University of Zurich
Department of Informatics

Dynamic and Distributed Information Systems

Master Thesis February 17, 2011

IfiPipes - The RDF UI Widget Framework

Clemens Wilding
of Winterthur ZH, Switzerland

Student-ID: 05-709-738
clemens.wilding@gmail.com

Advisor: **Jonas Tappolet**

Prof. Abraham Bernstein, PhD
Department of Informatics
University of Zurich
http://www.ifi.uzh.ch/ddis

Acknowledgements

I would like to thank Professor Abraham Bernstein for offering me the opportunity to write my Master Thesis at the DDIS group of the University of Zurich and for finding an adequate topic. Many thanks to Jonas Tappolet for his ideas and support during the time of the thesis. Special thanks to my office mates Matthias, Marc, Pascal and Yannick for endless discussions, problem solving and support during coffee breaks. Additionally I would like to thank my parents for giving me the possibility of studying and my girlfriend Daniela for her great support.

Abstract

This master thesis is about the creation of a semi-automatic data retrieval and visualization framework. The framework describes how semantic data can be retrieved from web services or SPARQL endpoints. By using Semantic Web technologies like RDF and OWL, the queried data is analyzed and depending on the type of data, a fitting visualization is displayed. An implementation of the framework is done by creating a web application in Java by using Google Web Toolkit. The application allows a user with knowledge of Semantic Web basics to query data sources from web services and allows him/her to upload new data sets. These can be visualized with little or no programming skills, which should make it easier for non-engineers to use semantic data. The application makes use of the cutting-edge tGraph framework to analyze and display temporal data. Special visualizations features provided in the web application are the display of information on a map, the use of temporal sliders to select time intervals and a tabular representation for larger data.

Zusammenfassung

Diese Masterarbeit handelt von der Erstellung eines halb-automatischen Datenabfragungs- und visualisierung-Frameworks. Das Framework beschreibt wie semantische Daten von Web Services oder SPARQL Endpoints geladen werden können. Mit Hilfe von Technologien des Semantischen Webs, wie RDF oder OWL werden die Daten analysiert und in Abhängigkeit der Art der Daten wird eine passende Visualisierung dargestellt. Eine Implementation des Frameworks wird durch das Erstellen einer Web-Applikation mit dem Namen *IfiPipes*, programmiert in Java mit der Hilfe des Google Web Toolkits, vorgenommen. Die Applikation erlaubt einem Benutzer mit Grundkenntnissen des Semantischen Webs die Abfrage von Datenquellen aus Web Services oder von hochgeladenen Daten. Dadurch wird dem Benutzer die Möglichkeit gegeben semantische Daten zu nutzen, ohne dass Programmierkenntnisse verlangt werden. *IfiPipes* benutzt das innovative tGraph Framework um temporale Daten zu analysieren und dann zu visualisieren. Die Besonderheiten in der Visualisierung sind das Darstellen von geographischen Daten auf einer Weltkarte, die Benützung von temporalen Schiebern um Zeitintervalle zu selektieren und eine tabellarische Übersicht, welche selbst für grosse Datensets flüssig läuft.

Table of Contents

1 Introduction

Nowadays more and more data is available on the web, therefore data aggregation and integration[1] services are more important than ever [Shadbolt et al., 2006; Berners-Lee, 2010]. The data on the web faces two big problems, the first one is that many companies provide no or little access to their data, as they hide the data behind so-called walled gardens. The second problem is the missing structure of most data, which makes its use and especially its reuse complicated and tiresome. By applying new technologies one can't solve the issue of non-accessible data sources, whereas the lack of structure in data can be fixed. Currently most of the integration services only work on specific data sources and can't be reused, because they are using proprietary APIs to access the data. Also most of the non-engineering people are excluded from using data integration services or mash-ups[2] as they are complex to build and require a greater knowledge of programming and/or web technologies [Yu et al., 2008].

The semantic web as a whole and especially the proposition of linked data offers a new solution to the previous stated problems [Berners-Lee et al., 2001]. The technologies behind the Semantic Web, such as the Resource Description Framework (RDF) and its various serialization formats, enable the description of data in the data itself [Hayes and McBride, 2004]. Linked data, is a specific use case of using the semantic web, is the result of using the method of the Tim Berners-Lee to publish structured (semantic) data [Berners-Lee, 2006]. With the help of linked data, data integrators can easily interlink the structured information from various sources, thus rendering the data more useful. So one can say the semantic web tackles the problem of unstructured data sources and the non-standard retrieval and thus enables reuse of tools and code. But it fails at solving the issue of overcomplexity, especially for non-engineers, as a fair amount of knowledge about web technologies is needed to extracting data. In contrast it adds a layer of complexity as one needs deep knowledge about many specifications like RDF, RDFS and OWL and the heavy documentation often scares the average web developer [Ankolekar et al., 2007].

Another problem of the semantic web (and most data stores) is the temporal aspect of the data. Often data stores contain aged or (temporal) inconsistent data, which deteriorates the experience

[1]The terms data integration and data aggregation are often used in different ways. In this thesis I'll follow the most common use: the term data integration depicts the process of combining two or more data sources, while data aggregation is the application of data mining algorithms to further enrich the information. Both of the terms don't necessarily result in the creation of a new integrated respectively aggregated data source.

[2][Merrill, 2006] defines mash-ups as web-based data integration applications, which aggregate and combine third-party data.

[Dean and McDermott, 1987]. The concept of specialised temporal data bases is relatively new compared to the age of conventional data bases[3] and even less explored in the semantic space. There is also not yet a reference implementation[4], but some work in adding temporal constraints to RDF and SPARQL is done [Gutierrez et al., 2005; Tappolet and Bernstein, 2009].

1.1 Motivation

The goal of this thesis is to twofold. The first goal is to create both a framework for describing the data integration service and the second one to represent the visualization styles depending on the type of the semantic data. The data integration framework acts as an interface for various data sources like SPARQL endpoints, web services returning RDF or files. The visualization framework describes different types of data visualizations and their relation to the data sources. It should be possible to extend both frameworks; the data integration framework should be extendable to create new data integration elements and the visualization framework must provide the extensibility to allow developers to create new visualizations and define their relation to the data sources in the framework.

The second goal is to implement these frameworks by mapping them to the application. The result of the implementation should be a web tool which allows the user to select various data sources and depending on the returned data, a corresponding visualization is drawn. So if one selects a data source containing geographical data, a visualization on a map is one possible outcome, whereas by selecting a temporal data source the application draws a temporal visualization. This can be solved by creating a visualization where one has a slide to go back and forth in time. If the data isn't specific enough for a graphical visualization, there should be a fall-back visualization. This visualization should be as specific as possible and prepared in a human readable form.

1.2 Suggested solution

The suggested solution for the frameworks is to create two ontologies, one for data integration and one for graph visualization. These ontologies will be written in OWL, the ontology language for the Semantic Web. As the application should be easy enough to use by the user who has a basic knowledge about the concepts of the Semantic Web, but lacks programming skills, the graphical representation of the data integration will be similar to the UI of Yahoo Pipes[5] or similar tools. It'll allow the user of the application to use graphical programming to select data sources and aggregate and manipulate data results fetched from the sources. The visualization ontology will describe the visualizations and provide mappings to different types of data. This includes at least temporal and geographical data. As ontologies are easily extendable a developer will be able to extend the framework to create both new pipe elements and new visualizations.

The web application will do the data analysis and integration on the server side, while the client side will be responsible for the user interface and the filtering of data. For reading and

[3]While data bases exist since the 1960s and SQL was developed in the 1970s, the the official temporal extension for SQL (T-SQL2) was specified in 1994 [Snodgrass et al., 1994]

[4]Although there exists a W3C working draft for a time ontology: `http://www.w3.org/TR/owl-time/`

[5]`http://pipes.yahoo.com/pipes/`

parsing RDF a Semantic Web framework like Jena[6] will be used and for reading temporal data the tGraph framework serves the purpose. To focus less on web development and more on delivering content, the web framework Google Web Toolkit[7] will be used. The application will use predefined widgets where possible, new widgets will be developed when necessary.

1.3 Structure

The structure of the thesis is derived from [Bernstein, 2005] and includes the following chapters:

Chapter 2 explains the background behind the thesis, thus explaining the concepts of the Semantic Web, data integration and data visualization. Section 2.1 is about the Semantic Web and covers excursions in the fields Linked Data and temporal semantic data. Section 2.2 briefly describes the theory of data integration, in the traditional way and extensively in the newer semantic way. The final section's topic is data and graph visualization. It consists of a listing of different types of data and graph visualization and shows benefits and disadvantages of each type.

Chapter 3 highlights related work in the fields data integration and data visualization with a special focus on Semantic Web applications. There is few related work which combines all topics covered in this thesis, but each category on its own provides more than enough material. The chapter shows not only complex web applications, but also recent progress in the research communities.

Chapter 4 describes *IfiPipes*, the resulting application. Section 4.1 describes the used software, both frameworks and libraries are presented. Section 4.2 is one of the most important parts of the thesis, because it presents the two ontologies which serve as a framework for the application. In the following sections the architecture and a few details of the implementation are getting a thorough look. The chapter concludes with a user guide and a selection of use cases, including a graphical presentation of the application.

Chapter 5 shows the limitations of the framework, the application and also the used technologies.

Chapter 6 lays out suggestions for future work, including the proposal of an universal visualization framework.

Finally, Chapter 7 draws the conclusion of the thesis and includes a personal recapitulation.

[6]http://openjena.org/
[7]http://code.google.com/webtoolkit/

2 Background

This chapter consists of the theories and concepts behind the thesis. The initial focus is on the term Semantic Web and its offsprings RDF, SPARQL and Linked Data. As an extension to the Semantic Web, a temporal framework is presented, which adds a validity to information entries.

Second the terms data aggregation and integration are brought closer to the reader. The thesis's focus is on data integration, especially semantic data integration. But the term data aggregation is explained shortly especially in the traditional statistical approach.

Third the focus is on data visualizations. This includes a discussion of the use of data and especially graph visualizations and presents several types of visualizations.

2.1 The Semantic Web

The Semantic Web, a term shaped by Tim Berners-Lee, the inventor of the World Wide Web (WWW) and director of the World Wide Web Consortium[1] (W3C), is a combination of various concepts and technologies with the goal to enable computer to understand the meaning respectively the semantics of data. The ultimate goal is that computers or agents can do many of our burdensome tasks, like finding the perfect appointment or the best driving route, as depicted by Berners-Lee in his influential paper [Berners-Lee et al., 2001]. His paper brought the term and the concepts behind the Semantic Web to a broader audience and since the publication in 2001 the Semantic Web is growing steadily. To achieve the goal of intelligent agents the Semantic Web is based on three concepts and one application:

- **Expressive Meaning**: by bringing a structure to the content of web pages, data becomes readable for software agents. This is based on the concept of hyperlinks where 'anything can link to anything', which results ultimately in the Internet of Things where anything has a URL [Gershenfeld et al., 2004];
- **Knowledge Representation**: the Semantic Web was built with compliance to the open world assumption, which says if contradictory values may exist for a statement[Drummond and Shearer, 2006]. In contrast the closed world assumption presumes that every statement

[1] `http://www.w3.org/`

which is not known is false. The challenge is therefore to provide an expressive language and rules for reasoning. The expressive language for the Semantic Web is called Resource Description Framework (RDF), which uses a natural language-like syntax.

- **Ontologies**: are collections of information which define classes of objects and relations among them. There exist many ontologies for different purposes, for example the FOAF ontology [2] is used to describe people and their social graph.

- **Agents**: result from applying the techniques of the Semantic Web to software. They are programs created by web developers which fetch content from various semantic data sources and process information to exchange them with other agents or present them to the user. The value of these agents is exponentially higher the more data is available on the web.

The set of standards is designed and developed by the W3C, which has several groups devoted to improve the Semantic Web[3]. The efforts concentrate on 5 topics[4], these are:

- **Linked Data**: the new main interest of many Semantic Web researchers, to interconnect data from various sources. The group concerns itself with working on standards like RDF, RDFa and GRDDL. Linked data is the main driver of the Semantic Web, as the focus towards releasing large, interlinked data sets. The more data is available, the more valuable it gets, as more conclusions can be drawn.

- **Vocabularies**: the group works on expanding the standards on the web ontology language (OWL) and the simple knowledge organization system (SKOS).

- **Query**: to retrieve semantic data the querying language SPARQL Protocol and RDF Query Language (SPARQL) was created. SPARQL is similar to SQL used for relational databases.

- **Inference**: the process of inferring or reasoning is the use of higher logic to find new relations between resources.

- **Vertical Applications**: the group focuses on building applications in the field Health Care and eGouvernment. While the first four groups are devoted to drive the technologies behind the Semantic Web further the last group creates ontologies and real-life applications.

The semantic web is not a replacement for the existing web, but like the Web 2.0, which added a social layer, the semantic web is an addition. Some people therefore call the semantic web one of the foundations of the Web 3.0 [Shannon, 2006; Spalding, 2007]. The technology stack of the Semantic Web is depicted in Figure 2.1, where one can see that the Semantic Web borrows a few technologies from the original web stack, like the concept of unique identifiers (URIs/IRIs) or the markup language (XML), which won't be subject of this thesis. But the next few subsections contain explanations of the fundamentals of the Semantic Web: RDF, SPARQL and OWL, respectively ontologies in general.

[2] `http://xmlns.com/foaf/spec/`
[3] see `http://www.w3.org/2001/sw/#groupsforadetailedoverviewofallgroups`
[4] see `http://www.w3.org/standards/semanticweb/` for even more details

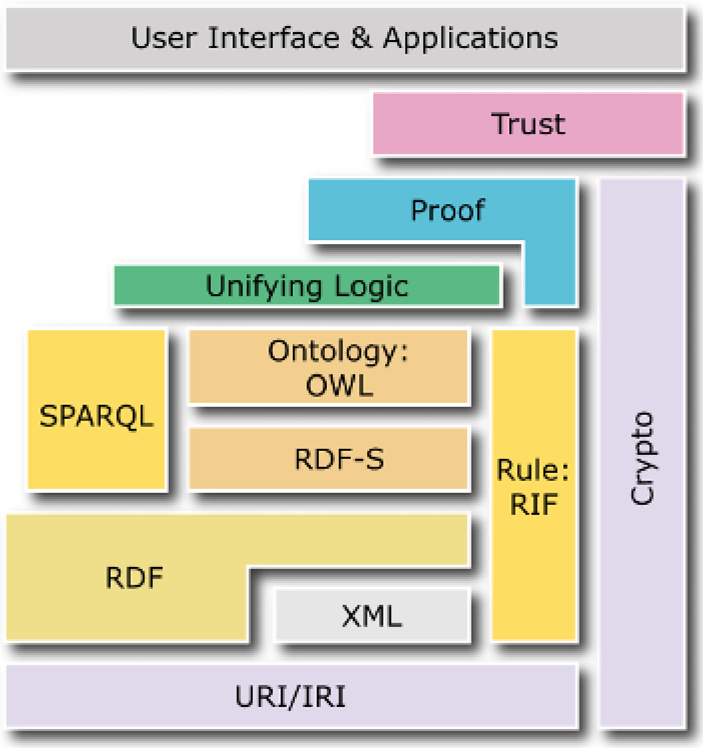


Figure 2.1: The Semantic Layer Cake or Semantic Web Stack. From [Bratt, 2007], p. 24.

2.1.1 The Resource Description Framework

The Resource Description Framework (RDF) is one of the foundations of the Semantic Web. It acts as a meta-data model and can also be used to model information. The origins of the framework go back to 1999, when the W3C published a specification of its data model and the XML-based syntax as a recommendation[5]. The specification was revised in 2004 and is stable since then. The recommendation is split in several parts including the concepts [Klyne et al., 2004], the semantics [Hayes and McBride, 2004] and the RDF/XML syntax specification [Beckett and McBride, 2004].

The main concepts behind RDF are:

- **Graph Data Model**: any statement in RDF is represented as a so-called triple and consists of a subject, a predicate and an object. This syntax resembles natural language and is more expressive than the traditional key/value (or in above words subject/object) store. The statement represented as a graph is depicted in Figure 2.2. It is a directed, unidirectional graph where the subject points always towards the object. One or more statements together form a graph, where the subjects and objects are called nodes and the predicates are often called properties.
- **URI-based Vocabulary**: nodes can be one of 3 types: URIs, literals or blank nodes, while properties are always URIs. URIs (Uniform Resource Identifiers) are always unique, so

[5]The old draft can be found on this page: `http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/`

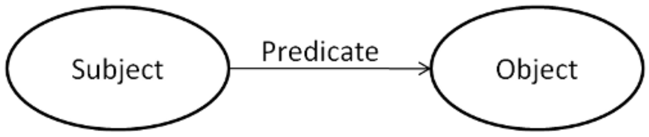


Figure 2.2: A statement consisting of a subject, a predicate and an object

different nodes must have different URIs. This enables the reuse of a URI for different statements or even different graphs. In contrast different literals may have the same value, but therefore can't be reused. While the concept of URIs and literals is easy to understand, blank nodes are more complex. They are a special type of node, they are unique in their graph, but they don't have a URI. The problem with blank nodes is that if several data sets are merged together the outcome of the merger isn't clear [Hayes, 2009]. There are many people regarding the inclusion of blank nodes as one of the biggest mistakes and even Tim Berners-Lee suggests you should give yourself a URI [6]. Also the first rule of publishing Linked Data is 'Use URIs as names for things' [Berners-Lee, 2009].

- **Datatypes and Literals**: RDF uses the existing XML datatypes to represent 'primitive' values, like integers, doubles, dates or strings. The only datatype specific for RDF is used to annotate XML in RDF: `rdf:XMLLiteral`. Literals represent a lexical form of these 'primitive' values and may or may not be restricted by datatypes. While typed literals are annotated with a datatype URI, plain literals are just strings.

There are several representations of RDF, the most often used is the representation in XML, called RDF/XML, which is part of the official RDF specification. XML is the main choice because when the specifications for the Semantic Web was created XML was the state-of-the-art markup language. The benefits of XML is that it is machine-readable and very expressive. But there is also some criticism that the syntax is too verbose and overly complex, resulting in less human-readable code and cumbersome writing [Crockford, 2006]. So there are a few notation formats that are less expressive, but also less complex and more human-friendly. N-Triples[7] or Notation3[8], both which are created by the Semantic Web Community, are an example of such notations. Very popular at the time is Turtle[9], the Terse RDF Triple language. Turtle is a subset of N3 and an extension for N-Triples and resembles the triple syntax used in the SPARQL query language. Turtle is easy to parse for a computer but nevertheless more readable than RDF/XML so it's the preferred alternative to XML. Another non-XML format is RDF/JSON[10], which is gaining popularity in the web-community, as JSON is the preferred way to encode messages in the web and thus the syntax is already known. JSON is called by its inventor the 'fat-free alternative to XML' and is easy to parse (especially in the JavaScript) and less 'bloated' than XML [Crockford, 2006; Alexander, 2008].

