UNIVERSITÄT KOBLENZ · LANDAU
Fachbereich 4: Informatik

DFKI Saarbrücken

# Graph-Based Visualization of RDF Soccer Data and Interaction Possibilities on a Handheld

Diplomarbeit

zur Erlangung des Grades eines Diplom-Informatikers
im Studiengang Computervisualistik

vorgelegt von

Philipp Heim

| | |
|---|---|
| Erstgutachter: | Prof. Dr.-Ing. Stefan Müller<br>(Institut für Computervisualistik, AG Computergraphik) |
| Zweitgutachter: | Dipl.-Ling. Daniel Sonntag<br>(DFKI Saarbrücken, Intelligente Benutzerschnittstellen) |

Koblenz, im Februar 2007

## Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

| | Ja | Nein |
|---|---|---|
| Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden. | ☐ | ☐ |
| Der Veröffentlichung dieser Arbeit im Internet stimme ich zu. | ☐ | ☐ |

.........................................................................

(Ort, Datum) (Unterschrift)

# Aufgabenstellung

Die Dialogführung mit einem Computer ist eingeschränkt. Ein wichtiger Grund hierfür ist die Tatsache, dass ein Computer nicht im menschlichen Sinne *"verstehen"* kann. Um eine Verständigung zwischen Mensch und Computer dennoch zu ermöglichen, müssen wir Verstehen simulieren. Für diese Simulation benötigt man semantische Informationen über den Dialog. Diese können mit Hilfe einer Ontologie bereitgestellt werden.

Die Diplomarbeit beschäftigt sich mit der grafischen Repräsentation eines solchen Dialogs. Dieser Dialog findet über einen Handheld (PDA) statt und besteht im Wesentlichen aus einer Frage und der darauf gegebenen Antwort durch den Computer.

Die semantischen Informationen, die benötigt werden, um Verstehen zu simulieren, werden bereits durch das Dialogsystem extrahiert. Die Aufgabe dieser Diplomarbeit ist herauszufinden, auf welche Weise man diese Informationen für die grafische Schnittstelle des Dialogs zwischen Computer und Mensch konkret nutzen kann.

Die Herausforderung besteht darin, die semantischen Informationen zu den Antworten des Computers in sinnvoller Weise für die Präsentation einzusetzen. Der Einsatz soll dem Verständnis von komplexen Resultaten und der Möglichkeit zur Navigation durch die Resultatstrukturen dienen. Auch sind die besonderen Konditionen auf einem PDA, wie geringer Platz, niedrige Auflösung und beschränkte Rechenleistung zu berücksichtigen. Hierfür müssen geeignete Lösungen gesucht und getestet werden.

Das Ziel dieser Diplomarbeit ist die Entwicklung einer grafischen Darstellungs- und Interaktionskomponente zur Nutzung von semantischen Informationen.

Schwerpunkte dieser Arbeit sind:

1. Literatur Recherche
2. Prototypentwicklung / Evaluation
3. Implementierung / Umsetzung des Prototypen
4. Dokumentation

Die Diplomarbeit wird am DFKI (Deutsches Forschungszentrum für Künstliche Intelligenz GmbH), Bereich IUI (Intelligent User Interfaces), innerhalb des SmartWeb Projektes in Saarbrücken durchgeführt.

# Abstract

In this thesis we searched for ways to enhance human accessibility to the current Semantic Web technology by enabling the visualization of knowledge. We built a graph-based user interface in which users can explore and adapt information that is arranged with respect to the relations within this information as well as its meaning. The data necessary to arrange information that way is extracted out of RDF descriptions.

The work in this thesis was done in the context of the SmartWeb project at the German Research Center for Artificial Intelligence (DFKI). The goal of the SmartWeb project is to lay the foundations for multimodal user interfaces to distributed and composable Semantic Web services on mobile devices.

Due to the use of a mobile handheld device, our graph-based user interface must be able to present content on a small screen size. For the space-efficient visualization we use a graphical fisheye view for graphs in combination with advanced interaction forms to offer details on demand. The use of a graphical fisheye view is a valuable tool for seeing both "local detail" and "global context" simultaneously.

Our graph presentation system needs to be able to deal with arbitrary RDF input data. Therefore, we structurally map this RDF input data to useful graph data for a visualization and than use an automatic layouter to arrange that graph data properly on the handheld. The automatic layouter applies a constraint-based method to find positions for all graph vertices with respect to their semantic relations. Due to the use of an automatic layouter, our graph-presentation system is ready for the integration into the SmartWeb system.

During the development of this thesis, we evaluated the usability of our approach in two stages. The first evaluation of our graph-presentation system showed that the users, especially those with less computer skills, had problems with the advanced interaction possibilities and with the extraction of information out of the distorted graph.

Drawing the right consequences from the first evaluation, the results of the second evaluation showed a clear improvement of usability and suitability to extract information out of the graph. This positive feedback of the second evaluation demonstrates, that our approach to arrange information in a

graph structure on a handheld with respect to its semantic relations can provide the users with a general survey of the information structure and thereby support their understanding of interrelations.

# Contents

# Chapter 1

# Introduction

Visualization is in many cases an important factor of success and value enhancer of information reception. For every specific type of information there are certain categories of visual representation that are more suitable than others. The use of a graph for the visualization of information has the advantage that it can capture the information structure and can model knowledge to a great extend. Therefore graphs are suitable to convey semantic relations between different elements and to provide an understanding of the overall information structure. To arrange information that way, semantic meta-data is needed.

In the approach described in this thesis, we rely on RDF descriptions as semantic meta-data. In the Semantic Web, RDF data is used to make meaning computer-processable. Machines can use those meta-data to build a view of how the individual terms within the information relate to each other and thereby become able to search websites and perform actions in a meaningful way. Even if the described information is intended for machines, there is no reason for not using RDF data for humans too, by facilitating their understanding through enhanced visualization.

## 1.1 Motivation

Our approach, described in this diploma thesis, is founded on the premise that machine web-based meta-data need not necessarily be only for machines, but can be used to offer humans a graphical representation of this meta-data to support their understanding of content.

In this thesis we develop a graphical user interface (GUI) for a mobile device, a handheld, which uses such meta-data for a better understanding of content. Our work is done in the context of the SmartWeb project whose aim it is to develop a context-aware, mobile, and multimodal user interface to the Semantic Web system [44].

Using meta-data for a graphical representation can improve the users' understanding of certain information pieces and of the relations between these pieces. With our additional graphical representation for the SmartWeb system, the dialog with the user could become more efficient because of a faster and more precise understanding of the presented information. A faster and more precise understanding would be especially useful for a dialog on a handheld because of its mobile aspect.

The use of meta-data to arrange information pieces with respect to their semantic relations could lead to an enhanced capacity to retrieve information. The capacity could be enhanced due to the fact that users are more familiar with this way of information arrangement, because humans itself encode information based upon its meaning (semantically) [34].

Just as the way to arrange information, the way to offer a new possibility to ask for new information could be also more familiar to the user than the already existing one.

In our GUI, the user should be able to use the already displayed information as a starting point to ask for additional information. By using this information as starting point, the received additional information is always semantically related to the already displayed one. Unlike to a complete new question, this way the received additional information could be smoothly integrated into the already displayed one. The user will therefore be able to iteratively refine his knowledge by following cross-references that lead to related information. Such a browsing functionality could be more intuitive for the user, especially if he does not know in advance what pieces of information are necessary to find an appropriate solution.

The use of meta-data to support the understanding of information in a graphical user interface could be a promising way to improve the multimodal dialog in the SmartWeb system. It could open up a new way of information reception on a handheld.

## 1.2 Starting Point

Consistent with the motivation given in the previous section, the starting point can be expressed as the question:

- Can RDF description be used not only to make the management and navigation of Web data easier to automate, but also to offer users a graphical representation of this description to support their understanding of content?

In this work we discuss ways of how this RDF description can be used suitably in a new GUI. Thereby we focus on ways for a concrete project, the SmartWeb project.

### 1.2.1 SmartWeb Project

The SmartWeb[1] project is organized by the German Research Center for Artificial Intelligence (DFKI)[2] in cooperation with several other research centers and companies. The goal of the SmartWeb project is to lay the foundations for multimodal user interfaces to distributed and compassable Semantic Web services on mobile devices [55].

The appeal of being able to ask a question to a mobile internet terminal and to receive an answer immediately has been renewed by the broad availability of information on the Web. Ideally, a spoken dialogue system that uses the Web as its knowledge base would be able to answer a broad range of questions.

SmartWeb provides a context-aware user interface, so that it can support the user in different roles, e.g. as a car driver, a motor biker, a pedestrian, or a sports spectator. One of the planned demonstrators of SmartWeb is a personal guide for the 2006 FIFA world cup in Germany that provides mobile infotainment services to soccer fans, anywhere and anytime.

The work in this thesis is done in the context of the SmartWeb project, more precisely in the context of the FIFA world cup scenario of this project. In this scenario soccer fans employ a handheld computer to request information available in the Semantic Web. The questions can be formulated freely in spoken natural language and can be supported by pointing gestures on the touch screen [45]. The domain of inquiry is focused on subjects surrounding the FIFA World Championships 2006, such as facts about the players and games. SmartWeb searches the Internet, in particular the Semantic Web, finds and formulates a natural language answer. In addition, the answer can include videos, graphics, texts, or other type of media. Natural language queries (perhaps supplemented with gestures) initiate an intelligent Semantic Web search. When all relevant data for answering the question has been collected, it is sent back to the handheld client and there presented to the user by a graphical user interface.

### 1.2.2 Semantic Web

The central idea of the Semantic Web initiative is to make the meaning of Web content machine accessible and processable [2]. The information in the Semantic Web is given a well-defined meaning due to meta-data. The meta-data consists of semantic information about the Web content and enables the development of sophisticated tools that can provide a much higher functionality in supporting human activities on the Web.

The Semantic Web relies on the combination of the following technologies:

[1]http://www.smartweb-project.de
[2]http://www.dfki.de

- *Explicit meta-data:* They allow Web pages to make their meaning accessible. For example, on a soccer club's Web Page, meta-data can identify names, final results, attendances, active players, goals etc.
- *Ontologies:* They describe the main concepts of a domain and their relationships. For example, a Sport Event ontology may contain concepts such as soccer matches, players, goalkeepers, and relationships such as subclass information (all goalkeepers are players).
- *Logical reasoning:* It makes it possible to draw conclusions from combining meta-data with ontologies.

The machine-readable descriptions enable content managers to add meaning to the content, thereby facilitating automated information gathering and research by computers. RDF and RDF Schema provide the basic core languages for the Semantic Web.

### 1.2.3 Resource Description Framework

Resource Description Framework[3] (RDF) is the W3C standard for encoding knowledge [53].

RDF provides a general, flexible method to decompose any knowledge into small pieces, called triples, with some rules about the semantics (meaning) of those pieces.

The foundation is breaking knowledge down into a labeled, directed graph. Each edge in the graph represents a fact, or a relation between two things. The edge in the example in figure 1.1, from the vertex michaelBallack labeled playsFor to the vertex chelseaFootballClub represents the fact that Michael Ballack plays for Chelsea Football Club.

The W3C specifications define an XML format to encode RDF. For our example, figure 1.2 shows how actually to write this in RDF.

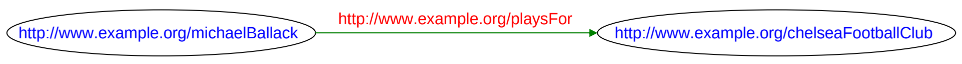

http://www.example.org/michaelBallack
http://www.example.org/playsFor
http://www.example.org/chelseaFootballClub

Figure 1.1: RDF knowledge expressed as a directed graph for the example: Michael Ballack plays for Chelsea Football Club.

A fact represented this way has three parts: a subject, a predicate (i.e., verb), and an object. The subject is what is at the start of the edge, the predicate is the type of edge (its label), and the object is what is at the end of the edge. The predicate expresses a relationship between the subject and the object.

[3]http://www.w3.org/RDF/

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:ex="http://www.example.org/">
  <rdf:Description rdf:about="http://www.example.org/michaelBallack">
    <ex:playsFor rdf:resource="http://www.example.org/chelseaFootballClub"/>
  </rdf:Description>
</rdf:RDF>
```

Figure 1.2: W3C XML format to encode RDF for the example: Michael Ballack plays for Chelsea Football Club.

In essence, RDF Schema is a primitive ontology language offering the following features:

- Organisation of instances in concepts (player, soccer player, stadium, score) and binary relations (happens at, held in, committed by).
- Subconcepts (all soccer players are players) and subrelations (everyone who score a goal in a soccer match is on a team) relationships.
- Domain (only a player can score) and range (one can score goals only) restrictions on relations.

### 1.2.4 Meta-Data

The SmartWeb system makes use of the SmartWeb integrated ontology SWIntO [36] in order to exchange data between its internal modules. SWIntO integrates various domain-specific ontologies that are relevant for mobile and intelligent user interfaces to open-domain question-answering and information services on the Web.

An answer found by the SmartWeb system consists of ontological instances which can include different media types. Every instance is of a certain concept defined in SWIntO and can have relations to other instances. This information is meta-data and it is encoded in RDF. It provides information about concepts and relations of instances.

To give an example of such meta-data, we will use the following dialog consisting of an asked question and a given answer. The question, asked by the user, is:

- How was Germany playing against Argentina in a world championship?

The answer found by the SmartWeb System on the server consists of four soccer matches of these teams in a world championship, the first 1958 in Sweden, the second 1966 in England, the third 1986 in Mexico and the last match 1990 in Italy. Along with this comes further information such as dates, locations, goals, players and so on. An abbreviated RDF representation of the answer found by the SmartWeb system is given in figure 1.3.

```
<rdf:RDF
   xmlns:j.1="http://smartweb.semanticweb.org/ontology/smartdolce#"
   xmlns:j.0="http://smartweb.semanticweb.org/ontology/smartsumo#"
   xmlns:j.2="http://smartweb.semanticweb.org/ontology/context#"
   xmlns:j.3="http://smartweb.semanticweb.org/ontology/smartmedia#"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:j.4="http://smartweb.semanticweb.org/ontology/swemma#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:j.6="http://smartweb.semanticweb.org/ontology/mpeg7#"
   xmlns:j.5="http://smartweb.semanticweb.org/ontology/discourse#"
   xmlns:j.7="http://smartweb.semanticweb.org/ontology/sportevent#"
   xmlns:j.8="http://smartweb.semanticweb.org/ontology/emma#" >

<rdf:Description rdf:about="http://smartweb.(...)#Argentinien_vs_Deutschland_FRG_29__Juni_1986">
   <rdfs:label>Argentinien vs. Deutschland (29.06.1986) 3:2 Finale</rdfs:label>
   <j.1:HAPPENS-AT rdf:resource="http://smartweb(...)semistruct#29. Juni 1986_interval"/>
   <j.7:inTournament rdf:resource="http://smartweb.(...)#SportEventOntology7Jan_Instance_10028"/>
   <j.7:inRound rdf:resource="http://smartweb.semanticweb.org/ontology/semistruct#id_31992"/>
   <j.7:heldIn rdf:resource="http://smartweb.(...)semistruct#Mexico_City_Azteca_Stadium"/>
   <j.7:attendance>114600</j.7:attendance>
   <j.7:officials rdf:resource="(...)Deutschland_FRG_29__Juni_1986_Berny_Morera"/>
   <j.7:matchEvents rdf:resource="(...)vs_Deutschland_FRG_29__Juni_1986_Jose_Brown23_Score"/>
   <j.7:matchEvents rdf:resource="(...)vs_Deutschland_FRG_29__Juni_1986_Jorge_Valdano55_Score"/>
   <j.7:matchEvents rdf:resource="(...)Deutschland_29__Juni_1986_Karl-Heinz_Rummenigge74_Score"/>
   <j.7:matchEvents rdf:resource="(...)_vs_Deutschland_FRG_29__Juni_1986_Rudi_Voller80_Score"/>
   <j.7:matchEvents rdf:resource="(...)Deutschland_FRG_29__Juni_1986_Jose_Burruchaga83_Score"/>
 </rdf:Description>
</rdf:RDF>

<rdf:Description rdf:about="http://(...)sportevent#SportEventOntology7Jan_Instance_10028">
   <rdf:type rdf:resource="http://smartweb.semanticweb.org/ontology/sportevent#FIFAWorldCup"/>
   <j.7:url>http://fifaworldcup.yahoo.com/06/en/p/pwc/r/1986.html</j.7:url>
   <j.7:NumberOfQualifiers>113</j.7:NumberOfQualifiers>
   <j.7:gender>Men</j.7:gender>
   <rdfs:label>WC 31.5.1986 - 29.6.1986, MEXICO (MEX)</rdfs:label>
   <j.7:NumberOfParticipants>24</j.7:NumberOfParticipants>
   <j.7:officialName>XIII Copa del Mundo - Mexico'86</j.7:officialName>
   <j.1:HAPPENS-AT rdf:resource="http://(...)#SportEventOntology_Instance_30037"/>
</rdf:Description>

<rdf:Description rdf:about="(...)Deutschland_29__Juni_1986_Karl-Heinz_Rummenigge74_Score">
   <rdfs:label>Tor von Karl-Heinz Rummenigge (0) in der 74. Minute</rdfs:label>
   <j.7:committedBy rdf:resource="(...)Deutschland_29__Juni_1986_Karl-Heinz_Rummenigge_PFP"/>
   <j.1:HAPPENS-AT rdf:resource="http://(...)#29-6-1986__:_timepoint+74_RelativeTimePoint"/>
   <j.7:opponentScoreAfterGoal>1</j.7:opponentScoreAfterGoal>
   <j.7:ownScoreAfterGoal>2</j.7:ownScoreAfterGoal>
   <rdf:type rdf:resource="http://smartweb.semanticweb.org/ontology/sportevent#ScoreGoal"/>
</rdf:Description>

