Proceedings of the 5th ACM Symposium on Computer Human Interaction for Management of Information Technology (New York, NY, USA), CHIMIT '11, ACM, 2011, pp. 4:1–4:10.

[YWT+10] Christopher C. Yang, Flaura Winston, Adam Townes, Xuning Tang, and Nancy Kassam-Adams, *A study on the user navigation path of a web-based intervention program – aftertheinjury.org*, Proceedings of the 1st ACM International Health Informatics Symposium (New York, NY, USA), IHI '10, ACM, 2010, pp. 449–453.

A. Appendix

A.1. RDF Schema of the trust graph

```
<?xml version="1.0"?>
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xml:base="http://www.w3.org/2001/XMLSchema#">

<rdfs:Class rdf:ID="Response" />

<rdf:Property rdf:ID="parentRequest">
    <rdfs:domain rdf:resource="#Response"/>
    <rdfs:range rdf:resource="#Response"/>
</rdf:Property>

<rdf:Property rdf:ID="ip">
    <rdfs:domain rdf:resource="#Response"/>
    <rdfs:range rdf:resource="string"/>
    <rdfs:comment>The IP address of the webserver</rdfs:comment>
</rdf:Property>

<rdf:Property rdf:ID="method">
    <rdfs:domain rdf:resource="#Response"/>
    <rdfs:range rdf:resource="string"/>
</rdf:Property>

<rdf:Property rdf:ID="url">
    <rdfs:domain rdf:resource="#Response"/>
```

```
27      <rdfs:range rdf:resource="string"/>
28  </rdf:Property>
29
30  <rdf:Property rdf:ID="domain">
31      <rdfs:domain rdf:resource="#Response"/>
32      <rdfs:range rdf:resource="string"/>
33  </rdf:Property>
34
35  <rdf:Property rdf:ID="hostname">
36      <rdfs:domain rdf:resource="#Response"/>
37      <rdfs:range rdf:resource="string"/>
38  </rdf:Property>
39
40  <rdf:Property rdf:ID="scheme">
41      <rdfs:domain rdf:resource="#Response"/>
42      <rdfs:range rdf:resource="string"/>
43      <rdfs:comment>HTTP access scheme (i.e. http, https)</
            rdfs:comment>
44  </rdf:Property>
45
46  <rdf:Property rdf:ID="fromCache">
47      <rdfs:domain rdf:resource="#Response"/>
48      <rdfs:range rdf:resource="boolean"/>
49      <rdfs:comment>Resource derived from the brwosers cache</
            rdfs:comment>
50  </rdf:Property>
51
52  <rdf:Property rdf:ID="statusCode">
53      <rdfs:domain rdf:resource="#Response"/>
54      <rdfs:range rdf:resource="positiveInteger"/>
55  </rdf:Property>
56
57  <rdf:Property rdf:ID="countryCode">
```

```
58      <rdfs:domain rdf:resource="#Response"/>
59      <rdfs:range  rdf:resource="string"/>
60      <rdfs:comment>Country code of the contents language</
            rdfs:comment>
61  </rdf:Property>
62
63  <rdf:Property rdf:ID="contentEncoding">
64      <rdfs:domain rdf:resource="#Response"/>
65      <rdfs:range  rdf:resource="string"/>
66      <rdfs:comment>The encoding of the response content</
            rdfs:comment>
67  </rdf:Property>
68
69  <rdf:Property rdf:ID="contentType">
70      <rdfs:domain rdf:resource="#Response"/>
71      <rdfs:range  rdf:resource="string"/>
72  </rdf:Property>
73
74  <rdf:Property rdf:ID="expires">
75      <rdfs:domain rdf:resource="#Response"/>
76      <rdfs:range  rdf:resource="dateTime"/>
77      <rdfs:comment>Time and date of the expiration date of the
            content</rdfs:comment>
78  </rdf:Property>
79
80  <rdf:Property rdf:ID="date">
81      <rdfs:domain rdf:resource="#Response"/>
82      <rdfs:range  rdf:resource="dateTime"/>
83      <rdfs:comment>Time and date of the server, when the reponse
            was send</rdfs:comment>
84  </rdf:Property>
85
86  <rdf:Property rdf:ID="setCookie">
```

```
87      <rdfs:domain rdf:resource="#Response"/>
88      <rdfs:range  rdf:resource="boolean"/>
89      <rdfs:comment>True whether it was tried to store a cookie on the
            client</rdfs:comment>
90  </rdf:Property>
91
92  <rdf:Property rdf:ID="entered">
93      <rdfs:domain rdf:resource="#Response"/>
94      <rdfs:range  rdf:resource="dateTime"/>
95      <rdfs:comment>The time when the request is created in the graph
            </rdfs:comment>
96  </rdf:Property>
97
98  <rdf:Property rdf:ID="tracker">
99      <rdfs:domain rdf:resource="#Response"/>
100     <rdfs:range  rdf:resource="string"/>
101     <rdfs:comment>Name of the tracker when the requested resource
            was identified as one</rdfs:comment>
102 </rdf:Property>
103
104 <rdf:Property rdf:ID="company">
105     <rdfs:domain rdf:resource="#Response"/>
106     <rdfs:range  rdf:resource="string"/>
107     <rdfs:comment>The company that operates the resource</
            rdfs:comment>
108 </rdf:Property>
109
110 <rdf:Property rdf:ID="serverLocation">
111     <rdfs:domain rdf:resource="#Response"/>
112     <rdfs:range  rdf:resource="string"/>
113     <rdfs:comment>The Country, the server is operating in</
            rdfs:comment>
114 </rdf:Property>
```

```
115
116 </rdf:RDF>
```