[6]As mentioned on his blog: `http://dig.csail.mit.edu/breadcrumbs/node/71`
[7]see `http://www.w3.org/TR/rdf-testcases/#ntriples`
[8]see `http://www.w3.org/DesignIssues/Notation3.html` (N3) for the specification of N3
[9]see `http://www.w3.org/TeamSubmission/turtle/forthespecificationofTurtle`
[10]see `http://n2.talis.com/wiki/RDF_JSON_SpecificationforthespecificationofRDF/JSON`

2.1.2 SPARQL

The name SPARQL is a recursive acronym and stands for SPARQL Protocol and RDF Query Language SPARQL. It can be used for querying RDF graphs and has a SQL-like syntax. The current W3C recommendation is SPARQL 1.0[11], while SPARQL 1.1 is a working draft. SPARQL 1.0 borrows the basic clauses from SQL, so users with SQL experience will have no problems switching, as they know almost all the keywords. A simple SPARQL query is shown below in Listing 2.1:

Listing 2.1: A simple SPARQL `SELECT` query

```
SELECT ?artist ?name
WHERE
{
  <http://ex.com/store#cd1> <http://ex.com/music#artist> ?artist .
  ?artist <http://xmlns.com/foaf/0.1/name> ?name.
}
```

The syntax in the `WHERE` clause is similar to Turtle, as mentioned before. The first part on the clause gets the object `?artist`, which is the subject in the second triple. The results of the query are the URI and the `foaf:name` of the artist of the cd1 in the store. The SPARQL query result encoded in XML is depicted in Listing 2.2:

Listing 2.2: The result of a simple SPARQL `SELECT` query encoded in SPARQL/XML

```
<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
  <head>
    <variable name="artist"/>
    <variable name="name"/>
  </head>
  <results>
    <result>
      <binding name="artist">
            <uri>http://ex.com/store#artist1</uri>
      </binding>
      <binding name="name">
            <literal xml:lang="en">Radiohead</literal>
      </binding>
    </result>
  </results>
</sparql>
```

There exists also a serialization of SPARQL query results in JSON from the W3C[12], which can be seen in the Listing 2.3.

[11]see `http://www.w3.org/TR/rdf-sparql-query/` for the specification
[12]see `http://www.w3.org/TR/rdf-sparql-json-res/`

Listing 2.3: The result of a simple SPARQL `SELECT` query encoded in SPARQL/JSON

```
{
  "head": { "vars": [ "artist" , "name" ]
  } ,
  "results": {
    "bindings": [
      {
        "artist": { "type": "uri"
                        , "value": "http://ex.com/store#artist1" } ,
        "name": { "type": "literal" , "value": "Radiohead" } }
    ]
  }
}
```

For web applications developers prefer to use the JSON encoding, especially when working on the client-side with JavaScript. Another advantage is the lower number of bytes, which can be crucial for web applications.

SPARQL allows more complex queries as well, for example one can query for literal values (`SELECT ?x WHERE { ?x ?p "cd" }`), using the `FILTER` keyword to use regular expressions to filter results (`FILTER regex(?name, "\^{}Radio")`) or using `OPTIONAL` to include resources which don't fulfil all criteria (`OPTIONAL { ?artist foaf:mbox ?mbox }`). Another important keyword is `CONSTRUCT` which allows one to query a graph and return a new graph according to the used query. A sample query is shown in listing 2.4:

Listing 2.4: A simple SPARQL `CONSTRUCT` query

```
CONSTRUCT { ?artist <http://xmlns.com/foaf/0.1/name> ?name }
WHERE { ?artist <http://ex.com/music#artistName> ?name. }
```

The result of this query in RDF/XML is shown in Listing 2.5:

Listing 2.5: The result of a simple SPARQL `CONSTRUCT` query

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:foaf="http://xmlns.com/foaf/0.1/">
  <rdf:Description>
    <foaf:name>Radiohead</foaf:name>
  </rdf:Description>
</rdf:RDF>
```

SPARQL 1.1 provides even more features, which are supposed to be included in the main specification, while guaranteeing no incompatibilities with the older version. Version 1.1 brings following new features

- **Aggregates**: apply expressions over groups of solutions. New clauses are for example `GROUP BY`, `COUNT` or `SUM`.

- **Subqueries**: one or multiple `SELECT` queries in the `WHERE` clause.

- **Negation**: allows the filtering of results depending on the presence of a graph pattern or not, by using the `NOT EXIST` respectively `EXISTS` clauses. It also allows to subtract solutions from the results by using `MINUS`.
- **Expressions in the `SELECT` clause**: SPARQL 1.1 allows the creation and reuse of variables by using the keywork `AS` to assign them to a value.

Extensions for SPARQL 1.1 include for example federated queries[13], which would allow one to write queries for more than one remote SPARQL endpoint, by using the `SERVICE` keyword. Federated queries could give the Linked Data community a huge boost as it allows the integration of multiple data sources in one query.

2.1.3 Linked Data

Linked Data is about setting guidelines for data to be published on the web and interlinking data sources. The usefulness of Linked Data is proportional to the availability of Linked Data on the web. Tim Berners-Lee coined the term in a design note in 2006 [Berners-Lee, 2006], where he set up simple rules for Linked Data. These four rules are:

1. Use URIs as names for things.
2. Use HTTP URIs so that people can look up those names.
3. When someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL).
4. Include links to other URIs. so that they can discover more things.

The rules should be quite intuitive for anyone familiar with a basic understanding of the Semantic Web, but there are some discussions about the implementation of these rules. The first rule, to give names for things, is there to ensure one can reference them from other data sets. The second rule, to use HTTP URIs instead of new URI schemes, exists because HTTP is an existing and widely used standard. The third rule, that information should be on the web and accessible via a URI, is to ensure that the data can be interlinked, by using existing technologies. The forth rule, to interlink data sources, exists to create a web of data, where the sum of the value of all information is greater than the sum of all its parts.

The current state-of-the-art guide to publish Linked Data on the web is explained in [Bizer et al., 2007], where there are more details on the implementation. An important part of the implementation is to distinguish between information and non-information resources. While information resources are resources available on the net, non-information resources point to physical things or concepts, which are not available on the web[14]. By accessing an information resource, the server returns a representation of the information resource and the HTTP response code `200 OK`. If one tries to access an non-information resource, which can't be accessed directly, the server returns a `303 SEE OTHER` message, to redirect the client to an information resource.

[13]see `http://www.w3.org/2009/sparql/docs/fed/service.xml` for the draft of the specification
[14]See [Lewis, 2007] for a more detailed explanation.

Another implementation issue is the problem that raw RDF code isn't very readable for humans, so it is suggested to use the HTTP mechanism content negotiation. The technique is enabled by creating three resources for each URI in a graph. First a non-information resource is created with the same URI as the graph, a so-called vocabulary URI. Second a URI pointing to an information resource is created, which serves raw RDF and third a URI pointing to another information resource is created, serving data represented in HTML. The client makes use of this by sending a `GET` request to the non-information resource. Depending on the accept header (either `Accept: application/rdf+xml` or `Accept: text/html`) the server returns with a `303 SEE OTHER` message, including the URI of the requested type of information resource, which the client resolves. Another important part of the publication of Linked Data are URI aliases, to ensure the interlinkage between data sources. The preferred was to tell two resources are the same is by using the `owl:seeAlso` keyword. To test if a Linked Data source is published correctly one can use the Linked Data validator VAPOUR[15].

The document also gives hints to avoid a few RDF features, most of them were already criticized by parts of the community. First the researchers discourage the use of blank nodes. Even older ontologies like FOAF started using URI references instead of blank nodes. Second RDF reification is discouraged, because its semantics are unclear and the statements are hard to query with SPARQL. The third point is SPARQL related as well: RDF collections and containers should be avoided, as SPARQL queries are cumbersome to write when querying collections or containers. It is suggested to create several triples using the same predicate, which is almost the same as using an unordered `rdf:list`.

The growth of Linked Data can best be seen by comparing the 2010 Linked Data cloud (see Figure 2.3), which compiles all greater semantic data sources, with versions from previous year. For example in 2007 there were only twelve data sets, while in the current version there are more than two hundred data sets and many of them are way bigger then before. The biggest and most important source is DBpedia[16], which extracts the information from Wikipedia and releases them as semantic data. The data isn't live data from Wikipedia, but rather there are dumps from previous Wikipedia versions. The current version of DBPedia, 3.6, uses dumps from October/November 2010. DBPedia is not only the biggest data source, but also very interlinked in the Linked Data cloud, as many data sets point to DBPedia.

There exists also a rating scheme for linked data, where the data is given one to five stars[17]:

*	make your stuff available on the web (whatever format) under a open license
**	make it available as structured data (e.g., Excel instead of image scan of a table)
***	use non-proprietary formats (e.g., CSV instead of Excel)
****	use URIs to identify things, so that people can point at your stuff
*****	link your data to other data to provide context

It can be seen, that one to three star data have nothing to do with the Semantic Web, but rather with traditional web applications and their APIs. Four star data may or may not use semantic

[15]see `http://validator.linkeddata.org/vapour` for the validation service.
[16]`http://dbpedia.org/`
[17]A more detailed explanation with examples for each rating can be found on: `http://lab.linkeddata.deri.ie/2010/star-scheme-by-example/`.

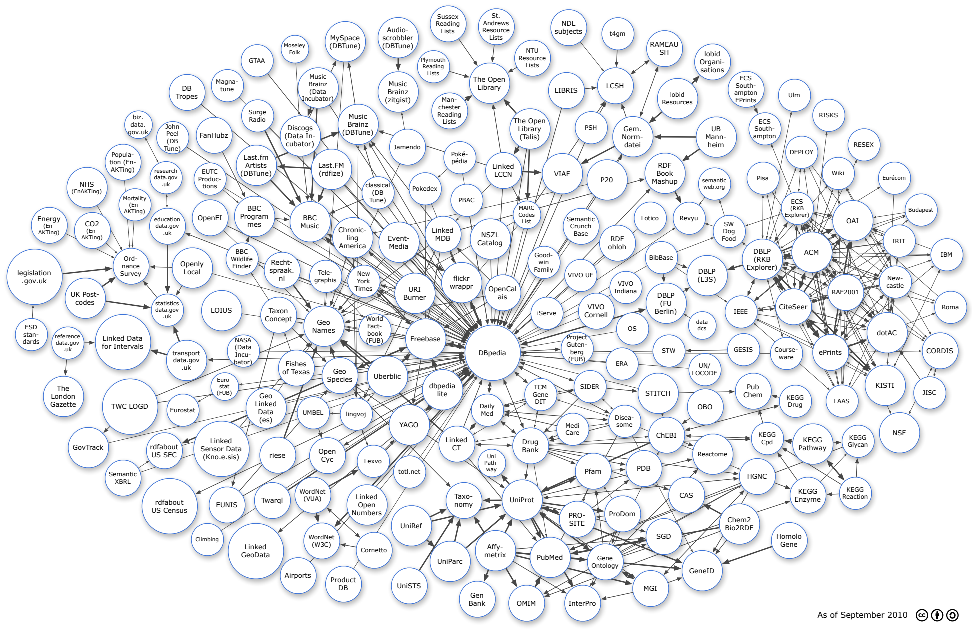


Figure 2.3: Linking Open Data cloud diagram, by Richard Cyganiak and Anja Jentzsch. http://lod-cloud.net/

technology to give things an URI, while only five star data unleashes the full power of the Semantic Web. The state of linked data so far is best described in [Bizer et al., 2009], where the authors show data sets and tools to use them and also list the current challenges in semantic research.

2.1.4 Temporal RDF

The rise of the Semantic Web resulted in the need for temporal annotated data, as most data isn't static, but changes from time to time. There were many suggestions, some tried to model time with OWL; others suggested to used Description Logic. Another approach was the temporal extension of RDF, which is an extension for regular RDF. A framework is presented in [Gutierrez et al., 2005; Gutierrez et al., 2006] and which addresses many issues of temporal information. One issue is the question of versioning vs labeling: Versioning creates a snapshot for each graph (a new snapshot is created if the graph is changed), which is a problem for some queries. Labeling on the other side annotates graphs or single triples with a time. The two versions of adding time dimensions correspond to the temporal dimensions in databases: Validity, which corresponds to labeling, ensures that the data is valid in the modeled world; transaction time, corresponding to versioning, is the time when data is actually stored in the database. Another issue is the question whether to use time points or time intervals. In [Tappolet and Bernstein, 2009] time intervals are the preferred choice, but time points can be expressed by setting an equal value for start and end time. The paper also presents a syntax for graphs, by using named graphs to annotate triples with a new temporal element, which results in quads. The authors also propose an extension to the query language SPARQL, called τ-SPARQL, and provide mappings form τ-SPARQL to regular SPARQL. A simple τ-SPARQL can be seen in Listing 2.6:

Listing 2.6: A simple τ-SPARQL `SELECT` query