(...)
```

Figure 1.3: Abbreviated RDF representation of the answer to the question: *"How was Germany playing against Argentina in a world championship?"* found by the SmartWeb system

The meta-data encoded in this RDF representation defines, for example, that a certain match between Germany and Argentina is of the concept *"soccer match"*. In this *"soccer match"* concept all relations to other concepts are described in an abstract form. So every concept defines the relation structure for all concrete instances of it. Such a relation is, for example, *"happens at"*. This relation has a *"soccer match"* as the subject concept and the *"date"* concept as the object. For a concrete instance of *"soccer match"* there must be a concrete instance of *"date"* for the *"happens at"* relation. So every instance has, depending on its concept, several relations to other instances. All relations defined by a concept are called the outgoing relations for the instances of this concept. These outgoing relations have this concept as the subject and other concepts as the object.

Every instance has arbitrary incoming relations. Incoming relations are all relations with the concept of this instance as the object. For a concrete instance of the concept *"date"*, the *"happens at"* relation is an incoming relation, because it leads from a *"soccer match"* to a *"date"* concept. Whether a relation is called incoming or outgoing relation depends on the concept we look at. For the *"soccer match"*, *"happens at"* is an outgoing relation, for the *"date"* concept it is an incoming one.

Not all relations are only between instances. Some are between an instance and a primitive. A primitive can be a literal or a number and has no outgoing relations.

For the presentation of this information to the user, every instance has labeling information. These labels stand for the semantic meaning of its instance.

In figure 1.3 the labels are marked with the tag *"rdfs:label"*. For example, a label for an instance of the concept *"tournament"* is *"WC 31.5.1986 - 29.6.1986, MEXICO (MEX)"*.

Along with the labels comes more information about this instance, like the number of participants or a outgoing relation *"happens at"* to an instance of a date. For every concept the number of outgoing relations can differ, starting from zero up to a few tens. The soccer match in figure 1.3, for example, has nine outgoing relations.

All the information in the RDF files is given by the SmartWeb system. The question is how to use this knowledge description for a graphical user interface on a handheld in a multimodal dialog system.

### 1.2.5 SmartWeb Architecture

The SmartWeb architecture consists of two basic processing blocks: handheld client and server system platform as shown in figure 1.4. The multimodal recognizers, the dialog system, and the Semantic Web access subsystems are located on this server. The user poses open-domain multimodal questions over the handheld client. The multimodal input is interpreted and

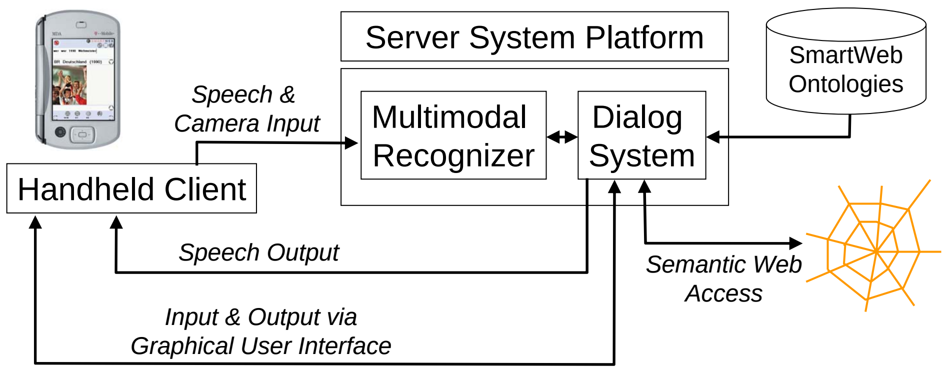


Figure 1.4: Distributed dialog processing in the SmartWeb system.

transmitted using UMTS or wireless LAN to a back-end server system. The handheld client and the server system exchange data in an XML format [44].

After the answer to an asked question has been found on the server side, it is sent back to the client, running on the user's handheld. It consists of a local Java based control unit which takes care of all I/O. It is connected to the GUI-controller, which is realized using Macromedia Flash with Actionscript. So the results are presented to the user via a Flash interface on the handheld client.

This thesis is written in the context of the SmartWeb project. Therefore the conditions are:

- A handheld client that uses a graphical user interface, shown in figure 1.5 and 1.6.
- Over this interface, the user can pose open-domain multimodal questions.
- The multimodal input is interpreted and transmitted using UMTS or wireless LAN to a back-end server system.
- The communication is realized due to a client-server architecture.
- The response to a question is presented on the mobile device and rendered on its screen (figure 1.6).
- A dialog is used for question answering functionality.

## 1.3 Task Description

In this thesis, the task is to develop a graphical representation of the answers found by the SmartWeb system. Additionally to the existing representation

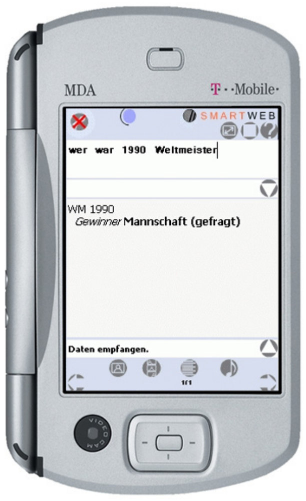


Figure 1.5: The current GUI for the SmartWeb system on a handheld client. The user has asked the question: “Who was the world champion 1990?”

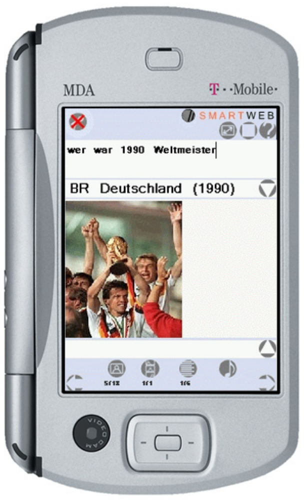


Figure 1.6: The response to this question is presented on the mobile device. The answer is “Germany”.

of the result data, consisting of media types such as text, picture, video, audio, and answer snippet (figure 1.6), the task is to offer the user a second result representation. In contrast to the current representation on the handheld, this new representation should deploy the meta-data to present the results in a way to support the understanding of its overall structure.

Due to the use of this meta-data, the user should better understand the complex interrelations in the result data and receive a different, a semantic point of view on it. We are using the term *semantic point of view*, because the meta-data identifies concepts and relationships and thereby gives information a well-defined meaning.

Furthermore, the second representation should offer a possibility to navigate through the result data. The user should be able to explore the received data through this semantically based graphical representation. We call such an exploration a *semantic navigation*.

Beyond the *semantic navigation*, the user should be able to ask for new information due to this second interface. In addition to the possibility to ask a concrete question, formulated freely in spoken natural language or typed in an input field (figure 1.5), this second interface should provide a way to directly use the presented results as a starting point to ask for new information.

Together with this new way to ask for information, the goal of the second GUI is to reach the following three benefits:

1. To support a new understanding of the received data by using the meta-data out of the RDF files.

2. To offer another way to explore the result data by a semantic navigation.

3. To provide another way to ask for new information, using the already received data as a starting point.

The challenge is to reach these benefits at a device with severe restrictions. The most severe restriction thereby is the size of the handheld. So a major effort to deal with is the small size and the low resolution of the screen.

Another problem is the restricted computing power of the device. Because of that, the amount of computation on the handheld has to be as small as possible. Thus it appears that the given kind of device has an important stake in finding the right solution for this task.

Finally, our solution should get integrated in the SmartWeb system and has to work with dynamically generated data.

## 1.4 Related Work

Displaying RDF data in a user-friendly manner is a problem addressed by various types of applications using different representation paradigms [41].

Web-based tools such as Longwell[4] use nested box layouts, or table-like layouts for displaying properties of RDF resources with varying levels of details.

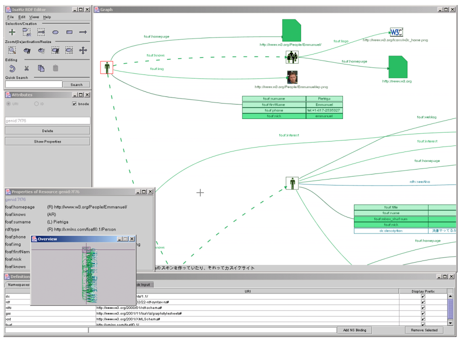
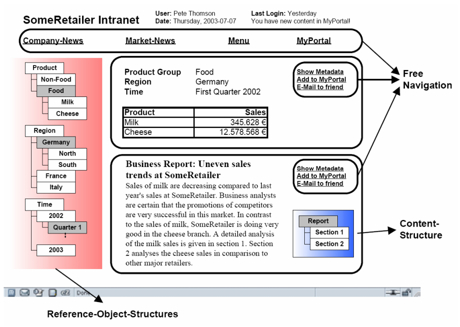


Figure 1.7: IsaViz is a visual environment for browsing and authoring RDF models represented as graphs.



Product Group Food
Region Germany
Time First Quarter 2002

| Product | Sales |
|---|---|
| Milk | 345.628 € |
| Cheese | 12.578.568 € |

**Business Report: Uneven sales trends at SomeRetailer**
Sales of milk are decreasing compared to last year's sales at SomeRetailer. Business analysts are certain that the promotions of competitors are very successful in this market. In contrast to the sales of milk, SomeRetailer is doing very good in the cheese branch. A detailed analysis of the milk sales is given in section 1. Section 2 analyses the cheese sales in comparison to other major retailers.

Figure 1.8: The MoSeNa-Approach aims at modeling complex, role-based and integrated navigation structures for structured and semi-structured data.

[4] http://simile.mit.edu/longwell/

Other tools like IsaViz [39] (see figure 1.7) represent RDF models as vertex-edge diagrams, explicitly showing their graph structure. IsaViz is a visual environment not only for browsing, but also for authoring RDF graphs and is one of the most widely seen tools for graph visualizations of RDF meta-data [40]. It produces the graphs for the W3C's RDF validator.

A third approach combines these paradigms and extends them with specialized user interface widgets designed for specific information items like calendar data, tree structures, or even DNA sequences, providing advanced navigation tools and other interaction capabilities: The MoSeNa-Approach [5] (see figure 1.8) and Haystack [43].

In our approach, we use a graph-based representation of the RDF data. We will think about ways to use this graph-based representation, consisting of vertices and edges, as a user interface for the handheld client.

One problem by visualizing data on a small display is lack of space. Several work has been done on generating space-efficient graphical user interfaces for mobile devices, like a handheld, or for displays with limited display resources in general [25]. We will discuss different ways of dealing with the limitation of space in section 3.4.

Another problem we have to deal with by solving the task described in section 1.3, is to automatically find a graph layout. We need an automatic graph layout, to deal with arbitrary RDF input data. This topic will be addressed in section 5.5.

## 1.5 Outline

The thesis consists of seven chapters.

Chapter 2 starts with the software specification and explains the software architecture. The main focus is on useful design patterns for a graphical user interface.

Chapter 3 deals with the implementation of our approach. The first section is about input dummy data for our first prototype. Section two deals with the concrete visualization, followed by a section about the interaction possibilities and their realization. We conclude the chapter with a discussion about the problem of limited space on the handheld.

In Chapter 4, we use a first evaluation of our first prototype to verify the decisions made so far and to get interesting impulses for further development steps.

In Chapter 5 we draw the consequences for further development of the software. We start with visual improvements of our GUI, which is followed by the important topic of automatic layout as a consequence of arbitrary input data. With the ability to deal with arbitrary input data, our software becomes ready to be integrated into the SmartWeb system.

In Chapter 6, we use a second evaluation to validate the hypotheses from the first evaluation and to identify problems.

The thesis finishes with Chapter 7 that consists of a summary of results and an outlook, giving suggestions for further development.

# Chapter 2

# Specification and Architecture

The steps for the development of our approach are, firstly, a specification of the software, secondly, a description of the architecture and thirdly, the implementation of the specification.

There are always several possible ways to realize a software specification, so pros and cons have to be discussed. Therefore prototyping seems to be a good approach for the software development process.

## 2.1 Prototyping

Prototypes are experimental and incomplete designs, which are cheaply and fast developed. Prototyping, which is the process of developing prototypes, is an integral part of iterative user-centered design because it enables designers to try out their ideas with users and to gather feedback [42].

The main purpose of prototyping is to involve the users in testing design ideas and get their feedback in the early stage of development, thus to reduce the time and cost of development. It provides an efficient and effective way to refine and optimize interfaces through discussion, exploration, testing and iterative revision [46]. Early evaluation can be based on faster and cheaper prototypes before the start of a full-scale implementation. The prototypes can be changed many times until a better understanding of the user interface design has been achieved with the joint efforts of both the designers and the users.

As a guideline for the development of an ergonomic interface we use the international standard ISO Norm EN ISO 9241.

### 2.1.1 Ergonomic Principles

The International Standard Organization has formulated guidelines for ergonomic requirements of computer-supported office work. The corresponding standard comprises hardware design, activity and task design as well as dialogue design.

ISO Norm EN ISO 9241, Volume 10, defines seven ergonomic principles for work-oriented software that have to be considered for the development of a successful interface. The seven ergonomic principles described therein are suitability for the task, self-descriptiveness, controllability, conformity with user expectations, error tolerance, suitability for individualization and suitability for learning.

## 2.2 Software Specification

A specification is a set of requirements. One of the main requirements concerning the implementation are clearly laid out interface components that follow the demands on ergonomic software, as it is described in section 2.1.1. Other important requirements regard the functionality of the software. In the following, an informal description of the functionality is given.

As input data, we receive a RDF file, as it is shown in figure 1.3. We need to find a way to convert this RDF data into a useful structure for a graphical user interface on a handheld client. The unconverted data expressed as a RDF graph is unsuitable to fit on a small display. This is due to the fact that in the RDF graph, every instance is represented by a separate vertex and a RDF file can consist of many instances.

On this account, we need a structural mapping from the RDF data, to a graph structure suitable for visualization on the small display of a handheld client.

After having proper graph data, the next step is to find a graph layout. The graph layout defines for every vertex of the graph data an x- and a y-coordinate, determining its position. Because the received answers are dynamically generated, we need an automatic graph layout to deal with this dynamic input data.

The graph layout, generated automatically, then needs to be presented to the user. Additionally to a proper visualization of this graph layout, the user must be able to interact with the graph appropriately. As mentioned in the task description in section 1.3, the user needs to be able to explore the result data and to ask for new information, using this graph representation.

Finally this has to be integrated into the SmartWeb system. We need the integration to ask for new information and to receive the answers from the SmartWeb system. Furthermore, our graphical user interface has to be integrated into the existing user interface of the SmartWeb system on the handheld client.

The software we are going to develop in this work is expected to fulfill these requirements regarding the functionality, illustrated in figure 2.1.

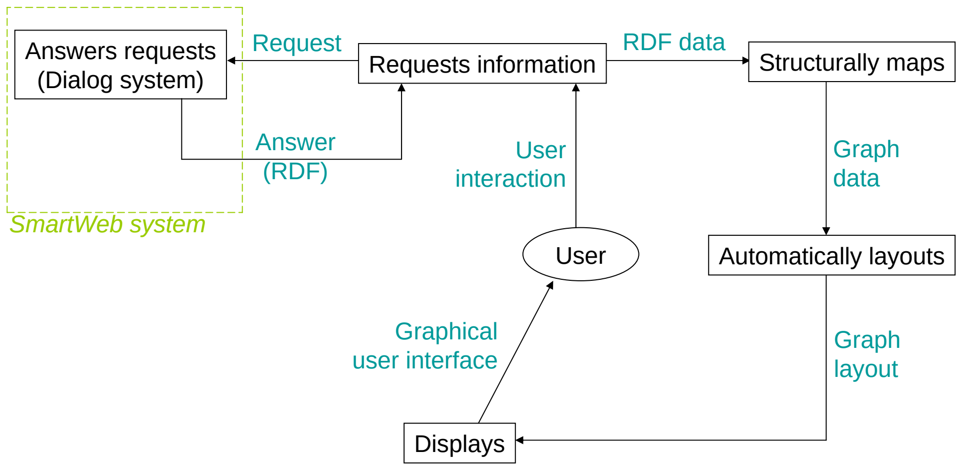


Figure 2.1: A data flow diagram, giving a survey of the data processing that has to be implemented in this diploma thesis.

To sum up, the software specifications are:

1. To receive dynamic input data encoded in RDF.
2. To structurally map these RDF data to useful graph structures for a presentation on a small display.
3. For this dynamic graph data, we have to automatically generate a graph layout.
4. Then the graph layout must be displayed on a handheld client.
5. The user needs to be able to interact with this graph to explore the information in it.
6. Additionally the user needs to be able to ask for new information, using this graph-based user interface.
7. Finally the software has to be integrated into the SmartWeb system.

## 2.3 Software Architecture

The software we are going to develop in this work is expected to provide an additional graphical user interface to make the meta-data of the results accessible for the user. In figure 2.2 we show a flow chart of the new graphical user interface. This new GUI is using Macromedia Flash[1] with Actionscript, because the already existing GUI of the SmartWeb system is using this technology.

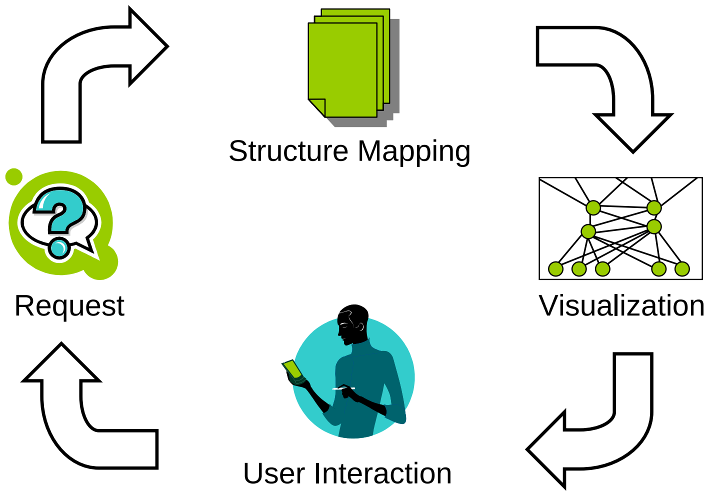


Figure 2.2: A flowchart of the new graphical user interface. The data out of the RDF file is structurally mapped and visualized. The user can interact with the graphical interface displayed on the handheld client. Additionally the user has the opportunity to request for new information, using this graph as starting point.

Our graph presentation system has to handle several complex tasks. It has to arrange correctly a given graph structure, response suitably to user interaction and communicate with the server. To find adequate software architecture for these tasks, we use design patterns.

> A pattern guides a designer by providing workable solutions to all of the problems known to arise in the course of design [4].

### 2.3.1 Design Pattern

Design patterns originated as an architectural concept by Christopher Alexander. He defines a pattern as follows: "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice".

He used it to refer to common problems of civil and architectural design, from how cities should be laid out to where windows should be placed in a room. The idea was initially popularized in his book "A Pattern Language"

[1]http://www.adobe.com/products/flash/flashpro/

[1]. Design patterns gained popularity in computer science after the book "Design Patterns: Elements of Reusable Object-Oriented Software" [24] was published in 1995. It describes patterns for managing object creation, composing objects into larger structures and coordinating control flow between objects.

For our graph-based user interface we are interested in patterns that are particularly applicable to the problems that arise in graphical user interfaces. In the following sections, we introduce three pattern used in the implementation of our user interface.

The first pattern we will look at is the Observer pattern and describes, how to update many objects when a single object changes state (figure 2.3). The second pattern is the Model-View-Controller (MVC) and the third pattern, the Delegation-Event Model. MVC is a combination of patterns that grew out of the Smalltalk language [28] and is used to structure user interfaces. The Delegation-Event Model is a core Java pattern that describes how to implement events and event handling.

### Observer Pattern

The Observer Pattern is a design pattern, used in computer programming to observe the state of an object in a program. The essence of this pattern is that one or more objects (called observers or listeners) are registered to observe an event which may be raised by the observed object (the subject).

The Observer Pattern defines a one-to-many dependency between objects (figure 2.3), so that if one object changes state, all its dependents will be notified and updated automatically [24].

For example, the Observer Pattern might be used to keep a pie chart and a bar chart in sync, or it might be used to let multiple objects respond to the completion of a game [31].

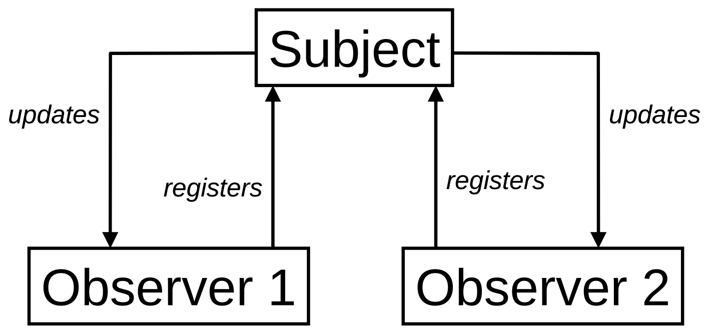


Figure 2.3: Observer pattern.

### Model-View-Controller Design Pattern

The Model-View-Controller (MVC) design pattern encompasses more of the architecture of an application than is typical for a design pattern. In the MVC paradigm the user input, the modeling of the external world and the visual feedback to the user are explicitly separated and handled by three types of objects, each specialized for its task [11].

MVC separates user interface code into three distinct classes:

- **Model:**
  Stores the data and application logic for the interface
- **View:**
  Renders the interface to the screen
- **Controller:**
  Responds to user input by modifying the model

If we look, for example, at a vertex of our graph, then the model stores data like the position and the size of the vertex, the view draws the vertex on screen, and the controller sets the data of the vertex in the model when the vertex is clicked (figure 2.4). The basic principle of MVC is the separation of responsibilities.

Separating the code that governs a user interface into the model, view, and controller classes yields the following benefits [31]:

1. It allows multiple representations (views) of the same information (model).
2. It allows user interfaces (views) to be easily added, removed, or changed, at both compile time and runtime.
3. It allows response to user input (controller) to be easily changed, at both compile time and runtime.
4. It promotes reuse.
5. It helps developers focus on a single aspect of the application at a time.

The MVC pattern is using the Observer Pattern to achieve a loose coupling between the object that changes (the subject) and the objects being notified of the change (the observers). The control flow works as follows:

1. The user interacts with the user interface, the views.
2. A controller handles the input event from the user interface (view).
3. The controller possibly modifies the model in a way appropriate to the user's action.

4. If the model has been modified, it notifies all its observers (views) of a change.
5. The observers (views) update themselves.
6. The user interface (views) waits for further user interactions, which begins the cycle anew.

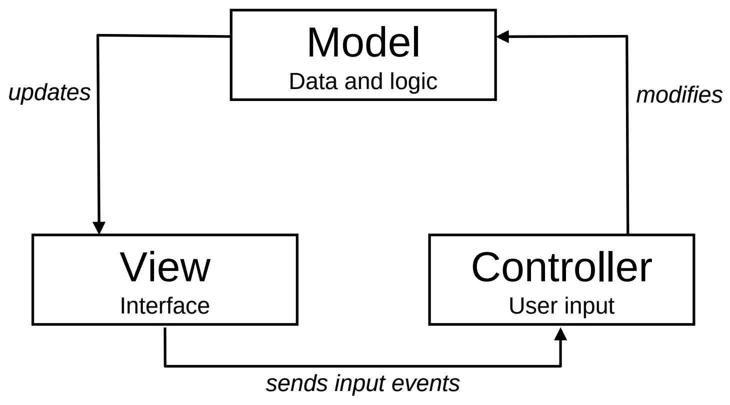


Figure 2.4: The MVC communication cycle.

This control flow starts from a user interaction, leads possibly to a modification of the model and the model then propagates the modification to all its views. This propagation from the model to its views can be called an up propagation, because the changes of the underlying model is propagated up to the views on the surface, the graphical interface (look at figure 2.6).

The propagation from the user interface to the controller can consequently be called a down propagation. However, the MVC pattern is not using a loose coupling between the views and their controllers. For every view there exists only one controller, handling the input events from it. The down propagation in MVC is implemented by a one to one relationship between the view and its controller. Furthermore, every controller controls only one model, however for every model there can exist several controllers to modify it. With these restrictions to the software architecture it is difficult to propagate a user interaction not only to one controller, but to all interested controllers in the system. For a loose coupling between the views and their controllers, too, we need another design pattern.

### Delegation-Event Model

The Delegation-Event Model is a general design for event broadcasting. In the Delegation-Event Model, an event is propagated from a *"Source"* object

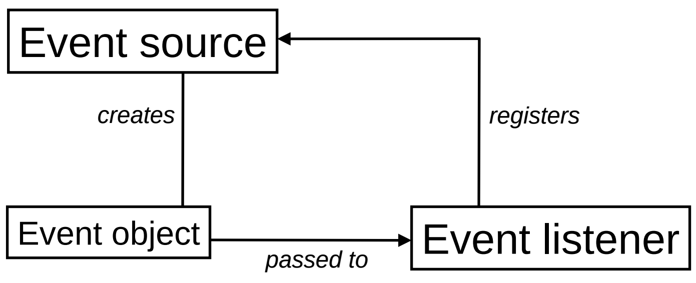


Figure 2.5: The Delegation-Event Model communication cycle.

to a *"Listener"* object by invoking a method on the listener and passing in the instance of the event subclass, which defines the event type generated[2].

The Delegation-Event Model has three main participants:

- **Event source:**
  An object that notifies other objects of some specific occurrence (some events)
- **Event listener:**
  The objects that will be notified by the event source if an event occurs
- **Event object:**
  An object that describes the nature of the event

An event source is, for example, a vertex view, and the event is a click of this vertex view by the user. If this event reaches only the connected controller, then only one vertex model will be modified, namely the model which is observed by the clicked view.

Having such a tight coupling between the view and its controller, prevents a flexible down propagation of user interactions as it is needed in a complex system like our graph presentation. A loose coupling for both propagation directions, down and up, is shown in figure 2.6. In this figure, a user interaction "click", can reach several controllers due to a loose coupled down propagation. These notified controllers possibly modify their models and the modified models then updates their observer (views) due to an up propagation. For a loose coupled down propagation we use the Delegation-Event Model.

In the Delegation-Event Model to notify an event listener of an event, the event source invokes an agreed-upon method on it. The event object is passed to the agreed-upon method by the event source, giving the event listener access to information about the event (figure 2.5).

[2]http://java.sun.com/j2se/1.5.0/docs/guide/awt/1.3/designspec/events.html

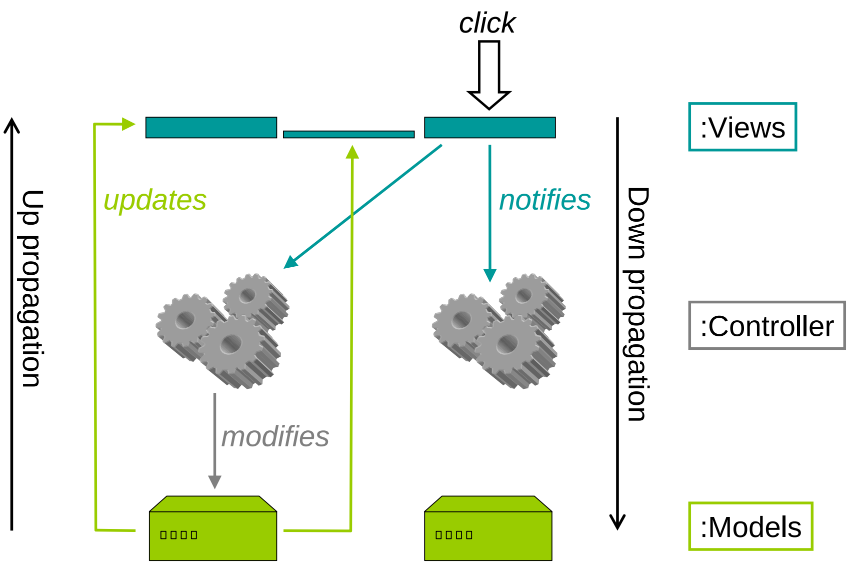


Figure 2.6: When the user clicks a vertex view, the event source broadcasts this event to all registered listeners (Controller), then the controller can modify their models if necessary and finally all modified models update their observers (Views).

For example, the event object holds the state, the identity and the type of the clicked vertex along with the event source, here a vertex view. Every controller (event listener), that is notified by the event source can react independently, depending on the information of the event object and the source.

Using both patterns, the Delegation-Event Model and the Observer, to create a loose coupling in both propagation directions, gives us the possibility to react flexible to user interaction in our graph presentation system.

### Differences between Observer Pattern and Delegation-Event Model

Why do we use two different pattern to attain loose coupling between elements? Structurally, the Delegation-Event Model has much in common with the Observer Pattern. The Observer Pattern has two basic participants, the subject class and the observer classes correspond to, the Delegation-Event Model has event source class, which broadcasts events, and event listener classes, which receive event notifications.

The Delegation-Event Model is designed to broadcast specifically known events rather than generic updates [31]. In Observer, the subject broadcasts updates to observers by invoking a generic update() method. But in the Delegation-Event Model, the event source broadcasts a specific type of event by invoking a custom method on its listener. Listeners receiving the event must implement an interface that defines the method invoked by the event source. Furthermore, in the Delegation-Event Model, any class can broadcast events; the event source does not need to be a subclass of some base event-broadcasting class.

Finally, in Observer, the subject should broadcast an update only in response to a state change. Events, by contrast, can be broadcasted for any reason deemed to be appropriate by the event source.

Generally speaking, the Observer Pattern works well used as an internal update mechanism within a discrete system, such as maintaining the position of a single vertex and its connected edges (look at section 2.3.1).

By contrast, the Delegation-Event Model is appropriate usually, when the granularity of events matters (that is, when events should correspond to individual methods rather than a single update() method). Event granularity is desirable for event sources that have a wide variety of unknown event listeners. For example, a rollout button uses the Delegation-Event Model to broadcast onClick() events to the world, but internally, it maintains its own visual appearance using Observer as part of MVC.

### 2.3.2 Conclusion

Using these three design patterns to build up the graphical user interface for our graph presentation system, makes it possible to maintain all kind of user events and to responses in a flexible way (figure 2.6).

Firstly, the division into model, view and controller offers a clear separation of responsibilities. Secondly, the up propagation and its organization due to the Observer Pattern, make the structure extremely flexible and the single elements independently applicable. And finally, the organization of the down propagation due to the Delegation-Event Model leaves open possibilities in how to respond to a user event to the system.

# Chapter 3

# Prototype Implementation

In this chapter we implement the software specifications described in section 2.2. In a first prototype we are going to implement these specifications in an experimental and incomplete way. The goal is to quickly put together a working model (a prototype) in order to test various aspects of the design, illustrate ideas or features and gather early user feedback. So the first prototype deals with fixed dummy input data only. The ability to deal with arbitrary input data will be implemented in the second prototype.

## 3.1 Dummy Data

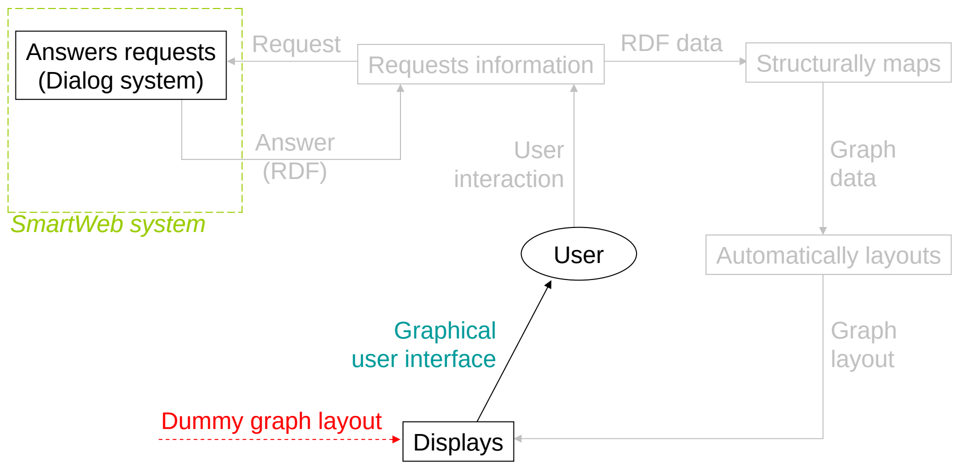


Figure 3.1: A data flow diagram, showing the use of dummy graph layout to simulate wide parts of the data processing.

With the use of dummy data, we simulate the question answering process to a large extent. Starting from a certain user question and its answer, given by the SmartWeb system, encoded in RDF, followed by a structural mapping to receive appropriate graph data and ending with an automatic generation of graph layout. In figure 3.1, the processes and the data flow, simulated by the use of the dummy graph layout, is marked light gray.

### 3.1.1 Dummy Graph Data

A graph data $G$ consists of vertices and edges, $G = (V, E)$. $V$ is a finte, non-empty set of vertices and $E$ is a set of edges (links between pairs of vertices).

As source for our dummy data, we take the answer to the question: *"How was Germany playing against Argentina in a world championship?"*. The answer given by the SmartWeb system, encoded in RDF, has already be shown in figure 1.3.

We manually map this RDF data to useful graph data for a representation on a small display. The goal is to receive graph data with viewer vertices as we would receive by directly mapping each instance of the RDF data to its own vertex. This means, the instances need to be grouped together based on certain criteria. A general implementation of this structure mapping is described in section 5.4, in connection with the integration into the SmartWeb system.

As a first approximation of a structure mapping for our dummy graph data, all instances with the same concept are grouped together to one vertex. All the relations between these concepts, represented by vertices, are mapped to edges. With this rough manual structure mapping of the answer, encoded in RDF, we receive dummy graph data consisting of a directed graph with cycles and labeled edges for a first approach to display such information on the handheld client.

For example, we structurally map all four soccer matches between Germany and Argentina to one vertex $V_1$ and all dates, these matches happened at, to another vertex $V_2$. Every match has a “happens at” relation to one of the dates. All four relations are mapped to one edge between $V_1$ and $V_2$ that is labeled with “happens at”, shown in figure 3.2.

### 3.1.2 Dummy Graph Layout

Having graph data consisting of vertices and edges, the next thing needed for a visualization is the dummy graph layout.

For every vertex we manually define an x- and a y-coordinate, determining its position. An approach to automatically layout arbitrary graph data will be described in section 5.5. For our dummy graph data we do not need a general solution for this problem.

By defining the vertex positions, we have to consider that related vertices should be placed next to each other and the edges between them must not cross. As a result of this layout process we get for every vertex an x- and a y-coordinate. For example, the position $(120, 140)$ for $V_1$ and the position $(60, 100)$ for $V_2$.

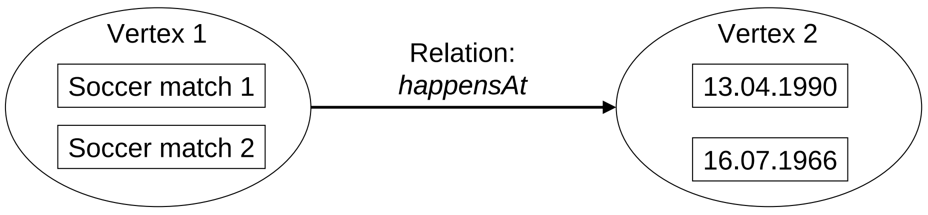


Figure 3.2: Visualization of a relation by an edge between two groups of instances.

## 3.2 Visualization

In this section about the visualization of the graph layout, achieved due to structure mapping, we deal with problems arising from limitation of space on the handheld display. Figure 3.3 illustrates the visualization.

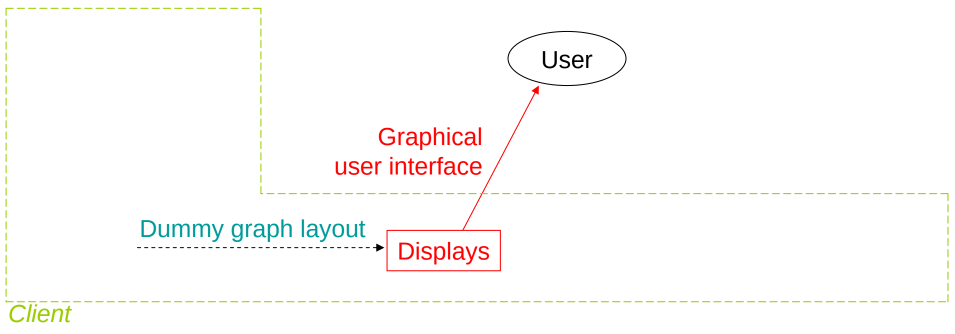


Figure 3.3: A data flow diagram to illustrate the visualization of the graph layout to the user.

### 3.2.1 Visualizing Instances and Relations

As described in section 3.1.1 about the dummy graph data, all instances with the same concept are grouped together and represented by a vertex. All incoming and outgoing relations of this vertex are represented by edges to other vertices, also groups of instances. Actually, for every instance of such a group, there exists a certain instance in another group for every incoming and outgoing relation, as it is shown in figure 3.2. Therefore, it is not enough to have edges between vertices only, but to provide for every instance of these vertices a separate edge.

This is the case because there is no chance to see which instance is related to which instance, if there are only edges between groups of instances and not between single instances. For example, several soccer match instances are grouped together in one vertex. This vertex has a relation *"happens at"* to a group of date instances, also represented by one vertex. Then there is no way to tell which match happens at which date. To visualize this direct relation between single instances, we need to draw a separate edge for every pair, consisting of a match and the date it happened. In figure 3.4, every soccer match is directly connected to its date instance in vertex 2.

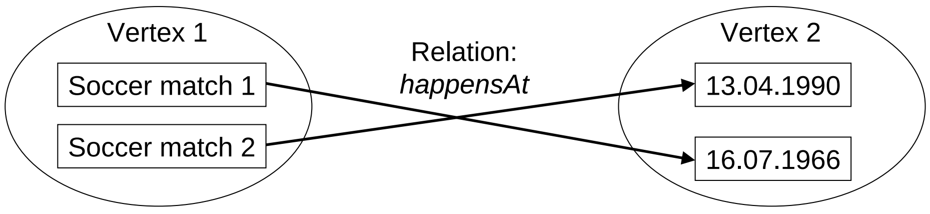


Figure 3.4: Visualization of a relation by edges for every pair of instances.

However, such edges between all instances are not practicable, because too many edges in the structure of the graph get confusing. This holds especially when visual space is limited as on the handheld client.

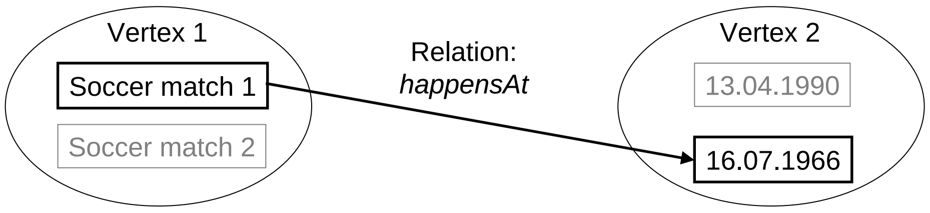


Figure 3.5: Visualization of a relation by an edge between only one active instance per vertex.

The idea is to offer only one active instance for every vertex. This one active instance of a vertex is the only visible instance of the group of instances, represented by this vertex. Because of this, we need to draw only one edge for every relation between two vertices. This edge represents only the relation between the two active instances of these vertices and therefore the active instances of all related vertices have to be consistent. This idea is shown in figure 3.5.

If, for example, the active date is *"16.07.1966"*, all the related vertices must have consistent active instances; like the active soccer match has to be exactly the one happens at this date.

Having only one active instance per vertex implies a mechanism to change the active instances. There must be a possibility to take another instance as active one out of every group of instances. The problem of the consistency between the active instances and the possibility to change them will be discussed in section 3.3.

### 3.2.2 Visualizing Many Edges

A problem by the visualization of the graph is the amount of outgoing relations. As it is shown in the exemplary input data, the RDF file in figure 1.3, a vertex can have many outgoing relations to other vertices. To visualize all of them as labeled edges in the graph would again lead to space problems. The more labeled edges are connected to one vertex, the harder is a visualization of these edges without overlapping.

To solve this problem of too many edges, we use the same strategy as we used for too many instances of the RDF graph, described in section 3.2.1. The idea is to have not all the outgoing edges of a vertex visible. Again, only a couple of all relations, available out of the RDF graph, are active, the rest is inactive. Only active relations are represented through edges in the graph visualization.

Having inactive relations invisible implies a mechanism to activate them. Otherwise the information of these inactive relations would be inaccessible for the user. The way to activate and deactivate relations over the graphical user interface will be described in section 3.3.2, about changing relations.

## 3.3 Interaction

In this section we are talking about the options for the user to interact with the graph. As mentioned in the last section, there must be a possibility to change the active instance for every vertex as well as a possibility to change the active relations.

### 3.3.1 Changing Instances

In our graph, the user can change the active instance for a vertex by using a roll out mechanism. By clicking on a roll out button, a menu rolls out and the user has the opportunity to define the active instance for the vertex by clicking on one of the instances in the roll out menu. For example, the user can change the active instance *“16.07.1966”* of a vertex to another date like *“29.06.1986”* (figure 3.6). After clicking on the inactive instance *“29.06.1986”*, this becomes the new active instance of that vertex. The old active instance *“16.07.1966”* becomes inactive and therefore only visible in the roll out menu.

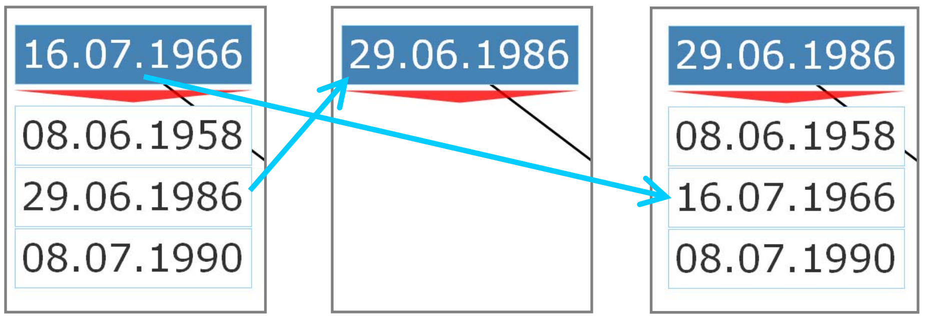


Figure 3.6: Roll out menu with several instances of the concept date. The user defines the instance *"29.06.1986"* as the new active instance for this vertex.

### Consistency Check

If the user changes the active instance of a vertex, there is a need to check whether all connected vertices and their active instances are still consistent (figure 3.7). The active instance of a vertex is called consistent, if all active incoming and outgoing relations are consistent. A relation is called consistent, if for all involved vertices, all their active instances are consistent.

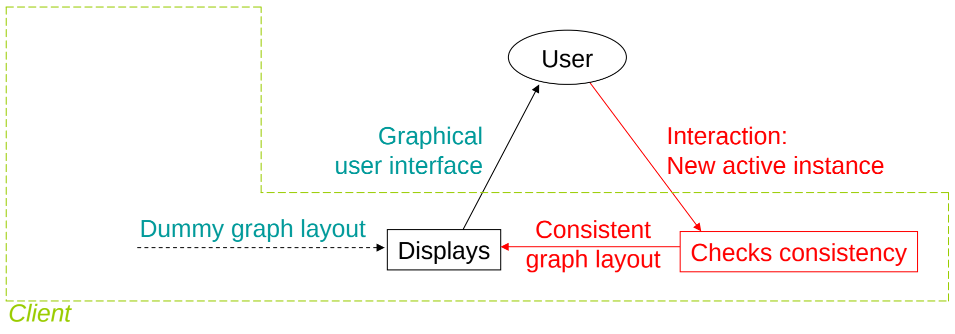


Figure 3.7: The data flow diagram extended by the consistency check.

To preserve a consistent graph, we need to implement a mechanism that runs through the graph and check all vertices whenever the user has changed an active instance. This mechanism is comparable to the link-state routing protocol [30]. Starting from its appearance the mechanism propagates the change done by the user in all directions through the topology. To avoid infinite loops and out-dated information due to propagation through the graph, we use two approaches.

Firstly, an approach to see whether new information is up-to-date by using a new version number for every change of an active instance. And secondly, an approach to solve the infinite propagation inside a loop, by using information about the distance to the source of the change. Whenever a change of an active instance is propagated through the graph, the distance of the propagated information is increased by every hop. The distance holds the information about the number of vertices, through which the change has already propagated. So at least by reaching the source of the change again, the distance of the propagated change is larger than the distance of the own information about this change and so the loop is broken.

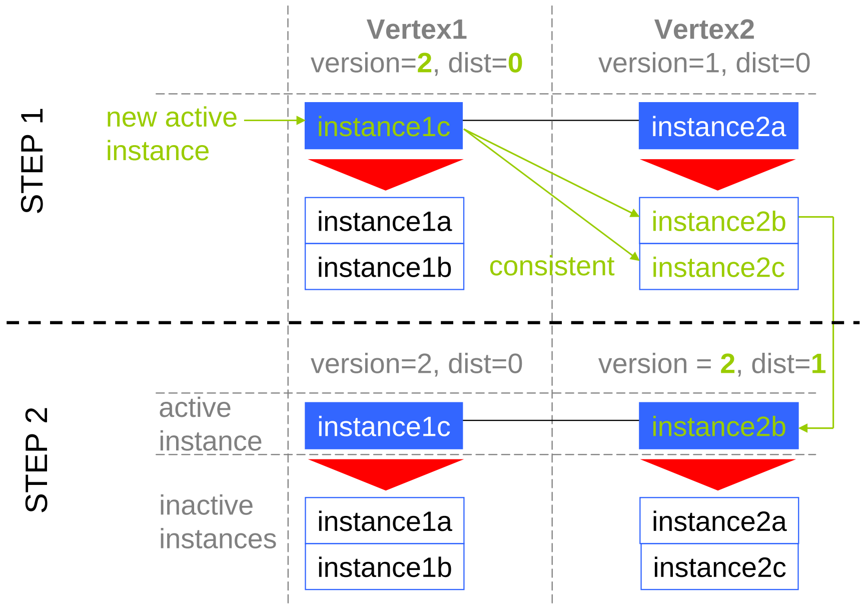


Figure 3.8: Every vertex in the graph propagates changes to related vertices until all vertices are consistent again.

Every change of the graph by the user will be propagated through the graph to hold consistency (figure 3.8). In Function 1, 2 and 3, we give a simplified version of the implementation. Every change is recursively propagated through the graph until all vertices are consistent again.

For the realization of the consistency check, every vertex needs to hold information about the last version of change and its distance to the source to this last change. Whenever a change is propagated to a vertex, the vertex has to compare the version and the distance of the propagated change with its own version and distance. Based on this information, every vertex handles its own update and the further propagation of this change.

If a change has a higher version (version1) as the last version saved by a vertex (version2), the vertex will update its version and distance, update its active instance if necessary and propagate the change further through the graph structure. If version1 of the incoming information is less than version2, vertex2 keeps its information and stops the further propagation.

Only if the versions are equal, the distance information is needed.

For the distance, if distance2, saved by vertex2, is higher as the propagated distance1, vertex2 is updated and the change is propagated. If distance2 is less or equal as distance1, there is a conjunction of the current consistent instances of vertex2 and the instances of vertex2 that are consistent with the current active instance of vertex1. Again, if this leads to an update of the active instance of vertex2, this is propagated through the graph.

**Function 1** propagate($newActiveInst$, $newVersion$, $newDistance$)