Listing A.1: RDF Schema of the trust graph

A.2. Internet only version: Privacy International ranking

Country	CP	SP	PE	DS	CI	DR	GA	AVG
Greece	4	3	4	-	1	-	3	3
Romania	3	3	4	-	2	3	2	2,8
Hungary	4	4	4	3	1	4	3	3,3
Slovenia	4	4	4	3	2	1	2	2,9
Portugal	4	4	3	2	2	-	-	3
Luxembourg	2	3	3	2	2	3	-	2,5
Germany	4	4	4	4	2	1	3	3,1
Italy	4	4	4	-	1	1	2	2,7
Estonia	3	3	4	-	2	-	3	3
Belgium	4	4	4	1	2	2	3	2,9
Czech Republic	4	3	4	1	1	2	2	2,4
Finland	3	3	3	1	3	3	2	2,6
Ireland	2	3	4	2	3	1	2	2,4
Malta	2	4	3	-	2	-	2	2,6
Poland	3	4	3	3	1	1	2	2,4
Spain	3	4	4	-	1	2	2	2,7
Austria	2	3	2	1	2	4	2	2,3
Cyprus	3	3	3	-	1	-	-	2,5
Latvia	3	2	2	2	2	-	2	2,2
Netherlands	2	4	4	1	1	1	2	2,1
Slovakia	4	3	3	-	2	1	1	2,3
Sweden	3	2	3	2	2	1	1	2
Denmark	3	2	2	1	2	1	1	1,7

Bulgaria	3	2	3	-	1	2	2	2,2
Lithuania	3	3	2	-	1	3	-	2,4
France	3	2	3	1	2	1	1	1,9
UK	1	2	2	1	1	1	2	1,4
Canada	4	4	2	2	3	4	3	3,1
Argentina	4	4	2	2	2	2	-	2,7
Iceland	4	4	4	3	3	2	2	3,1
Switzerland	4	4	2	2	2	2	2	2,6
New Zealand	2	2	3	2	1	3	2	2,1
South Africa	4	1	1	2	2	1	-	1,8
Japan	3	1	1	2	3	4	3	2,4
Australia	1	2	2	2	2	4	2	2,1
Israel	4	3	3	2	2	2	1	2,4
Brazil	3	2	1	2	2	2	2	2
Norway	3	2	3	1	2	2	2	2,1
India	3	1	1	-	1	-	1	1,4
Philippines	3	2	1	-	1	2	1	1,7
US	3	1	1	2	1	3	2	1,9
Thailand	2	2	2	-	1	2	1	1,7
Taiwan	2	2	1	-	1	3	2	1,8
Singapore	1	1	1	2	1	3	1	1,4
Russia	3	2	1	1	1	1	1	1,4
China	2	2	1	1	1	1	1	1,3
Malaysia	1	2	1	1	1	3	1	1,4

Table A.1.: Internet only version: Privacy International ranking

See Privacy International ranking in section 3.4 Existing trustworthiness ratings for explanation of the table entries. Information based on [Int07].