```
SELECT ?start ?end ?name
WHERE
{
  [?start, ?end] <http://ex.com/music#artist1> foaf:name ?name.
}
```

The query return the name of the artist and the time in which the artist is valid, in this context the time during the musician is/was active. To model data which is valid from existence on uses the keyword `EVER`; data which is valid until now is annotated with `NOW`.

2.2 Data Combination

Data combination covers both approaches data integration and data aggregation. The process of data integration is the merging of different data sources and combination of them. Data integration often results in creating a data warehouse, a centralized data storage containing all information. By using semantic technologies like ontologies, semantic data integration allows the easier combination of these data sources. Data aggregation is the process of applying data mining algorithms to enrich the data gained from one or multiple sources. In an ubiquitous Semantic Web the two data combination methods would approach each other more and more, as the data mining algorithms could be replaced by semantic metadata.

The field of data combination is growing during recent years. As more and more information is created, especially on the web, the combination, aggregation and manipulation of data becomes interesting for companies and individuals. By using APIs, developers create tools which allow the user to create data mash-ups.

2.2.1 Data Aggregation

The space of data aggregation is mainly dominated by the statistical research of social sciences [Clark and Avery, 1976]. Through combination and data mining one can know more about single entries of the data sources. Data aggregation has also raised privacy concerns in recent years, especially with the rise of the internet [Solove, 2002]. There are several companies competing in the data aggregation space, not by creating software which can be used for data aggregation, but rather by having huge databases of aggregated personal data.[18].

Data aggregation could be enhanced by using the Semantic Web. First one could use a reasoner to enrich the data, by finding new connections or spotting contradictions in the data source. Second by querying the Semantic Web, respectively linked data sources one can find more information about the data entries. This may result in better aggregation results, as more data means better results from data mining algorithms.

[18]see `http://www.acxiom.com/Pages/Home.aspx` or `http://www.lexisnexis.com/` for more information on data aggregator companies

2.2.2 Data Integration

Data integration is currently widely discussed in the enterprise space [Halevy et al., 2006]. The research in this area is one of the oldest topics in database research, but there is no ‘silver bullet‘ solution, which would work for all kinds of databases and all kinds of data [Ziegler and Dittrich, 2004a]. The traditional approach of data integration is the combination of several data sources in one central database, a so-called data warehouse. The problem is often that the databases don’t have the same format. Some data may be stored in a relational data base, while other data is stored in flat files and other data is encoded in XML. Another problem is that there are many data entries depicting the same thing, but with a different description. If one has a working data model, the work is less tiresome, but still can rarely be automated. The structural problem to connect several databases is rather easy, in comparison to the combination of the data. One of the biggest problem are data entries with the same name, but totally different meanings. So the semantics of data integration are much harder to grasp, as the structural problems of having different databases or the syntactical problem of having different syntax for storing data is.

2.2.3 Semantic Data Integration

The technologies of the Semantic Web can help to facilitate the problem of unclear semantics. If one applies the practices of linked data, one can use ontologies and by using the `owl:sameAs` keyword one can describe the same resources in different data sources [Gardner, 2005]. Another problem which is easier to solve by using semantic tools is the combination of data sources [Ziegler and Dittrich, 2004b]. Even if the data sources aren’t interlinked one can combine them without a problem and by using reasoning or data mining one may find relations between entries. The Semantic Web also solves the data of double entries as each triple is unique in the graph. By using reasoning and ontologies also contradictions in the data sources can be found.

By using the principles of linked data, it isn’t even necessary to combine the data sources to a data warehouse, but with the upcoming SPARQL 1.1 an federated queries it is possible to make queries on several data sources. This results in a looser coupling and smaller and more specific data sources, which results in better maintainability and often faster query results. Another option would be to add a semantic layer on top of the existing data bases. This option is often the best approach as there are many legacy systems, which shouldn’t or can’t be touched, as they are still critical to operate. By adding this semantic integration layer, one can create views of the data sources, without meddling with the data itself. The semantic layer now allows a few possibilities a traditional layer or wrapper can’t provide: First one can interlink the data entries from several sources to point out double entries and second one can make links to another linked data sources. This results in enriched data, for example the address data base can be combined with geographical data from the GeoNames web service[19] [Kobilarov et al., 2009].

[19]`http://www.geonames.org/`

2.3 Data Visualisation

Data visualization is the science of visual representation of data[20]. The used data can either be raw data or be abstract data. The two main categories of data visualization are statistical graphics and thematic cartography. While the cartographic visualization mostly use geographical data, statistical visualizations can be applied to all data domains. Data visualization is a very old field, there exist town maps from more then 8000 years ago. In the 17th century statistical visualizations were born and the scientific use of data visualization began in the 19th century. In the second half on the twentieth century the visualization science gets a huge boost as the computer progresses rapidly. With the help of computer programs scientists can finally visualize large data sets and use less abstracted data. Also the computer helps with the analysis of data sources, so new conclusions can be found and the visualization can be rearranged faster [Spence, 2001]. This is especially helpful if one doesn't know what data to expect. The computer age results in two new achievements, first many new visualizations are created, mainly in the field of 3-dimensional visualizations and second the ubiquity of visualizations. As nowadays much more data is produced then even five years ago this can lead to an information overload. With the help of visualizations of the newly created data humans can process these large sums of data faster. In [Sears and Jacko, 2008] the author mentions six points how information visualization helps cognition:

1. **Increased Resources**: offload work from cognitive to perceptual system. Graphics can often be consumed in parallel, whereas text can only be consumed in serial.
2. **Reduced Search**: visualization group information together which results in reducing search. Also visualizations can represent large amounts of data in small spaces.
3. **Enhanced Recognition of Patterns**: recognition is easier than recall for humans. By organizing visual data in structures pattern emerge.
4. **Perceptual Inference**: visual representations make some problems obvious. Also computers can use their computing power to help with complex visualizations.
5. **Perceptual Monitoring**: visualization allow the monitoring of large numbers of events in parallel, if the visualization is organized so that changes stand out.
6. **Manipulable Medium**: instead of static diagrams visualization can be explored by changing the parameters leading to the visualization.

The points mentioned above lean towards the field of human-computer-interaction, a interdisciplinary field which combines the fields of computer science, design and behavioral sciences.

2.3.1 Graph Visualization

As the concept of the Semantic Web is to store data as a graph it is common sense to create graph visualizations of data. Unlike data or information visualization the graph visualization science has its origin in the mathematical graph theory. Graph theory normally uses dots to represent

[20]There are many definitions of data visualization, information visualization and scientific visualization. In this thesis I will follow the definitions of [Friendly and Denis, 2001], which are also explained in detail on: `http://datavis.ca/milestones/index.php?page=varieties+of+data+visualization`.

vertices and arcs to display edges between the connected vertices. Depending on the condition if a graph is directed or not, edges are displayed as arrows pointing in the direction of the vertex. While the basics of graph visualization are almost always edges and vertices, the layout of these visualization can change dramatically. The choice of layout is important, as each layout has its advantages and its problems. A few aesthetics one should apply to a general graph layout are listed below [Cruz and Tamassia, 1998]:

- Minimize crossings
- Minimize area
- Minimize bends
- Maximize slopes
- Maximize smallest angle
- Maximize display of symmetries
- Keep edge length uniform
- Distribute vertices uniformly

Sometimes it is impossible to apply all the aesthetics, so one should try to maximize a certain number, in order to draw a good looking layout [Di Battista et al., 1998]. There are many types of graph layouts, a list of the most important ones can be found in Table 2.1:

Layout Type	Description	Remarks
General Graph Layout	The plain graph representation with dots and arcs as vertices and edges.	If possible the aesthetics listed above should be applied.
Planar Graph Layout	A graph is planar if none of its edges crosses another.	While planar graphs are often desirable, it isn't always possible to create a planar graph, especially for larger graphs with many edges. If possible general graphs are *planarized* and then a planar algorithm is applied.
Directed Graph Layout	Directed graphs have directed edges, which means they point to at least one vertex.	Directed graphs are parted in acyclic and cyclic graphs. Cyclic graphs are harder to display, so it is often tried to cut a few edges to remove cycles.
Tree Layout	Trees are often used to represent hierarchical graphs. Vertices are placed on different levels to display the hierarchy.	The layout is good for hierarchical graphs and graphs with a small width. If the hierarchy is too deep the clarity of the graph deteriorates.
Circular Layout	The nodes are arranged on a circle, to emphasise group and tree structures.	The circular layout is preferred for ring or star topologies and works very well for displaying networks.
Hierarchical Layout	The hierarchical layout is similar to the tree layout, it places the vertices on layers to display the hierarchy.	In contrast to the tree layout, where each leaf can only have one parent, the hierarchical layout allows the reunion of branches.
Organic Layout	The organic layout displays similar nodes together. This results in organic looking graphics, resembling structures found in nature.	The organic layout is often used by applying a force-base algorithm[21].
Orthogonal Layout	The orthogonal layout displays graphs in rectangular forms.	The orthogonal layout produces good results for medium sized graphs and results in compact drawings with few crossings.

Table 2.1: List of Graph Layouts

2.3.2 Types of Information Visualization

There exist an infinite number or visualizations, from simple textual visualization to 3- or more-dimensional visualizations. Each of this visualizations is best suited for a specific data format. In [Shneiderman, 2002] the author creates a taxonomy of data types. The seven data types of the taxonomy are:

- **1-dimensional**: linear data types are documents, lists or source code and are arranged in a sequential manner. Visualization issues are font, color and size usage and scrolling for large data sources.
- **2-dimensional**: data which can be displayed on a map or planar data. Issues are finding connections between items or filting of data.
- **3-dimensional**: data from real-world objects like atoms or animals. Usually created by computers, the issues of this visualization are navigating in the 3rd dimension and display of relationships.
- **Temporal**: temporal data can be displayed as time lines. Shneiderman classifies temporal data type that is separate from 1-dimensional data, as it is widely used and vital for many visualizations. Issues are the filtering of events before, after or during a time interval.
- **Multi-dimensional**: data is often multidimensional (e.g. most data from databases). Issues are displaying this additional dimensions in a 2- or 3-dimensional space. Solutions are the omission of dimensions or the enabling / disabling of dimension with additional controls.
- **Tree**: hierarchical structures where each item has only one parent. Issues of the visualization are the displaying of too broad or too deep trees.
- **Network**: network data has often many connections and can't be arranged in a tree in an aesthetically pleasing way. Issues are large data sets and nodes with many connections.

Depending on the data types above, there are many types of visualizations. A number is presented below:

- **Tabular Visualization**: the tabular visualization is one of the simplest visualizations and resembles the textual representation. The data is arranged in a table where one row equals one data entry and one column equals a category. If the data is at least semi-structure the tabular visualization is one of the easiest visualizations to create. It is good for small to medium data sizes, for large sets the clarity gets lost.
- **Chart Visualization**: a chart is a visualization of numerical data a special symbols. There are many types of chart visualizations, where each one has a special use case. Typical chart visualizations are bar charts, pie charts or line charts. Charts are less useful for non-numerical data and show only aggregates.
- **Map Visualization**: a visualization of data as points on a map. The map visualization works best for geographical data. It can also be combined with other visualization, for example one can draws charts on the map. A special type of map visualization is the heat map, where region of the map are colored according to the data.

- **Treemap**: is a visualization where data is displayed in nested rectangles. A treemap is a suitable visualization for hierarchical data. Sometimes colors are applied to the rectangles to further distinguish them.
- **Temporal Visualization**: usually on a time- or story line. Interactive visualizations make often use of a slider to change the time value. One can say temporal visualizations are a special kind of multi-dimensional visualizations, where the time is another dimension. Temporal visualizations can be used in addition to one of the other visualizations.
- **Mindmap**: often visualized in a circle (similar to the organic layout for graph visualizations), where the most important node of the data is in the middle and there are connections from there to other nodes. Mindmaps are often used when brainstorming. A computer visualization seldom uses the mindmap to visualize data.
- **Multi-dimensional Visualization**: use the power of three or more dimensions to display more facets about the data. While 3-dimensional visualizations are common nowadays and often selected to have not only two, but three axis, four- and more-dimensional visualizations are still in its infancy.
- **Website**: a mixed form of visualization and can consist of a number of other visualizations. Semantic data is often displayed on a website. Depending on the data and type of website, one or more visualizations are selected. The simplest choice for semantic data is often to visualize it in a tabular form.
- **Graph Visualization**: the most common layouts are presented in Table 2.1 [22].

In [Shneiderman, 2002] the author Ben Shneiderman also presents the Information Visualization Seeking Mantra: 'Overview first, zoom and filter, then details-on-demand'. Shneiderman argues that the steps for using an information visualization are always the same. First present the user with a zoomed out overview of the visualization. Second the user zooms into the visualization to the point of his/her interest. Next there are several possibilities, the user may want to use a filter to remove uninteresting items or he/she might want to select an item or a group to gather more details about it. Other guidelines for creating visualization information are that one should allow the user to see relationships between the items on demand and the application should keep a history of all commands the user is doing, to allow functions like undo or replay. At last the user should be allowed to extract the data or parts of it in a useful format.

2.3.3 Semantic Data Visualization

One can argue that because the Semantic Web is in its core a graph, it has to be visualized as such The authors of [Schraefel and Karger, 2006] argue, that it rarely makes sense to display RDF data as a graph. The strongest argument they provide is that the internet is in it's core also a graph, but is rarely depicted as such. The problem with semantic data as graph visualization is mainly the usability. A list of problems can be found in the paper, the most important are: graphs don't make use of our daily used layouts, they don't scale for large data sets and editing is often hard.

[22]For more visualization types the reader is referred to an article showing modern approaches in data visualization (`http://www.smashingmagazine.com/2007/08/02/data-visualization-modern-approaches/`) and a periodic table of visualizations (`http://www.visual-literacy.org/periodic_table/periodic_table.html`).

By applying the guidelines of information visualizations one can weaken the problem, but instead of a rash determination on one type of visualization, one should rethink other possibilities. One possibility is to display data like on a website, with multiple visualizations depending on the data. These sites are often single-purpose, they only fit a special data source. The other possibility is to create a multi-purpose user interface, which is less specified for the data, for example a tabular representation for RDF data. A vision of the Semantic Web is that one has only to create user interfaces for most specific tasks and after analysis of the data the best visualization is chosen.

3
Related Work

There is a lot of related work, both in the field of data integration and in the field of data visualization. There are also many web-based applications covering one of these fields, but the novelty of this application lays in the combination of the two features, which is rather seldom. If one adds technologies of the semantic web, the number of applications gets even smaller. But the chapter covers all the fields and shows many applications which influenced *IfiPipes*.

3.1 Data Aggregation and Data Integration

In the field data aggregation and data integration there are many tools. The focus is on tools which are influential on the *IfiPipes* application or are outstanding in research or practice.

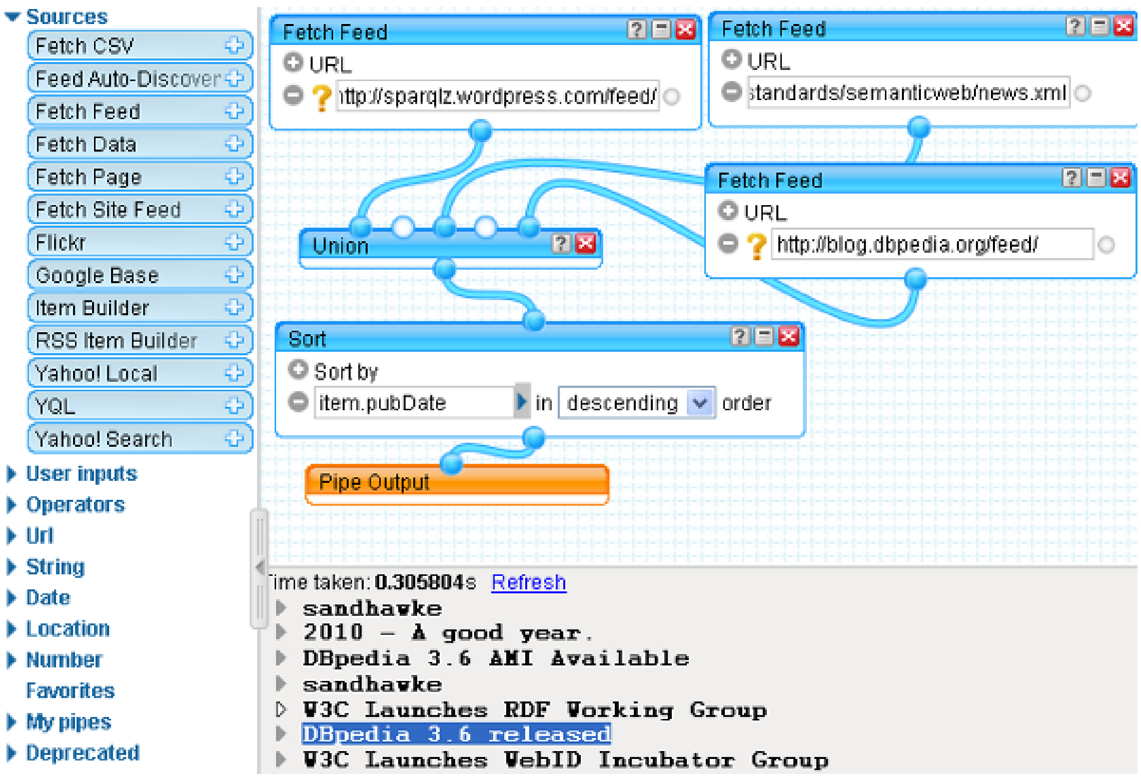
3.1.1 Enterprise Data Integration

Data integration tools are mostly used by enterprises. There exist several concepts of data integration. Enterprise information integration (EII) is the process of providing an abstraction layer to the company-owned data sources. This integration layer results in a single interface for all data in an organization. 'Extract, transform, load' (ETL) on the other hand results often in a data warehouse, where the data is extracted from the single databases, then transformed if needed and at last loaded into the new target. The advantage of ETL is a much better performance, as the queries don't have to query several sources. The advantages of EII is to have an access to 'live' data and that no data warehouse is needed. Summarized the benefit of ETL is speed, while the benefit of EII is cost. If EII could solve the problem of analyzing queries better, to query only the needed data sources, it'll result in a huge performance boost. But even if the problem at query start is optimally solved, it doesn't mean it is optimal at run-time. In [Halevy et al., 2005] the authors point out the two approaches are fundamentally different, thus they can co-exist. So it is in the hand of the company to decide which approach suits its business better. An unsolved problem which exists in both approaches is the mapping of semantics in the databases. By using machine learning and data algorithms one can at least semi-automate the process[1].

[1]A list of data and entreprise integration research projects can be found on `http://www.ifi.uzh.ch/~pziegler/IntegrationProjects.html`

3.1.2 Yahoo! Pipes

Yahoo Pipes[2] is a traditional web application created by Yahoo! in 2007. Yahoo Pipes allows the aggregation and manipulation of web feeds, internet sites and other sorts of contents on the web. One of the biggest advantages of Yahoo Pipes is its graphical interface, which allows graphical programming of these data mash-ups, as it can be seen in Figure 3.1.



URL: ıttp://sparqlz.wordpress.com/feed/
URL: standards/semanticweb/news.xml
URL: http://blog.dbpedia.org/feed/
Sort by item.pubDate in descending order

```
ime taken: 0.305804s
sandhawke
2010 – A good year.
DBpedia 3.6 AMI Available
sandhawke
W3C Launches RDF Working Group
DBpedia 3.6 released
W3C Launches WebID Incubator Group
```

Figure 3.1: A screenshot of a Pipe fetching and combining three feeds in Yahoo! Pipes[3]

The user interface of Yahoo! Pipes consists of three parts. On the left side there is the Library pane, which is a list of all possible Modules [Inc, 2011; Trevor, 2008]. The Library is parted in several groups, where the Modules are grouped by functionality. The different groups have different Modules, the most important ones are listed below[4]:

- Sources: Sources are Modules which encapsulate a data source (i.e. an RSS feed).
- Operators: Operators enable data manipulation functions like sort or filter and aggregation functions like count.
- User Inputs: User Input Modules act as input fields for the users of a Pipe and can be filled before running the Pipe.

[2] `http://pipes.yahoo.com/pipes/`
[4] The full list of Modules can be found on `http://pipes.yahoo.com/pipes/docs?doc=modules`.

- Data Types: There are several groups for different data types, like String, Data or Url.

The Canvas is the main area of the Pipe in the center of the application. By dragging an entry from the Library to the Canvas it creates a Module. It also acts as the workspace to assemble Pipes and allows the dragging of Modules on the whole workspace. After the creation of Pipes the most important thing is the drawing of connections between the Modules. The wiring of the connections is the heart of the Pipes application, by connecting Modules one creates a data mash-up. Each output of a Module is also the input of another Module, so the data streams are passed from Module to Module. It is even possible to use a previous create Pipe which acts as a normal Module and connect this Pipe to other Modules.

The third part of the application is the Debugger pane in the bottom of the page and shows the output, depending on the selected Module (shown in orange in Figure 3.1. To finish a Pipe one has to connect the final Module to the Output Module. Not shown is the Menu, which allows the user to save a Pipe, change its name or create a new Pipe.

The concept behind Yahoo! Pipes resembles the Unix pipes and filters architecture [Baker, 1987]. Pipes had some success in mash-up development, especially because of its graphical programming paradigm, which enabled user with little programming abilities to create data mash-ups. Users who want to use Pipes without a graphical interface can use YQL[5], a SQL-like query language for the web.

Missing features of Yahoo! Pipes are the fact, that they don't allow to pipe semantic data and the lack of an integrated visualization, to display the selected data. The output can be fetched as RSS or JSON and then visualized by the developer, but an automatic solution to visualize the outcome is missing.

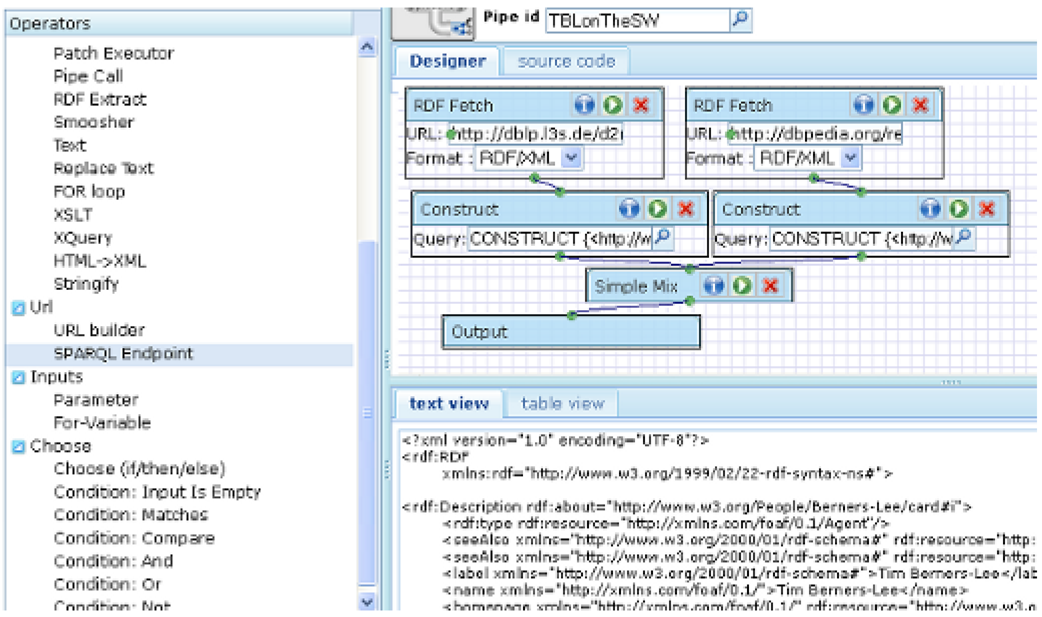
3.1.3 DERI Pipes

DERI Pipes[6] is the answer of the Digital Enterprise Research Institute (DERI) to Yahoo! Pipes. DERI is a research institute with focus on the Semantic Web, which consits of three groups localized in Europe, Asia and the USA. With DERI Pipes one can create data mash-ups using semantic technologies as RDF and SPARQL. The user interface, as shown in Figure 3.2 is very similar to the UI of Yahoo! Pipes.

The left pane consists as of the Operators (analogue to the Modules in Yahoo! Pipes). Many of the Operators have the same or at least a similar function[8]. But instead of fetching RSS feeds or HTML data, Operators can fetch complex queries with the help of SPARQL or even whole data sources by getting RDF. As the power of the Semantic Web to combine and interlink data sources is higher, the value of the semantic Pipes is higher [Le-Phuoc et al., 2009]. The application allows the user to get the output of a Pipe as RDF data or as raw data (depending on the type of output). It also provides a webpage with the results displayed in a faceted browser style, by using Exhibit[9] to display the data.

DERI Pipes is an open-source project written in Java and is using the web framework ZK[10]. The problem with further development of DERI Pipes is that it isn't actively developed at the

[5]`http://developer.yahoo.com/yql/`
[6]`http://pipes.deri.org/`
[8]A list of all Operators can be found on `http://pipes.deri.org:8080/pipes/doc/operators.html`
[9]`http://www.simile-widgets.org/exhibit/`
[10]`http://www.zkoss.org/`



URL: http://dblp.l3s.de/d2
Format : RDF/XML
URL: http://dbpedia.org/re
Format : RDF/XML
Query: CONSTRUCT {<http://w
Query: CONSTRUCT {<http://w

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

<rdf:Description rdf:about="http://www.w3.org/People/Berners-Lee/card#i">
    <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Agent"/>
    <seeAlso xmlns="http://www.w3.org/2000/01/rdf-schema#" rdf:resource="http:
    <seeAlso xmlns="http://www.w3.org/2000/01/rdf-schema#" rdf:resource="http:
    <label xmlns="http://www.w3.org/2000/01/rdf-schema#">Tim Berners-Lee</lat
    <name xmlns="http://xmlns.com/foaf/0.1/">Tim Berners-Lee</name>
    <homepage xmlns="http://xmlns.com/foaf/0.1/" rdf:resource="http://www.w3.o
```

Figure 3.2: A screenshot of a pipe fetching and combining two data sources about Tim Berners-Lee in Deri Pipes[7]

time. While the mailing-list is still active, there isn't much activity ongoing and the development stopped in 2009. Another problem is the Draw2D-port in ZK, which isn't documented very well and the development of it has stalled as well. All the problems result in the fact, that DERI Pipes lacks usability in modern browsers.

3.1.4 Google Fusion Tables

Google Fusion Tables[11] is the product of Alon Halevy, a well-known researcher in the field of data integration. Fusion Table allows the user to upload tabular data from CSV or Excel-files to the web application. The uploaded data can then be merged, aggregated and filtered. One can also merge data with data from the web or with another table. Fusion Table further provides visualizations of data sources by using the Google Visualization API. Visualizations include table visualization, visualizations on Google Maps or visualization in a timeline, depending on what data is available. Fusion Tables lacks the functionality using data sources stored in semantic formats and also doesn't allow to interlink to existing data.