```
Require: The new active instance to be propagated as well as the new
    version and the distance to the source of the user interaction
 1: var $vertex1$:Vertex = $newActiveInst.getVertex()$;
 2: var $activeOutRelations$:Array = $vertex1.getActiveOutRelations()$;
 3: var $activeInRelations$:Array = $vertex1.getActiveInRelations()$;
 4: // Forward Propagation
 5: **while** $activeOutRelations.hasMoreElements()$ **do**
 6:    var $outRel$:Relation = $activeOutRelations.getNextElement()$;
 7:    var $consistentInstances$:Array = $inRel.getObjects(newActiveInst)$;

 8:    runUpdate($consistentInstances$, $newVersion$, $newDistance$);
 9: **end while**
10: // Backward Propagation
11: **while** $activeInRelations.hasMoreElements()$ **do**
12:    var $inRel$:Relation = $activeInRelations.getNextElement()$;
13:    var $consistentInstances$:Array = $inRel.getSubjects(newActiveInst)$;

14:    runUpdate($consistentInstances$, $newVersion$, $newDistance$);
15: **end while**
```

**Function 2** runUpdate(*consistentInstances*, *version*1, *distance*1)

**Require:** A list of consistent instances, the current version and distance

```
var $vertex2$:Vertex = $consistentInstances[0].getVertex()$;
var $version2$:Number = $vertex.getVersion()$;
var $distance2$:Number = $vertex.getDistance()$;
// if the new information has a higher version
if $version2 < version1$ then
  $vertex2.setVersion(version1)$;
  // we have to increase the distance at each step
  $vertex2.setDistance(distance1 + 1)$;
  commit($consistentInstances, vertex2$);
else if $version2 == version1$ then
  // if the distance of vertex1 is less
  if $distance2 > (distance1 + 1)$ then
    $vertex2.setDistance(distance1 + 1)$;
    commit($consistentInstances, vertex2$);
    // else their distances are equal or the vertex2 is closer
  else
    var $consistInst2$:Array = $vertex2.getConsistentInstances()$;
    var $list$:Array = conjunction($consistentInstances, consistInst2$);
    commit($list, vertex2$);
  end if
end if
return
```

**Function 3** commit(*instances*, *vertex*)

**Require:** A list of consistent *instances* and the *vertex* to be updated and committed

**Ensure:** The Array *instances* must not be empty

```
$vertex.setConsistentInstances(instances)$;
if $!instances.contains(vertex.getActiveInstance())$ then
  $vertex.setActiveInstance(instances[0])$;
  propagate($instances[0], vertex.getVersion(), vertex.getDistance()$);
end if
```

### 3.3.2 Changing Relations

Choosing which instance is the active one for a vertex is not the only way for the user to adapt the displayed information. As mentioned in section 3.2 about the visualization, the user needs also the possibility to control which relations are active and which are not (look at figure 3.9).

When a new graph structure is displayed the first time, it has an initial setting. In this initial setting, every vertex has an initial active instance and initial active relations to other vertices. Starting from this initial setting, the user can change it by choosing other active instances or relations. The decision, which relations and instances are active, defines what information is shown by the graph. Therefore we let the user control what information to show.

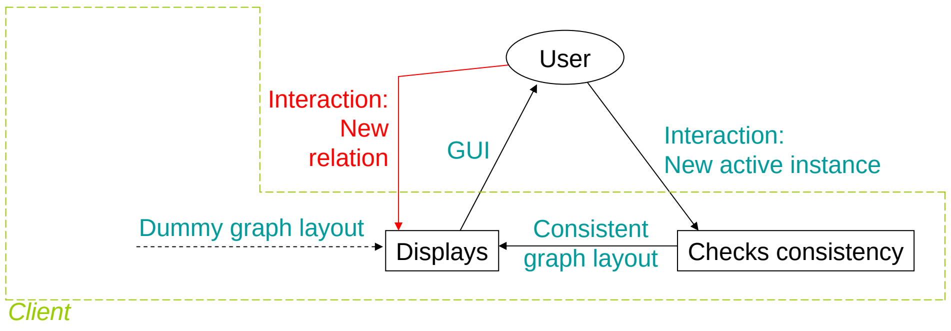


Figure 3.9: The data flow diagram extended by a new user interaction, the change of relations.

So, for every relation we need a labeled edge and a possibility to activate or deactivate it. One idea is to have a close button for every relation, for example an “X” on the right upper corner as in figure 3.10. By clicking the close button of a relation, this relation is no longer visible and so there is space to display other relations. All the inactive relations are grouped in a pool, represented by another vertex. This pool holds all the inactive relations of a vertex, or in other words, all the inactive outgoing relations of the concept of this vertex, and offers a way to select one for activation. By activation, this relation is taken out of the pool and displayed as a labeled edge between two vertices.

The first idea to visualize the interactive functionality by a closing button has advantages and some disadvantages. One advantage is the fact, that the user can organize whether a relation is displayed or not, as he likes. So it is possible, for example, to deactivate all existing relations of a vertex or to display all relations, even if they are overlapping. The disadvantage is that the organization is time-consuming. In average, two work steps are needed to activate a new relation. One step is needed to free space by closing an

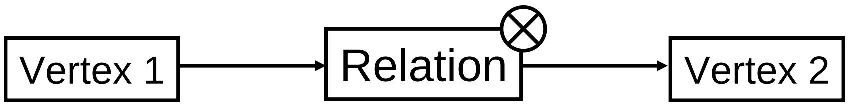


Figure 3.10: A relation vertex with a close button.

old relation and another to activate a new one.

In order to simplify the organization, we combine these two work steps into one. One relation will be replaced by another and thereby the two steps, to close the old and to open the new relation, become one and the same work step (figure 3.11). The advantage of this simplification is an easy handling due to the use of a roll out menu. The disadvantage is, that it is no longer possible to arrange as many relations as one likes. This is because every new relation is a replacement of an old relation and therefore the number of active relations remains always constant.

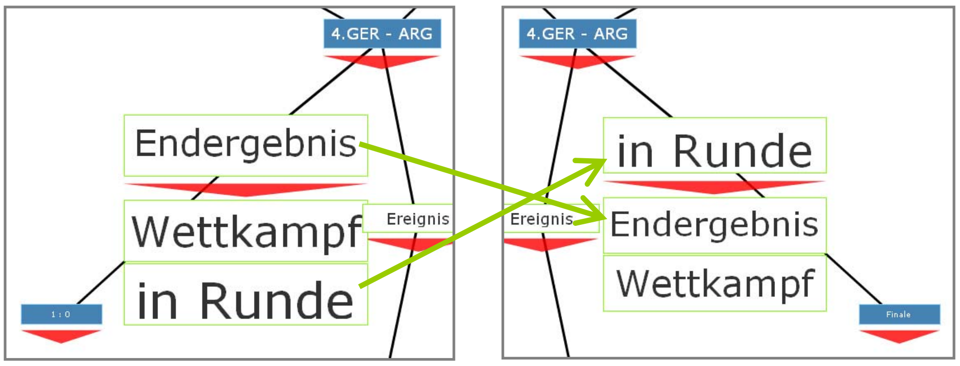


Figure 3.11: The active relation “Endergebnis” (final results) is replaced by the relation “in Runde” (in round).

In figure 3.10 and 3.11, a relation is represented by two edges and a vertex. The reason for not using just a labeled edge for the graphical representation of a relation is the extended interactive functionality. Every relation can be activated and deactivated. With this extension a relation is no longer just a labeled edge, but has become one element of the graph, a vertex. Now we have to kinds of vertices, a vertex representing an instance and a vertex representing a relation between instances. For a differentiation between these two kinds of vertices, we call the vertex representing a relation *rvertex*. A vertex representing an instance is still called vertex.

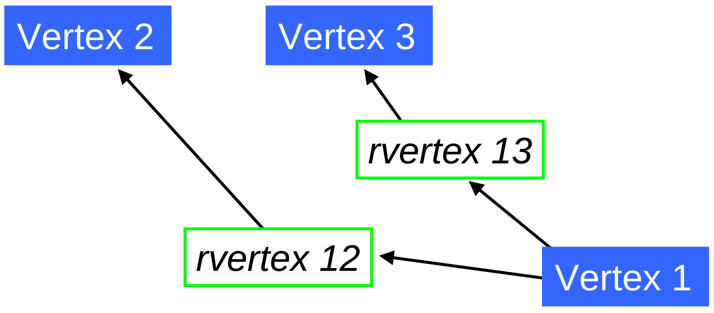


Figure 3.12: With the two articulated joints "*rvertex 12*" and "*rvertex 13*", complex structures become clearly arranged.

To make these two different kinds of vertices better distinguishable for the user, we use different colors. As shown in figure 3.12, *rvertices* have a green border and vertices are blue.

### Edge Label Placement

One reason to represent a relation by a vertex, a *rvertex*, is its extension of interactive functionality. Another reason to use a *rvertex* is, that it eases the problem of edge label placement.

The problem of automatic label placement is important [17], and has applications in many areas including Cartography [12], GIS[21] and Graph Drawing [15]. Unfortunately, even in simple settings, the problem turns out to be NP-hard [12, 19]. Using a *rvertex* instead of an edge label, simplifies the problem of automatic placement. This is due to the fact that dividing one edge into two edges simplifies the effort to avoid edge crossing, because the shape of the edge becomes more flexible (figure 3.12). Furthermore, with the edge label as a well-defined element of the graph, an *rvertex*, it is easier to deal with the overlapping of the labels.

The problem is reduced to the question, where to place the *rvertex*. The best position for a *rvertex* between two vertices is the straightest way without overlapping.

Before we take a look at how to organize the *rvertex* positions, we have to reconsider the given conditions. Firstly, there are two types of relations, outgoing and incoming ones. So by placing the *rvertices* we have to consider whether it represents an incoming or an outgoing relation. Secondly, not all available relations are active. Like described in the second idea about changing relations, there is always the same number of active relations for every vertex on the screen. This is because the user can only replace one *rvertex* by another one.

Taking these conditions into account we come to the idea that every ver-

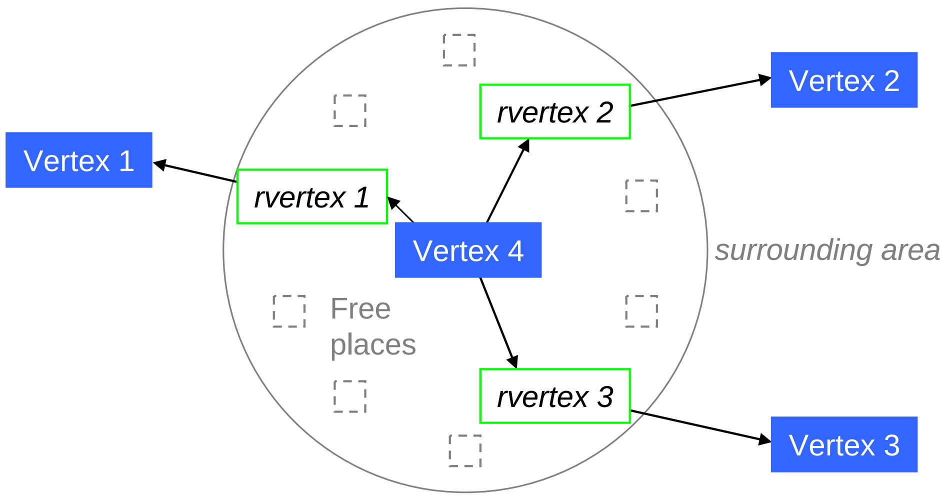


Figure 3.13: Every vertex organizes its outgoing relations, represented by *rvertices*, itself.

tex must organize its outgoing relations itself. Therefore every vertex gets some surrounding area for doing so (figure 3.13). Every representation of an active outgoing relation of a certain vertex is placed inside its surrounding area. Therefore every vertex organizes a ring of possible places for its *rvertices*. Such a possible place has either the status free or busy. If the user replaces an *rvertex* by another *rvertex*, the place of the old *rvertex* will be set to free. After freeing the old place, a new position for the new *rvertex* must be found. This happens by a computation of the closest distance. So we take the straightest connection available to the vertex to which the new *rvertex* is leading.

For every vertex we define 10 places, organized in a ring around this vertex (figure 3.13), available for outgoing relations. At any time there are for every vertex at most 4 outgoing relations active. Therefore there always exists a proper place for a new *rvertex*, because the number of free places is never less than 6.

## 3.4 Limited Space

The next problem to deal with is the limitation of space on the handheld client. As already mentioned in the task description, to find appropriate ways to deal with this problem is a key point of this work.

Displaying a graph structure, one question is: How do we get the positions for all the vertices, the graph layout?
We are going to discuss this position problem later in the section 5.5. As a first approximation, we use dummy graph layout. As described in section

3.1.2, we choose the positions for all the vertices in the dummy graph layout manually.

Assuming these positions do exist, another problem occurs. On a limited screen there is not enough space to display every graph in its full amount. If every label of every vertex is of a readable size, we will shortly be running out of space.

So the question is: How to display a more complex graph on a small screen?
If we view the graph in a way all labels can be read easily, the display will only show a part of the graph. So the user has no overview of the whole structure and therewith looses orientation. If we, however, try to display the whole graph on one screen, the labels will get unreadable small. If the labels get to small, the user will not be able to get information out of the graph.

> A sufficiently large set of data, or a sufficiently small display area, will force a change in perspective, or orientation, since the *"natural"* form of the structure cannot be displayed due to insufficient display resources [25].

### 3.4.1 Perspective Techniques

Two categories of resolution techniques exist, undistorted and distorted orientations [29]. These perspective techniques are used in conjunction with a particular visual representation to create a visual interface and can be used with a variety of other data structures in addition to hierarchies.

#### Undistorted Orientation

Undistorted orientation techniques display the information stored in a hierarchy in sections, thus restricting the display to show only a portion of the overall structure. These techniques are frequently used in conventional interfaces and can generally restrict the usability of a visual interface if not properly augmented with additional navigational information.

1. **Paging**: The simplest form of undistorted orientation is paging, which splits the hierarchical structure into a linear series of pages, similar to pages in a book. Each page can then be accessed by page numbers, or through a simple "next page, previous page" command.

2. **Scrolling**: Another popular form of undistorted orientation is scrolling. This technique creates a virtual display area large enough to display the entire hierarchy, and then maps this display to the real one. By manipulating the x and y coordinates of the virtual display, the entire hierarchy can be seen, one portion at a time.

### Distorted Orientation

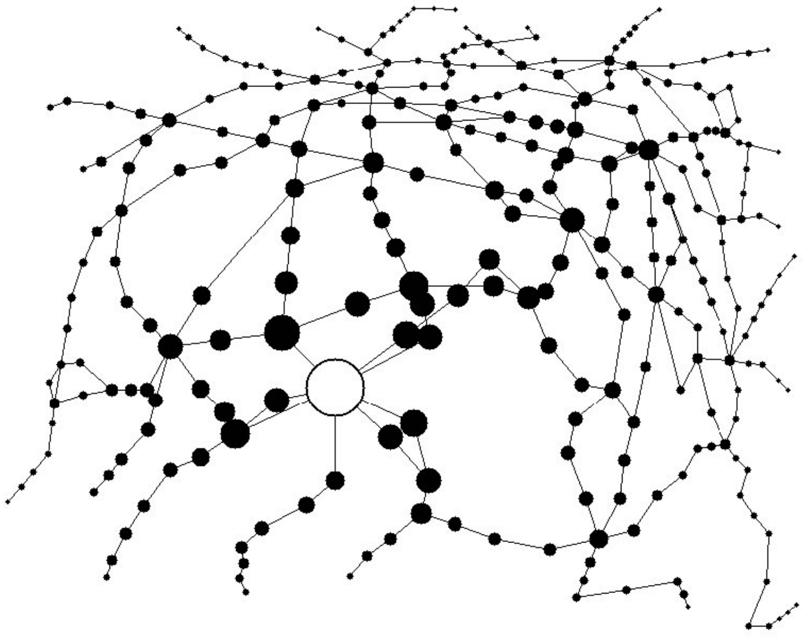


Figure 3.14: A graph structure distorted by a fisheye view.

The second category of orientation techniques modifies the natural view of the structure of the hierarchy by applying a mathematical transformation to the visual representation, so that a central area, or focus, dominates the display. The remaining areas of the structure are compressed, thus allowing most, if not all, of the original structure to be visible. These techniques are considered more advanced than their undistorted counterparts, since they attempt to map the entire hierarchical structure inside the display area and provide visual cues for both global and local information [29]. By analyzing the characteristics of the mathematical formula responsible for the visual transformation, a distorted orientation technique can be classified as either continuous or uncontinuous.

We only take a look on the continuous one, because non-continuous techniques fail to provide a smooth transition between the focus area and the context areas [25]. An example of continuous magnification transformation is a Fisheye View.

### 3.4.2 Fisheye View

The fundamental motivation of a fisheye strategy is to provide a balance of local detail and global context. In many contexts, humans often represent their own *"neighborhood"* in great detail, yet only major landmarks further away (figure 3.14). This suggests that such views ("fisheye views") might be useful for the computer display of large information structures like programs, data bases, online text etc. [20].

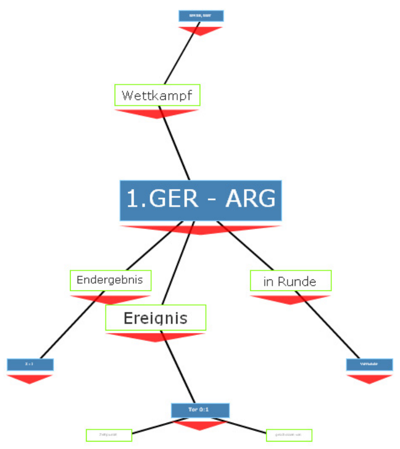


Figure 3.15: A distorted graph structure with “1.GER - ARG” as the current focus vertex.

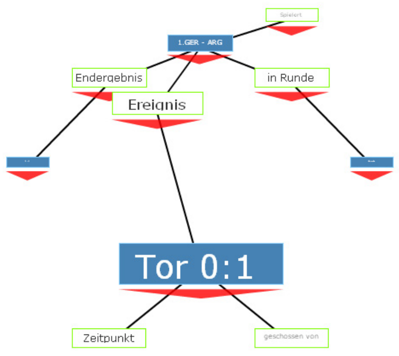


Figure 3.16: A distorted graph structure with “Tor 0:1” as the current focus vertex.

A fisheye camera lens is a very wide angle lens that magnifies nearby objects while shrinking distant objects. It is a valuable tool for seeing both local detail and global context simultaneously [49]. The Fisheye View is unique in the way that it uses a system of priorities and thresholds to determine what information should be presented and what information should be suppressed [23].

Using a fisheye view to distort our graph is a solution for the space problem on the display of the handheld client. The fisheye view makes it possible for the user to see both, local detail and global context simultaneously.

By using this fisheye approach to deal with the limited space, we add a new interaction form to our user interface. Additionally to the two possibilities of interaction, namely the change of relations and the change of active instances, the user can choose the focus for the fisheye view.

This focus interaction happens by clicking on a vertex or a *rvertex* in the graph. By choosing a new focus all positions and details change. For every vertex of the graph the position and the detail must be recomputed dependent on the distance to the new focus. Such a distortion of our graph-based user interface can be seen in figure 3.15 and 3.16. The integration of the fisheye distortion into the data flow of our graph presentation system is illustrated in figure 3.17.

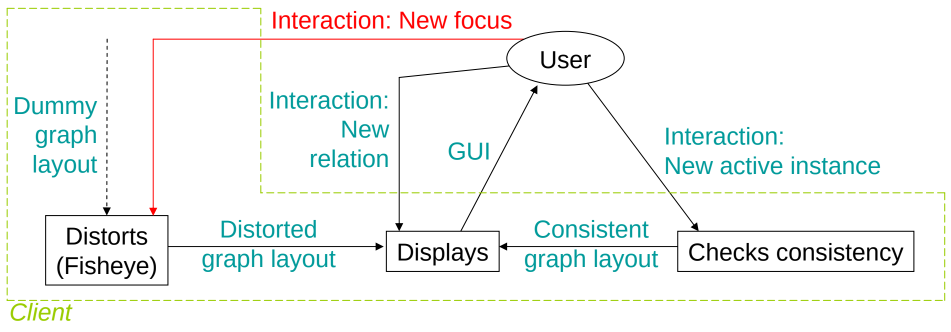


Figure 3.17: The data flow diagram extended by a new user interaction, the choose of the current focus vertex.

The user can click on vertices to get a detailed view on this region of the graph. If the user clicks on a vertex and sets the focus point, he shows an interest in this region. So clicking on a vertex is a kind of a request for detailed information about this vertex.

### Implementation of the Fisheye Distortion

In Function 4 and 5, the computation of a new fisheye position for a vertex is described. Intuitively, the position of a vertex in the fisheye view depends on its position in the normal view and its distance from the focus. Our implementation of the fisheye distortion is based on the algorithm described in [49].

**Function 4** computeFisheyePosition($x$:Number,$y$:Number,$focus$:Object)

**Require:** The undistorted coordinates $x$ and $y$ as well as the current $focus$