A.3. SPARQL queries

This section contains SPARQL queries that are either used for the calculation of the trust score or other analysis.

Average third party cookies over all domains

```
PREFIX tg: <http://trustgraph.org/schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX fn: <http://www.w3.org/2005/xpath-functions#>

SELECT AVG(?cookie) AS ?allAVG WHERE {{
SELECT SUM(?cookie) AS ?cookie ?parentRequest ?childDomain
    WHERE {{
  SELECT ?cookie ?parentRequest ?childDomain WHERE {
    ?childRequest tg:isA 'Request' .
    ?parentRequest tg:isA 'Request' .
    ?childRequest tg:parentRequest ?parentRequest .
    ?parentRequest tg:domain ?parentDomain .
    ?childRequest tg:domain ?childDomain .
    ?childRequest tg:setCookie ?cookie .
    FILTER NOT EXISTS { ?childRequest tg:tracker ?tracker } .
    FILTER(fn:not(?parentDomain = ?childDomain))
  }
} UNION {
  SELECT "0"^^xsd:float AS ?cookie ?parentRequest ?childDomain
      WHERE {
    ?childRequest tg:isA 'Request' .
    ?parentRequest tg:isA 'Request' .
    ?childRequest tg:parentRequest ?parentRequest .
    ?parentRequest tg:domain ?parentDomain .
    ?childRequest tg:domain ?childDomain .
    FILTER NOT EXISTS { ?childRequest tg:tracker ?tracker } .
```

```
25      FILTER NOT EXISTS { ?childRequest tg:setCookie ?cookie } .
26      FILTER(fn:not(?parentDomain = ?childDomain))
27    }
28 }}
29 }}
```

Listing A.2: Average third party cookies over all domains

Average third party cookies of a specific domain

```
1  PREFIX tg: <http://trustgraph.org/schema#>
2  PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
3
4  SELECT AVG(?cookie) AS ?avg WHERE {{
5  SELECT SUM(?cookie) AS ?cookie ?parentRequest ?childDomain
       WHERE {{
6    SELECT ?cookie ?parentRequest ?childDomain WHERE {
7      ?childRequest tg:isA 'Request' .
8      ?parentRequest tg:isA 'Request' .
9      ?childRequest tg:parentRequest ?parentRequest .
10     ?parentRequest tg:domain ?parentDomain .
11     ?childRequest tg:domain ?childDomain .
12     ?childRequest tg:setCookie ?cookie .
13     FILTER NOT EXISTS { ?childRequest tg:tracker ?tracker } .
14     FILTER(fn:not(?parentDomain = ?childDomain)) .
15     FILTER(?parentDomain = <%s>)
16   }
17 } UNION {
18   SELECT "0"^^xsd:float AS ?cookie ?parentRequest ?childDomain
         WHERE {
19     ?childRequest tg:isA 'Request' .
20     ?parentRequest tg:isA 'Request' .
21     ?childRequest tg:parentRequest ?parentRequest .
```

```
22      ?parentRequest tg:domain ?parentDomain .
23      ?childRequest tg:domain ?childDomain .
24      FILTER NOT EXISTS { ?childRequest tg:tracker ?tracker } .
25      FILTER NOT EXISTS { ?childRequest tg:setCookie ?cookie } .
26      FILTER(fn:not(?parentDomain = ?childDomain)) .
27      FILTER(?parentDomain = <%s>)
28    }
29  }}
30  }}
```