A solution for using semantic data in Fusion Tables would be to use another tool created by Google, Refine[12]. Refine is a desktop application that allows the import of various data sources and the clean up, transformation and extension of the imported data. By exporting the data as CSV one can import it into Fusion Tables. Refine has some semantic capabilities, as it's the successor of a product created by Freebase, a company working in the semantic space, which was acquired by Google. So the data can be connected with data from Freebase[13], which is now a large

[11]http://www.google.com/fusiontables/Home
[12]http://code.google.com/p/google-refine/
[13]http://www.freebase.com/

graph of things on the Semantic Web. One can even export data as RDF, with an extension[14].

3.2 Data Visualization

Data visualization software is often used in the fields of statistics and medical sciences, as they deal with large numbers of data entries. In recent time enterprises started to use information visualization to display large data sets. Semantic data visualization is still in it's infancy, but there are a few tools and applications. Another trend is to create visualizations, which can be changed and then shared or that visualization can be even 'forked' [Gorman, 2010]. As the data is freely available others can take the same data to create different visualizations.

3.2.1 IBM Many Eyes

Many Eyes[15] is a web site created by IBM, where people can upload data sets and choose visualizations for their data. The visualizations have to be chosen by the user, there is no data analysis. All data sets and all visualization are public and other people can play with the visualization or even create new visualization based on existing data sets. Users can also comment on both data sets and visualizations and share them with others. Many Eyes supports a number of visualizations, from statistical visualizations and charts, to tag clouds and maps. There are also a few faceted browsing options to filter data from the visualization and a text search to highlight single items.

3.2.2 Microsoft Pivot

Pivot is a software application from Microsoft Live Labs for visualizing massive amounts of data. As the research group was discontinued, the desktop application was removed from the homepage, but the web viewer called **PivotViewer**[16] is still online, as it is one of the show cases of Microsofts programming language Silverlight. Pivot uses a faceted browsing approach to filter data categories and uses a AJAX-enabled search to find specific data entries. Pivot allows the user to arrange the data in a tile view or in columns, where similar items are grouped together. The application uses high-resolution images and with the help of Microsofts Deep Zoom, the application can be zoomed very smoothly. Pivot's user interface can be seen in Figure 3.3.

Pivot can't visualize semantic data directly, the data has to be transformed to special XML templates. The Virtuoso Quad Store, a semantic data store developed by OpenLink Software, has a built-in visualization engine based on faceted browsing and Pivot[18].

[14]The homepage of the Google Refine RDF extension cat be found at: `http://lab.linkeddata.deri.ie/2010/grefine-rdf-extension/`

[15]`http://www-958.ibm.com/software/data/cognos/manyeyes/`

[16]`http://www.microsoft.com/silverlight/pivotviewer/`

[18]Kingsley Idehen, a proponent of the Semantic Web and CEO of OpenLink Software, creates many visualizations using semantic with Pivot. A list of collections can be found in his delicios account: `http://www.delicious.com/kidehen/pivot_collection_app`

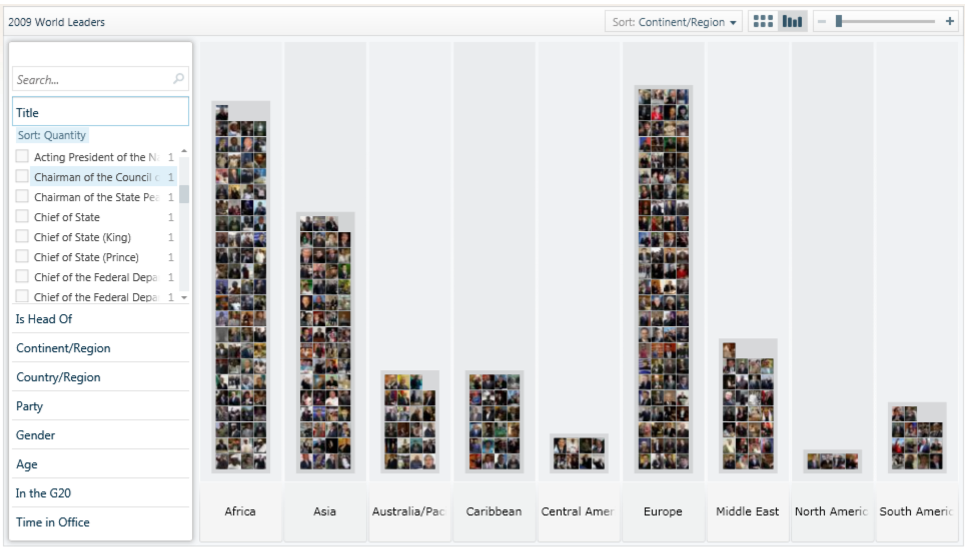


Figure 3.3: The user interface of Microsoft PivotViewer showing the leaders of the world, categorized by continent/region.[17]

3.2.3 Google Public Data Explorer

Google Public Data Explorer[19] is the newest web application for information visualization. The Public Data Explorer has a few predefined data sets and provides four visualizations for each set. These visualizations are Line chart, Bar chart, Map chart and Bubble chart. The application uses features from Trendalyzer, an application created by well-known statistician Hans Rosling[20]. The Public Data Explorer allows the user to upload his/her own data set, which can be previewed, published and shared. To upload data, a user has to describe their data sets using the Dataset Publishing Language (DSPL), which is an open, XML-based format. Although the format is open, one still has to learn a new language in order to use their own data sets. The user interface of the application feels really great. One can go back and forth in time and even press a play button to automatically advance in time. Also the filtering is very expressive and many visual clues are provided to find the right data. The user interface of the Public Data Explorer is depicted in Figure

3.2.4 Graph Visualization Tools

prefuse[21] is a visualization toolkit for creating rich interactive data visualizations [Heer et al., 2005]. prefuse is open-source and implemented in Java. There exists also a Flash implementation, called prefuse flare. While prefuse is mainly a graph visualization toolkit, it can also visualize non-graph data.

Jung[22] is also a visualization library, implemented in Java and is open-sourced. Jung contains

[19] http://www.google.com/publicdata/home
[20] For an overview of Hans Rosling's greatest talks the reader is referred to http://www.economist.com/node/21013330
[21] http://prefuse.org/
[22] http://jung.sourceforge.net/

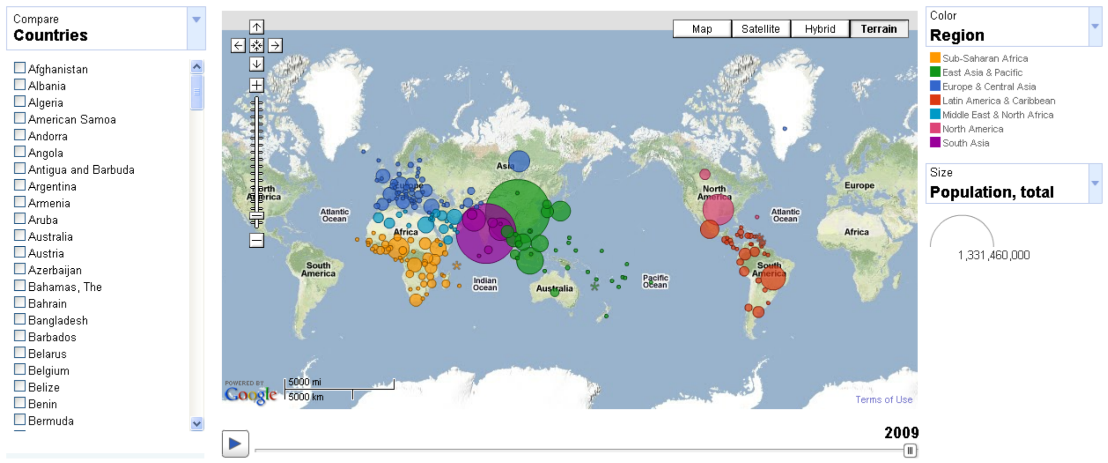


Figure 3.4: The user interface of Google Public Data Explorer showing the total population of the countries of the world.

various implementations of graph and network algorithms. Jung also allows the user to interact with the visualizations to change the layout or to move nodes. Also the framework enables filtering, which declutters the visualization.

Processing[23] is a programming language and environment and can be used to create interactive and static images and animations. Processing runs in the Java Virtual Machine and can be extended with Java. There is a sister project which implements Processing in Javascript, called Processing.js. Processing was developed as a teaching language within a visual context, but it developed into a whole toolkit.

Tulip[24] is a data and graph visualization framework implemented in C++. Tulip is very strong for large data sets and is able to visualize graphs having up tp 1'000'000 elements [Auber, 2003]. Tulip allows the user to interact with the visualization and follows Ben Shneiderman's mantra: 'Overview first, zoom and filters, then details on demand'[25]. Tulip is mainly used in bioinformatics and other research fields which work with large data sets.

3.2.5 Fresnel

Fresnel[26] is a vocabulary for specifying how RDF graphs are presented[27]. Fresnel wants to stop the practice that developers have to determine for each application and tool which information is shown and how it is presented. This is done by using concepts of lenses and formats. Lenses determine which properties of an RDF graph or resource are shown and how these properties are ordered. Formats define how resources and properties are displayed and provides relationships to styling languages like CSS [Bizer et al., 2005]. Fresnel allows the selection of resources with SPARQL and its own selector language, FSL.

[23] `http://www.processing.org/`
[24] `http://tulip.labri.fr/TulipDrupal/`
[25] See Subsection 2.3.2 for more information about Shneiderman's guidelines for information visualization.
[26] `http://www.w3.org/2005/04/fresnel-info`
[27] The specification for Fresnel can be found on: `http://www.w3.org/2005/04/fresnel-info/manual/`

While Fresnel is a good approach to display data in tabular or textual form, Fresnel's use for more sophisticated visualizations is currently small. The problem is, that most visualizations rely on more then pure CSS. The problem could be alleviated when every browser supports vector graphics like SVG or more powerful style languages. A first step in this direction is done with CSS3, which allows animations without using JavaScript. Fresnel is used in a few applications, but most of them seem to be discontinued or not working. Also there is little activity in the Fresnel working community, their mailing list[28] is updated scarcely. One member proposes a new format, called SPARQL Web Pages[29], which allows user to use HTML snippets that contain SPARQL queries, which is like a stripped variant of Fresnel.

3.2.6 Semantic Data Visualization Tools

One of the earliest graph visualization tools for the Semantic Web is **RDF Gravity** [30]. RDF Gravity enables the import of RDF data or OWL ontologies and visualizes them as a graph. One can filter data by applying global and local filters and the application includes a full-text search. For more filtering the application allows to create SPARQL queries for additional filtering. RDF Gravity is implemented as a Java web application and created using the JUNG framework for the visualization and the Jena framework as semantic API. A screenshot of the interface is shown in Figure 3.5.

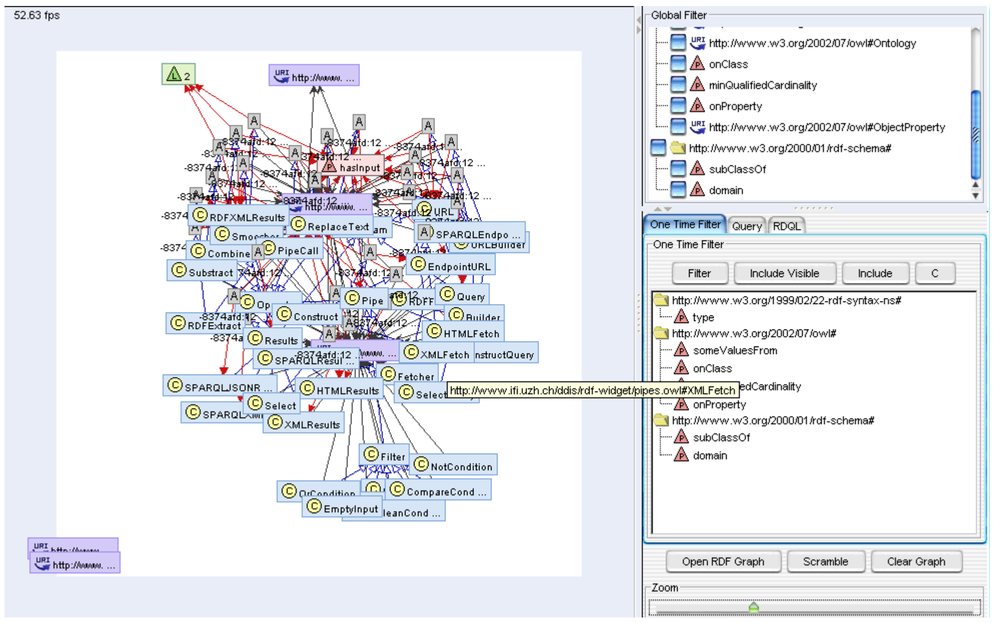


Figure 3.5: The user interface of RDF Gravity. The loaded ontology is the Pipes ontology of the thesis.

One can see that the visualization doesn't provide much clarity. By applying filters the visualization gets better, but the problem of displaying RDF data as a graph is clearly visible.

[28] `https://sympa-roc.inria.fr/wws/arc/fresnel`
[29] `http://uispin.org/`
[30] `http://semweb.salzburgresearch.at/apps/rdf-gravity/index.html`

mSpace[31] uses a totally different approach, they use semantic data and display it in a iTunes-like column UI, to explore and select data. Depending on the selection in the columns the content changes. This is a more website like approach, which makes the selection much easier. Optional there is also a full-text search in place.

Exhibit[32] is a publishing framework for web pages. Exhibit is implemented in JavaScript and enables developers to create views of their data. The data is loaded in templates and visualized in a faceted browsing manner. Views include maps or tabular visualizations and are defined by special data templates. While Simile doesn't allow the import of RDF data, there exists a web service called **Babel**[33], which can convert the data from various formats, including RDF/XML to Exhibit JSON. Exhibit is used by Deri Pipes to visualize the queried data in a faceted web page.

A similar tool is **Longwell**, which visualizes RDF data in a faceted browser. Unlike Exhibit, Longwell isn't a web application, but a desktop application written in Java. All three tools, Exhibit, Babel and Longwell are created by teams from the Simile research project[34], which is a joint project run by the W3C and the computer science group of the MIT. Simile focuses strongly on semantic technologies and is one of the leading Semantic Web research groups.

There exist several semantic web browser, which allow the traversing of certain data set. **ThinkBase**[35] for example allows to browse through the Freebase data set, while **ThinkPedia**[36] allows a user to browse through Wikipedia articles [Hirsch et al., 2009]. **DBpedia**[37], the semantic clone of Wikipedia, has also a faceted browsing visualization, where it allows the filtering of results.

Tabulator[38] on the other hand is a generic data browser for the Semantic Web [Berners-Lee et al., 2006]. Tabulator is both a web application and a browser add-on. Tabulator uses faceted browsing to display data about Semantic Web data sources. If data is annotated as temporal or geographical data, it can be displayed on a timeline respectively on a map. While Tabulator is an interesting proof of concept the user interface is bulky and the querying is pretty slow.

[31]http://www.mspace.fm/
[32]http://simile-widgets.org/exhibit/
[33]http://service.simile-widgets.org/babel/
[34]http://simile.mit.edu/
[35]http://thinkbase.cs.auckland.ac.nz
[36]http://thinkpedia.cs.auckland.ac.nz/
[37]http://dbpedia.neofonie.de/browse/
[38]http://www.w3.org/2005/ajar/tab

4

IfiPipes

This chapter explains the application of the learned theory in previous chapters. In includes an overview of the software used in the application, presents roughly the architecture and points at details of the implementation. Furthermore the user interface is shown, both as sketch and as application. The chapter concludes with a user guide and a few selected use cases.

4.1 Used Software

IfiPipes, the application resulting from this thesis uses various free and open-source software. It is written in Java Version 6 and the author used Eclipse Helios with the GWT Plug-In as development environment. For the ontology development the ontology editor Protégé was used. The application is designed for the Mozilla Firefox browser and runs with a few quirks also in Google Chrome. The application is deployed on a server running an Apache Tomcat instance. More software is listed in the next few subsections below.

4.1.1 Google Web Toolkit

Google Web Toolkit[1] (GWT) is an open-source development toolkit created by Google and allows the development of complex web applications. Contrary to traditional web development, in which the client side is often done in HTML, CSS and extensive JavaScript, the toolkit enables the developer to write browser-based applications only in Java, without the need to write JavaScript. The Java source code is compiled to JavaScript and is compatible with almost all browsers and browser versions. It also relieves the developer with the sometimes tedious work to write AJAX (Asynchronous JavaScript and XML) calls, by using its own optimised Remote Procedure Call (RPC) system, which every Java developer already knows. Furthermore GWT includes various Swing-like UI components and allows the user to create more widgets.

In opposition to similar Java toolkits like ZK[2] or Vaadin[3], GWT distinguishes between client-side and server-side programming. This results in faster client-side only operation, but leaves the

[1]`http://code.google.com/webtoolkit/`
[2]See `http://www.zkoss.org/` for more information. Deri Pipes (see Related Work) uses the ZK toolkit.
[3]`http://vaadin.com/home`. Vaadin uses GWT widgets to display webpages.

developer with more work. GWT allows only few selected libraries on the client-side and restricts the datatypes used in RPCs to mostly primitive types. On the server-side every programming language can be used, although it makes sense to use Java or at least a language running on the Java Virtual Machine (JVM). Google itself uses GWT in a few of it's own projects, in particular Adwords, Moderator and the now discontinued Wave. The version used in the application is GWT 2.1, as version 2.2 was released in the last days of writing. The new version brings support for HTML 5 functionality like canvas, audio or video, which provide currently no benefit for the application and even introduces a bug when Google Maps for GWT is uses[4].

There are several other frameworks or libraries extending GWT and adding features not provided by the basic version. This project made use of following extensions or libraries:

- **SmartGWT**[5] for more information on SmartGWT.: SmartGWT is a framework on top of GWT and includes a large set of widgets. It is geared towards larger, JavaScript-heavier applications, as it consists of a huge codebase and the initial loading time is therefore slow. SmartGWT is release under two licence; there is a free LGPL version and and a commercial professional edition, which includes more features on the server-side, like data binding or advanced security functions.
- **GWTUpload**[6]: GWTUpload is a small library, consisting of a UI widget and a servlet to upload files to the server. Like GWT it is written in Java, but compiles into JavaScript.
- **Google Maps for GWT**[7]: The Google Maps Library for GWT allows one to access the Maps API via Java instead of JavaScript.

4.1.2 Jena

Jena[8] is an open-source framework for developing Semantic Web applications written in Java. Jena originated from HP and is one of the biggest Semantic Web-related frameworks. It includes a triple store, an easy-to-use programming API for both RDF and OWL and a SPARQL query engine. It is also one of the very few frameworks which has not only a RDFS reasoner, but also an OWL reasoner is included.

The heart of the Jena API is the Model, which is a set of statements and another representation of the semantic graph. To fill a Model one can either add statements in Java or parse them from a file or the web. Jena supports the import of various RDF formats like RDF/XML, N3 or OWL.