```
var $focusX$:Number = $focus.getX()$;
var $focusY$:Number = $focus.getY()$;
// dBoundX: Horizontal distance between focus and the boundary of the screen toward the vertex position
var $dBoundX$:Number = 0;
// dBoundY: Vertical distance to the boundary
var $dBoundY$:Number = 0;
if $focusX > x$ then
    $dBoundX = -focusX$;
else
    $dBoundX = screenWidth - focusX$;
end if
if $focusY > x$ then
    $dBoundY = -focusY$;
else
    $dBoundY = screenHeight - focusY$;
end if
// distX: Horizontal distance between the focus and the vertex
var $distX$:Number = $x - focusX$;
// distY: Vertical distance
var $distY$:Number = $y - focusY$;
// pos: The new focus position
var $pos$:Object = new Object();
// We divide distX by dBoundX so that the argument to transform() is normalized to be between 0 and 1
$pos.x$ = transform($distX/dBoundX$) $* dBoundX + focusX$;
$pos.y$ = transform($distY/dBoundY$) $* dBoundY + focusY$;
return $pos$;
```

**Function 5** transform($x$:Number)

**Require:** $x$ must be positive and $distortion > 0$ to change $x$

```
// transform is monotonically increasing and continuous for $0 \leq x \leq 1$
return $(((this.distortion + 1) * x) / (this.distortion * x + 1))$;
```

### Flashlight Approach

Initially, the information in the graph is limited to a certain result, for example the result to the question *"How was Germany playing against Argentina in a world championship?"*. The idea for further development is to offer the possibility to deliver potential additional information to the user. If the user reaches the limits of this initial information, the graph system should automatically request for additional information. By offering the user stubs of additional information, the user can choose on his own, which path to follow. That way the graph gets dynamically expanded.

In this first prototype of a graph presentation system, we are not dealing with dynamic data from the server, but with fixed dummy data, as described in section 3.1. The information represented by the graph is limited to this dummy data. When the user shows an interest in a certain vertex by clicking on it, the amount of local details available to display is limited to the available information out of the dummy data. For this reason we do not need a connection to a server, yet.

Anyway, in the further development of the semantic graph, the user should become able to really request for new information by clicking on a vertex. So the graph presentation system sends a request for more detailed information for this certain vertex and its surrounding area to the server and presents the response.

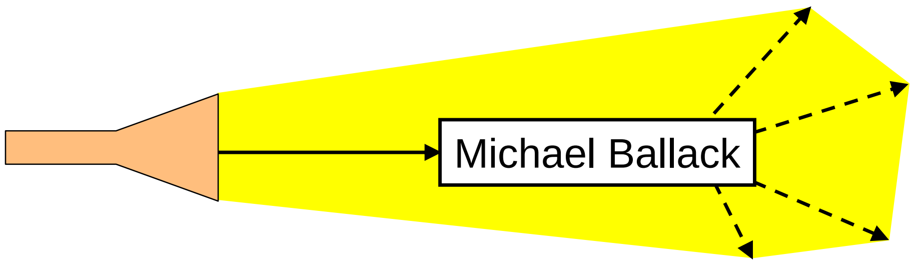


Figure 3.18: Using a flashlight is a metaphor for offering some stubs to reach more information in a specific direction.

This approach is comparable to the use of a flashlight. Pointing with a flashlight to certain spots of interest in a dark space is a way to discover more information. The dimly lit surrounding area provides an information basis of what could be a worthwhile point to spot. Like the dark silhouettes around, the stubs of additional information provide indications of worthwhile directions for information discovery. Using a flashlight is a metaphor for offering some stubs to reach more information in a specific direction. We call this approach the *flashlight approach.*

This *flashlight approach* is an alternative way to ask for more information. Pointing with the flashlight to certain spots in order to explore new information, is just like asking a question in order to receive an answer. The difference between these two ways is the fact that the flashlight only allows to explore information, which is connected to the graph, in contrast to a new question, which offers access to any kind of information. Using the *flashlight approach* for exploration, the user is locally bound to the available stubs of additional information in the current graph.

If we receive as initial result a couple of soccer matches, then an available stub is, for example, the vertex holding all the players that have scored the goals in these matches (figure 3.18), but without any additional information in terms of outgoing relations to other vertices. In case the user is interested in the players and clicks on this vertex, a request is sent to the server and the response is integrated into the graph. A response could consist of additional information like the team they are playing for or goals they have scored.

Together with the other forms of interaction, the change of instances (section 3.3.1), the change of relations (section 3.3.2) and the change of the focus vertex for the fisheye distortion (section 3.4.2), the *flashlight approach* offers a specific semantic browsing function. Using this interaction possibilities, the user becomes able to semantically navigate through the information space.

## 3.5 Semantic Navigation

Navigation can be differentiated into three types: Navigation by content-structures, free navigation and semantic navigation [5].

Navigation by content-structures is constituted by dependencies within a complex content that contains several sub-contents. Like a book containing several chapters, which are hierarchically ordered, content may consist of several sections or chapters. Users can switch from one chapter to another by clicking links.

Free navigation allows users to navigate without any implications for the semantic context or structural dependencies. This concept is needed to link external resources and to provide shortcuts in the information space.

The third and most comprehensive navigation structure is the semantic navigation. In [9], semantic navigation is defined as a way to build up and navigate views according to the logical organization given by ontologies.

The RDF graph, which lays the foundations of our graph-based user interface, has a logical structure given by ontologies [36]. It offers thereby a visual environment for browsing RDF models represented as graphs. Such an environment has already been developed in other applications, as it is described in section 1.4, about related work.

Even if already many visualization tools have been developed, there is no *"single view"* that will appeal to all the users and tasks requiring visualization. Therefore, many different ways to customize and extend the visualizations are needed [50].

In this diploma thesis we develop a visualization tool, offering the possibility for a semantic navigation on a handheld. As already mentioned, we get information as an answer for a question like *"How was Germany playing against Argentina in a world championship?"*. This result consists of information pieces, called instances, and semantic information about their concepts and relations. This semantic information is the logical organization of the instances. In our approach the user has the opportunity to both, build up views by changing the active relations and instances (look at section 3.3) and navigate views by moving the focus (look at section 3.4). With the definition of the active relations, the user controls the dimensions, which span the information space for a semantic navigation. For example, the information space is spanned by the dimensions *"Region"*, *"Time"*, and *"Product"* with their corresponding instances (e.g. *"France"* and *"Germany"* for the dimension *"Region"* and *"Milk"* and *"Cheese"* for the dimension *"Product"*).

Such an approach for a semantic navigation in an information space is used in the MoSeNa-Approach [5], too. However, in this approach the identification of dimensions like *"Region"* and *"Product"* must be specified beforehand by information systems engineers and not dynamically by the user. In the example concerning *"regions"*, *"products"* and *"time"* navigation structure in [5] is limited to these three specified dimensions. For example, the user could have the possibility to navigate along different regions producing the same product, maybe milk. If there are several regions producing milk, the system offers a semantic navigation through all of them. So the user gets the possibility to semantically explore the data along these three dimensions.

By defining a fixed dimension space beforehand and regardless of the structure and the relations of the current content, the system looses important dimensions for a semantic navigation. Therefore the approach in [5] offers, for example, no possibility to show the semantic relation between products of the same company, even if this information is available. So maybe milk is produced in several regions, but only by two companies. In this system the user has no influence on the question, what dimensions are available for a semantic navigation, and therefore, in the example, no opportunity to navigate through all places of production for only one of these two companies. The different perspectives are limited by the beforehand defined dimensions and not by the user or by the possible relations inside the data.

In contrast to the approach described in [5], in this thesis we take a more flexible approach. In this way the full advantages of semantic relations are delivered up to the graphical user interface. To give an example, we go back to the matches between Argentina and Germany and take a look at the dimensions available at this data.

Firstly, there are four soccer match instances grouped in one vertex.

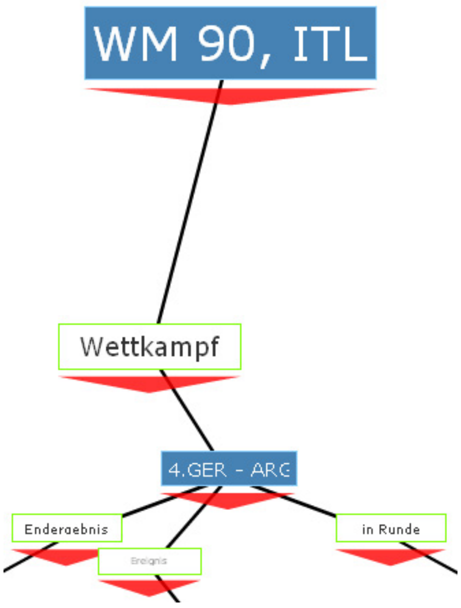


Figure 3.19: The graph from the point of view of information about certain world championships.

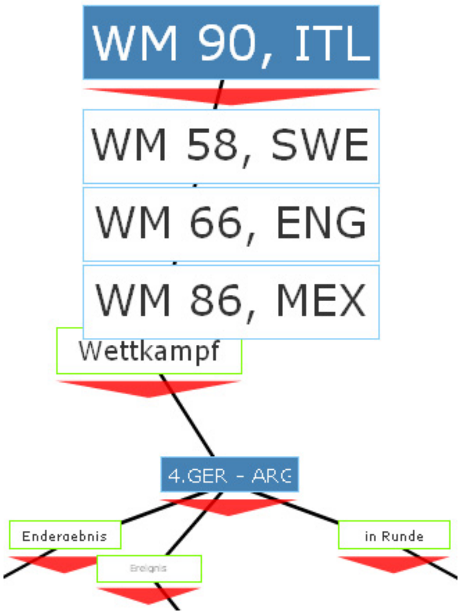


Figure 3.20: Changing the active tournament leads to updates of active instances throughout the graph.

The active instance for this vertex is the first match between Germany and Argentina during the world championship 1958.

Secondly, there are lots of relations to other vertices, also groups of instances, like dates, results and so on. The user can decide whether a relation is of interest or not (look at section 3.3). Every relation is a dimension spanning the information space. The relation *"in tournament"*, for example, spans a new dimension and therefore a new point of view. If the user is interested in all matches in a certain tournament, then he can take this point of view by just activating this relation. After activation, the user has the opportunity to get all soccer matches for a certain championship (figure 3.19). This is possible because of the consistency check in section 3.3.1. Consistency means, that starting from the user interaction all connected vertices must hold a consitent active instance. In other words, if the user changes an instance of a vertex, the whole graph gets updated in a way that all relations are consistent (figure 3.20). After such an update, the graph shows a new point of view. In our example the graph shows all information under the perspective of a certain world championship.

In this approach, the structure of the information defines the dimensions spanning the space for the semantic navigation. For every concept all possible relations to other concepts are graphically represented. Even if only few can be active at the same time, all can be activated by the user. And even though only the active relations build the current dimensions for a semantic navigation, all the inactive relations can be chosen as dimensions, too. Therefore the amount of possible dimensions is huge and the user can decide how to span the information space.

To sum up, the difference between our approach and the approach, described in [5], is the flexible possibility of semantic navigation for the user. This is possible by building the dimensions for the semantic navigation dynamically depending on the information structure, instead of defining a fixed amount of dimensions for the semantic navigation. This way the possibilities for a semantic navigation are extracted from the current information structure and not from a structure defined beforehand by information systems engineers.

# Chapter 4

# First Evaluation

After a first prototype has been developed, we need to get a first feedback from the users. This first feedback can be used to verify the decisions made so far and to get interesting impulses for further development steps.

Software ergonomic methods have to be included in the design process of software from the beginning. It is still necessary though, that the design process also contains an evaluation after each stage of accomplishing the prototype.

## 4.1 Evaluation Framework

In this section we define the framework for the evaluation. Firstly, we shortly describe again our prototype, the program that will be evaluated. Secondly, we state the program implementation objectives in measurable terms. Thirdly, the evaluation questions are formulated and finally we describe the context of the evaluation. This procedure will ensure that the evaluation reflects the program's goals and objectives.

### 4.1.1 First Prototype

We are going to evaluate the first prototype of our graph-based user interface at its current state of implementation. This includes everything mentioned in chapter 3.

So we have a graphical interface, where semantic relations and concepts are visually represented by a dynamic and interactive graph structure. This graph structure is distorted due to a fisheye view. The user has the opportunity to set the focus of the fisheye view to gain detailed information. Furthermore, he can change the content and modify the structure of the semantic graph by changing the active relations and instances (look at section 3.3).

Because the integration into the SmartWeb system is not yet completed, the content of the graph is limited to a fixed set of dummy information. This dummy information consists of the answer to the question "How was Germany playing against Argentina in a world championship?". The user has to imagine he would have asked this question and is interested in the given answer represented by the interactive graph displayed on the handheld client.

The answer consists of four soccer matches with additional data about the dates, goals, players and locations. The user can navigate through this information and interact with the graph in the above mentioned way.

### 4.1.2 Implementation Objectives

As a next step we decide what should get evaluated, because not much insight can be gained from the answers to a question like *"Is it a good system or not?"*.

Therefore we formulate some implementation objectives in measurable terms. This is to determine the kinds of information we need and avoid the problem of collecting more information than is actually necessary.

The four implementation objectives are:

1. **Interaction**: The possibilities to interact with the graph are easily to understand.

2. **Information Extraction**: It is possible to extract information out of the graph structure and the vertex labels.

3. **Differentiation**: The user gets aware of the difference between an instance and a relation.

4. **Dependencies**: The user realizes the dependencies between related active instances.

The idea is to present the user the dummy graph about the soccer matches between Germany and Argentina and ask questions about the information represented by the graph. If the representation is good and the means of interaction are well designed and easy to understand, the answers to these questions help to verify these objectives.

### 4.1.3 Evaluation Questions

Questions to be addressed in the evaluation concern two areas. The first area deals with questions about the implementation objectives and their achievement. Associated with this is the question about the barriers or problems that have been encountered.

The second area focuses on whether evaluation results vary as a function of characteristics of the participants or not. So the questions is: Did some participants change more than others and, if so, what explains this difference? One explanation could be, for example, characteristics of participants like the level of their computer skills.

### 4.1.4 Context of the Evaluation

This first evaluation took place during the CV-Tag 2006 at the Campus Koblenz of the University Koblenz-Landau. During this event, several projects were presented by students. In this context we presented our first prototype, too. This CV-Tag was a good option to attract potential participants with different background.

So during this event we presented our first prototype to visitors and asked to take part in its evaluation. Our prototype, implemented on a handheld client, was given to the participants for the evaluation.

## 4.2 Evaluation Procedure

This section provides detailed descriptions of the practices and procedures that are used to answer evaluation questions pertaining to our implementation objectives.

First, we identify sources of evaluation information and then we describe in detail for every implementation objective the information needed to determine if it is being attained and the method to collect this information.

### 4.2.1 Sources of Information

The most common method to obtain evaluation information is the survey of system properties with an interview or questionnaire [37]. In addition to a written questionnaire afterwards, we use own observations while the user handles certain tasks during the evaluation. The user has to fulfill these tasks, upon which he then gains experience. This method is most suitable, if a prototype is available and if potential members of the target group agree in participating [32].

The questionnaire, which can be found in the appendix, consists of nine questions about the usability and the usefulness of the graph-based interface. Additionally we ask about particulars, like the age and gender, and about the knowledge of computers, particularly of PDAs. These questions are statements like *"Navigation through the graph is a good way to collect information"* and the user has to rate them. Rating those statements means, the user has the possibility to agree with the statement, stay neutral or disagree with it.

### 4.2.2 Information Needed and Methods Used

For the four implementation objectives, several kinds of information are needed. In the following we formulate for each objective the information needed to determine if it is being attained and to assess barriers and facilitators. In addition we specify how this information will be collected (the instruments and procedures) and who will collect it.

1. Type of information needed and methods used to collect them for the first objective, concerning the **Interaction**:

   (a) We need information if and how participants use the interaction forms with the graph. Interaction forms are the click on a vertex to set the focus for the fisheye view and the click on a rollout button to open a rollout menu.

   So we need information whether the user understands and uses the forms of interaction with the graph or not.

   (b) The information needed for the first objective is collected by observations. The person performing the evaluation needs to observe, if the participant realizes the possibilities to interact with the graph and uses this interaction forms knowingly and goal orientated.

   Additionally, several questions about the interaction forms and their usability are asked in the questionnaire.

2. Type of information needed and methods used for the second objective, about **Information Extraction**:

   (a) Here we need information about how participants extract information out of the graph. Thereby it is important to determine the quality of the extracted information. We have to check, if this information is correct and whether it was easy to extract it out of the graph or not.

   (b) To collect the information to determine the second objective we use a combination of concrete questions in the questionnaire afterwards and a certain task to be performed by the user. The task is:

   - Find an arbitrary result for a soccer match between Germany and Argentina.

   Solving this task is only possible, if the user combines two separated information out of the graph to a new one. Therefore the user just has to look at the graph and does not need to change anything, because the relation *"result"* is initially active. He has to understand, that the three connected vertices *"GER - ARG"*,

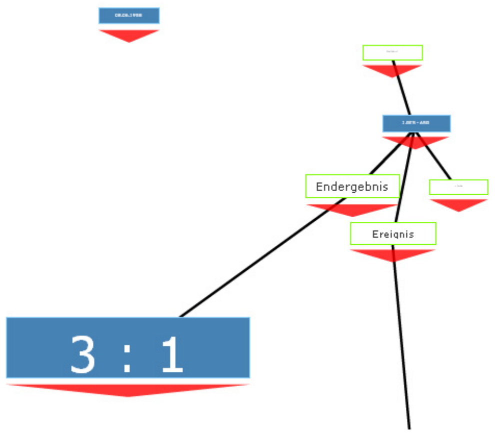


Figure 4.1: Out of this graph the user needs to get the information to name an arbitrary result for a soccer match between Germany and Argentina.

*"3:1"* and *"result"* together form the answer for this task (figure 4.1).
The given answer to this task serves for the determination of the second objective, **Information Extraction** out of the graph structure.

3. For the third objective, the **Differentiation** between an instance and a relation, we need the following:

   (a) Firstly, information about how the participants employ the graph. Secondly, information whether they are able to distinguish between a relation and an instance. And thirdly, in case they are able to distinguish, do they interpret the two different kinds of vertices correctly.

   (b) The special task to determine the third objective is to get the correct answer to the question:

      - Find a concrete date for an arbitrary soccer match between Germany and Argentina.

      This task demands the activation of an initially inactive relation, namely *"happens at"*. Therefore the user needs to understand the meaning of a relation as well as the meaning of an instance and how to activate a new relation.
      The quality and correctness of the given answer is the information basis for the validation of the third objective.

4. And finally the required information and useful methods for the fourth objective, to realize the **Dependencies** between related active instances:

(a) For this objective we need information about the use of the roll out menus. This holds for a change of an active instance as well as for the change of an active relation.

Of special interest is the way a participant uses such a change to gain new information. Does he realize the update of the graph due to such a change?

(b) The special task for the last objective is:

- Find the result of the soccer match between Germany and Argentina in the world championship 1966.

This is the most challenging task for the user. For the right answer, the user has to change the active instance of the vertex with all Championships to *"WM 66, ENG"* and then he has to activate the right relations to get the information about the result. Further the user needs an understanding of the dependencies between the active instances. This means, he needs to understand that a change of an active instance can lead to further changes in the graph due to propagation. This is described in section 3.3.1 about the consistency check.

So collecting the information of the answer to this given task gives us a possibility to assess the forth objective, whether the user realizes the **Dependencies** between related active instances.

## 4.3 Evaluation Results

In total twenty participants were interviewed during the first evaluation. The computer skills of the twenty participants range from inexperienced to professional and are uniformly distributed. The same holds for the distribution of the gender, there is almost the same number of men and women. The distribution of computer skills is shown in figure 4.2. The average age of the twenty participants is 26.6 years, spreading between 43 and 21 years.

In the following we present the information we collected in order to determine the four implementation objectives, introduced in section 4.1.2. The data involves tabulating frequencies and classifying free text comments into meaningful categories.

### Results Regarding Interaction

The results regarding the first objective, **the possibilities to interact with the graph are easily to understand**:

This does not hold in case of 42 percent of the participants (figure 4.3). The most influential information for this high number of negative feedback

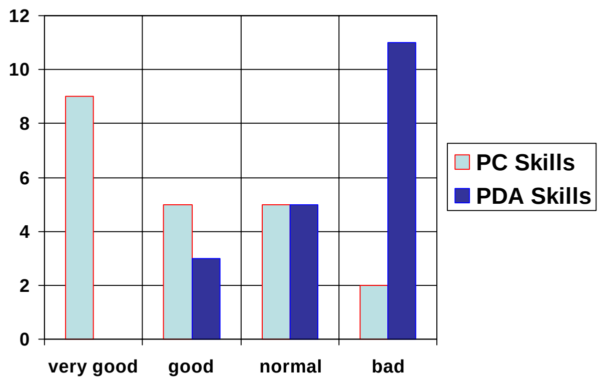


Figure 4.2: The data out of the questionnaire about the computer skills of the participants.

were the answers to the questionnaire statement: *"The interaction is difficult"*. 73 percent agreed with this negative statement in the questionnaire. Comments were, for example:

1. The reaction time of the system is too slow.
2. The area available to click on is too small.

The second most influential information for the 58 percent of negative feedback regarded the answers to the statement: *"The behavior of the graph is hard to predict"*. 63 percent of the users agreed with this statement in the questionnaire.

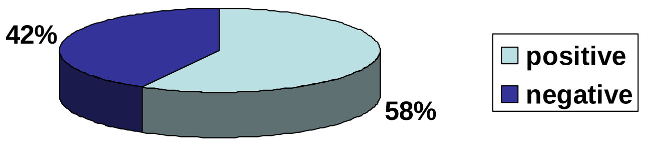


Figure 4.3: 42 percent of all participants did not agree with the first objective: The possibilities to interact with the graph are easily to understand.

### Results Regarding Information Extraction

The results regarding the second objective, **it is possible to extract information out of the graph structure and the vertex labels**:

Only 25 percent of the participants failed to reach this objective. An interesting point is here to distinguish between the result for participants with good computer skills and participants with bad skills. The negative percentage rises to 59 for the group with less computer skills and drops to 10 for the professionals (figure 4.4).

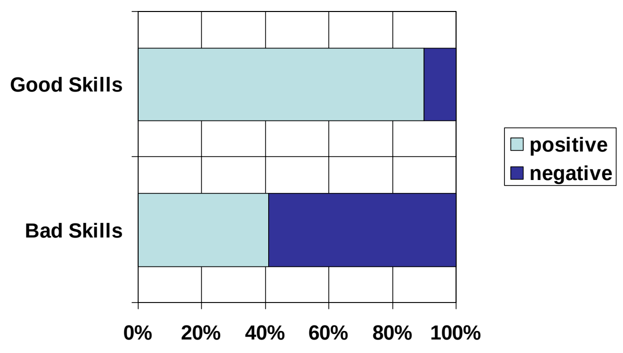


Figure 4.4: 59 percent of all participants with bad computer skills did not match with the objective: It is possible to extract information out of the graph structure and the vertex labels.

These 59 percent negative feedback needs some explanation, so we take a look at the information of this group of participants. The two explanations given by the participants are:

1. The labels are too small and therefore too hard to read.
   62 percent agreed with this.

2. To extract the required information out of the graph is difficult.
   Half of the users agreed, 52 percent. The comments were that the vertices around the focus vertex should get enlarged. So the surrounding area is too small to read the labels.

### Results Regarding Differentiation

The results regarding the third objective, **the user gets aware of the difference between an instance and a relation**:

Here the percentage of negative feedbacks is low. Only 15 percent did not agree with it. This time we take a look at the information from the observation. While monitoring the users, it seems that they were not always able to interpret the edges between two vertices as a semantic connection. Therefore they tried to get the information about the date of a soccer match from the relation *"happens at"* itself and not from the related instance (figure 4.5 and 4.6).

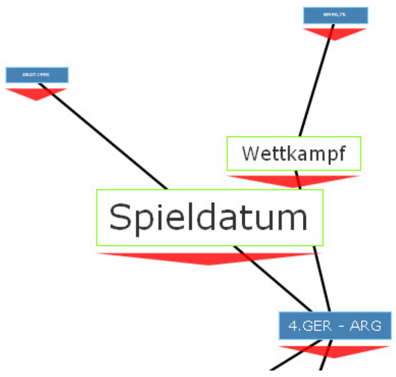


Figure 4.5: If the relation "Spieldatum" (happens at) is the current focus, it is almost impossible to read the date in the upper left corner, because the vertex is too small.

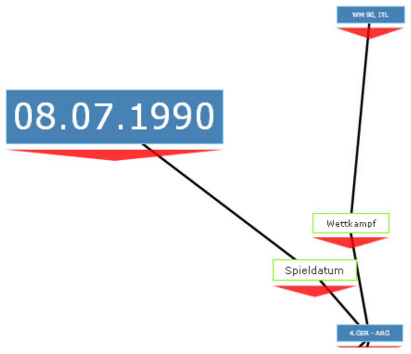


Figure 4.6: If the date itself has the focus, the connected relations are too tiny for visual inspection.

### Results Regarding Dependencies

The results regarding the fourth objective, **the user realizes the dependencies between related active instances**:

This last objective gained 45 percent negative feedback. If we take a look again at the two groups of participants divided by the computer skills, there is a difference of almost 30 percent between those groups. The low skilled group reaches 67 percent negative feedback (figure 4.7) and the professional group only 38 percent.

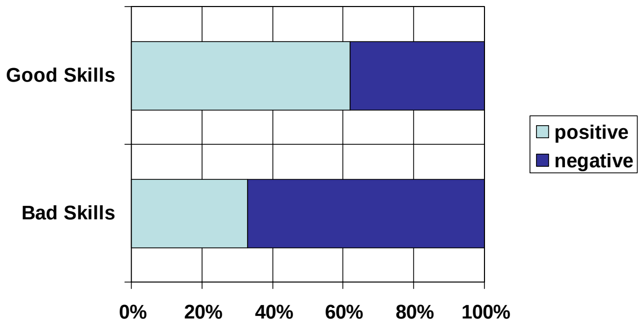


Figure 4.7: 67 percent of all participants with low computer skills disagreed with the objective: The user realizes the dependencies between related active instances.

This was mainly, because the users did not realize the change of the active instances due to a change of a certain active instance in the graph. So the dependencies between related instances were not realized.

### 4.3.1 Insights Gained From Free-Text User Input

After filling in the questionnaire the participants were asked to write down some general remarks on the prototype. To give an impression of the written answers, here are some samples of the positive and the negative remarks.

The headline for the first free text area on the back side of the questionnaire is: *"What did I like especially about this interaction form?"*. Positive remarks were:

- All information is displayed with regard to its context
- The representation through a graph is good
- The amount of information is visible as well as what information leads to what information
- The animation helps to understand the interrelations
- New and efficient metaphor to find information
- Small gadget

The second free text area on the back side has the headline: *"What needs immediate change?"*, and negative remarks were:

- The font is to small
- The roll out button should be easier to recognize
- A change in the graph should be more obvious
- The reaction time is to slow
- The labels are sometimes misleading
- A help tool is needed to explain the functionalities
- The interaction form using the pen and the touch screen is difficult, especially because the areas to click are to small for the pen

These remarks match with the results of the evaluation in total. The results show, that our prototype still contains several obscurities.

On one hand the visualization has several weak points. First and foremost these are the readability of the labels and the visual effects for changes in the graph content as well as the graph structure.

On the other hand, our graph prototype is only able to represent the semantic structure of the answer to one certain question. So it is not possible to represent arbitrary answers through the semantic graph-based interface, yet. On this account the user cannot explore new information beyond the dummy data, because the integration of new information into the graph is also not implemented, yet.

In the first section (section 5.1) of the next chapter, we will discuss the consequences arising out of the first evaluation.

# Chapter 5

# Further Development

After the first evaluation, we are now going to draw the consequences out of the evaluation results. If our analysis has been correct, we will be able to find the right ways to improve our graph presentation system.

## 5.1 Consequences Arising out of the First Evaluation

For every implementation objective we can look at the evaluation results to see whether it is being attained or not. Additionally we get information about factors that were barriers to attain this objective. Out of these factors we draw consequences by identifying areas in which changes may be needed for future implementation.

#### Consequences Regarding Interaction

The barrier to attain the first objective, **the possibilities to interact with the graph are easily to understand**, consists mainly of three problems. The area available to click on is too small, the reaction time of the system is too slow and the behavior of the graph is hard to predict.

An idea to solve the first problem is to increase the clicking area for the buttons. This should make the interaction with the graph easier.

The problem regarding the reaction time is more complicated to deal with. Due to the use of Flash for the graphical interface in combination with a handheld client, the system needs some time to compute the graph movements. Even if we try to optimize these computations on the handheld client, the animation of the graph will remain a complex operation.

A way to reduce this complexity is to hold the number of visible vertices as small as possible. Holding the number small, reduces the amount of computation to animate the movements of the whole graph. Every visible part of the graph moves visually animated from its old screen position to its new one, accordingly to the current focus position. Until now, all the vertices inside of a certain distance to the focus vertex are visible. Even if the vertex has no active relation to another vertex at all, as shown in figure 5.1. There is no management of the visibility of vertices as well as no organization of vertex positions, yet.

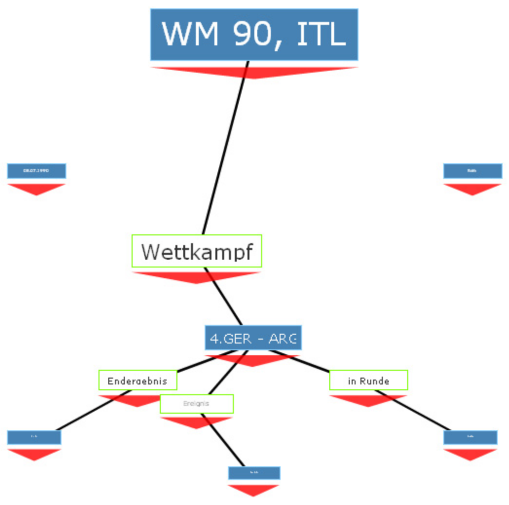


Figure 5.1: Two loose vertices, left and right, without any connection to other vertices, but still visible.

Organizing the visibility of vertices dynamically will decrease the amount of computation for the handheld client. We need such a dynamic organization anyway, in case of representing arbitrary graph data.

The third problem to reach the first objective is the difficultly to understand the behavior of the graph. A response therefore could be to highlight the current focus vertex. In the first prototype exists no visual mark for the focus vertex except for the distortion due to the fisheye view. Marking the focus in an additional way could make the interpretation of the distortion easier and thereby the understanding of the behavior in general.

### Consequences Regarding Information Extraction

The users' comments, most closely associated with the second objective, **it is possible to extract information out of the graph structure and the vertex labels**, are attributable to the problems encountered in implementing this objective. The comments were:

- The labels are too small and therefore too hard to read.
- To extract the required information out of the graph is difficult.

A solution could be just to increase the text size of all vertex labels. By increasing the labels in equal measure in addition to the fisheye distortion described in section 3.4.2, which increases vertices depending on their distance to the focus vertex, the readability would be improved. However, we have to be careful with this, because the space for increasing all labels is not available on the handheld client. We will think about a refinement of the fisheye distortion to improve the readability in section 5.2.2.