Listing A.3: Average third party cookies of a specific domain domains

Average trackers of all domains

```
1  PREFIX tg: <http://trustgraph.org/schema#>
2  PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
3
4  SELECT AVG(?tracker) AS ?allAVG WHERE {{
5    SELECT SUM(?tracker) AS ?tracker ?parentRequest ?childDomain
         WHERE {{
6          SELECT COUNT(DISTINCT(?tracker)) AS ?tracker ?
               parentRequest WHERE {
7            ?childRequest tg:isA 'Request' .
8            ?parentRequest tg:isA 'Request' .
9            ?childRequest tg:parentRequest ?parentRequest .
10           ?childRequest tg:tracker ?tracker .
11           ?parentRequest tg:domain ?parentDomain
12         }
13         GROUP BY ?parentRequest
14   } UNION {
15         SELECT "0"^^xsd:float AS ?tracker ?parentRequest WHERE
               {
16           ?childRequest tg:isA 'Request' .
```

```
17            ?parentRequest tg:isA 'Request' .
18            ?childRequest tg:parentRequest ?parentRequest .
19            ?parentRequest tg:domain ?parentDomain .
20            FILTER NOT EXISTS { ?childRequest tg:tracker ?tracker }
                  .
21          }
22          GROUP BY ?parentRequest
23    }}
24 }}
```

Listing A.4: Average third party cookies of a specific domain domains

Average trackers of a specific domain

```
1  PREFIX tg: <http://trustgraph.org/schema#>
2  PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
3
4  SELECT AVG(?tracker) AS ?avg WHERE {{
5    SELECT SUM(?tracker) AS ?tracker ?parentRequest ?childDomain
         WHERE {{
6          SELECT COUNT(DISTINCT(?tracker)) AS ?tracker ?
               parentRequest WHERE {
7            ?childRequest tg:isA 'Request' .
8            ?parentRequest tg:isA 'Request' .
9            ?childRequest tg:parentRequest ?parentRequest .
10           ?childRequest tg:tracker  ?tracker  .
11           ?parentRequest tg:domain ?parentDomain .
12           FILTER ( ?parentDomain = <%s>) .
13         }
14         GROUP BY ?parentRequest
15   } UNION {
16         SELECT "0"^^xsd:float AS ?tracker ?parentRequest WHERE
               {
```

```
17            ?childRequest tg:isA 'Request' .
18            ?parentRequest tg:isA 'Request' .
19            ?childRequest tg:parentRequest ?parentRequest .
20            ?parentRequest tg:domain ?parentDomain .
21            FILTER ( ?parentDomain = <%s>) .
22            FILTER NOT EXISTS { ?childRequest tg:tracker ?tracker }
                  .
23          }
24          GROUP BY ?parentRequest
25    }}
26 }}
```

Listing A.5: Average trackers of a specific domain

Retrieve a domain's company

```
1 PREFIX tg: <http://trustgraph.org/schema#>
2
3 SELECT ?domain WHERE {
4   ?domain tg:company ?company .
5   FILTER (?company = '%s') .
6 }
```

Listing A.6: Retrieve a domain's company

Retrieve the country of a domain

```
1 PREFIX tg: <http://trustgraph.org/schema#>
2
3 SELECT ?country WHERE {
4   ?request tg:isA 'Request' .
5   ?request tg:domain ?domain .
```

```
6   ?request tg:serverLocation ?country .
7   FILTER(?domain = <%s>)
8 }
9 GROUP BY ?country
10 ORDER BY DESC(COUNT(?country))
11 LIMIT 1
```

Listing A.7: Retrieve the country of a domain

Retrieve all domains operated by a company

```
1 PREFIX tg: <http://trustgraph.org/schema#>
2
3 SELECT ?domain WHERE {
4   ?domain tg:company ?company .
5   FILTER (?company = '%s')
6 }
```

Listing A.8: Retrieve all domains operated by a company

Check whether enough information for a trust score calculation is available

```
1 PREFIX tg: <http://trustgraph.org/schema#>
2
3 SELECT COUNT(?request) AS ?count WHERE {
4   ?request tg:isA 'Request' .
5   ?request tg:domain ?domain .
6   FILTER(?domain = <%s>) .
7   FILTER NOT EXISTS { ?request tg:parentRequest ?parentRequest
        }
```

```
8 }
```

Listing A.9: Check whether enough information for a trust score calculation is available

A.4. Sourcecode

The whole source code is provided on the attached CD-ROM.