4.1.3 tGraph

The tGraph framework created by Jonas Tappolet is based on Jena and NG4J - Named Graphs API for Jena and enables the querying of temporal semantic data. Data can either by retrieved by

[4]As mentioned in a GWT forum thread: `http://groups.google.com/group/google-web-toolkit/browse_thread/thread/65f1184b6e5f4f95`

[5]See `http://code.google.com/p/smartgwt/` respectively `http://www.smartclient.com/product/`

[6]`http://code.google.com/p/gwtupload/`

[7]`http://code.google.com/p/gwt-google-apis/`

[8]`http://openjena.org/` provides many resources about Jena and the Semantic Web, including an comprehensive tutorial on RDF and the Jena API.

using Java or tSparql[9] queries. Like Jena the tGraph framework also provides both, an in-memory and a persistent storage, by using the Virtuoso Quad Store[10].

4.2 Ontologies

IfiPipes makes great use of the two ontologies created for the application. The goal was to describe both the operators to query data sources and the visualization corresponding to the queried data.

4.2.1 Pipe Ontology

The ontology describing the pipes, the main element of the application, is implemented in OWL. The structure of the ontology is as follows:

- `Pipe`: `Pipe` is the superclass of all concrete `Pipe`s. Each subclass of a `Pipe` must have a name represented as a String.
  - **`Builder`**: `Builder` are elements that can be used to create `URL`s.
  - **`Fetcher`**: a `Fetcher` queries data from a data source. If a `Fetcher` follows after a `Builder` the output from the `Builder` is used by the `Fetcher`.
  - **`Condition`**: a `Condition` is an element which allows the add conditions to `Pipe`s.
  - **`Filter`**: `Filter` can be used to manipulate the output of elements by combining elements or replacing text.
- **`Stream`**: a `Stream` is the output or the input of a `Pipe`. A `Stream` has a value which is used to serialize `Pipe`s.
  - **`Query`**: a `Query` is a representation of a Sparql Query.
  - **`Results`**: `Results` come in several types and result mostly of a fetch operation.
  - **`URL`**: a `URL` is mainly created by a `Builder`.

Each of the four subclasses of `Pipe`s has more subclasses which are mapped to Java classes in the source code. For example the OWL class `SPARQLEndpointBuilder` points to the Java class `SPARQLEndpointBuilder.java`. Like its siblings the class `SPARQLEndpointBuilder` has not only it's hierarchical parent as super class, but also various property restrictions, modelled in OWL. One of the restrictions is the name, modelled with a data property called `name`. The restriction is of type `owl:hasValue`. This is a workaround to the fact that OWL doesn't allow default values for properties [Quirolgico et al., 2004].[11] Each Pipe has also at least one `hasInput` restriction and exactly one `hasOutput` restriction. In the case of the `SPARQLEndpointBuilder` the input can either be a `Query` or a `URL`, so is has two `hasInput` restrictions, and the output is a special type of `URL`, a `EndpointURL`. One can see that the `Pipe`s are not regular, 'dumb' pipes according to the Pipes and Filter architecture [Baker, 1987; Meunier, 1995], which have the same output as

[9]tSparql is a temporal extension to SPARQL.
[10]`http://virtuoso.openlinksw.com/rdf-quad-store/`
[11]The workaround is described in detail on SemanticOverflow, a Q&A site for the Semantic Web: `http://www.semanticoverflow.com/questions/807/expressing-default-values-in-owl`.

the input and thus allowing all combinations of pipes. In contrast the application uses 'intelligent' pipes which restrict the combination of pipes by matching the output of the first pipe with the input of the next pipe. The OWL/XML representation of the `SPARQLEndpointBuilder`, which is one of the more complex classes, can be seen in Listing 4.1.

Listing 4.1: The OWL class SPARQLEndpointBuilder represented in OWL/XML

```
<owl:Classrdf:about="#SPARQLEndpointBuilder">
    <rdfs:subClassOf rdf:resource="#Builder"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasInput"/>
            <owl:someValuesFrom rdf:resource="#Query"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#name"/>
            <owl:hasValue>SPARQL Endpoint Builder</owl:hasValue>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasOutput"/>
            <owl:someValuesFrom rdf:resource="#EndpointURL"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasInput"/>
            <owl:someValuesFrom rdf:resource="#URL"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

4.2.2 Visualization Ontology

The ontology describing the visualizations is implemented in OWL as well. The ontology is split in two parts, where one part describes the data source, respectively the results from a query and the other part defines the visualization. The ontology imports the Geo Vocabulary[12] from the W3C, which is used to describe the visualizations. The part of the ontology which is used to describe the queried data source can be seen in following list:

[12] `http://www.w3.org/2003/01/geo/`

- **`QueryResult`**: super class of all query results.
  - **`RDF`**: represents query results in the RDF format.
    - **`GeoRDF`**: RDF with annotated geographical data. Has `geo:lat` and `geo:long` as value for its `keywords` property.
    - **`TemporalRDF`**: RDF with annotated temporal data. Has the String 'isQuad' as value for its `keywords` property.
  - **`SparqlResults`**: represent query results in the SPARQL results format
    - **`GeoSparqlResults`**: results from a SPARQL query with annotated geographical data. Has the comma separated String 'lat,long,geo,latitude,longitude' as value for its `keywords` property.
    - **`TemporalSparqlResults`**: results from a SPARQL query with annotated temporal data. Has the comma separated String 'time,from,to,until,now' as value for its `keywords` property.

The hack to use only one data property to store multiple values is ugly and should be changed in the future. This part of the ontology is loaded as soon as the user runs a Pipe and the query results are analyzed. The part of the ontology which describes the visualization is displayed in the following list:

- **`Visualization`**: super class of all visualizations
  - **`MapsViz`**: used for displaying data on a map. `MapsViz` needs `GeoProperties` to get selected.
  - **`PlainTextViz`**: used as a fallback visualization or if a user wants to see raw JSON / XML code.
  - **`TableViz`**: used to visualize unspecific data in tabular form. Also a fallback visualization as all semantic data can be presented in tables.
  - **`TempViz`**: used to display temporal controls. `TempViz` needs `TempProperties` to get selected.
- **`Property`**: super class for all properties.
  - **`GeoProperty`**: property with annotated geographical data.
  - **`TempProperty`**: property with annotated temporal data.

This part of the ontologies describes the visualizations. Although the description might be valuable for either the user or the program, the application makes currently no use of it. It is suggested to implement the part of the ontology further, maybe even create a new ontology for it.

4.3 Architecture

IfiPipes is built as a classical client-server application. The client-side of the application is written in Java and with the help of GWT compiled in JavaScript. It manages mostly the user interface of the application, but also sends requests to query data sources and matches Operators. The main class of the application is `IfiPipes` which implements the `EntryPoint` interface and the method `onModuleLoad()`. In this specific case the application calls `draw()` on the class `ApplicationWindow`, which contains all graphical modules of the application and is implemented as a a Singleton. By looking at the class one can see the architecture of *IfiPipes* has some weaknesses, as the class acts both as controller and as view. The same flaw can be found in other classes too, for example the class `Menu` contains business logic as well. A UML class diagram of the application is shown in Figure 4.1.

The server-side is written in Java as well and the resulting WAR-file after compilation can be loaded in an Apache Tomcat instance. Communication between the server and the client works via remote procedure calls (RPC), these are translated in AJAX calls during the compilation. A GWT RPC consists of three components as seen in the example below:

- An interface (`DataFetcherService`) which extends `RemoteService` and declares the method signature of all RPC methods.
- An asynchronous interface (`DataFetcherServiceAsync`) declare to the service that will be called from the client-side.
- An implementation of the `DataFetcherService` (`DataFetcherServiceImpl`) that extends `RemoteServiceServlet`.

Furthermore one has to include the servlets in the web application deployment descriptor (web.xml in the war/WEB-INF/ folder) and add `<servlet>` and `<servlet-mapping>` elements. The servlet configuration can be seen in listing 4.2.

Listing 4.2: Snippet showing the servlet configuration for the DataFetcherService from web.xml

```
<servlet>
  <servlet-name>dataFetcher</servlet-name>
  <servlet-class>
    ch.uzh.ifi.ddis.pipes.server.DataFetcherServiceImpl
  </servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>dataFetcher</servlet-name>
  <url-pattern>/ifipipes/dataFetcher</url-pattern>
</servlet-mapping>
```

In addition to the three classes and the servlet configuration, one has to annotate the service with `@RemoteServiceRelativePath("dataFetcher")`, else the servlet container Jetty throws an exception when running.

4.4 User Interface Design

The user interface of *IfiPipes* is sketched in Figure 4.2. The idea follows the general user interface standard for western culture to have the controls on the left side and the menu on top.

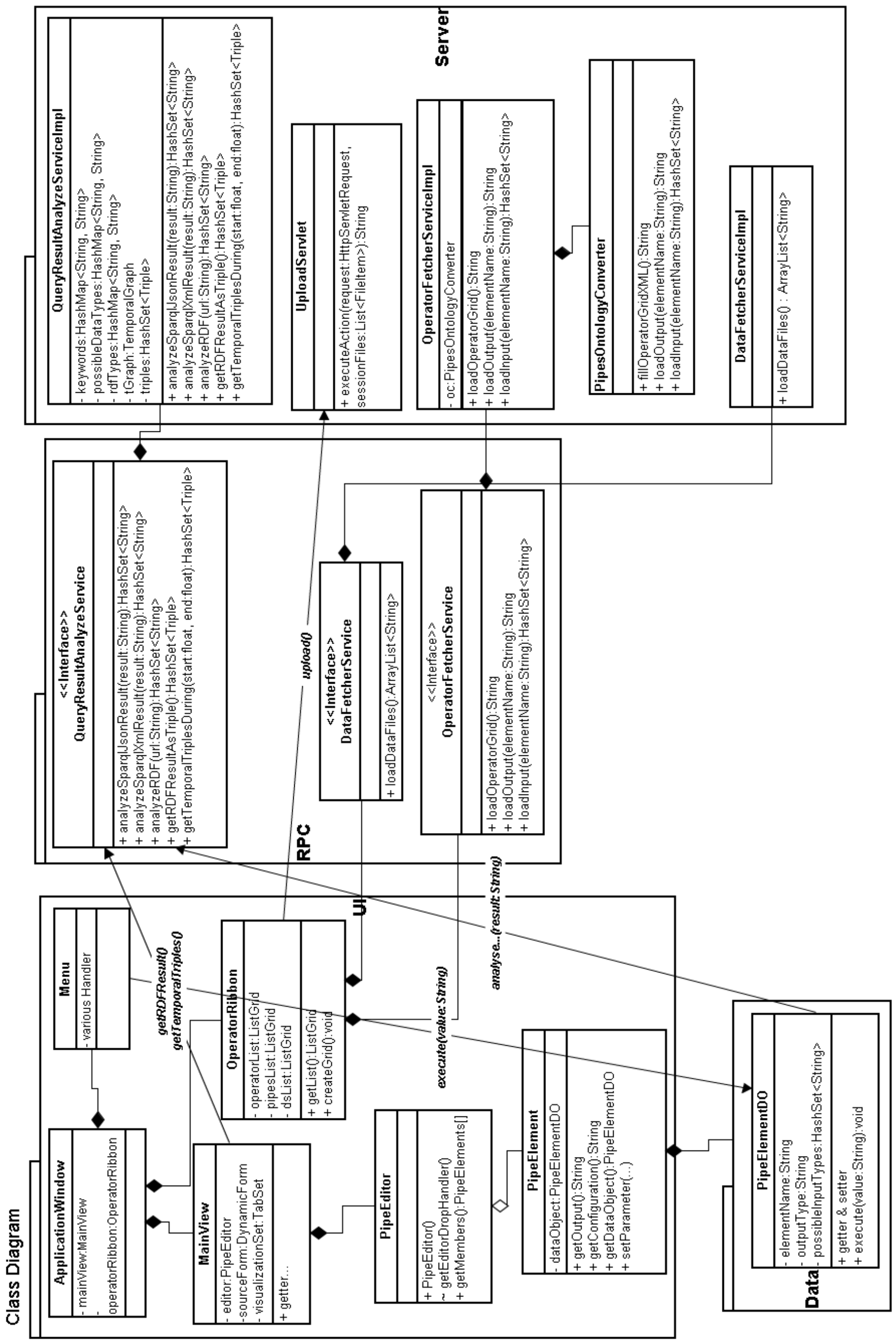

Class Diagram
UI
ApplicationWindow
- mainView:MainView
- operatorRibbon:OperatorRibbon
Menu
- various Handler
MainView
- editor:PipeEditor
-sourceForm:DynamicForm
- visualizationSet:TabSet
+ getter...
OperatorRibbon
- operatorList:ListGrid
- pipesList:ListGrid
- dsList:ListGrid
+ getList():ListGrid
+ createGrid():void
PipeEditor
+ PipeEditor()
~ getEditorDropHandler()
+ getMembers():PipeElements[]
PipeElement
- dataObject:PipeElementDO
+ getOutput():String
+ getConfiguration():String
+ getDataObject():PipeElementDO
+ setParameter(...)
Data
PipeElementDO
- elementName:String
- outputType:String
- possibleInputTypes:HashSet<String>
+ getter & setter
+ execute(value:String):void
getRDFResult()
getTemporalTriples()
execute(value:String)
analyse...(result:String)
upload()
RPC
<<Interface>>
QueryResultAnalyzeService
+ analyzeSparqlJsonResult(result:String):HashSet<String>
+ analyzeSparqlXmlResult(result:String):HashSet<String>
+ analyzeRDF(url:String):HashSet<String>
+ getRDFResultAsTriple():HashSet<Triple>
+ getTemporalTriplesDuring(start:float, end:float):HashSet<Triple>
<<Interface>>
DataFetcherService
+ loadDataFiles():ArrayList<String>
<<Interface>>
OperatorFetcherService
+ loadOperatorGrid():String
+ loadOutput(elementName:String):String
+ loadInput(elementName:String):HashSet<String>
Server
QueryResultAnalyzeServiceImpl
- keywords:HashMap<String, String>
- possibleDataTypes:HashMap<String, String>
- rdfTypes:HashMap<String, String>
- tGraph:TemporalGraph
- triples:HashSet<Triple>
+ analyzeSparqlJsonResult(result:String):HashSet<String>
+ analyzeSparqlXmlResult(result:String):HashSet<String>
+ analyzeRDF(url:String):HashSet<String>
+ getRDFResultAsTriple():HashSet<Triple>
+ getTemporalTriplesDuring(start:float, end:float):HashSet<Triple>
UploadServlet
+ executeAction(request:HttpServletRequest, sessionFiles:List<FileItem>):String
OperatorFetcherServiceImpl
- oc:PipesOntologyConverter
+ loadOperatorGrid():String
+ loadOutput(elementName:String):String
+ loadInput(elementName:String):HashSet<String>
PipesOntologyConverter
+ fillOperatorGridXML():String
+ loadOutput(elementName:String):String
+ loadInput(elementName:String):HashSet<String>
DataFetcherServiceImpl
+ loadDataFiles() : ArrayList<String>

Figure 4.1: A stripped UML class diagram of IfiPipes[13]

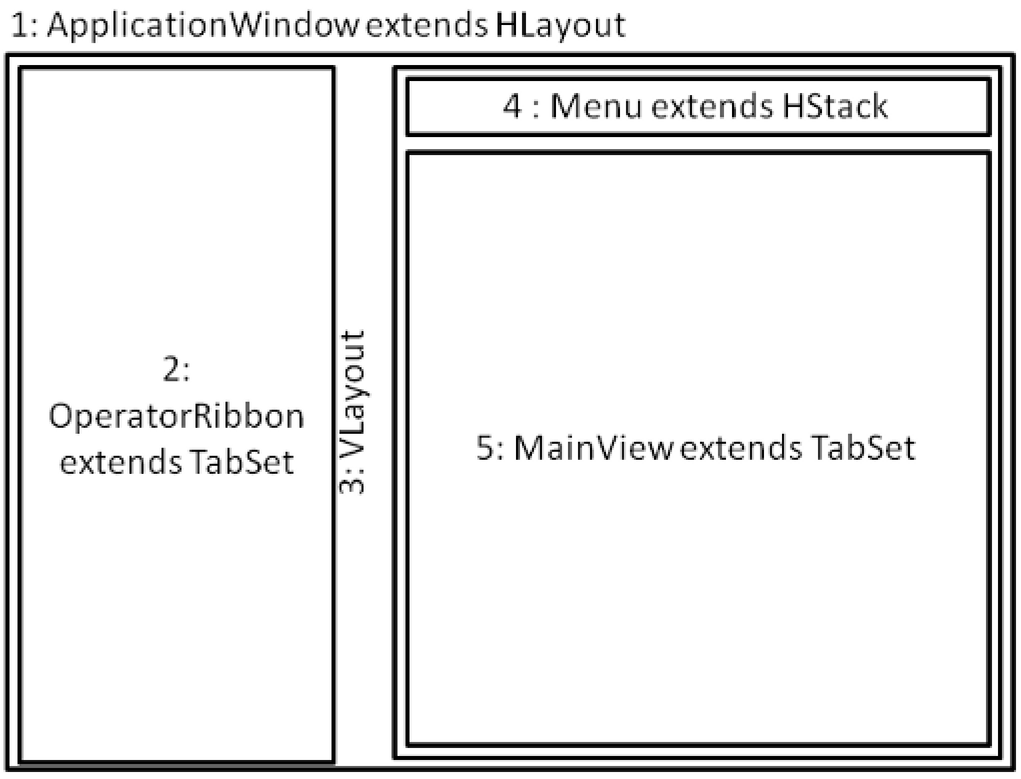

1: ApplicationWindow extends HLayout
2: OperatorRibbon extends TabSet
3: VLayout
4 : Menu extends HStack
5: MainView extends TabSet

Figure 4.2: The sketched user interface of *IfiPipes*

One can see that the `ApplicationWindow` (1) class contains all other graphical modules of the application. It extends the class `HLayout`, a container from the SmartGWT framework. The `ApplicationWindow` has two direct subelements, one is the `OperatorRibbon` (2) and the other is a simple `VLayout` (3). The `OperatorRibbon` extends the class `TabSet` and consists of three `Tab`. The three `Tabs` are called 'Operators', 'Pipes' and 'Data' and all `Tabs` contain a `ListGrid` that displays the contents of each tab in a tabular form. The `VLayout` contains two other panes, on the top there is the `Menu` (4), which extends the `HStack` class and the middle and bottom part are occupied by the `MainView` (5) which extends `TabSet` as well. The `Menu` consists of three `Buttons`: 'New Pipe', 'Save Pipe' and 'Run Pipe'. The `MainView` contains three `Tabs`: The `PipeEditor`, the source pane and the visualization `TabSet`. The implemented user interface can be seen in Figure 4.3.

4.5 Implementation Details

This section shows a few details of the implementation, which are essential for the understanding of the application. For more information the reader is referred to the source code, which accompanies the thesis.

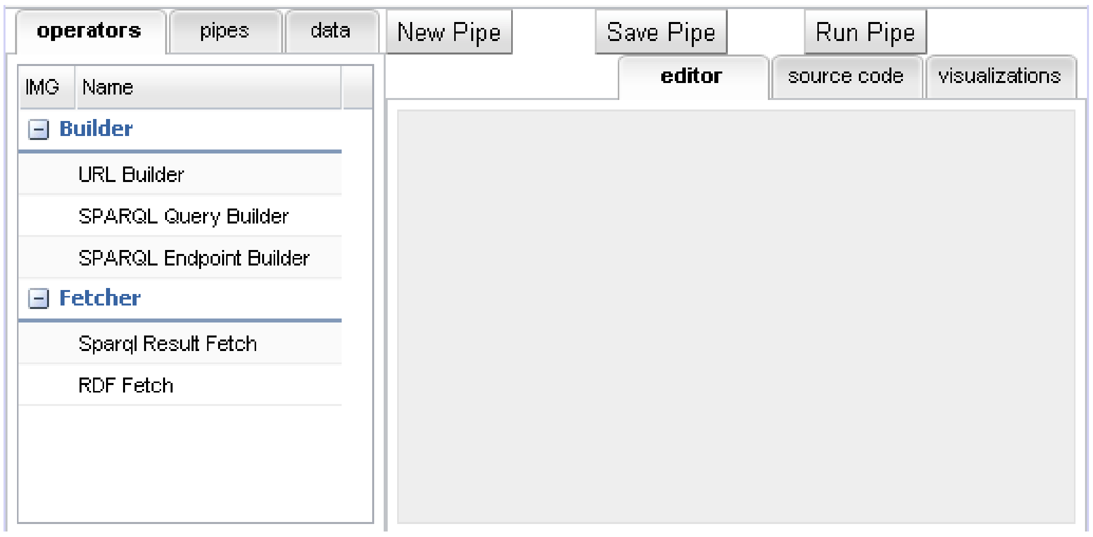


Figure 4.3: The user interface of *IfiPipes*.

4.5.1 Loading the Ontologies

The Pipes ontology is loaded asynchronously with an AJAX call when the application is started. On slower computer it is visible, that the Operator ribbon isn't filled from beginning, but the grid is populated when the ontology is finally parsed and written in an XML document, which is the data source for the grid. This XML is overwritten each time when the user starts the application, to automatically push changes in the ontology. The parsing of the ontology makes use of Jena's frame API[14]. At first the names of the Operators need to be loaded from the ontology, these are stored as value restrictions, as explained in Subsection 4.2.1. First the ontology is loaded in a Jena `OntModel`, which extends the basic graph with a few OWL-specific methods. Second the `OntProperty` name is got from the graph and all of its restrictions are stored in a iterator. Third the iterator is used to iterate over all restrictions and if there are `owl:hasValue` restrictions they are written to XML. A Java code snippet for this procedure can be found below:

Listing 4.3: Snippet from the fillOperatorGridXML method in the OntologyConverter class