Another solution for the problems to accomplish an easy information extraction is to hide all vertices that have no connection at all to the focus. Because, if there are vertices visible on the screen, not connected at all, it is confusing to the user (figure 5.1).

#### Consequences Regarding Differentiation

The analysis of the barriers for the third objective, **the user gets aware of the difference between an instance and a relation**, leads to the judgment, that the vertex, to which a relation is leading to, has no sufficient visibility. Therefore, the idea of a refinement of the fisheye distortion, to increase those vertices of current interest to the user, seems to be a solution for this problem, too. If the focus lies on a relation, then the vertex, this relation is leading to, has to be increased. In case the focus lies on a normal vertex all the outgoing relations have to be increased. As a consequence the user possibly recognizes the related information more likely because it is more visible.

#### Consequences Regarding Dependencies

For the fourth objective, **the user realizes the dependencies between related active instances**, we found that the user did not recognize these dependencies, maybe because the visual effects are too tiny. An idea to counter this is a better visual response to an actualization of instances due to a change of a certain active instance in the graph. Whenever a vertex gets a new active instance due to an actualization, it could flash to signalize this change.

Another idea is not to highlight the updated vertices, but all the edges leading to them. With such a highlighting, the user would become able to even see the way of the update through the graph.

## 5.2 Visual Improvements

The evaluation of the prototype reveals that a visual improvement of the interface is needed. The analysis of the output information has clearly identified the areas in which changes will be needed.

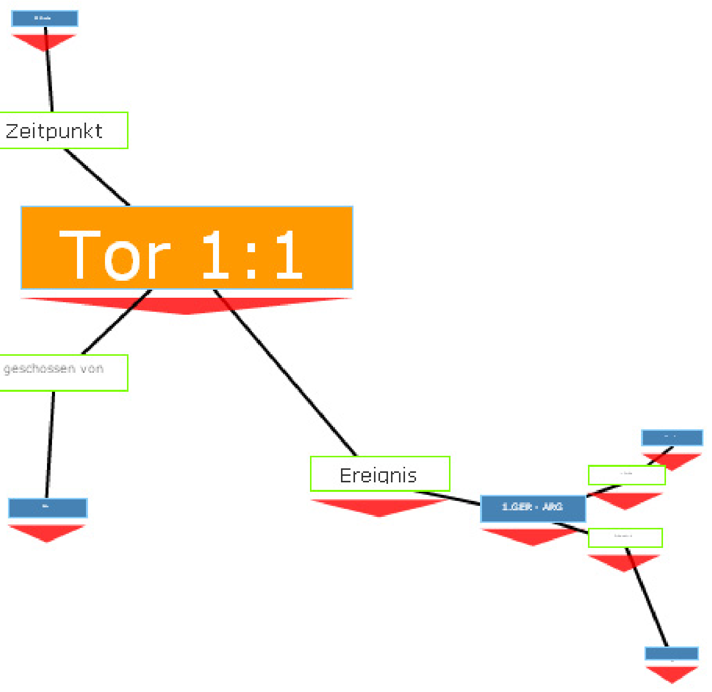


Figure 5.2: The focus vertex has a special color to be more visible.

The first area is the support of visual perception. The graphical representation has to highlight changes and point out important information in general.

One of such important information is the position of the focus vertex. The first improvement is to increase the visual effect of the focus by a certain color, as we can see in figure 5.2. Additionally to the increase of text size due to the fisheye distortion, the extra color helps the user to identify the focus vertex and thereby to understand the behavior of the whole graph.

Another important information is the update of active instances due to a change of a certain active instance by the user. Whenever the user changes the active instance of a vertex, the system runs a consistency check through the whole graph, to check whether the related vertices have to update their active instances, too. Highlighting every updated vertex due to a change of its color makes every update more visible to the user. Such highlighted vertices are shown in figure 5.3.

Along with this coloring of updated instances comes another idea. At the present state of visual improvements, the coloring affects only the active instances of updated vertices. As described in section 3.3.1, the user can change the active instance for a vertex due to a roll out mechanism that makes the inactive instances become visible in a roll menu. So the idea

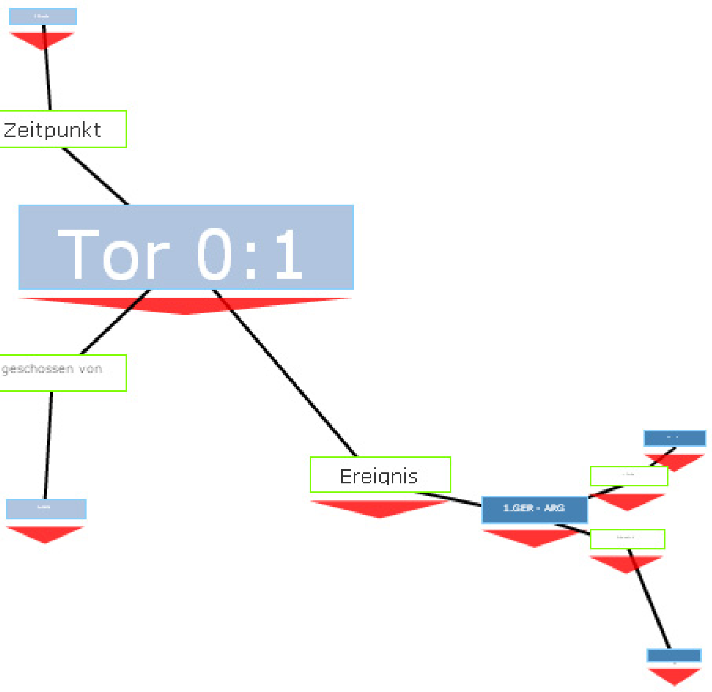


Figure 5.3: The user has changed the active scored goal from *"Tor 1:1"* to *"Tor 0:1"*. All the updated vertices are marked with a special color, here light blue.

is to deliver additional information about the consistency by coloring also inactive instances. To gain this additional consistency information about inactive instances, the consistency check needs a refinement.

## 5.2.1 Refinement of the Consistency Check

As described in section 3.3.1, all instances are related to other instances, no matter whether they are active or inactive. The difference is that the active instances are permanently visible, while the inactive instances are only temporary visible in the roll out menus. So the user can see the relationships of the active instances only. For example, the user can see for the active soccer match, the date it happened at, *"08.06.1958"*, and one of the goals, scored at this active soccer match, *"Tor 0:1"*.

If the user changes the active soccer match, the consistency of all the related vertices and their active instances have to be checked, as described in section 3.3.1. So for our example, if the active date instance *"08.06.1958"* is no longer consistent with the active soccer match, it has to be updated. The same holds for the active goal instance *"Tor 0:1"*.

The consistency check, in the form it was implemented in section 3.3.1, changes the active instance of a vertex, if it is not consistent anymore. Whenever an active instance changes, it is highlighted by a change of its color, as described in section 5.2. However, the primary consistency check is only responsible for a consistent active instance per vertex and thereby provides no information about the inactive instances and their consistency. To provide information whether inactive instances are also consistent or not, we need a refinement of the consistency check. This information can then be represented by the color of the inactive instances.

### Consistency of Inactive Instances

In the primary consistency check, implemented in section 3.3.1, we defined consistency in the following way:

- The active instance of a vertex is called consistent, if all active incoming and outgoing relations are consistent.

Because this first definition concerns only active instances and their consistency, a second definition for inactive instances and their consistency is needed. We define the consistency of inactive instances in the following way:

- An inactive instance is called consistent, if it has a connection to a consistent instance for all incoming relations.

Two things have to be pointed out, firstly, this second definition is only valid for inactive instances and secondly, the consistency of inactive instances depends only on incoming relations.

The most important consequence of the implementation of this second definition is the division of inactive instances into consistent instances and inconsistent instances. By visualizing these two groups differently, the user gains the information whether the change to another instance as the active one, causes a change of related instances or not. It is thereby possible to predict the change of the graph content, just by looking at the color of the inactive instances.

In figure 5.4, in STEP 1, the user clicks the inactive instance *"2.GER-ARG"*. Just by looking at the color of *"2.GER-ARG"*, he knows that this inactive instance is inconsistent and thereby causes a change of related instances. In STEP 2, after the refined consistency check, all the goals, scored by the new active match *"2.GER-ARG"* are marked with white to show their consistency. As a consequence of the goal *"Tor 1:1"* being consistent, the player *"Völler"*, which has scored it, is marked consistent, too.

The distinction between consistent and inconsistent inactive instances offers the possibility for a further visual improvement. The idea is to show for certain vertices only consistent inactive instances in the roll out menu and hide the inconsistent inactive instances. This is particularly suitable for vertices where the number of inactive instances is too huge to be properly

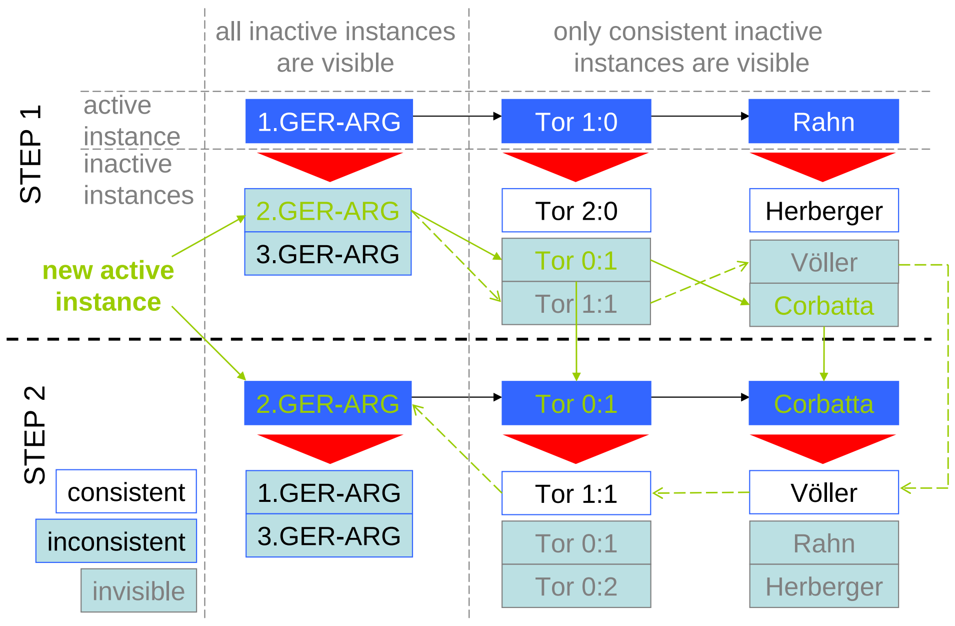


Figure 5.4: The propagation and visualization of the consistency of inactive instances, possible due to the refinement of the consistency check.

displayed, for example the goals scored in all the matches between Germany and Argentina. There are more than ten goals scored in four matches and displaying all of them would confuse the user and could not be properly arranged on the small screen.

Because of this, all vertices are divided into two groups, as shown in figure 5.4. In the roll out menu of the first group, all inactive instances are visible. In the roll out menu of the second group, only consistent inactive instances are visible. The vertex holding all the soccer matches, for example, always shows all inactive instances in the rollout menu (figure 5.5); however, the vertex holding all scored goals shows only the consistent ones (figure 5.6). Therefore, only those goals scored in the current active soccer match are visible in the roll out menu. If the user changes the active goal, the active soccer match remains the same.

### 5.2.2 Refinement of the Fisheye Distortion

Another consequence arising out of the first evaluation is to increase the text size of the vertex labels. Because we do not have the space to increase all labels in equal measure, we need a way to increase the size of only those labels of current interest to the user.

With the fisheye distortion, described in section 3.4.2, we are already using a mechanism to increase the size of certain labels. The evaluation

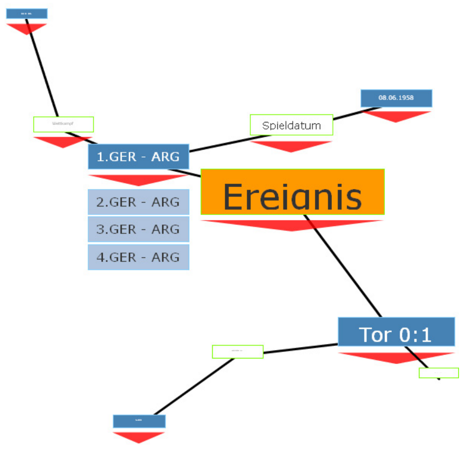


Figure 5.5: In the rollout menu with the soccer matches, all inactive instances are visible even if they are inconsistent. Inconsistent inactive instances are dark blue. The label *"Ereignis"* refers to the scored goals.

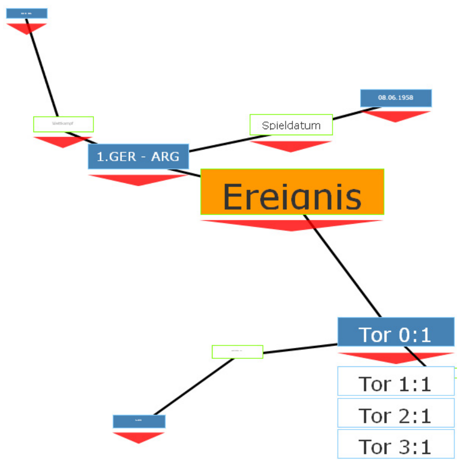


Figure 5.6: In the rollout menu with the scored goals, only the consistent inactive instances are visible. Consistent are all goals that were scored in the active soccer match. Consistent inactive instances are white.

showed, that the fisheye view at its current state of implementation offers no optimal distortion of the graph to easily extract information out of it. Therefore we have to think about ways to refine the fisheye distortion. The goal of such a refinement is to increase the text size of the vertex labels in a way, more adapted to the current demands of the user. It must be pointed out, that this refinement idea uses a new concept to increase certain labels, in addition to the distortion of the graph due to the fisheye view.

So firstly, we have to define current demands of the user. The text size of which vertices have to be increased in which situations?

If the user chooses an active instance as focus vertex, then the labels of all active outgoing relations of this new focus vertex have to be increased. So if, for example, the vertex with a certain soccer match between Germany and Argentina has the focus, then all active outgoing relations are of interest. Outgoing relations are, for example, *"happens at"* and *"final result"*. We assume that there is a connection between the interest in a certain instance and the interest in related information, like here the final result and the date for the soccer match.

In case of one of the relations having the focus, the interest of the user moves to the certain vertex this relation is leading to. So, if the user focuses on a relation, the label of the vertex this relation is leading to must be increased. Here the assumption is, that if the user is interested in the relation *"geschossen von"* (*"scored by"*), he will be highly interested in the real information, namely the player who really scored the goal. In figure 5.7 the refined fisheye distortion is shown. Due to the refinement, the text size of the vertex label, the relation *"geschossen von"* (*"scored by"*) is leading to, is increased. In comparsion, figure 5.8 shows the fisheye distortion at its first implementation, as it was described in section 3.4.2. In the first implementation, the text size of a vertex label depended only on its distance to the focus vertex and not on its relation to this vertex.

With the refinement of the fisheye distortion, the text size gets more adapted to the current demands of the user. Whether this refinement is appropriate to improve the extraction of information out of the graph or not, will be seen in the second evaluation.

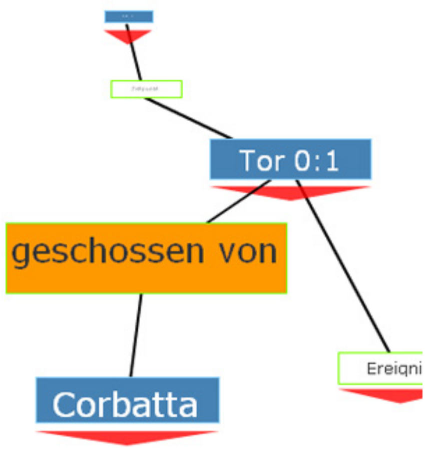


Figure 5.7: The text size of the player *"Corbatta"*, who has scored the goal (*"geschossen von"*), is increased due to the refinement of the fisheye distortion.

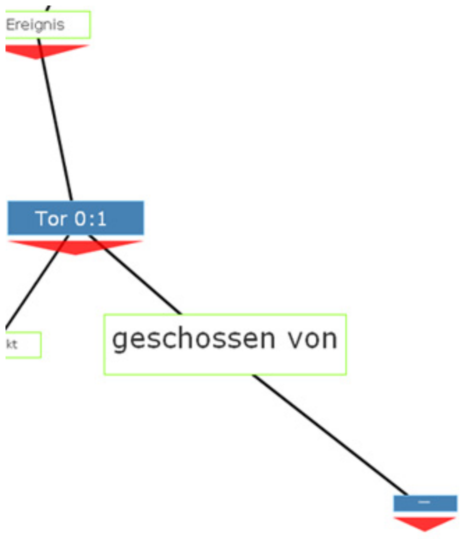


Figure 5.8: With the primary fisheye distortion, the player is almost invisible.

## 5.3 Integration into SmartWeb

Our graph presentation system is in the first instance developed as a part of the SmartWeb system. It serves as a new semantic point of view for the user, however, it offers not only a new perspective on the result data, but also a new interaction form to explore this information space. Furthermore, it should be possible, by using this graph-based user interface, to ask for new information. These software specifications are described in section 2.2.

At the current state of implementation, our second prototype is only able to deal with dummy data. As described in section 3.1, this dummy data represents the information of the answer to the question *"How was Germany playing against Argentina in a world championship?"*. As a first approximation, we assumed that this is the only graph to display and therefore we could find appropriate positions for the vertices, the dummy graph layout, manually.

However, for an integration into the SmartWeb system, our graph presentation system has to be able to handle arbitrary input data, encoded in RDF, and hence be able to request for this RDF data from the SmartWeb system (look at figure 5.9).

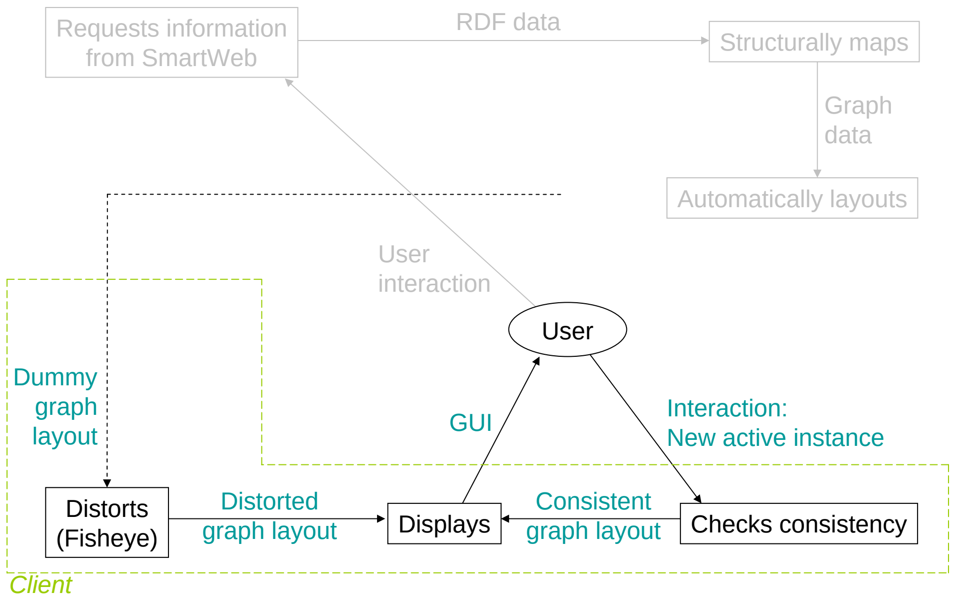


Figure 5.9: The integration of the graph presentation system into the SmartWeb system as a data flow diagram.

As it is shown in figure 5.9, for an integration we need mainly two things:

1. A structure mapping to convert the RDF input data into useful graph data for a representation on a small display.

2. An automatic layouter to get the positions for all the vertices of the graph data, the graph layout.

## 5.4 Structure Mapping

In the first prototype, we manually mapped the RDF data of the answer to the question *"How was Germany playing against Argentina in a world* *championship?"* to useful dummy graph data. In this section, a general implementation of a structure mapping for arbitrary RDF input data is described. The corresponding data flow diagram is shown in figure 5.10.

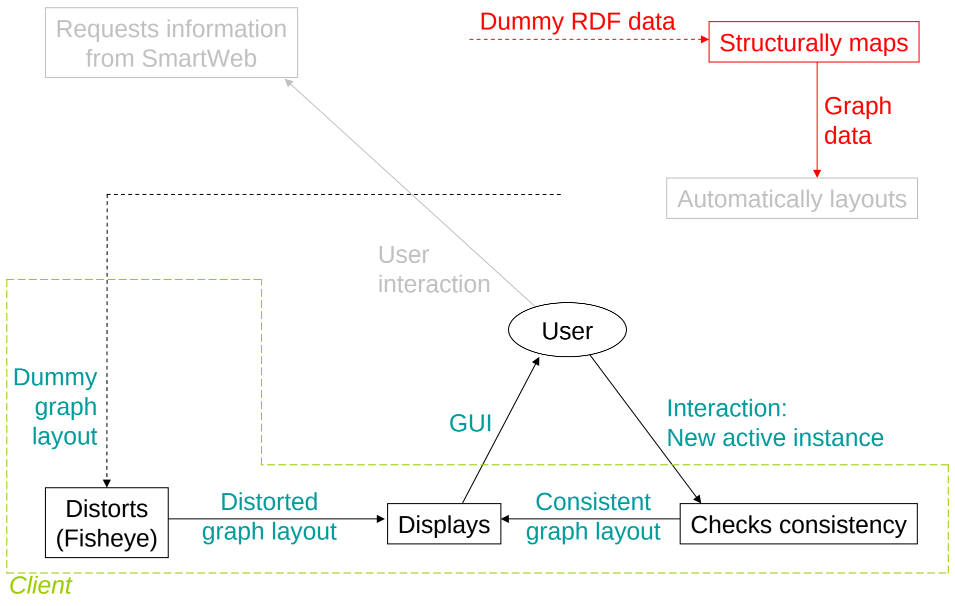


Figure 5.10: The Data flow diagram extended by a structure mapping.

Again, the reason for this structure mapping is to receive graph data with fewer vertices as we would receive by directly mapping each instance of the RDF data to its own vertex. Reducing the vertices needed for a representation of the RDF data, makes them more suitable to be displayed on the small screen of the handheld client. This means, the instances are grouped together based on certain criteria. In our manual structure mapping in section 3.1.1, we used as a criteria the concept of the instances.

It is possible to group the instances by their concepts, but maybe that is hard to implement. This is because, even if two instances are of the same concept that does not mean they share the same relationships. This does not apply for instances with different incoming relations. The incoming relations of an instance can differ from other instances of the same concept, because the incoming relations are not defined by the concept, but depend on the context.

This is, for example, the case for the concept *"date"*. An incoming relation from the concept *"soccer match"* to the concept *"date"* is *"happens at"*. However, another date instance can have other incoming relations, for example a relation *"starts at"* from a concept *"championship"* that defines a starting point for a time period.

Even if both instances are of the same concept *"date"*, their contexts are different. One date stands for an exact day of a soccer match and another date just stands for the beginning of a championship.

If both instances were represented by only one vertex, the edges connected to this vertex would be incorrect. This is because the incoming relations *"happens at"* and *"starts at"* are not valid for all the dates inside this vertex.

For this reason, our criteria to group instances together are the incoming and outgoing relations of the instances. In this approach different instances are only grouped together to one vertex, if they share exactly the same incoming and outgoing relations. So, only if the instances stand in the same contextual structure to all other vertices and their groups of instances. An instance, sharing the same concept, but having different incoming relations, can not belong to the same vertex. In this case, a new vertex is needed for all the instances, standing in this contextual structure.

In case of the example about dates, all instances of the *"date"* concept are divided into two groups of instances with different contextual structures. Therefore we need two different vertices, one for each group. All the date instances in one group have *"happens at"* as an incoming relation from their soccer matches and all the instances in the other group have *"starts at"* as an incoming relation from their championships.

To sum up, all instances of the RDF data are grouped by their contextual structure. Every group is then represented by one vertex in the new graph data. This approach for a structure mapping is applicable for arbitrary RDF input data.

## 5.5 Automatic Graph Layout

If we have to display graph data, how do we get the positions for all the vertices?

In the first prototype, we manually defined an x- and a y-coordinate to determine the positions of every vertex (look at section 3.1.2). This method was sufficient to receive a dummy graph layout for our fixed graph data. To receive a graph layout for arbitrary graph data, the appropriate positions for the vertices must be found automatically, as shown in figure 5.11. If the user changes a relation, the layout needs to be updated.

**Constraints** The three constraints for our automatic graph layout are:

1. All vertices must be inside a fixed space on the handheld.
2. Vertices must not overlap.
3. Related vertices must be placed next to each other.

In addition to these constraints for the automatic graph layout, a number of aesthetic criteria should be considered as far as possible [16].

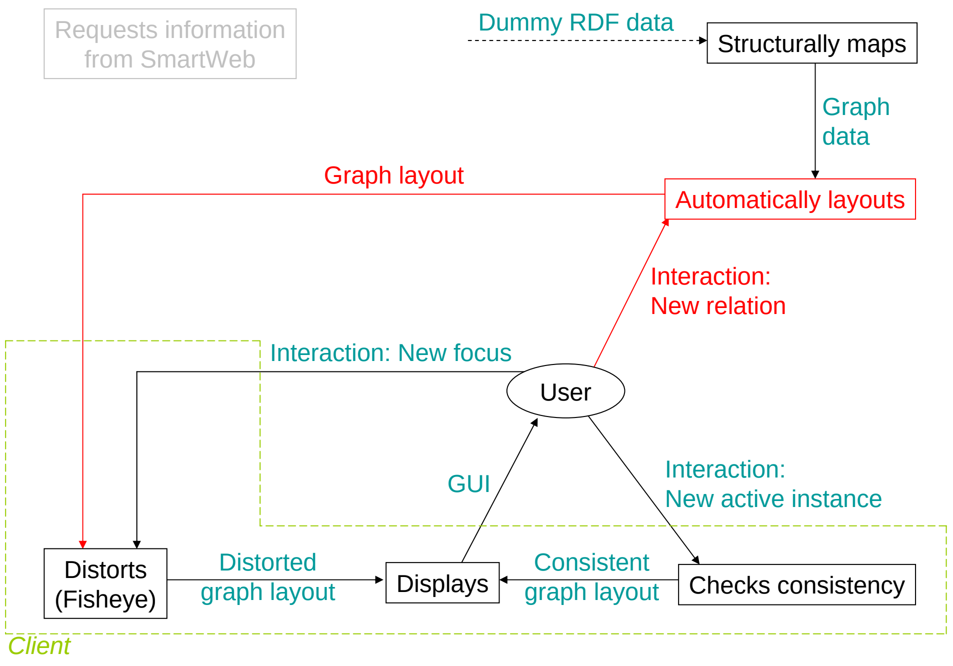


Figure 5.11: The graph layout for arbitrary graph data is found automatically and, in case of a relation change, it is updated appropriately.

**Aesthetic Criteria** For a small display on a mobile device, our four Aesthetic Criteria are:

1. Avoid edge crossings.
2. Keep edge lengths uniform.
3. Distribute vertices uniformly.
4. Keep vertices conform with user expectations.

There has been significant research on graph drawing algorithms [15]. In general two approaches are used to automate graph layout: the algorithmic approach and the declarative approach [52].

In the **algorithmic approach**, the idea is to devise an algorithm that is supposed to produce a graph that meets a pre-specified set of criteria. Force-directed algorithms are a class of algorithms for drawing graphs in an aesthetically pleasing way. Though this approach usually is computationally efficient, it is difficult to meet all criteria, and changing criteria or changing priority usually involves developing a new algorithm. Because the algorithm usually runs in polynomial time, it is able to deal with large problems [13].

The **declarative approach** is the most direct. In this approach, the layout of the graph is specified by a user-defined set of constraints, and it is generated by solving the system of constraints.

While this constraint-based method is a direct and expressive way of formalizing constraints into a computer program, the program itself usually takes a long time to compute a graph layout and is generally inefficient.

In this diploma thesis we try both approaches to find the best implementation of an automatic graph layout; the force directed method and the constraint based method.

### 5.5.1 Force Directed Method

A particularly unique and successful strategy for drawing graphs, which meet many of the aesthetic criteria, is the force directed method [22]. Unlike most algorithms rooted in theory, the force directed simulates a system of natural forces to find a graph layout.

The Force-Directed method defines a system of forces acting on the vertices and edges to find a minimum energy state [18].

The *"spring-embedder"* is the original force directed method, where all the edges are modelled as stretchable springs of different length which oscillate until the system reaches equilibrium [18]. Other force-directed methods produce physical models from subatomic forces of attraction and repulsion [22] and energy minimization using simulated annealing [14]. A survey of force directed methods can be found in [10].

For our problem, which consists in finding positions for vertices of a given graph structure, the force directed method seems to be appropriate. The method starts with an initial random placement of the vertices and tries to reach a stable minimum energy state. Due to the fact of random vertex positions right from the start of the computation, we never receive an empty solution. So the user will never receive an empty screen, even if the displayed solution of a graph layout is not optimal.

On the one hand, this is an important advantage of the force directed method. However, on the other hand, it is a disadvantage, because there exists no way to tell in advance, after what time the solution satisfies certain constraints. This is because the force directed method just tries to find the minimum of a global energy level for all pairs of vertices and does not care for certain constraints, as for example: Vertices must not overlap. Even in the optimal solution of the force directed method, there is no guaranty that vertices do not overlap.

For our graph layout, we choose for a first rough implementation of such a force directed method CCVISU[1] [6]. CCVISU is a lightweight tool for co-change visualization and force-directed graph layout, using the well-known algorithm of Barnes and Hut[3]. Although the tool was originally developed for computing clustering layouts of software systems, CCVISU is applicable to many graph layout problems.

[1]http://directory.fsf.org/ccvisu.html

Our rough implementation with CCVISU confirms the described disadvantage of the force directed method. Especially for a graph layout to be displayed on a small screen, so for our graph layout, the limited constraint satisfaction capability is of great disadvantage. Due to the lack of space on the handheld display, this disadvantage can easily lead to overlapping vertices and thereby to a reduction of readability.

### 5.5.2 Constraint-based Method

The second approach to automate graph layout, is the constraint-based method.

A central idea of this approach is to address automated layout by using constraint processing techniques [54] to represent and process causal design principles and perceptual criteria about human visual abilities for structuring and organizing documents.

The design of an aesthetically pleasing layout is characterized as a combination of a general search problem in a finite discrete search space and an optimization problem [27].

This problem is called the "Constraint Satisfaction Problem" or CSP. Formally speaking, a constraint satisfaction problem (or CSP) is defined by a set of variables, $X_1, X_2, ..., X_n$, and a set of constraints, $C_1, C_2, ..., C_m$ [47]. Each variable $X_i$ has a nonempty domain $D_i$ of possible values. Each constraint $C_i$ involves some subset of the variables and specifies the allowable combinations of values for that subset. A state of the problem is defined by an assignment of values to some or all of the variables, $\{X_i = v_i, X_j = v_j, ...\}$. An assignment that does not violate any constraints is called a consistent or legal assignment. A complete assignment is one in which every variable is mentioned, and a solution to a CSP is a complete assignment that satisfies all the constraints.

In section 5.5, we have formulated our problem of automatic graph layout in three constraints and four aesthetic criteria. Fortunately the problem to satisfy these constraints belongs to the simplest kind of CSPs, it involves only variables that are discrete and have finite domains, namely vertex positions. A position consists of an x- and y-coordinate and both coordinates are discrete numbers. The domains for these coordinates are restricted by the width and height of the handheld display.

If the maximum domain size of any variable in a CSP is $d$, then the number of possible complete assignments is $O(d^n)$, where $n$ is the number of variables. In the worst case, therefore, we cannot expect to solve finite-domain CSPs in less than exponential time that is, exponential in the number of variables.

We can roughly classify constraint-based languages and systems by using one of two approaches: the perturbation model or the refinement model [8].

In both cases constraints restrict the values that variables may take on.

In the **perturbation model**, at the beginning of an execution cycle variables have specific values associated with them that satisfy the constraints. The values of one or more variables are perturbed, and the task of the system is to adjust the values of the variables so that the constraints are again satisfied. The perturbation model has often been used in constraint-based applications such as the interactive graphics systems Sketchpad[51], ThingLab 1[7], Magritte[26], and Juno[35], and user interface construction systems such as Garnet [33].

In contrast, in the **refinement model**, variables are initially unconstrained; constraints are added as the computation unfolds, progressively refining the permissible values of the variables.

### 5.5.3 Perturbation Model

For the perturbation model we use the DeltaBlue algorithm [48]. Delta Blue is an efficient incremental algorithm for satisfying hierarchies of multi-way constraints using local propagation [48].

When building interactive user interfaces, the displayed information changes frequently and therefore the set of constraints. It is thus useful to have an incremental constraint satisfaction algorithm, one that can take advantage of previous computations rather than starting over each time there is a change in the set of constraints. The key idea behind DeltaBlue is to associate extra information with the constrained variables so that the solution graph can be updated incrementally, when a constraint is added or removed without examining more than a small fraction of the entire constraint hierarchy.

This is extremely useful for our graph representation in case of an extension or an update of the graph structure (look at section 3.3). So the DeltaBlue algorithm tries to solve every new constraint by a local search in the affected area, in contrast to a complete new search for all variables in the refinement model.

Even if the DeltaBlue algorithm seems to fulfill our demands of automatic graph layout with user interaction better, the effort for the implementation of this approach is enormous. The reason for this enormous effort lies in the fact that the SmartWeb system is implemented in Java and we have not found a Java implementation of the DeltaBlue algorithm. Available implementations of DeltaBlue are in C[2] and Delphi[3]. This leaves three possibilities, firstly, to implement it in Java ourselves, secondly, to build an interface between the implementation of this algorithm and the main SmartWeb system or thirdly, to switch to the refinement model.

[2]http://www.cs.washington.edu/research/constraints/deltablue/

[3]http://www.wack.co.za/article/articleview/177/1/11/

Before we decide to shoulder the effort of a new implementation, we have to check whether there is an easier way available with the refinement model.

### 5.5.4 Refinement Model

In the refinement model the approach to solve CSPs is based on removing inconsistent values from variables' domains until the solution is found. This is called forward checking. Whenever a variable X is assigned, the forward checking process looks at each unassigned variable Y that is connected to X by a constraint and deletes from Y 's domain any value that is inconsistent with the value chosen for X.

If a partial solution violates any of the constraints, backtracking is performed to the most recently instantiated variable that still has alternatives available. The term backtracking search is used for a depth-first search that chooses values for one variable at a time and backtracks it, when a variable has no legal values left to assign. An intelligent approach to backtracking is to go all the way back to one of the set of variables that caused the failure. A backjumping algorithm that uses conflict sets is called conflict-directed backjumping.

For the implementation of the refinement model, we use the Choco Constraint Programming System[4]. Choco is a Java library for constraint satisfaction problems (CSP) and constraint programming (CP). It is built on an event-based propagation mechanism with backtrackable structures.

Implementing an automatic graph layout, we have to meet the following three demands:
Firstly, it has to find positions for all vertices and these positions have to satisfy the three constraints, formulated in section 5.5.
Secondly, the four aesthetic criteria, formulated together with these constraints, should be considered as far as possible and thirdly, the computation of the solution has to be fast because of the interactive aspect of our system.

#### Implementing Constraints and Aesthetic Criteria

A vertex position consists of an x- and a y-coordinate. Unfortunately there exists no variable type in our constraint system for such a tuple structure. On this account we need to divide a vertex position into two constraint variables. One variable holds the x-coordinate of a position and another variable the y-coordinate. The domain of possible values for these variables is limited to the fixed space on the handheld, reserved for the graph presentation.

Now, if we want to formulate a constraint for the distance between two vertex positions $P_1$ and $P_2$, we have to formulate constraints for distances between two x-coordinates $(x_1, x_2)$ and two y-coordinates $(y_1, y_2)$.

[4]http://www.choco-solver.net

We formulate for all pairs of vertices a first distance constraint to avoid the overlapping of two or more vertices (figure 5.12). This minimum distance constraint prevents overlapping by setting a minimum separation distance value between all vertices.

As a second distance constraint, we additionally formulate for all pairs of related vertices a maximal distance constraint to place them next to each other (figure 5.12). With this second distance constraint, we also avoid edge crossings and keep edge lengths uniform.

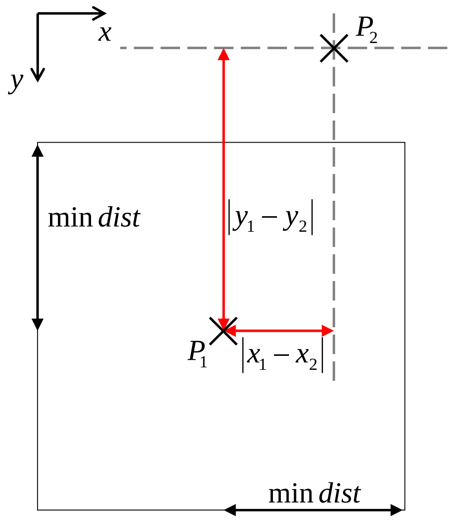


Figure 5.12: A rough approach to set a minimum separation distance (*mindist*) between two points ($P_1$ and $P_2$).

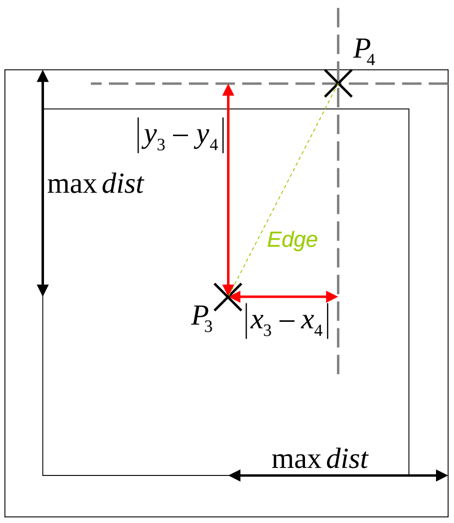


Figure 5.13: To place related points ($P_3$ and $P_4$) next to each other, we additionally set a maximal separation distance (*maxdist*) between them.

Our first idea to compute the distance between two positions is to use the Pythagorean Theorem:

$$distance = \sqrt{|x_1 - x_2|^2 + |y_1 - y_2|^2}$$

However, this is not possible, because there is neither power nor radical implemented in the Choco system, yet. Available are, at the best, elementary calculation types, like addition and subtraction, because more complex types are too computationally intensive due to the fact that every calculation type has to get integrated in the constraint system. So the constraint system needs to be able to backtrack, prune and search through every calculation type, for every variable and over the whole range of values. Thereby the constraint system shortly runs out of time in case of a complex calculation type like, for example, power.

Fortunately we do not need the exact distance; we only need to formulate something like: *"Two positions must not get closer or must not get farther than a certain number of pixels"*. Therefore we can take a simpler approach to prevent such a constraint, as it is shown in figure 5.12 and figure 5.13.

In this approach we define the shape of all positions with the same distance to a certain point as no longer being a circle, but as being a square. So a constraint defines a square around a vertex position of a certain intersection that keeps all other positions at bay. For doing so, it is enough to formulate a constraint that either $x_1$ has a certain distance to $x_2$ or $y_1$ has this distance to $y_2$. With the variable "*dist*" as the distance between the two vertex positions, the constraint can be described by the following formula:

$$(|x_1 - x_2| > dist) \vee (|y_1 - y_2| > dist)$$

If this constraint holds for two positions $P_1$ and $P_2$, then the distance between both positions is at least "*dist*", as it is in figure 5.12.

Unfortunately there exists also no way to formulate an absolute value in a constraint and that is the reason why something like:

$$(|x_1 - x_2| > dist)$$

needs to be formulated as:

$$((x_1 - x_2) < -dist) \vee ((x_1 - x_2) > dist)$$

The implementation of the constraints between related vertices can be found in function 6 and 7. Even if the formulation of a constraint seems to be very complicated, we have not found another library on the Internet sufficient for our concrete task.

The popular *"Java Constraint Library"*[5] (JCL), for example, contains no possibility to connect constraints with an "AND" or an "OR". However, we need this kind of constraints for the formulation of certain distances between vertices. So we take the Choco System to formulate our constraints and search for solutions.

**Function 6** relateVertices($problem$,$x1$,$x2$,$y1$,$y2$)