```java
Iterator<Restriction> i = nameProp.listReferringRestrictions();
while (i.hasNext()) {
  Restriction r = i.next();
  if (r.isHasValueRestriction()) {
    name = r.asHasValueRestriction().getHasValue()
      .asLiteral().getLexicalForm();
    ...
  }
}
```

[14]See `http://jena.sourceforge.net/how-to/rdf-frames.html` for a tutorial on RDF frames and the Jena frame API.

The loading of the XML into the Operator ribbon is fairly straightforward and uses SmartGWT's DataSource implementation, which automatically loads the XML and creates the desired fields. The possible inputs and the possible output of each Operator aren't loaded at start, but rather when the user drops an Operator on the Editor, they are loaded on demand. Again an AJAX call is used, which parses the ontology on the server and returns a the results to the client. The possible inputs and the output are also stored as restriction, this time as a `owl:someValuesFrom` restriction. The extraction of these restriction works in a similar fashion, but it slightly different, as the following code snippet shows:

Listing 4.4: Snippet from the loadOutput method in the OntologyConverter class

```
Iterator<OntClass> superClasses =
  model.getOntClass(PIPES_NS + elementName).listSuperClasses(true);
while (superClasses.hasNext()){
  final OntClass cl = superClasses.next();
  if (cl.isRestriction()) {
    if (cl.asRestriction().getOnProperty().equals(outputProp)
      && cl.asRestriction().isSomeValuesFromRestriction()) {
      return cl.asRestriction().asSomeValuesFromRestriction()
        .getSomeValuesFrom().getLocalName();
    }
  }
}
```

The Visualization ontology isn't loaded at start time, but rather when the user runs a Pipe for the first time. The

4.5.2 User Interface Implementation

The `PipeEditor`, which contains the Pipes created by the user, extends the SmartGWT container `HStack`. When a user drops an Operator from the grid onto the Editor, an instance of the class `PipeElement` is drawn and is added to the As all Operators share some visual features, like the header with the name of the element and or the remove button[15], these are implemented in the super class. The class provides a few other methods, which throw an exception in the base class and thus have to be implemented by its children. Two methods are used to get either the configuration to serialize a element or the output of an Operator to pass it to the next one, a third is used to set the fields of a `PipeElement`, if the output and the possible input of two adjacent Operators match.

4.5.3 Operator Matching

The matching code of the Operators can be found in the `Menu` and is done by String matching. In Subsection 4.5.1 the loading and parsing of the ontologies is explained. The parsed output and inputs are stored in the `PipeElementDO`, the data object of each Operator. The actual matching

[15]The header and the remove button should be on the same row. Although the implementation is correct, and I filed a bug report (`http://forums.smartclient.com/showthread.php?t=14645`), the outcome is not the expected one.

takes places when the user runs a Pipe. It is done by String matching and a snippet can be found in Listing 4.5.

Listing 4.5: Snippet of the onClick method from the Menu.RunPipeClickHandler class

```
if (!checkIfPossiblePipe()) {
  return;
}
PipeElement[] members = ApplicationWindow.getInstance().getMainView()
    .getEditor().getMembers();
if (members.length > 1){
  for (int i = 0; i < members.length-i; i++) {
    PipeElement ithElement = members[i];
    PipeElement ithPlusOneElement = members[i+1];
    ithPlusOneElement.setParameter(ithElement.getOutput(),
        ithElement.getDataObject().getOutputType());
  }
}
// execute the last element
members[members.length -1].getDataObject()
    .execute(members[members.length -1].getOutput());
```

The snippet shows the method checks at beginning if the Pipe is possible, which means all outputs and outputs match. If it is valid, the code iterates over all Operators and uses the parameter of the selected element to set the value of the matching field of the next element. Finally the last `PipeElement` is executed, which is often a Fetcher. The execution of the Pipe leads to a request to a data source.

4.5.4 Querying and Analyzing Data Sources

There are three ways to query data sources in *IfiPipes*. The first is to use a SPARQL endpoint, which return SPARQL results encoded in JSON. As most web services uses JSONP, which allows one to circumvent the single-origin policy, the approach uses GWT's `JsonpRequestBuilder`. The object is requested by calling `requestObject(url, callback)` and if the request is returned the callback method will run. The second way queries a SPARQL endpoint as well, but this time the SPARQL results are encoded in XML. Therefore GWT's `RequestBuilder` is used to send a GET request to the server, ba calling `sendRequest(data, callback)`. If the request to the SPARQL endpoint is successful and the result, either JSON or XML, is returned, the data is sent to the server, where it is analyzed. The third approach fetches RDF encoded in various formats. Unlike the other two methods which are called on the client-side, the RDF fetching happens on the server-side and make use of Jena's `Model.read(url)` functionality. As the data is already on the server it is analyzed.

The analysis for all formats is done in the `QueryResultAnalyzeServiceImpl` class. The class loads the Visualization ontology the first time a user runs a pipe, more exactly in its constructor. The analysis is pretty simple, for SPARQL results keywords describing a the ontology are compared with the requested data. If the keywords match, the visualization is added to the pool of possible visualizations. After finishing the matching the list of visualizations are returned to the client, where it is are presented to the user in dialog.

4.5.5 Visualizing Data

The visualizations of the data happens in created in the `ApplicationWindow`.The display of the data depends greatly on which visualization is used, which format are the results in and is the data temporal or not. As there are three different formats and four visualizations, there are unfortunately many cloned lines of code and a lot of condition. One example of a visualization can be found in Listing 4.6.

Listing 4.6: Snippet of the getCorrespondingSparqlVisualization method from the ApplicationWindow class

```
for (int i = 0; i < jsonData.size(); i++) {
  final JSONObject bindingObject = jsonData.get(i).isObject();
  double lat = Double.parseDouble(bindingObject
    .get("lat").isString().stringValue());
  double lng = Double.parseDouble(bindingObject
    .get("long").isString().stringValue());
  LatLng point = LatLng.newInstance(lat, lng);
  final Marker marker = new Marker(point);
  marker.addMarkerClickHandler(new MarkerClickHandler() {
    ...
  });
  map.addOverlay(marker);
}
```

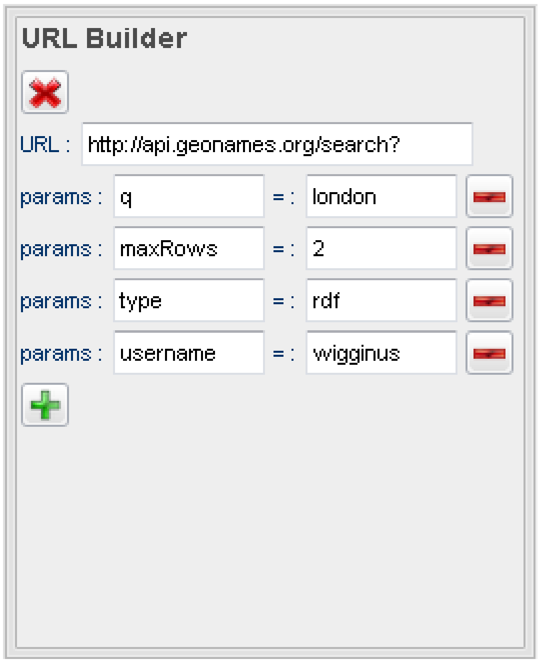
The snippet shows the creation of a the geographic visualization for SPARQL results encoded in JSON. The map is already loaded asynchronously at beginning and the method adds `Marker` to the map to display the results. The other visualization are similar, but slightly less complex.

4.6 User Guide

The design of *IfiPipes* is pretty intuitive, but nevertheless a user guide for the application is provided. *IfiPipes* allows the combination of so-called Operators to create Pipes. Pipes consist of at least one Operator which fetches a data source. All Operators can be dragged from the Operators grid in the left side of the application. As soon as the dragged Operator is dropped onto the Editor, the graphical representation of the Operator is drawn. The Operators can be freely arranged on the Editor, to change the order. An explanation of all Operators can be found below in Subsection 4.6.1.

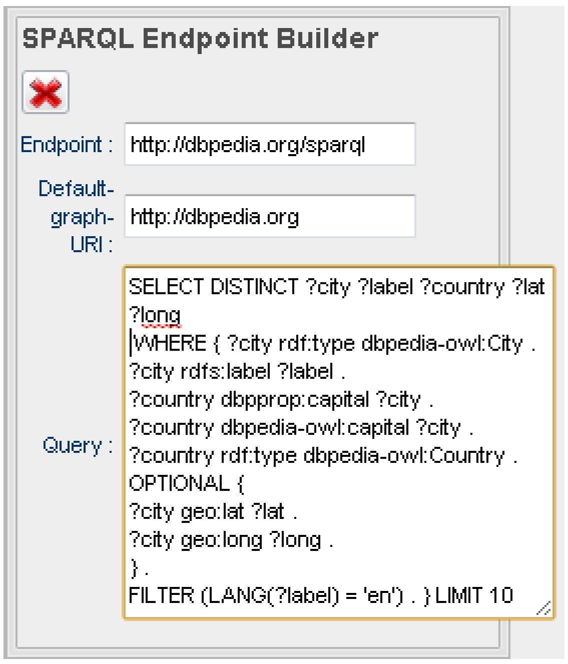
4.6.1 List of Operators

There are four types of Operators: Builder, Fetcher, Filter and Conditions. Builder are graphical helpers to create URLs, endpoint URL or SPARQL queries. The URL Builder accepts a base URL and a variable number of parameters. The inputs for the SPARQL Endpoint Builder are fixed, it accept the URL of the SPARQL endpoint, the URI of the default graph and the SPARQL query. Depending on the endpoint SPARQL 1.1 may or may not be supported. The two implementations for URL Builder (a) and SPARQL Endpoint Builder (b) can be seen in Figure 4.4.



URL : http://api.geonames.org/search?
params : q =: london
params : maxRows =: 2
params : type =: rdf
params : username =: wigginus

(a) URL Builder



Endpoint : http://dbpedia.org/sparql
Default-graph-URI : http://dbpedia.org
Query :

```
SELECT DISTINCT ?city ?label ?country ?lat
?long
WHERE { ?city rdf:type dbpedia-owl:City .
?city rdfs:label ?label .
?country dbpprop:capital ?city .
?country dbpedia-owl:capital ?city .
?country rdf:type dbpedia-owl:Country .
OPTIONAL {
?city geo:lat ?lat .
?city geo:long ?long .
} .
FILTER (LANG(?label) = 'en') . } LIMIT 10
```

(b) SPARQL Endpoint Builder

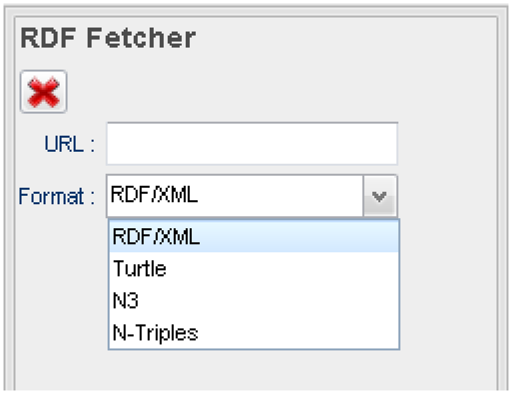
Figure 4.4: Builder Operators

Fetcher are Operators which accept a URL as input and fetch data from a local or remote source. While the RDF Fetcher (a) loads RDF data in different formats, the SPARQL Result Fetcher (b) sends a SPARQL query to a SPARQL endpoint and receives SPARQL results, either encoded in XML or JSON. The two Fetcher Operators are depicted in Figure 4.5.

4.6.2 Running a Pipe

By combining a Builder with a Fetcher one creates a runnable Pipe. By clicking on the Run Pipe button in the menu the Pipe is analyzed and the program decides if its a valid Pipe or not. The validity check compares input and output of all the Operators and if they match, the combination is valid. A Pipe is only valid im all combinations are valid. If the Pipe is not executable a warning pops up and the user is prompted to fix the Pipe setup. If the Pipe is executable the selected data source is queried. As soon as the response from the data source is there, the requested data is analyzed by the application and a window pops up, as seen in Figure 4.6.

The window allows the user to choose various visualizations. Depending on the analyzed data one or more visualizations are selected. If the user is unhappy with the selection, he or she can change the selection and the program tries to fit the data in the selected visualizations nev-

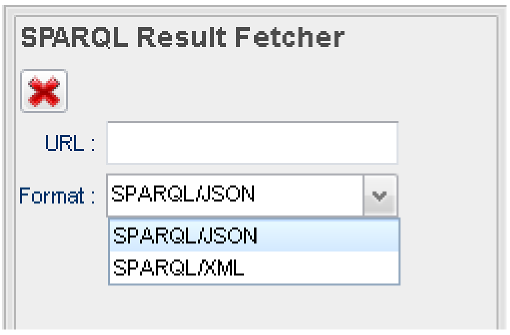

RDF Fetcher

URL :

Format : RDF/XML

RDF/XML
Turtle
N3
N-Triples

(a) RDF Fetcher


SPARQL Result Fetcher

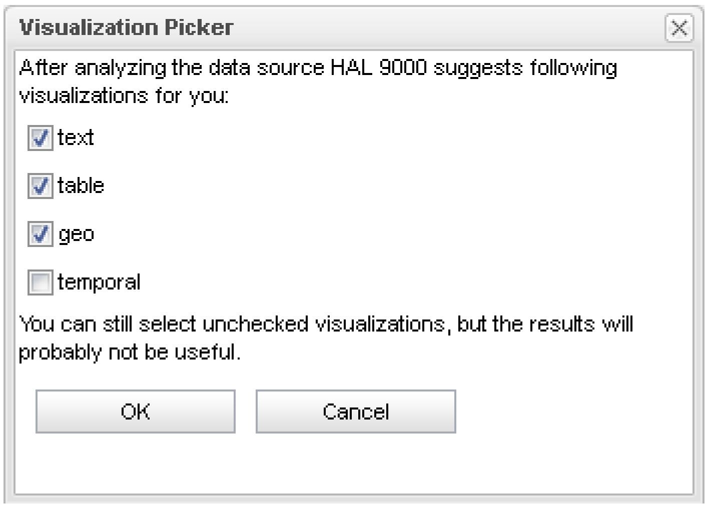
URL :

Format : SPARQL/JSON

SPARQL/JSON
SPARQL/XML

(b) SPARQL Result Fetcher

Figure 4.5: Fetcher Operators


Visualization Picker

After analyzing the data source HAL 9000 suggests following visualizations for you:

[✓] text

[✓] table

[✓] geo

[ ] temporal

You can still select unchecked visualizations, but the results will probably not be useful.

OK Cancel

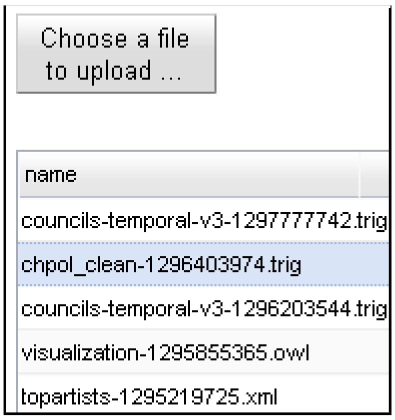
Figure 4.6: The Visualization Selection Screen

ertheless. After selecting at least one visualization from the visualization picker the application switches to the visualization tab and displays the selected visualizations in tabs.

4.6.3 Uploading a Data Set

The application allows the user to upload data sets, which aren't already accessible via HTTP. To upload a data source, the data tab in the controls on the left has to be selected, which can be seen in Figure 4.7.

There one can find already existing data sets or upload new ones. To upload a data set the user has to click on the upload button and a dialog to choose a file is displayed. The application supports data sets up to ten MB and the following file types: .xml, .rdf, .owl, .n3 and .trig. The file is uploaded in the background and as soon as the upload is finished, it is displayed on top of



councils-temporal-v3-1297777742.trig
chpol_clean-1296403974.trig
councils-temporal-v3-1296203544.trig
visualization-1295855365.owl
topartists-1295219725.xml

Figure 4.7: The data tab of IfiPipes

the list. The name of the data set is appended a timestamp, so the data sets won't be overwritten by equal named file. To use a file as a data source, one can drag the file from the list into the URL of an RDF Fetcher and run the Pipe like normal.

4.6.4 Visualizing Capitals of the World

The section will present a use case for IfiPipes, to show the reader what the application is capable of. The use case has the goal to visualize all capitals of the world on a map. As there is plenty of data available on the web describing countries and political data, the data will be fetched from a web service. In this concrete case a SPARQL endpoint will be queried, this use case concentrates on the DBPedia endpoint. DBPedia is the semantic clone of Wikipedia and the largest linked data source. On DBPedia one can find an almost unlimited resources, especially when looking for facts.

The first action of the user is to start the application and wait for the Operators to load. When the grid with the Operators appears on the left side, the application is fully loaded and ready for use. At beginning the Editor remains empty, a state which is going to change soon. To start with the Pipe, a SPARQL Endpoint Builder is selected on the Operators grid and dropped onto the editor, the main area with a grey background. After the Operator drew itself, the user starts filling in the fields. The URL for the DBPedia endpoint (`http://dbpedia.org/sparql`) and the default graph URI (`http://dbpedia.org`) can both be found on their homepage, the user copies and pastes the values in the respective fields. The SPARQL query has to be filled in manually, for the brevity of the use case it is provided in Listing 4.7.

Listing 4.7: SPARQL Query to list all capitals of the world

```
SELECT DISTINCT ?city ?label ?country ?lat ?long
WHERE {
        ?city rdf:type dbpedia-owl:City .
        ?city rdfs:label ?label .
        ?country dbpedia-owl:capital ?city .
        ?country rdf:type dbpedia-owl:Country .
        ?country dbpprop:sovereigntyType ?type .
        ?city geo:lat ?lat .
        ?city geo:long ?long .
        FILTER (LANG(?label) = 'en') .
        LIMIT 100 .
}
```

After filling in the values, the user selects a Sparql Endpoint Fetch and drop it right next to the Builder. This completes a simple runnable Pipe. As DBPedia supports both SPARQL/JSON and SPARQL/XML formats, the user leaves the default value to SPARQL/JSON. To run the Pipe the user clicks on the 'Run Pipe' button on top. After a short waiting period, the visualization picker is shown. The user selects the desired visualizations and waits for the application to draw them. When the application has finished with the task, the user may switch tabs to see different visualizations and play with them. A possible outcome of this result can be seen in Figure 4.8.

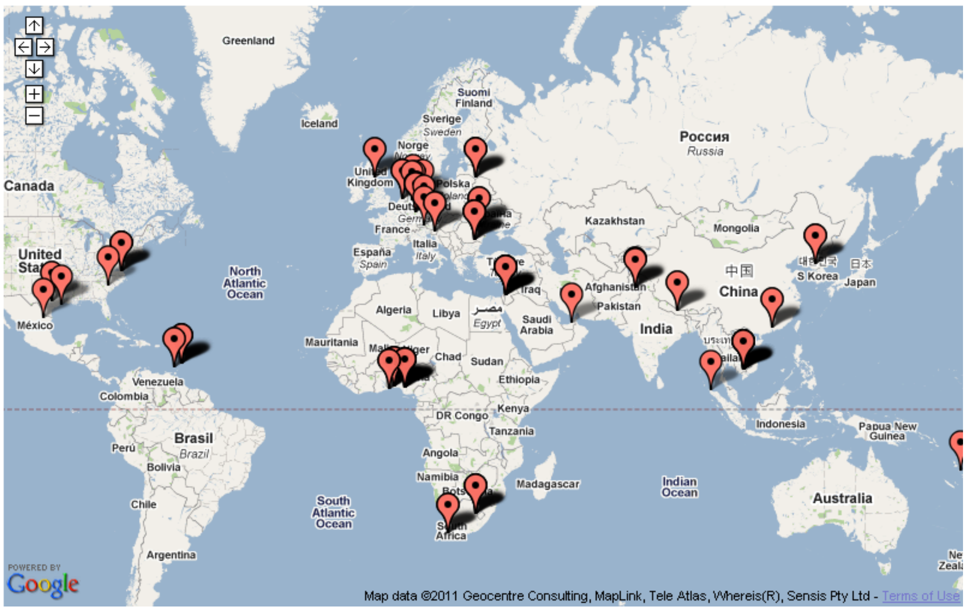


Figure 4.8: The Map Visualization of IfiPipes

If one sees the figure and thinks this isn't right, there aren't that many capitals, they are right. The problem is that DBPedia thinks even states or regions are capitals and therefore they are shown as well.

5 Limitations

It is almost impossible to write a thesis and apply the learned knowledge without limitations in ones work, especially if the task has to be done in a relatively short time. This chapter shows limitations in both the framework, respectively the ontologies, and the resulting application. Also limitations in the used technologies are displayed, which have mostly an effect of the speed and the snappiness of the application.

5.1 Framework

The framework, which consists of the two ontologies is pretty straightforward. The Pipes ontology is rather small and simple, which works fine for the requirements of the application. Although it probably could be extended, it would at the same time increase the complexity and reduce both flexibility and extensibility. The Visualization ontology is also lightweight, but would probably need some redesign. The keywords which describe a data source are stored in a single value as a comma separated String, instead of modelling each keyword as a data property and reusing existing ontologies. This design leads to a very simple mapping between the keywords from the ontology and the keywords extracted from the queried data. While this String comparison is working, it doesn't use the power of the Semantic Web, which could enforce the mapping by using reasoning.

Another issue with the Visualization ontology is the description of the visualization. Whereas the ontology includes a brief description of the visualization, is isn't used in the application. The reason for this is the fact, that it is hard to map the description of the visualization to the actual visualization. The task is even more tiresome when using a framework like GWT, where most libraries and even reflection can't be used on the client, where to visualization is created.

5.2 Application

At the time of writing *IfiPipes* hasn't an option to save created Pipes. A procedure to serialize and store Pipes or single Operators could enable the reuse of Pipes. As it is a Semantic Web application it would be obvious to store the Pipes in RDF, but the implementation is open, one could use a data base or store them as flat files. If one would choose to serialize the Pipes in XML, it would be nice if other users could import their serialized Pipes, created in Deri Pipes.

IfiPipes implements four visualizations: two of them are textual representations and the other two are graphic visualizations. One is a geographical visualization on a map and the other displays temporal data. The temporal visualization isn't a stand-alone visualization, but rather an extension to one of the other visualizations. As the visualizations are the most important part for the user, it would be great, if one could create more visualizations. Similarly the users could benefit from a larger number of Operators. Some of these are defined, but not yet implemented, while some developers might have ideas for completely new Operators.

If the application is used by multiple users, it would be important to implement some sort of caching. This is especially true, if the user makes extensive use of the temporal visualization. Temporal changes aren't filtered on the client side, but rather an AJAX request is sent each time the time slider is changed. For large data sets this results in long waiting times for the user and deteriorates the user experience.

5.3 Used Technology

One of the biggest limitations of the application is the absence of one feature in the frameworks. Whereas it would have been nice to have a Microsoft Visio like editor, which allows the wiring of elements and the total rearrangement of them, none of the investigated frameworks provides this ability well enough. Draw2D[1], which is a JavaScript library devoted to the task to create these Visio-like drawings, could have been used, but the bindings to the frameworks are either very old or still in pre-alpha state[2].

Another quirk with GWT is that the architecture is split in client- and server-side. This results in snappier client-side interactions, but has the problem that only very few Java libraries can be used. Unfortunately there is no Semantic Web or RDF library, so all client-side code must be implemented using mostly primitive data types and almost no libraries. The other option is to always send data back and forth between client and server. This results in a similar architecture if one would use a server-side framework, but one has to write the all the requests. The application uses both approaches, all work using the ontologies is done on the server side, while the client-side parses the returned data from a request to prepare the data for the visualizations.

Another limitation is the speed of the temporal framework, which takes quite long to load a file into memory. This is especially crucial in web applications, where time is an important factor. Generally all semantic stores aren't on par with modern relational data bases, if only speed is measured, so it isn't the fault of the tGraph implementation. The fact that SmartGWT is primary an enterprise framework and therefore has a huge code base increases the load time. While most speed restrictions can't easily be solved, the result is nevertheless that the application has a long loading time and the response times are sometimes slow.

[1] `http://www.draw2d.org/draw2d/`

[2] ZK has a repository devoted to a Draw2D implementation, is hasn't been updated in more then two years: `http://thinkcap.svn.sourceforge.net/viewvc/thinkcap/plugins/ZKDiagram/` and for GWT there exists a project page for a similar project, but most of the bindings aren't implemented yet: `http://code.google.com/p/gwt-draw2d/`

6 Future Work

The future work chapter is divided in two sections. The first part presents a vision of a universal visualization framework, whereas the second part shows where future work can be done regarding the ontologies and the application.

6.1 Universal Visualization Framework

The thought-provoking Universal Visualization Framework (UVF) is a framework used to describe all kinds of data. Depending on how fine-granular the data set is and of course on the nature of the data itself, the application which implements the UVF displays a different visualization. The visualization would be as specific as possible and the application would let the user interact with it and provide controls for filtering and searching, as proposed in [Shneiderman, 2002]. If the user is unsatisfied, he/she could select another visualization. These other visualizations would be ranked by the application as well and would be displayed in a list or tile view, sorted by their ranking. The application should be able to find out if a visualization is frequently rejected and by using machine learning algorithms the framework should be improved. The application should allow the user to switch between broader and more fine-granular visualizations. While the UVF sounds like a daydream of most information visualization researchers, which it is nowadays, it could become real in the future. Like Tim Berners-Lee vision of the Semantic Web [Berners-Lee et al., 2001], the vision of the UVF is similarly futuristic, but nevertheless both desirable and necessary. Rather than creating one framework to rule them all, it would be more convenient to create a number of frameworks, each for a specific use case. Similar to linked data, frameworks are uploaded on the web, where they can be accessed without restrictions and are interlinked with each other. This results in a few improvements to the current state of visualizations. First, it is increasingly easier to use a visualization for non-engineers. Second, the concept of visualization 'forking'[1] is born. As both frameworks and data are open, everyone can either reuse an existing visualization with a new, fitting data set or can create a new visualization with already existing data sets.

[1]The term fork comes is used in software development and describes the process of copying the source code of a software, to work independently on it.

It would make sense to implement these frameworks as OWL ontologies. The concept is already known from linked data, so little learning is necessary. Furthermore OWL is a perfect solution to create meta-ontologies, as the import of external ontologies is straightforward. The mapping between the data and the data description could be realized by probabilistic reasoning, like suggested in [Gilson et al., 2008].

6.2 IfiPipes

There are a few limitations in the application, as mentioned in Chapter 5. Future work could include a developer fixing these limitations. First, one could implement the saving functionality, which enables the reuse of Pipes and the sharing of Pipes with other users. Second, it is always nice to have more features or in this case visualizations. While as the map, the temporal and the textual/tabular visualization are pretty straightforward and aren't very specific, one could create visualization exactly for very specific data sets. An example would be the specification for a politician visualization, depending on the agenda of their party and the actual political profile. Third, the mentioned caching would improve the speed of the application and make the visualizations more interactive. Speed, especially for web applications, is almost always the biggest issue of users. Forth, the architecture of the application could be redesigned and the Model View Presenter pattern could be applied more strictly. Fifth, the user interface could be beautified by an interaction designer. Instead of using predefined widgets of SmartGWT it would be nice to have custom designed modules, which look less 'enterprisey'. While this isn't a needed for an academical application, it would be needed if one has the goal to gain non-researcher as users. An extreme approach of this redesign would be the scrapping of the whole application and only use the framework, to build a new implementation.

The ontologies and especially the mapping between ontology and data could be improved as well. Whereas the Pipes ontology is useful for the task, the Visualization ontology is less advanced. Instead of using keywords stored as `owl:hasValue` restrictions it would be better if the ontology made use of existing resources and their URIs. For example the map visualization could be described using already defined properties and classes from the W3C geo vocabulary[2]. To further improved the mapping between the data and its description, one could use probabilistic reasoning as it is done by [Gilson et al., 2008] and the hypothetical UVF.

[2]www.w3.org/2003/01/geo/wgs84_pos

7 Conclusions

This Master Thesis was about the creation of a framework for information visualization. The resulting *IfiPipes* framework consists of two ontologies, the Pipes ontology defines data retrieval and aggregation and the Visualization ontology is about the description of data sources and the actual visualizations. *IfiPipes* implements these ontologies to create a web application which allows the user to select information from data sources distributed all over the web. Furthermore one can query these data sources and to visualize the queried results. The queries can be sent by using web services or even SPARQL endpoints. Although there are a few similar solutions in the same space, none of them implements all features of IfiPipes. Most data aggregation and visualization services provide either a full set of visualizations, but don't use semantic data or they focus on semantic data, but provide no or very few visualizations.

The Master Thesis also likes to solve the issue that there is almost no reuse in visualizations. Like conditional mash-ups, visualizations are created for one use only and break if the API, respectively the data changes. By using the technologies of the Semantic Web it would allow developers to reuse visualizations, if they are fed with similar data. Therefore the concept of visualization taxonomies and even higher-level visualization ontologies should gain more traction in the future. Eventually this results in the creation of the Universal Visualization Framework presented in Section 6.1. Similar to the vision of the Semantic Web, the Universal Visualization Framework brings structure and thus reuse for visualizations, which leads to ubiquitous visualization options in the daily life.

A Appendix

A.1 Contents of the CD-ROM

The CD-ROM which accompanies the master thesis contains following data:

- The Folder 'IfiPipes' contains the Eclipse project of IfiPipes including the source code, the ontologies, the used libraries and all graphics.
- The Zip file 'IfiPipes.zip' contains the same files as the folder 'IfiPipes', but is compressed.
- The PDF file 'MasterThesis.pdf' contains the written part of this thesis
- The PDF file 'Zusfsg.pdf' contains the abstract in German
- The PDF file 'Abstract.pdf' contains the abstract in English

A.2 The Pipes Ontology

Listing A.1: The Pipes Ontology encoded in N3

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl2xml: <http://www.w3.org/2006/12/owl2-xml#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix pipes: <http://www.ifi.uzh.ch/ddis/rdf-widget/pipes.owl#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
<http://www.ifi.uzh.ch/ddis/rdf-widget/&owl;Thing> a owl:Class.
pipes:AndCondition a owl:Class;
  rdfs:subClassOf pipes:Condition.
pipes:BooleanCondition a owl:Class;
  rdfs:subClassOf pipes:Condition.
pipes:Builder a owl:Class;
```

```
  rdfs:subClassOf pipes:Pipe.
pipes:Combine a owl:Class;
  rdfs:subClassOf pipes:Filter,
    _:bnode109643136,
    _:bnode927796992.
pipes:CompareCondition a owl:Class;
  rdfs:subClassOf pipes:Condition.
pipes:Condition a owl:Class;
  rdfs:subClassOf pipes:Pipe.
pipes:Construct a owl:Class;
  rdfs:subClassOf pipes:Filter,
    _:bnode1811010112.
pipes:ConstructQuery a owl:Class;
  rdfs:subClassOf pipes:Query.
pipes:EmptyInput a owl:Class;
  rdfs:subClassOf pipes:Condition.
pipes:EndpointURL a owl:Class;
  rdfs:subClassOf pipes:URL.
pipes:Fetcher a owl:Class;
  rdfs:subClassOf pipes:Pipe.
pipes:Filter a owl:Class;
  rdfs:subClassOf pipes:Pipe.
pipes:hasInput a owl:ObjectProperty;
  rdfs:domain pipes:Pipe.
pipes:hasOutput a owl:ObjectProperty;
  rdfs:domain pipes:Pipe.
pipes:HTMLFetch a owl:Class;
  rdfs:subClassOf pipes:Fetcher,
    _:bnode773569984,
    _:bnode263870144.
pipes:HTMLResults a owl:Class;
  rdfs:subClassOf pipes:Results.
pipes:imgURL a owl:DatatypeProperty.
pipes:MatchCondition a owl:Class;
  rdfs:subClassOf pipes:Condition.
pipes:name a owl:DatatypeProperty.
pipes:NotCondition a owl:Class;
  rdfs:subClassOf pipes:Condition.
pipes:OrCondition a owl:Class;
  rdfs:subClassOf pipes:Condition.
pipes:Pipe a owl:Class;
  rdfs:subClassOf <http://www.ifi.uzh.ch/ddis/rdf-widget/&owl;Thing>,
    _:bnode1956216896,
    _:bnode1301310272.
pipes:PipeCall a owl:Class;
```

```
  rdfs:subClassOf pipes:Filter ,
    _:bnode918776768 ,
    _:bnode118663360 .
pipes:Query a owl:Class;
  rdfs:subClassOf pipes:Stream .
pipes:RDFExtract a owl:Class;
  rdfs:subClassOf pipes:Filter ,
    _:bnode2101423680 ,
    _:bnode1156103488 .
pipes:RDFFetch a owl:Class;
  rdfs:subClassOf pipes:Fetcher ,
    _:bnode1727910400 ,
    _:bnode1063983552 ,
    _:bnode492176640 ,
    _:bnode1529616768 .
pipes:RDFXMLResults a owl:Class;
  rdfs:subClassOf pipes:Results .
pipes:ReplaceText a owl:Class;
  rdfs:subClassOf pipes:Filter ,
    _:bnode171750208 ,
    _:bnode865689920 .
pipes:Results a owl:Class;
  rdfs:subClassOf pipes:Stream .
pipes:Select a owl:Class;
  rdfs:subClassOf pipes:Filter ,
    _:bnode628255744 .
pipes:SelectQuery a owl:Class;
  rdfs:subClassOf pipes:Query .
pipes:Smoosher a owl:Class;
  rdfs:subClassOf pipes:Filter ,
    _:bnode409184384 ,
    _:bnode1965344576 .
pipes:SPARQLEndpointBuilder a owl:Class;
  rdfs:subClassOf pipes:Builder ,
    _:bnode1894002368 ,
    _:bnode326084672 ,
    _:bnode1363524800 ,
    _:bnode1903130048 .
pipes:SPARQLJSONResults a owl:Class;
  rdfs:subClassOf pipes:Results .
pipes:SPARQLQueryBuilder a owl:Class;
  rdfs:subClassOf pipes:Builder ,
    _:bnode856562240 ,
    _:bnode699597952 .
pipes:SPARQLResultFetch a owl:Class;
```

```
  rdfs:subClassOf pipes:Fetcher,
    _:bnode1665695872,
    _:bnode1520489088,
    _:bnode483048960,
    _:bnode1073111232,
    _:bnode1737038080.
pipes:SPARQLXMLResults a owl:Class;
  rdfs:subClassOf pipes:Results.
pipes:Stream a owl:Class;
  rdfs:subClassOf _:bnode1292182592.
pipes:Substract a owl:Class;
  rdfs:subClassOf pipes:Filter,
    _:bnode254742464,
    _:bnode1301417728.
pipes:URL a owl:Class;
  rdfs:subClassOf pipes:Stream.
pipes:URLBuilder a owl:Class;
  rdfs:subClassOf pipes:Builder,
    _:bnode1956109440,
    _:bnode399949248,
    _:bnode637490880.
pipes:value a owl:DatatypeProperty.
pipes:XMLFetch a owl:Class;
  rdfs:subClassOf pipes:Fetcher,
    _:bnode1582596160,
    _:bnode1674931008.
pipes:XMLResults a owl:Class;
  rdfs:subClassOf pipes:Results.
<http://www.ifi.uzh.ch/ddis/rdf-widget/pipes.owl> a owl:Ontology.
_:bnode1063983552 a owl:Restriction;
  owl:hasValue "RDF Fetch";
  owl:onProperty pipes:name.
_:bnode1073111232 a owl:Restriction;
  owl:hasValue "images/SPARQLResultFetch.png";
  owl:onProperty pipes:imgURL.
_:bnode109643136 a owl:Restriction;
  owl:onProperty pipes:hasOutput;
  owl:someValuesFrom pipes:RDFXMLResults.
_:bnode1156103488 a owl:Restriction;
  owl:onProperty pipes:hasOutput;
  owl:someValuesFrom pipes:RDFXMLResults.
_:bnode118663360 a owl:Restriction;
  owl:onProperty pipes:hasOutput;
  owl:someValuesFrom pipes:Stream.
_:bnode1292182592 a owl:Restriction;
```

```
  owl:onProperty pipes:value;
  owl:someValuesFrom <http://www.ifi.uzh.ch/ddis/rdf-widget/&xsd;string>.
_:bnode1301310272 a owl:Restriction;
  owl:onProperty pipes:name;
  owl:someValuesFrom <http://www.ifi.uzh.ch/ddis/rdf-widget/&xsd;string>.
_:bnode1301417728 a owl:Restriction;
  owl:onProperty pipes:hasOutput;
  owl:someValuesFrom pipes:RDFXMLResults.
_:bnode1363524800 a owl:Restriction;
  owl:onProperty pipes:hasInput;
  owl:someValuesFrom pipes:URL.
_:bnode1520489088 a owl:Restriction;
  owl:onProperty pipes:hasInput;
  owl:someValuesFrom pipes:EndpointURL.
_:bnode1529616768 a owl:Restriction;
  owl:onProperty pipes:hasOutput;
  owl:someValuesFrom pipes:RDFXMLResults.
_:bnode1582596160 a owl:Restriction;
  owl:onProperty pipes:hasInput;
  owl:someValuesFrom pipes:URL.
_:bnode1665695872 a owl:Restriction;
  owl:onProperty pipes:hasOutput;
  owl:someValuesFrom pipes:SPARQLJSONResults.
_:bnode1674931008 a owl:Restriction;
  owl:onProperty pipes:hasOutput;
  owl:someValuesFrom pipes:XMLResults.
_:bnode171750208 a owl:Restriction;
  owl:onProperty pipes:hasInput;
  owl:someValuesFrom pipes:Stream.
_:bnode1727910400 a owl:Restriction;
  owl:hasValue "images/RDFFetch.png";
  owl:onProperty pipes:imgURL.
_:bnode1737038080 a owl:Restriction;
  owl:onProperty pipes:hasOutput;
  owl:someValuesFrom pipes:SPARQLXMLResults.
_:bnode1811010112 a owl:Restriction;
  owl:onProperty pipes:hasOutput;
  owl:someValuesFrom pipes:ConstructQuery.
_:bnode1894002368 a owl:Restriction;
  owl:onProperty pipes:hasOutput;
  owl:someValuesFrom pipes:EndpointURL.
_:bnode1903130048 a owl:Restriction;
  owl:hasValue "SPARQL Endpoint Builder";
  owl:onProperty pipes:name.
_:bnode1956109440 a owl:Restriction;
```

```
  owl:hasValue "URL Builder";
  owl:onProperty pipes:name.
_:bnode1956216896 a owl:Restriction;
  owl:onProperty pipes:imgURL;
  owl:someValuesFrom <http://www.ifi.uzh.ch/ddis/rdf-widget/&xsd;string>.
_:bnode1965344576 a owl:Restriction;
  owl:onProperty pipes:hasOutput;
  owl:someValuesFrom pipes:RDFXMLResults.
_:bnode2101423680 a owl:Restriction;
  owl:onProperty pipes:hasInput;
  owl:someValuesFrom pipes:Results.
_:bnode254742464 a owl:Restriction;
  owl:minQualifiedCardinality "2"^^<&xsd;nonNegativeInteger>;
  owl:onClass pipes:RDFXMLResults;
  owl:onProperty pipes:hasInput.
_:bnode263870144 a owl:Restriction;
  owl:onProperty pipes:hasOutput;
  owl:someValuesFrom pipes:HTMLResults.
_:bnode326084672 a owl:Restriction;
  owl:onProperty pipes:hasInput;
  owl:someValuesFrom pipes:Query.
_:bnode399949248 a owl:Restriction;
  owl:onProperty pipes:hasOutput;
  owl:someValuesFrom pipes:URL.
_:bnode409184384 a owl:Restriction;
  owl:minQualifiedCardinality "2"^^<&xsd;nonNegativeInteger>;
  owl:onClass pipes:RDFXMLResults;
  owl:onProperty pipes:hasInput.
_:bnode483048960 a owl:Restriction;
  owl:hasValue "Sparql Result Fetch";
  owl:onProperty pipes:name.
_:bnode492176640 a owl:Restriction;
  owl:onProperty pipes:hasInput;
  owl:someValuesFrom pipes:URL.
_:bnode628255744 a owl:Restriction;
  owl:onProperty pipes:hasOutput;
  owl:someValuesFrom pipes:SelectQuery.
_:bnode637490880 a owl:Restriction;
  owl:onProperty pipes:hasInput;
  owl:someValuesFrom pipes:URL.
_:bnode699597952 a owl:Restriction;
  owl:onProperty pipes:hasOutput;
  owl:someValuesFrom pipes:Query.
_:bnode773569984 a owl:Restriction;
  owl:onProperty pipes:hasInput;
```

```
   owl:someValuesFrom pipes:URL.
_:bnode856562240 a owl:Restriction;
   owl:hasValue "SPARQL Query Builder";
   owl:onProperty pipes:name.
_:bnode865689920 a owl:Restriction;
   owl:onProperty pipes:hasOutput;
   owl:someValuesFrom pipes:Stream.
_:bnode918776768 a owl:Restriction;
   owl:onProperty pipes:hasInput;
   owl:someValuesFrom pipes:Pipe.
_:bnode927796992 a owl:Restriction;
   owl:minQualifiedCardinality "2"^^<&xsd;nonNegativeInteger>;
   owl:onClass pipes:RDFXMLResults;
   owl:onProperty pipes:hasInput.
```