```
1: // Builds a maximal distance constraint between two vertices
2: $problem.post(this.buildConstraint($
   $problem, x1, y1, x2, y2, maxDistanceX, maxDistanceY, false));$
3: // Builds a minimal distance constraint between two vertices
4: $problem.post(this.buildConstraint($
   $problem, x1, y1, x2, y2, minDistanceX, minDistanceY, true));$
```

[5]http://liawww.epfl.ch/JCL/

**Function 7** buildConstraint($problem$,$x1$,$x2$,$y1$,$y2$,$limitX$,$limitY$,$max$)

```
1: Constraint $c$;
2: **if** $max$ **then**
3:     // A maximal distance constraint
4:     $c = problem.or(problem.or($
       $problem.lt(problem.minus(x1, x2), -limitX),$
       $problem.gt(problem.minus(x1, x2), limitX)$
       $), problem.or($
       $problem.gt(problem.minus(y1, y2), limitY),$
       $problem.lt(problem.minus(y1, y2), -limitY)));$
5: **else**
6:     // A minimal distance constraint
7:     $c = problem.and(problem.and($
       $problem.leq(problem.minus(x1, x2), limitX),$
       $problem.geq(problem.minus(x1, x2), -limitX)$
       $), problem.and($
       $problem.leq(problem.minus(y1, y2), limitY),$
       $problem.geq(problem.minus(y1, y2), -limitY)));$
8: **end if**
9: **return** $c$;
```

After having implemented all three constraints and two of four aesthetic criteria, formulated in section 5.5, the implementation of only two aesthetic criteria is missing. The first aesthetic criteria, we are going to implement in the following, is to distribute vertices uniformly, the second aesthetic criteria, we are going to implement, is to keep vertices conform with user expectations.

**Distribute Vertices Uniformly** An implementation of the aesthetic criteria, to distribute vertices uniformly, is an optimization that tries to spread the graph as much as possible over the available screen.

The Choco System offers a way to implement optimization. It is possible to search for a solution and thereby try to maximize or minimize a certain variable. We use this to implement as much spreading of the positions as possible. Therefore we need a variable denoting the objective value.

The variable to be optimized during the solving step is the total distance between all not related vertices (look at figure 5.14). The Choco System tries to maximize this total distance variable while searching for vertex positions. By doing so, the vertices are spread as widely as possible over the display.

**Keep Vertices Conform With User Expectations** If the user changed a relation in our first prototype (section 3.3.2), the positions of the vertices did not change, a relation only was replaced by another relation. All the

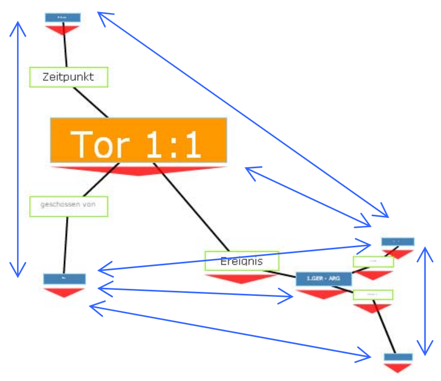


Figure 5.14: The automatic layouter tries to maximize the total distance between all not related vertices to distribute vertices uniformly.

vertices inside of a certain distance to the focus vertex were visible, even if the vertex had no active relation to another vertex at all. This problem was introduced in section 5.1.

If the user changes a relation in our second prototype, the positions of the vertices can change. This is because the automatic layouter tries to find optimal positions for all visible vertices. Vertices without any active relation to visible vertices are deactivated and become invisible. Having free space on the screen available, the layouter will try to rearrange the still visible vertices in a way to distribute them uniformly, as described above. This can lead to a completely new graph layout.

A completely new graph layout contradicts the aesthetic criteria to keep vertices conform with user expectations. Therefore we implement an optimization that leaves as many vertex positions untouched as possible in case of a new graph layout.

To leave always all vertex positions untouched is impossible on a restricted screen space. If we are possibly not able to leave all of them untouched for the next display of the graph, we need to define an order in which to try to leave them untouched.

The idea is to rate the vertex positions regarding to their importance for the user. A measurement for this importance is the time-lag to the last click by the user. Rating the old vertex positions this way, we get something like a trace of the user clicks. All the vertices that have been clicked recently keep their position in case of a new graph layout, all the vertices that have not been clicked recently release their position more easily. This mechanism is illustrated in figure 5.15 and 5.16.

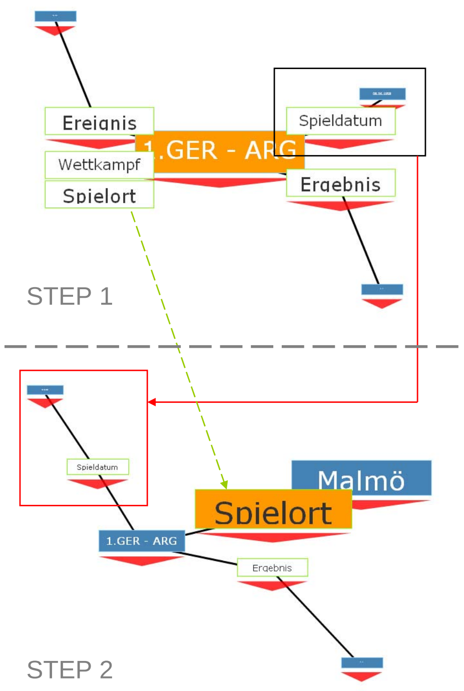


Figure 5.15: If a vertex (marked with the black box in STEP 1) has not been clicked recently, it releases its position easily (marked with the red box in STEP 2) in case of an update.

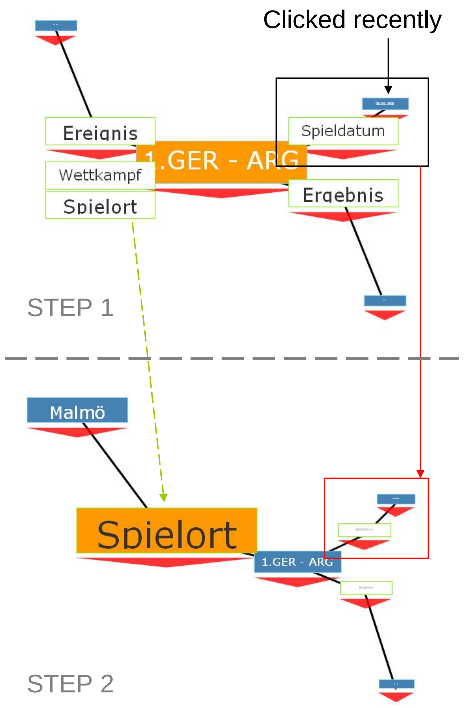


Figure 5.16: If a vertex has been clicked recently, the automatic layouter tries to hold its position (red box in STEP 2) in case of an update, here a change of relations (new relation: *"Spielort"*).

### Performance Problems

Our graph presentation system is an interactive system and thereby highly sensitive to performance problems. This is because the reaction time is an important factor for the usability of the system.

The efficient satisfaction of constraints is essential to the performance of constraint-based user interfaces [48]. The Choco System uses a systematic search and therefore gets into trouble in case of an increasing solution space. In our system, the solution space grows exponentially, because every new vertex needs a new constraint for every already existing one. So if we add just one vertex, we have to add a constraint for all already existing vertices to clear their relation for the right placement. So the exploration of the whole solution space could become quickly inapplicable for an interactive system in case there is no way to improve the efficiency. There are several methods to find solutions more efficiently.

One way is to use problem-specific knowledge to speed up the search. For example, choosing the variable with the fewest *"legal"* values first. This method is called the minimum remaining values (MRV) heuristic. It also has been called the *"most constrained variable"* or *"fail-first"* heuristic, the latter because it picks a variable that is most likely to cause a failure soon, thereby pruning the search tree [47].

Another way to speed up the search is to reduce the solution space. As described in section 5.5.4, every vertex position is represented by two constraint variables. The domain of possible values for these variables is limited to the fixed space on the handheld, reserved for the graph presentation. The screen resolution of the handheld client is $320 * 240$ pixel. For a search over a solution space of 320 times 240 possible positions for one vertex, the solution space for 10 vertices becomes $(320 * 240)^{10}$. This is a number with 49 digits!

The idea is to discretize the solution space to narrow down the search. We do not need a resolution as high as the display resolution of the handheld for the different vertex positions. It is enough to simplify the solution space for one vertex to 32 times 24 possible positions and thereby reduce the search period enormously (The solution space of possible positions for 10 vertices drop to a number with 29 digits). After receiving the vertex positions we just have to multiply them by ten and we have adequate data.

A different way to deal with performance problems is to use a faster computer. Even with all the described methods to find solutions more efficiently implemented in our automatic graph layouter, the handheld client is much to slow to find vertex positions in appropriate time. Thus we compute the automatic graph layout on the server and not on the handheld client. However, the fisheye computation stays at the client side, because the computational cost is far from the cost to compute an automatic graph layout.

With the automatic graph layouter on the server side and the graphical user interface on the client side, we have to implement a proper communication between them.

## 5.6 Server Client Communication

Our graph-based user interface on the handheld client needs the graph layout from the automatic graph layouter. Therefore the graph layout needs to be delivered from the server to the client. On the other side, the user interactions need to be delivered from the client to the server to change the graph layout accordingly. This data flow is shown in figure 5.17.

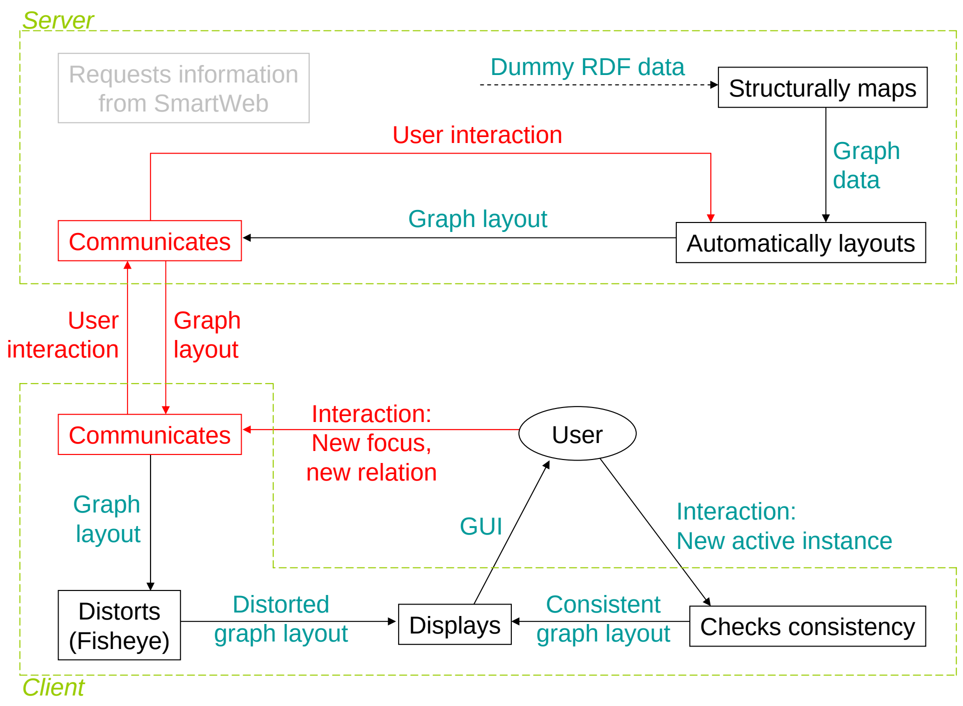


Figure 5.17: The data flow diagram of the whole graph presentation system extended by a server client communication.

### 5.6.1 Data Structure

An exchange of data between the server application and the client application is already organized in the SmartWeb system. Therefore we can just hook up with this exchange mechanism to deliver the graph layout to the client and the user interaction to the server.

In the SmartWeb system, all the data exchanged between server and client is organized by an XML structure. We extend this XML structure by an new tag, *dynamicgraph*, for the data to be exchanged in our graph presentation system, the graph layout and the user interaction.

Figure 5.18 exemplifies a graph layout being encoded inside the *dynamicgraph* tag. The graph layout consists mainly of vertices, their positions and relations. The instances are encoded in pairs inside the relations and the current focus vertex can be found in the footer.

The graph layout, encoded in XML, is sent from the server to the client to be displayed on the handheld. The user interacts with the graph-based user interface on the handheld client. If the user changes a relation or chooses a new focus vertex, this interaction is sent back to the server, also encoded in XML (look at figure 5.17). The third possible user interaction, to change an instance, is handled by the consistency check (section 3.3.1) on the handheld