A.3 The Visualization Ontology

Listing A.2: The Visualization Ontology encoded in N3

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix visualization:
  <http://www.ifi.uzh.ch/ddis/rdf-widget/visualization.owl#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix wgs84_pos: <http://www.w3.org/2003/01/geo/wgs84_pos#>.
@prefix owl2xml: <http://www.w3.org/2006/12/owl2-xml#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
<http://www.ifi.uzh.ch/ddis/rdf-widget/&owl;Thing> a owl:Class.
<http://www.ifi.uzh.ch/ddis/rdf-widget/&wgs84_pos;Point> a owl:Class.
visualization:basedOn a owl:ObjectProperty;
  rdfs:domain visualization:Visualization.
visualization:GeoProperty a owl:Class;
  rdfs:subClassOf <http://www.ifi.uzh.ch/ddis/rdf-widget/&wgs84_pos;Point>,
     visualization:Property.
visualization:GeoRDF a owl:Class;
  rdfs:subClassOf visualization:RDF,
     _:bnode1446542208.
visualization:GeoSparqlResults a owl:Class;
  rdfs:subClassOf visualization:SparqlResults,
     _:bnode109617984,
     _:bnode1665778176.
visualization:keywords a owl:DatatypeProperty;
  rdfs:comment "add keywords as a comma-seperated list";
  rdfs:domain visualization:SparqlResults;
  rdfs:range <http://www.ifi.uzh.ch/ddis/rdf-widget/&xsd;string>.
```

```
visualization:MapsViz a owl:Class;
  rdfs:subClassOf visualization:Visualization ,
    _:bnode1073028928.
visualization:PlainTextViz a owl:Class;
  rdfs:comment "a textual visualization of the returned values"^^<&xsd;string>;
  rdfs:subClassOf visualization:Visualization.
visualization:possibleDatatypes a owl:DatatypeProperty;
  rdfs:comment "add datatypes as a comma-seperated list";
  rdfs:domain visualization:SparqlResults;
  rdfs:range <http://www.ifi.uzh.ch/ddis/rdf-widget/&xsd;string>.
visualization:Property a owl:Class.
visualization:QueryResult a owl:Class;
  rdfs:subClassOf <http://www.ifi.uzh.ch/ddis/rdf-widget/&owl;Thing>.
visualization:RDF a owl:Class;
  rdfs:subClassOf visualization:QueryResult.
visualization:SparqlResults a owl:Class;
  rdfs:subClassOf visualization:QueryResult.
visualization:TableViz a owl:Class;
  rdfs:subClassOf visualization:Visualization.
visualization:TemporalRDF a owl:Class;
  rdfs:subClassOf visualization:RDF,
    _:bnode483131264.
visualization:TemporalSparqlResults a owl:Class;
  rdfs:subClassOf visualization:SparqlResults ,
    _:bnode699515648,
    _:bnode2039291456.
visualization:TempProperty a owl:Class;
  rdfs:subClassOf visualization:Property.
visualization:TempViz a owl:Class;
  rdfs:subClassOf visualization:Visualization.
visualization:Visualization a owl:Class.
<http://www.ifi.uzh.ch/ddis/rdf-widget/visualization.owl> a owl:Ontology;
  owl:imports <http://www.w3.org/2003/01/geo/wgs84_pos>.
_:bnode1073028928 a owl:Restriction;
  owl:onProperty visualization:basedOn;
  owl:someValuesFrom visualization:GeoProperty.
_:bnode109617984 a owl:Restriction;
  owl:hasValue "float,double";
  owl:onProperty visualization:possibleDatatypes.
_:bnode1446542208 a owl:Restriction;
  owl:hasValue "http://www.w3.org/2003/01/geo/wgs84_pos#lat,
    http://www.w3.org/2003/01/geo/wgs84_pos#long";
  owl:onProperty visualization:keywords.
_:bnode1665778176 a owl:Restriction;
  owl:hasValue "lat,long,geo,latitude,longitude";
  owl:onProperty visualization:keywords.
_:bnode2039291456 a owl:Restriction;
  owl:hasValue "int,float,double";
  owl:onProperty visualization:possibleDatatypes.
_:bnode483131264 a owl:Restriction;
  owl:hasValue "isQuad";
  owl:onProperty visualization:keywords.
_:bnode699515648 a owl:Restriction;
  owl:hasValue "time,from,to,until,now";
  owl:onProperty visualization:keywords.
```

List of Figures

List of Tables

List of Listings

Bibliography

[Alexander, 2008] Alexander, K. (2008). RDF/JSON: A Specification for serialising RDF in JSON. In *Workshop on Scripting for the Semantic Web*. Citeseer.

[Ankolekar et al., 2007] Ankolekar, A., Krötzsch, M., Tran, T., and Vrandecic, D. (2007). The two cultures: Mashing up Web 2.0 and the Semantic Web. In *Proceedings of the 16th international conference on World Wide Web*, pages 825–834. ACM.

[Auber, 2003] Auber, D. (2003). Tulip-a huge graph visualization framework. *Graph Drawing Software*.

[Baker, 1987] Baker, P. (1987). Pipes and filters. *Byte*, 12(12):215–217.

[Beckett and McBride, 2004] Beckett, D. and McBride, B. (2004). RDF/XML syntax specification (revised). *W3C recommendation*, 10.

[Berners-Lee, 2006] Berners-Lee, T. (2006). Linked data. *International Journal on Semantic Web and Information Systems*, 4(2).

[Berners-Lee, 2009] Berners-Lee, T. (2009). Linked data - design issues. `http://www.w3.org/DesignIssues/LinkedData.html`.

[Berners-Lee, 2010] Berners-Lee, T. (2010). Long Live the Web. *Scientific American Magazine*, 303(6):80–85.

[Berners-Lee et al., 2006] Berners-Lee, T., Chen, Y., Chilton, L., Connolly, D., Dhanaraj, R., Hollenbach, J., Lerer, A., and Sheets, D. (2006). Tabulator: Exploring and analyzing linked data on the semantic web. In *Proceedings of the 3rd International Semantic Web User Interaction Workshop*, volume 2006. Citeseer.

[Berners-Lee et al., 2001] Berners-Lee, T., Hendler, J., Lassila, O., et al. (2001). The semantic web. *Scientific American*, 284(5):34–43.

[Bernstein, 2005] Bernstein, A. (2005). So what is a (diploma) thesis? a few thoughts for first-timers. Technical report, Dynamic and Distributed Information Systems Group, Univerity of Zurich, Switzerland.

[Bizer et al., 2007] Bizer, C., Cyganiak, R., and Heath, T. (2007). How to publish linked data on the web. *Retrieved June*, 20:2008.

[Bizer et al., 2009] Bizer, C., Heath, T., and Berners-Lee, T. (2009). Linked data-the story so far. *International Journal on Semantic Web and Information Systems*, 5(3):1–22.

[Bizer et al., 2005] Bizer, C., Lee, R., and Pietriga, E. (2005). Fresnel-a browser-independent presentation vocabulary for rdf. In *Proceedings of the Second International Workshop on Interaction Design and the Semantic Web, Galway, Ireland*. Citeseer.

[Bratt, 2007] Bratt, S. (2007). Semantic web, and other technologies to watch. `http://www.w3.org/2007/Talks/0130-sb-W3CTechSemWeb`.

[Clark and Avery, 1976] Clark, W. and Avery, K. (1976). The effects of data aggregation in statistical analysis. *Geographical Analysis*, 8(4):428–438.

[Crockford, 2006] Crockford, D. (2006). JSON: The fat-free alternative to XML. In *Proc. of XML*, volume 2006.

[Cruz and Tamassia, 1998] Cruz, I. and Tamassia, R. (1998). Graph drawing tutorial. `http://www.cs.brown.edu/~rt/papers/gd-tutorial/gd-constraints.pdf`.

[Dean and McDermott, 1987] Dean, T. and McDermott, D. (1987). Temporal data base management. *ARTIFICIAL INTELLIG.*, 32(1):1–55.

[Di Battista et al., 1998] Di Battista, G., Eades, P., Tamassia, R., and Tollis, I. (1998). *Graph drawing: algorithms for the visualization of graphs*. Prentice Hall PTR Upper Saddle River, NJ, USA.

[Drummond and Shearer, 2006] Drummond, N. and Shearer, R. (2006). The open world assumption. *Presentation, The Univ. of Manchester*.

[Friendly and Denis, 2001] Friendly, M. and Denis, D. (2001). Milestones in the history of thematic cartography, statistical graphics, and data visualization. *web document, available at http://www. math. yorku. ca/SCS/Gallery/milestone*, 174.

[Gardner, 2005] Gardner, S. (2005). Ontologies and semantic data integration. *Drug Discovery Today*, 10(14):1001–1007.

[Gershenfeld et al., 2004] Gershenfeld, N., Krikorian, R., and Cohen, D. (2004). The Internet of Things. *Scientific American*, 291(4):76–81.

[Gilson et al., 2008] Gilson, O., Silva, N., Grant, P., and Chen, M. (2008). From web data to visualization via ontology mapping. In *Computer Graphics Forum*, volume 27, pages 959–966. Wiley Online Library.

[Gorman, 2010] Gorman, S. (2010). Open Data: Why the Crowd Can Be Your Best Analytics Tool. `http://mashable.com/2010/12/30/crowd-data-analytics/`.

[Gutierrez et al., 2005] Gutierrez, C., Hurtado, C., and Vaisman, A. (2005). Temporal rdf. *The Semantic Web: Research and Applications*, pages 93–107.

[Gutierrez et al., 2006] Gutierrez, C., Hurtado, C., and Vaisman, A. (2006). Introducing time into RDF. *Knowledge and Data Engineering, IEEE Transactions on*, 19(2):207–218.

[Halevy et al., 2005] Halevy, A., Ashish, N., Bitton, D., Carey, M., Draper, D., Pollock, J., Rosenthal, A., and Sikka, V. (2005). Enterprise information integration: successes, challenges and controversies. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 778–787. ACM.

[Halevy et al., 2006] Halevy, A., Rajaraman, A., and Ordille, J. (2006). Data integration: The teenage years. In *Proceedings of the 32nd international conference on Very large data bases*, pages 9–16. VLDB Endowment.

[Hayes, 2009] Hayes, P. (2009). Blogic or now what's in a link? `http://videolectures.net/iswc09_hayes_blogic/`.

[Hayes and McBride, 2004] Hayes, P. and McBride, B. (2004). RDF semantics. *W3C recommendation*, 10:38–45.

[Heer et al., 2005] Heer, J., Card, S., and Landay, J. (2005). Prefuse: a toolkit for interactive information visualization. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 421–430. ACM.

[Hirsch et al., 2009] Hirsch, C., Hosking, J., and Grundy, J. (2009). Interactive visualization tools for exploring the semantic graph of large knowledge spaces. In *Workshop on Visual Interfaces to the Social and the Semantic Web (VISSW2009)*. Citeseer.

[Inc, 2011] Inc, Y. (2011). Yahoo! pipes - documentation. `http://pipes.yahoo.com/pipes/docs`.

[Klyne et al., 2004] Klyne, G., Carroll, J., and McBride, B. (2004). Resource description framework (RDF): Concepts and abstract syntax. *Changes*.

[Kobilarov et al., 2009] Kobilarov, G., Scott, T., Raimond, Y., Oliver, S., Sizemore, C., Smethurst, M., Bizer, C., and Lee, R. (2009). Media meets semantic web–how the BBC uses DBpedia and linked data to make connections. *The Semantic Web: Research and Applications*, pages 723–737.

[Le-Phuoc et al., 2009] Le-Phuoc, D., Polleres, A., Hauswirth, M., Tummarello, G., and Morbidoni, C. (2009). Rapid prototyping of semantic mash-ups through semantic web pipes. In *Proceedings of the 18th international conference on World wide web*, pages 581–590. ACM.

[Lewis, 2007] Lewis, R. (2007). Dereferencing http uris. *Draft Tag Finding*, 31:2007–05.

[Merrill, 2006] Merrill, D. (2006). Mashups: The new breed of Web app. *IBM Web Architecture Technical Library*.

[Meunier, 1995] Meunier, R. (1995). The pipes and filters architecture. In *Pattern languages of program design*, page 440. ACM Press/Addison-Wesley Publishing Co.

[Quirolgico et al., 2004] Quirolgico, S., Assis, P., Westerinen, A., Baskey, M., and Stokes, E. (2004). Toward a formal common information model ontology. In *Web Information Systems–WISE 2004 Workshops*, pages 11–21. Springer.

[Schraefel and Karger, 2006] Schraefel, M. and Karger, D. (2006). The pathetic fallacy of rdf. In *International Workshop on the Semantic Web and User Interaction (SWUI)*, volume 2006.

[Sears and Jacko, 2008] Sears, A. and Jacko, J., editors (2008). *Information Visualization*, chapter 26, pages 509–543. Lawrence Erlbaum Associates.

[Shadbolt et al., 2006] Shadbolt, N., Hall, W., and Berners-Lee, T. (2006). The semantic web revisited. *Intelligent Systems, IEEE*, 21(3):96–101.

[Shannon, 2006] Shannon, V. (2006). A 'more revolutionary' web. *The New York Times*.

[Shneiderman, 2002] Shneiderman, B. (2002). The eyes have it: A task by data type taxonomy for information visualizations. In *Visual Languages, 1996. Proceedings., IEEE Symposium on*, pages 336–343. IEEE.

[Snodgrass et al., 1994] Snodgrass, R., Ahn, I., Ariav, G., Batory, D., Clifford, J., Dyreson, C., Elmasri, R., Grandi, F., Jensen, C., K
"afer, W., et al. (1994). TSQL2 language specification. *ACM SIGMOD Record*, 23(1):65–86.

[Solove, 2002] Solove, D. (2002). Access and Aggregation: Privacy, Public Records, and the Constitution. *Minnesota Law Review*, 86(6).

[Spalding, 2007] Spalding, S. (2007). How to define web 3.0. `http://howtosplitanatom.com/news/how-to-define-web-30-2/`.

[Spence, 2001] Spence, R. (2001). *Information visualization*. Addison-Wesley Reading, MA.

[Tappolet and Bernstein, 2009] Tappolet, J. and Bernstein, A. (2009). Applied temporal rdf: Efficient temporal querying of rdf data with sparql. *The Semantic Web: Research and Applications*, pages 308–322.

[Trevor, 2008] Trevor, J. (2008). Doing the mobile mash. *Computer*, 41(2):104–106.

[Yu et al., 2008] Yu, J., Benatallah, B., Casati, F., and Daniel, F. (2008). Understanding mashup development. *IEEE Internet Computing*, pages 44–52.

[Ziegler and Dittrich, 2004a] Ziegler, P. and Dittrich, K. (2004a). Three Decades of Data Intecrationall Problems Solved? *Building the Information Society*, pages 3–12.

[Ziegler and Dittrich, 2004b] Ziegler, P. and Dittrich, K. (2004b). User-specific semantic integration of heterogeneous data: The sirup approach. *Semantics of a Networked World*, pages 44–64.