```
<?xml version="1.0" encoding="UTF-8"?>
<dynamicgraph>
        <vertex>
                <id>match1:Match</id>
                <xpos>120</xpos><ypos>160</ypos><active>true</active>
                <showOnlyValids>false</showOnlyValids>
        </vertex>*
        <relation>
                <relid>match1-result1:Endergebnis</relid>
                <rellabel>Endergebnis</rellabel><active>true</active>
                <startvertex>match1:Match</startvertex>
                <pair>
                        <startid>match1</startid>
                        <startlabel>1.GER - ARG</startlabel>
                        <endvertex>result1:Result</endvertex>
                        <endid>result1</endid><endlabel>3 : 1</endlabel>
                </pair>*
        <relation>*
        <footer>
                <newfocus>
                        <vertid>match1:Match</vertid><instid>match1</instid>
                </newfocus>
        </footer>
</dynamicgraph>
```

Figure 5.18: The XML structure to deliver the graph layout from the server to the client.

client and needs no communication with the server.

Figure 5.19 exemplifies the XML structure to deliver the user action, to choose a new focus. Even if the fisheye distortion (section 3.4.2) is computed on the client, a new focus vertex can lead to a request for new information from the SmartWeb system, as it is described in the *flashlight approach* in section 3.4.2. However, at the current progress of the integration into the SmartWeb system, it is not possible to really request for new information, yet. This topic is part of the outlook in section 7.2.

As the user action, to choose a new focus, the action, to change a relation, can also lead to a request for new information from the SmartWeb system. Such a request will happen whenever the user interaction is an exploration of information outside the already received result data. The response to this request from the SmartWeb System then is send back to the layouter, to update the graph layout. Finally, this updated graph layout is forwarded to the graph user interface on the handheld client. The XML structure to deliver the user action, to change a relation, is shown in figure 5.20.

```
<?xml version="1.0" encoding="UTF-8"?>
<dynamicgraph>
        <action>
                <graphcommand>newFocus</graphcommand>
                <vertexid>match1:Match<vertexid>
                <instanceid>match1<instanceid>
        </action>
</dynamicgraph>
```

Figure 5.19: The XML structure to deliver the user interaction, to choose a new focus vertex, from the client to the server.

```
<?xml version="1.0" encoding="UTF-8"?>
<dynamicgraph>
        <action>
                <graphcommand>newRelation</graphcommand>
                <newid>match1-date1:Spieldatum</newid>
                <oldid>match1-result1:Endergebnis</oldid>
        </action>
</dynamicgraph>
```

Figure 5.20: The XML structure to deliver the user interaction, to change a relation, from the client to the server.

### 5.6.2 Conclusion

We found a way to balance the disadvantages of the Choco System in comparison with the DeltaBlue algorithm without causing too much effort. Our graph layouter finds the vertex positions automatically, solves the performance problems, described in section 5.5.4, and fulfills the constraints and aesthetic criteria, formulated in section 5.5. However, we still have to proof this in the second evaluation.

To sum up, our graph presentation system gets dummy RDF data and structurally maps it to appropriate graph data, as described in section 5.4. This graph data consists of instances, grouped to vertices and relations between those vertices. To present this graph data to the user, the automatic graph layouter tries to find proper positions for all the vertices. The problem is formulated as a Constraint Satisfaction Problem. The solution is an optimized graph layout with a memory function for old vertex positions (section 5.5.4). If the user interacts with the graph, the system is able to update and optimize the graph accordingly. Even the extension of the graph due to new information is possible but not implemented, yet.

In the next section we will think about limits of our approach and ways to handle them.

## 5.7 Limits of the Constrained-based Method

Compared to a force directed method, where a solution is always returned (look at section 5.5.1), a constrained-based method, like our refinement approach, returns a solution only if it satisfies all existing constraints (look at section 5.5.2). If there exists no solution that satisfies all the constraints, no solution will be returned. This happens either if the number of vertices is to large to fit on the available space or if the constraints are inconsistent. Actually, in both cases the constraints are inconsistent, only the reasons differ. To avoid an empty screen due to inconsistent constraints, we formulate three ideas how to handle this case.

The first idea, and probably the worst one, is to enlarge the space by a scrolling mechanism. If the amount of information does not fit on the available screen space, the user must scroll to reach all the vertices of the graph. Thereby, the user looses a general survey of the graph structure. However, during the whole development of the graph interface, we have tried to keep the advantage of a general survey in mind, so we should not start losing sight of it now.

The second idea is to increase the fisheye distortion. The distortion of the fisheye view defines the intensity of shrinking distant objects and magnifying nearby objects in relation to the focus (look at section 3.4.2). The mechanism is the same as by a lens of a fisheye camera. If the lens has a wide angle, the visible area is enormous, but the distortion is enormous, too. So we could change the intensity of the distortion depending on the amount of required space for the graph. However, there is also a threshold for the distortion of the graph, because at a certain intensity of distortion the remote vertices are getting too small to even be clicked on.

The third idea is to reduce the amount of vertices to display and thereby to reduce the amount of constraints to satisfy. Unlike the first two ideas that only offer a solution for inconsistent constraints due to a lack of space, this would be a solution for inconsistent constraints in general.

Whenever the automatic layouter is not able to find a solution to satisfy all the constraints, the amount of active vertices will be reduced to only those vertices with an active input or output relation to the current focus vertex. So only vertices with a direct connection to the focus vertex will stay visible. Due to this reduction, the inconsistency will be eliminated and a graph layout can be found.

In figure 5.21, we illustrate this procedure with our example of the soccer matches between Germany and Argentina together with their results, dates and scored goals and three additionally added relations *“TEST1”*, *“TEST2”*

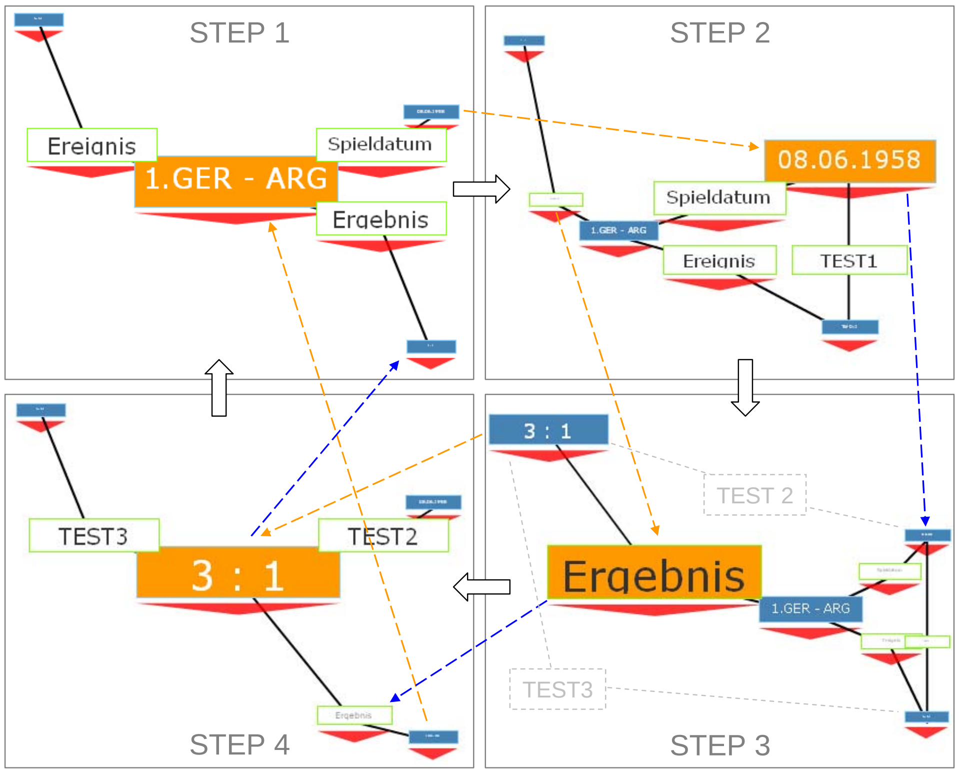


Figure 5.21: Four screenshots of our graph-based user interface and the way it handles inconsistent constraints.

and *"TEST3"*. At the beginning, in STEP 1, all three additional relations are inactive. Having all the relations active, the constraints will become inconsistent and the amount of active vertices will be reduced.
STEP 1 starts with the soccer match vertex as focus and the three active outgoing relations to the result, the date and the scored goals. Every relation between two vertices forces their position to hold a certain distance to each other as described in section 5.5.4. The constraints between the vertices in STEP 1 are consistent and so positions for all four vertices are found.
In STEP 2, the focus changes to *"08.06.1958"* and thereby activates the output relation *"TEST1"* between the date and the result[6]. The activation of this new relation leads to the formulation of a new constraint of the distance between the date and the result vertex. As can be seen in STEP 2, this constraint forces the connected vertices to stay closer together.
The two relations *"TEST2"* and *"TEST3"*, connected by light gray dotted lines in STEP 3, are drawn in to better understand the change to STEP 4. Both relations are inactive and will be activated not until STEP 4. If

[6]The mechanism to activate available relations of the focus vertex is described in the *flashlight approach* in section 3.4.2

these relations are activated, the according constraints will force the related vertices to stay closer together, as described above. To find a solution to satisfy all these constraints is impossible, because vertex *“Tor 3:1”* can not have the same distance to the soccer match, the date and the result vertex with the relation *“TEST1”* still active.

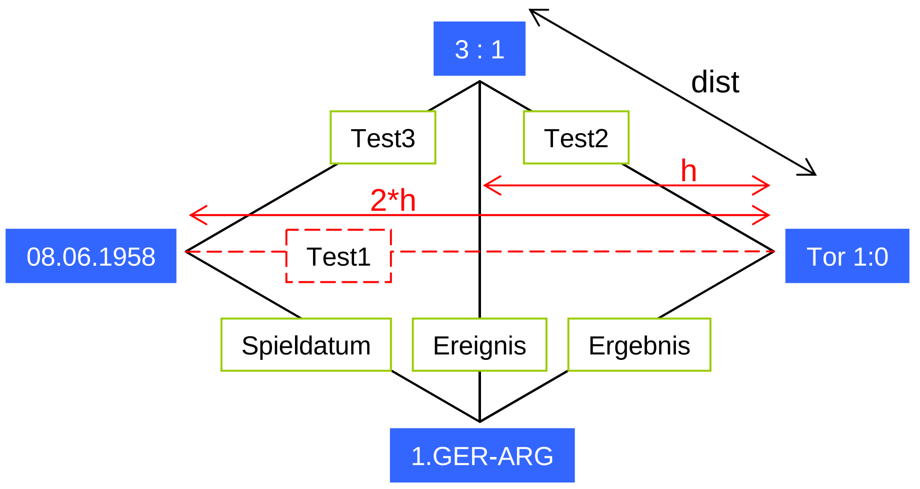


Figure 5.22: An illustration of the distances between four vertices constrained by fife active relations. The constraint of the sixth relation *“Test1”* can not be satisfied.

Figure 5.22 shows the distances between all four vertices together with their six relations. Having fife relations active and thereby the same distance *dist* between the four related vertices, makes it impossible to have the sixth relation *“Test1”* active, too. This is because the fife active relations form a parallelogram consisting of two equilateral triangles where the distance of the one inactive relation *“Test1”* is $2 * h$. The altitude $h$ of an equilateral triangle is defined as $h := dist/2 * \sqrt{3}$. If we insert this in $2 * h$ we get:

$$2 * h = 2 * dist/2 * \sqrt{3} = dist * \sqrt{3} > dist, \forall dist > 0$$

We proofed, that the distance $2 * h$ in figure 5.22 is greater than *dist* for all $dist > 0$. So it is impossible for all four vertices to remain equally aloof if the distance is greater than zero.

In STEP 4 (figure 5.21) on this account the automatic layouter is not able to find a solution to satisfy all the constraints of the six active relations and therefore the active vertices are reduced. Only vertices with a direct connection to the current focus stay active and thereby visible. By clicking on the soccer match vertex, the constraints become inconsistent again, what leads us back to STEP 1.

# Chapter 6

# Second Evaluation

In this chapter we evaluate our graph presentation system the second time. The goal is to validate the drawn consequences from the first evaluation and identify problems still existent to attain the implementation objectives. This second evaluation thereby serves also as the basis for the final conclusion and the summary of results.

Even if our graph presentation system is now able to deal with arbitrary input data, the information presented to the user in the second evaluation remains the same as in the first evaluation. Having the same information available through the graph interface, makes it easier to compare the results of the first and the second evaluation. Besides the better comparability, the interface for the graph layouter to request for new information from the SmartWeb on the server platform is not completely executable, yet. This problem is also part of the outlook, in section 7.2.

## 6.1 Evaluation Framework

The framework of the second evaluation is almost the same as the framework of the first evaluation, described in section 4.1. Firstly, we look at the program that will be evaluated. Secondly, we state the implementation objectives in measurable terms. Thirdly, the evaluation questions are formulated and finally we describe the context of the evaluation.

### 6.1.1 Second Prototype

Again we are going to evaluate the current state of our graph presentation system, the second prototype. However, there are several improvements in the visual representation and in the reaction forms in comparison to the first prototype.
The main points are:

- The graph layout is computed dynamically.

- The visual occurrence has been improved.
- The interactivity of the graph is better and changes are clearly visible.

Unfortunately, the integration into the SmartWeb system is not completed, yet. Because of this, the content of the graph is still limited to a fixed set of dummy information. This dummy information consists of the answer to the question: *"How was Germany playing against Argentina in a world championship?"*. Again, the idea is to present the user the dummy graph about the soccer matches between Germany and Argentina, ask questions about the information represented by the graph and after this every participant fills out a questionnaire containing nine questions about the usability of the program.

Even if this seems to be nearly the same prototype to evaluate as it was in the first evaluation, the mode of operation has changed completely. The vertex positions are adapted to the currently visible graph structure and all information displayed on the handheld client is provided due to a server client connection. All user interaction is delivered to the server, where the data is analyzed by the graph layouter and thereupon new information is delivered back to the client to update the graph.

The graph presentation system to be evaluated consists of not only a graph-based graphical user interface, implemented on the handheld client, but also of a graph layouter on the server. This second program controls the displayed graph as well as the communication between graph interface and graph layouter.

### 6.1.2 Implementation Objectives

The four implementation objectives remain the same as in the first evaluation. By doing so, we make sure, that the new results are comparable to the old ones.

The four implementation objectives are:

1. **Interaction**: The possibilities to interact with the graph are easily to understand.
2. **Information Extraction**: It is possible to extract information out of the graph structure and the vertex labels.
3. **Differentiation**: The user gets aware of the difference between an instance and a relation.
4. **Dependencies**: The user realizes the dependencies between related active instances.

### 6.1.3 Evaluation Questions

In this second evaluation we validate whether the drawn conclusions from the last evaluation have been correct or not. Therefore we deal again with the question about the achievement of objectives and possibly occurring barriers. However, this time we have a result out of the first evaluation to compare with.

The main question is:

- Were the improvements, implemented as a consequence of the first evaluation, appropriate to fix the weak points of the first prototype?

To answer this question, we confront the results of the first evaluation and the resultant consequences with the results of the second evaluation. If the attainment of implementation objectives has improved, the consequences, drawn from the first evaluation, had been correct. If the second evaluation reveals the opposite, we have to analyze the evaluation information to identify the barriers that have been encountered.

### 6.1.4 Context of the Evaluation

The second evaluation took place at the campus Saarbrücken of the Saarland University. The server was running on a notebook and had a direct connection to the handheld client. The handheld was plugged in an access point, connected to the notebook via an USB cable.

## 6.2 Evaluation Procedure

The practices and procedures to answer evaluation questions pertaining to the implementation objectives have not changed. We need exactly the same types of information, the sources of this information remain the same and the methods for collecting them, too. This is because the types of information needed, is guided by the objectives we assess and they have not changed either.

Again, the needed information is collected by the written questionnaire and own observations while the user handles certain tasks during the evaluation, as described in section 4.2.2. The used questionnaire can be found in the appendix.

## 6.3 Evaluation Results

In total, twenty participants were interviewed during the second evaluation. This is exactly the same number as it was for the first evaluation, what makes it easy to compare the results. This holds also for the computer skills of the twenty participants. The distribution is nearly the same as it is for the first evaluation. However, no participant out of the first evaluation was interviewed in the second evaluation.

### Results Regarding Interaction

In the first evaluation, 42 percent of the participants did easily understand the interaction possibilities, the remaining 58 percent did not. To reduce this high number of negative feedback, we did several improvements.
Firstly, we decided to increase the clicking area for the buttons.
Secondly, we reduced the amount of visible vertices on the display by an automatic graph layout, described in section 5.5. Due to this, vertices with no relation to other visible vertices are dynamically set invisible. This helped to increase the reaction time of the program, because it unloaded the amount of computations.
And thirdly, we marked the focus vertex with a special color, to make the current focus position more obvious (section 5.2).

As a result of these improvements, the amount of negative feedback dropped to only 16 percent in the second evaluation, as shown in figure 6.1.

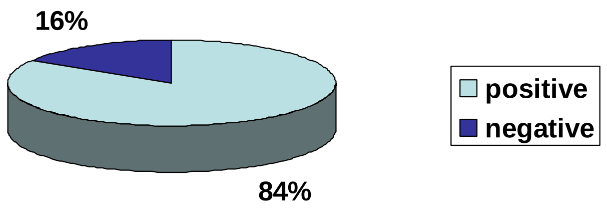


Figure 6.1: The possibilities to interact with the graph are easily to understand. This objective was attained in case of 84 percent of the participants in the second evaluation.

### Results Regarding Information Extraction

In the first evaluation, 59 percent of the group with less computer skills were confronted with problems by extracting information out of the graph structure. The amount of negative results for professionals was only 10 percent.

To correct this huge amount of negative feedback from unskilled participants, we decided to increase the text size of special vertex labels by a refinement of the fisheye distortion. The idea of this refinement was to increase those vertices of current interest to the user and by doing so to ease the extraction of information out of the graph (look at section 5.2.2).

Another point to support the information extraction was to dynamically hide vertices that have no connection at all. So the automatic graph layouter helped a lot to improve the graphical user interface.

With these ideas, the percentage of negative feedback felt to zero in the second evaluation. However, zero percent does not mean, everything concerning this objective went perfectly. There are always several factors that decide whether an objective holds for one participant or not. And therefore, such a number is always only a tendency and not a fact. Nevertheless, in comparison to the first evaluation, it is an improvement.

### Results Regarding Differentiation

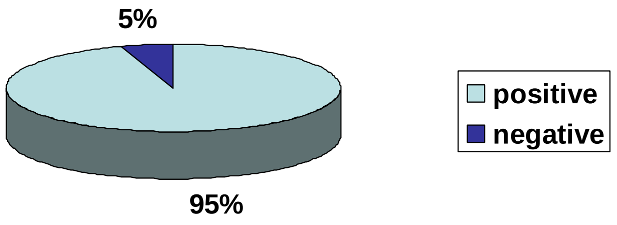


Figure 6.2: The user gets aware of the difference between an instance and a relation. Only 5 percent disagreed with this third objective in the second evaluation.

The third objective, regarding the **Differentiation** between an instance and a relation, was attained soonest. Only 15 percent were not aware of the difference between an instance and a relation in the first evaluation.

A barrier to attainment of this objective during the first evaluation was the fact, that the participants of the evaluation did not realize the connection between a relation like *"happens at"* and the information itself, the certain date instance. This was mainly because the vertex, this relation was leading to, was too small to get recognized by the user.

As a consequence, we increased those vertices of current interest to the user in the refinement of the fisheye distortion in section 5.2.2. Due to this refinement, the text size of the vertex label a relation is leading to will be increased, if this relation has the focus. This helps to understand the difference between an instance and a relation. Therefore, the result of the second evaluation dropped to only 5 percent of negative feedback (figure 6.2) for the objective to make an instance and a relation distinguishable.

### Results Regarding Dependencies

The last objective, regarding the **Dependencies** between related active instances, gained 45 percent negative feedback in the first evaluation. This happened mainly because the users did not realize changes in the graph due to a change of a certain active instance. So the dependencies between related instances were not realized.

The idea was to counter this bad feedback by a better visual response due to a change of a certain active instance in the graph. When ever an instance changes, it flashes to signalize this change (section 5.2). This improvement gained nothing at all and the percentage of the second evaluation is again 45 percent.

One reason for this stagnation of the percentage could be the faint visual effect in case of an update of active instances in the graph. Apparently this faint effect has not been noticed by the user. So maybe we need to increase the intensity of this effect to gain attention.

Another, more complicated reason could be the fact, that a new relation always replaces an old relation. Replacing an existing relation means to replace also the information by other information, so for example the information about the date by the information about the final result. If this replacement mechanism is not yet understood by the user, he could replace information that is still needed to answer a certain task.

The concrete task to validate this statement was: *"Name the result of the soccer match between Germany and Argentina in the world championship 1966"*. For the right answer, the user has to change the active instance of the vertex with all Championships to *"WM 66, ENG"* and then to activate the right relations to get the information about the result. If the user replaces exactly the relation, leading to the Championships, by the relation *"result"*, he could get confused because he cannot see whether this is still the right Championship. This problem is part of the chosen interaction form, the replacement, and therefore changing it will be difficult.

So to deal with this problem, it would be appropriate to offer an interactive tutorial. The purpose of this interactive tutorial would be to assist users in learning how to use our graph-based user interface. This idea is part of the outlook in section 7.2.

### 6.3.1 Conclusion

Generally speaking, this second evaluation shows a change to the better. This holds for almost all areas. The possibilities to interact with the graph are faster to understand, the extraction of information out of the graph structure and the vertex labels works well and gradually the user becomes able to distinguish between an instance and a relation.

Nevertheless, one area has not improved at all. The amount of users that realize the dependencies between related active instances remains constant. The reasons for this stagnancy and ways to counter them have been discussed above.

Altogether, the second evaluation showed the general correctness of the drawn conclusions from the first evaluation.

# Chapter 7

# Conclusion

In this thesis we developed a semantic graph-based user interface, which can provide new and interesting visual presentations and interaction possibilities.

Due to a graph-like representation of result data, the user gets a new understanding of the overall result structure and the relations within. Furthermore, the user receives another way to explore the content of this result and at the same time he can adapt this content to his current demands.

## 7.1 Summary of Results

We developed an additional graphical user interface for the SmartWeb system to make meta-data accessible for the user. The meta-data is used to arrange content in a way that the user better understands the semantic relations within this content.

For the graphical presentation of the content and its meta-data we chose a dynamic and interactive graph. Through the graph-based user interface, the user gets a semantic perspective due to the visualization of semantic relations and contextual information. Furthermore, this graphical interface offers the possibility to ask for more information or to show details on demand.

During the development of this semantic graph-based interface, we took the restrictions of the handheld device into consideration. We found a way to deal with the restricted computing power and the small screen size of the handheld client.

To reduce the amount of computation on the handheld we mainly do two things: Firstly, we deactivate every vertex with no connection to the current focus and thereby decrease the amount of computation to animate the movements of the whole graph. Secondly, we compute the automatic layout on the server in order to solve the performance bottleneck on the handheld client. On the server, we use problem-specific knowledge and the discretization of the solution space to narrow down the search for a graph

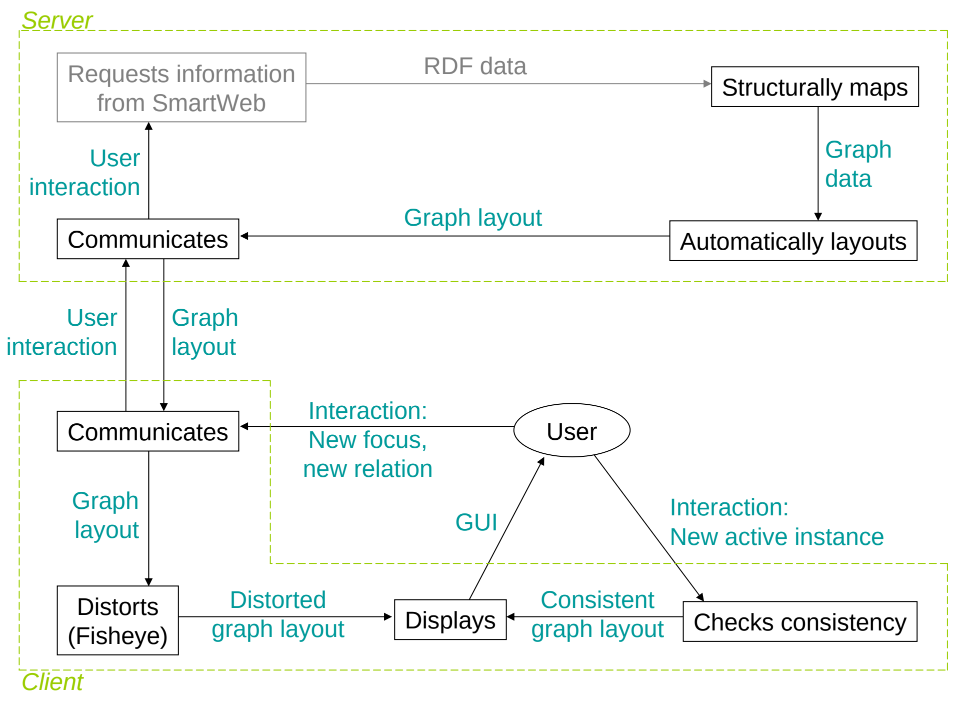


Figure 7.1: For the integration of our graph-presentation system into the SmartWeb system we have to implement an interface, which makes it possible to request information from the SmartWeb system and receive the answer.

layout to increase the reaction time of our interactive system.

To offer the user an overview and details on the small screen size at the same time, we use a fisheye view in combination with advanced interaction forms to strike a balance between these two extremes. Using a fisheye view is a valuable tool for seeing both local detail and global context simultaneously.

By clicking on a certain vertex of the graph, the user can change the focus point for the fisheye distortion. This interaction form is a way to ask for additional detailed information related to this new focus. At the current state of implementation, the displayable information is limited to dummy result data (figure 7.1).

Through the use of an automatic layout to arrange the vertices of the graph, our graph presentation system is able to deal with arbitrary graph data. Our automatic layouter tries to find vertex positions that satisfy certain constraints and aesthetic criteria. Even if the constraints are inconsistent and no solution exists to satisfy all of them, we implemented a way to handle this situation properly. Due to this, our graph presentation system is prepared to get integrated into the SmartWeb system.

During the development of our system, we used two evaluation phases to involve the users in testing design ideas and get their feedback in the early stage of development. The results of the two evaluations gave us information about the achievement of implementation objectives and barriers that were encountered. These feedbacks were useful sources of suggestions for the further improvement of our graph presentation system.

To sum up, in this diploma thesis we developed a promising approach to use meta-data for a semantic graph-based user interface on a handheld client as an additional GUI for the SmartWeb system.

## 7.2 Outlook

In this section we give an outlook to useful and necessary expansions of the already implemented components of our graph presentation system.

The integration of our graph presentation system into the SmartWeb system is not completed yet. Our graph-presentation system cannot be used to really ask for new information and our graphical user interface is not integrated into the SmartWeb GUI. Even though the integration is not completed, all important preconditions are fulfilled.

As it is shown in figure 7.1, for a full integration we need an interface between our graph presentation system and the SmartWeb system to request information and to receive the answer. With such an interface, our system would be able to request for new information to expand the displayed graph. An idea for this interface could work as follows:
Whenever the user clicks on a vertex representing a group of instances, for example soccer players, this user interaction is sent to the server. On the server our graph presentation system could use the new interface to ask whether further information to the clicked vertex is available. For such a request we could deliver all the instances of the clicked vertex to the SmartWeb system and receive thereupon all their outgoing relations and the instances these relations are leading to. If this information is not already part of the presented graph, the graph will be expanded by it.

For the integration of our graph-based user interface (figure 7.3) into the original SmartWeb GUI (figure 7.2), we have to specify how a switch between both graphical representations could be performed. The user could, for example, switch between these interfaces by clicking on a certain button, one for each representation. If the user switches from one interface to another, the information represented by the first interface has to be in accordance with the information represented by the second one. That way both graphical user interfaces stay up to date, even if the user expands the displayed information, changes the focus or asks a complete new question. The user would have the impression of interacting with only one information source rather than with two. So the interfaces would serve as two different points of view on the same thing.

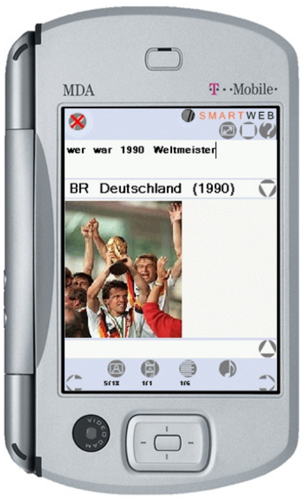


Figure 7.2: The original GUI of the SmartWeb system. The user has asked the question "Who was the world champion 1990?" and the given answer is "Germany".

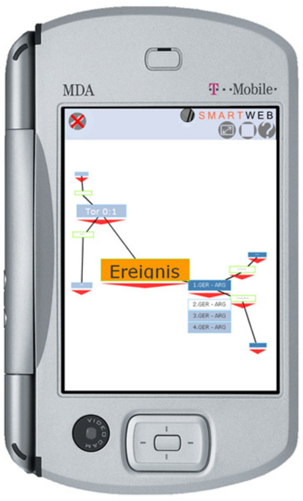


Figure 7.3: This is how an integration of the semantic graph-based GUI into the original GUI could look like.

Furthermore, the results of the second evaluation showed that the attainment of the implementation objective regarding the dependencies between related active instances (section 6.3) had not improved in comparison to the results of the first evaluation. One suggestion to counter this stagnation is to increase the intensity of the visual effect in case of an update of active instances in the graph. Increasing the intensity of the visual effect could gain the attention of the users in case of an update and thereby help to better understand the dependencies between related active instances.

Another suggestion for further improvement mentioned in the second evaluation was to offer an interactive tutorial to assist users in learning how to use our graph-based user interface. In such an interactive tutorial, the user could follow on-screen instructions, whereupon he could do the tutorial exercises and get feedback depending on his actions. That way, the user would become familiar with the interaction forms and misuse would be prevented.

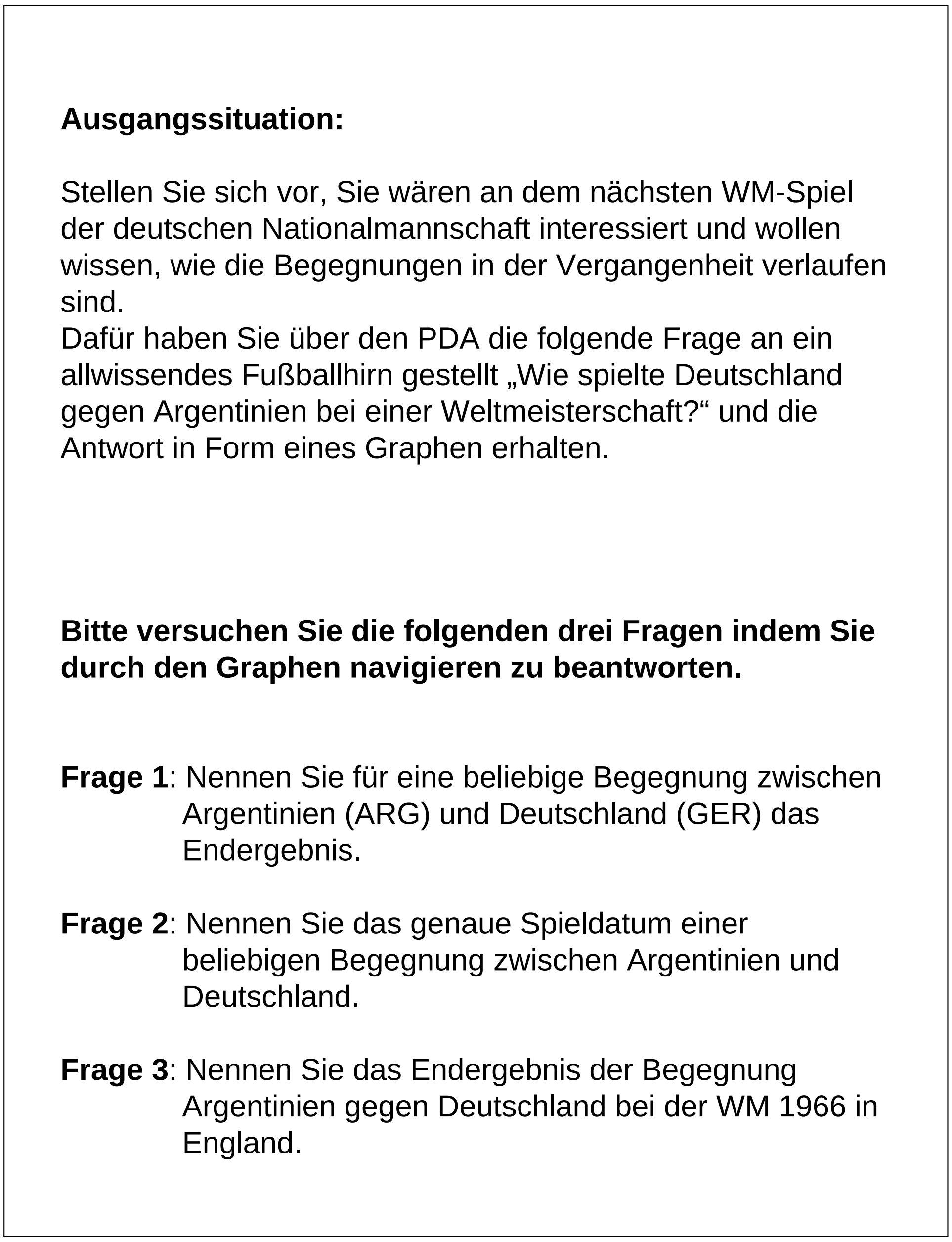
A suggestion for an additional functionality could be the expansion of the interaction forms by speech or other input and output modes. This would fit perfectly in the multimodal user interface of the SmartWeb system, where users can use speech and gestures to ask for information. On a mobile device with a small visual interface and keypad, a word may be difficult to type but very easy to express by speech. Systems that integrate complementary modalities to yield a highly synergistic blend potentially can function more robustly than unimodal systems that involve a single recognition-based technology such as speech, pen, or vision [38].

A promising field for multimodality in our graph-based user interface is the movement through the graph structure. Here the involvement of speech commands to navigate through the graph could be of special advantage for the user. This is because of the mobile aspect of the device as well as the mobile aspect of the SmartWeb scenario, described in section 1.2.1. So if the user cannot spare a second hand to click with the pen on the screen of the handheld, he could use speech as another input mode. Using speech to move the focus of the fisheye view and thereby navigate through the graph could be realized in several ways.

One way could be to just state directions to move to, as for example *"left"* or *"down"*. In response to such a user utterance, the fisheye focus could move on to the next best vertex that fits the stated direction best. Additionally, there could also be the possibility to undo such a movement by saying something like *"back"* or *"undo"*.

Another way to move through the graph structure by speech command could be to specify the direct goal of the movement, a label of a concrete vertex. When the user says a label of an existing vertex in the graph, the focus would move to this vertex. However, distant vertices can possibly not be readable for the user because of the shrinking due to the fisheye distortion. Therefore it could be impossible for the user to state the label of the goal vertex. Our graph presentation system can be easily expanded by missing functionalities like mulimodal interaction forms, especially due to the use of Design Patterns (section 2.3.1) like the Model-View-Controller and the Observer.

# Questionnaire



**Ausgangssituation:**

Stellen Sie sich vor, Sie wären an dem nächsten WM-Spiel der deutschen Nationalmannschaft interessiert und wollen wissen, wie die Begegnungen in der Vergangenheit verlaufen sind.
Dafür haben Sie über den PDA die folgende Frage an ein allwissendes Fußballhirn gestellt „Wie spielte Deutschland gegen Argentinien bei einer Weltmeisterschaft?“ und die Antwort in Form eines Graphen erhalten.

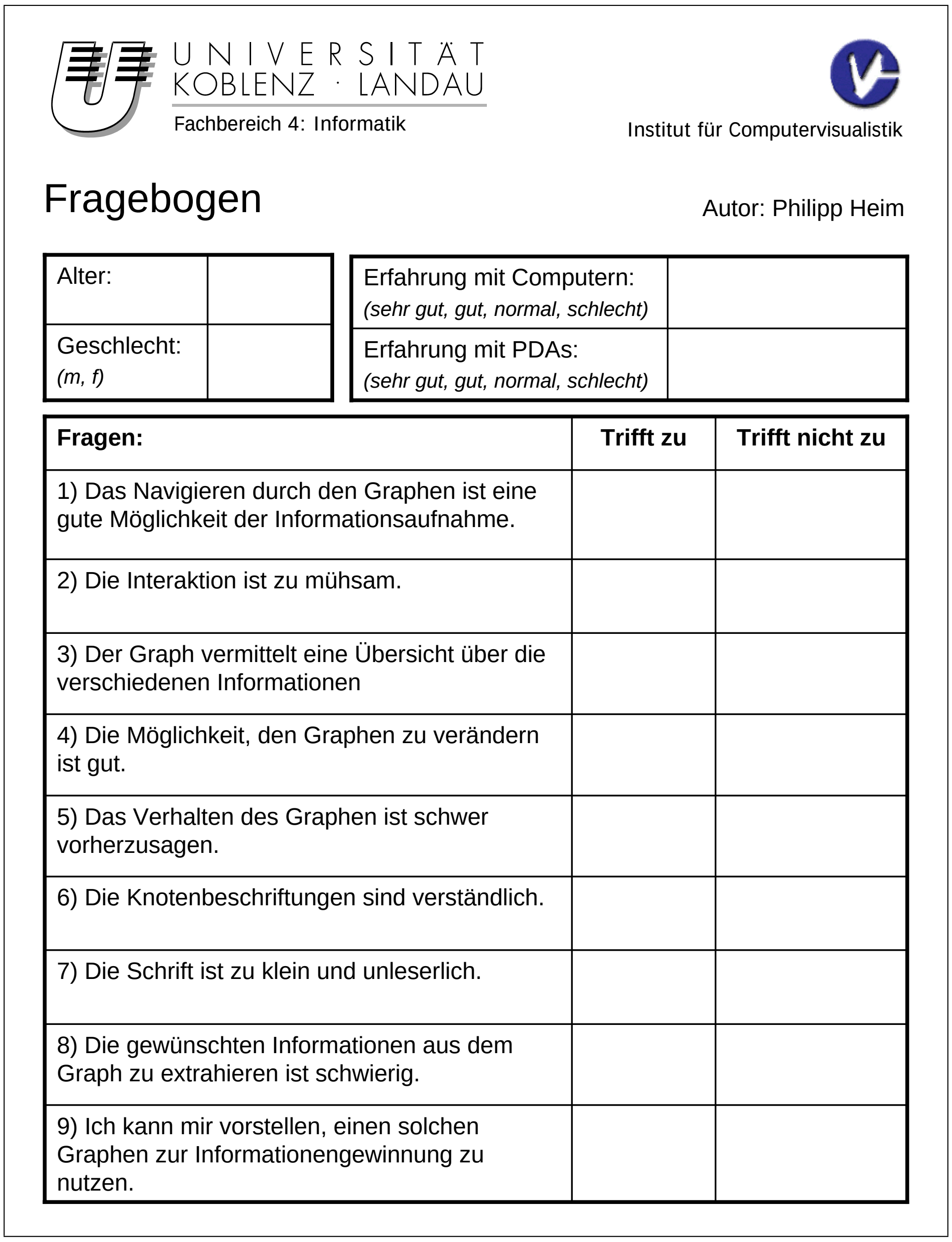
**Bitte versuchen Sie die folgenden drei Fragen indem Sie durch den Graphen navigieren zu beantworten.**

**Frage 1**: Nennen Sie für eine beliebige Begegnung zwischen Argentinien (ARG) und Deutschland (GER) das Endergebnis.

**Frage 2**: Nennen Sie das genaue Spieldatum einer beliebigen Begegnung zwischen Argentinien und Deutschland.

**Frage 3**: Nennen Sie das Endergebnis der Begegnung Argentinien gegen Deutschland bei der WM 1966 in England.

Figure 4: The introductory directions that are given in order to explain the situation to the participants of the evaluation, plus the three task, the user tries to solve by using the semantic graph on the handheld client.



UNIVERSITÄT KOBLENZ · LANDAU
Fachbereich 4: Informatik

Institut für Computervisualistik

Fragebogen

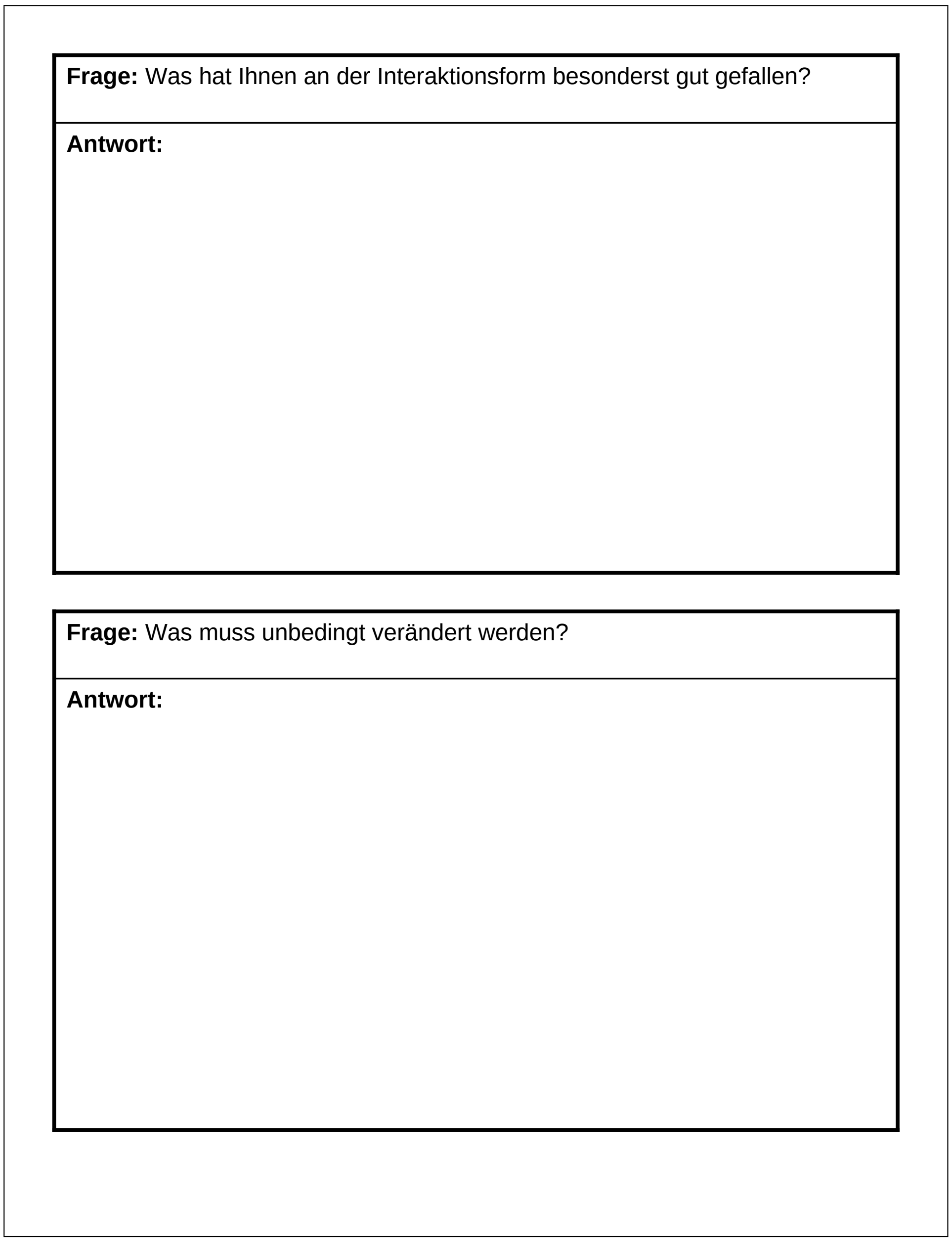
Autor: Philipp Heim

| | |
|---|---|
| Alter: | |
| Geschlecht: *(m, f)* | |

| | |
|---|---|
| Erfahrung mit Computern: *(sehr gut, gut, normal, schlecht)* | |
| Erfahrung mit PDAs: *(sehr gut, gut, normal, schlecht)* | |

| **Fragen:** | **Trifft zu** | **Trifft nicht zu** |
|---|---|---|
| 1) Das Navigieren durch den Graphen ist eine gute Möglichkeit der Informationsaufnahme. | | |
| 2) Die Interaktion ist zu mühsam. | | |
| 3) Der Graph vermittelt eine Übersicht über die verschiedenen Informationen | | |
| 4) Die Möglichkeit, den Graphen zu verändern ist gut. | | |
| 5) Das Verhalten des Graphen ist schwer vorherzusagen. | | |
| 6) Die Knotenbeschriftungen sind verständlich. | | |
| 7) Die Schrift ist zu klein und unleserlich. | | |
| 8) Die gewünschten Informationen aus dem Graph zu extrahieren ist schwierig. | | |
| 9) Ich kann mir vorstellen, einen solchen Graphen zur Informationengewinnung zu nutzen. | | |

Figure 5: The questionnaire consists of nine questions about the usability and the usefulness of the semantic graph-based interface. Additionally it asks about particulars, like the age and gender, and about the knowledge of computers, particularly of PDAs.



**Frage:** Was hat Ihnen an der Interaktionsform besonderst gut gefallen?

**Antwort:**

**Frage:** Was muss unbedingt verändert werden?

**Antwort:**

Figure 6: The back side of the questionnaire gives participants the possibility to write down some general remarks on the prototype.

# List of Algorithms

# List of Figures

# Bibliography

[1] C Alexander, S. Ishikawa, and M. Silverstein. *A Pattern Language.* Oxford University Press, Oxford, UK, 1977.

[2] Grigoris Antoniou and Frank van Harmelen. *A Semantic Web Primer.* The MIT Press, 2004.

[3] Josh Barnes and Piet Hut. A hierarchical o(n log n) force-calculation algorithm. *Nature*, pages 446–449, 1986.

[4] Kent Beck and Ward Cunningham. Using pattern languages for object-oriented programs. Technical Report CR-87-43, Apple Computer, Inc., http://c2.com/doc/oopsla87.html, 1987.

[5] C. Becker, J.and Brelage, K. Klose, and M. Thygs. Conceptual modeling of semantic navigation structures: The mosena-approach. In *Proceedings of the Fifth ACM International Workshop on Web Information and Data Management*, pages 118–125, 2003.

[6] Dirk Beyer. Co-change visualization. In *ICSM (Industrial and Tool Volume)*, pages 89–92, 2005.

[7] Alan Borning. The programming language aspects of thinglab, a constraint-oriented simulation laboratory. *ACM Trans. Program. Lang. Syst.*, pages 353–387, 1981.

[8] Alan Borning, Robert Duisberg, Bjørn N. Freeman-Benson, Axel Kramer, and Michael Woolf. Constraint hierarchies. In *OOPSLA*, pages 48–60, 1987.

[9] Roberto Boselli and Flavio De Paoli. Semantic navigation through multiple topic ontologies. Semantic Web Applications and Perspectives, SWAP 2005, 2005.

[10] Franz J. Brandenburg, Michael Himsholt, and Christoph Rohrer. An experimental comparison of force-directed and randomized graph drawing algorithms. In *Graph Drawing*, pages 76–87, 1995.

[11] Steve Burbeck. Applications programming in smalltalk-80(tm): How to use model-view-controller (mvc). www.math.rsu.ru/smalltalk/gui/mvc.pdf, 1987.

[12] Jon Christensen, Joe Marks, and Stuart Shieber. An empirical study of algorithms for point-feature label placement. *ACM Transactions on Graphics*, pages 203– 232, 1995.

[13] Rogier Van Dalen and Mike Spaans. On automated graph layout. http://www.tech.port.ac.uk/ addist/Layout.pdf, 2001.

[14] Ron Davidson and David Harel. Drawing graphs nicely using simulated annealing. *ACM Transactions on Graphics*, pages 301–331, 1996.

[15] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs: an annotated bibliography. Technical report, Brown Univ., 1993.

[16] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs.* Prentice Hall, 1999.

[17] Nikos Drakos. Application challenges to computational geometry: Cg impact task force report. Technical report, Princeton University, 1996.

[18] P. Eades. A heuristic for graph drawing. *Congressus Numerantium*, pages 149–160, 1984.

[19] M. Formann and F. Wagner. A packing problem with applications to lettering of maps. In *Proceedings of the 7th ACM Symposium on Computational Geometry*, pages 281–288, 1991.

[20] Arno Formella and Jörg Keller. Generalized fisheye views of graphs. In *Graph Drawing*, pages 242–253, 1995.

[21] H. Freeman. *Computer name placement*, pages 445–456. Longman, London, 1991.

[22] Thomas M. J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement. *Software - Practice and Experience*, pages 1129–1164, 1991.

[23] G. W. Furnas. Generalized fisheye views. In *CHI '86: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 16–23. ACM Press, 1986.

[24] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, New York, NY, 1995.

[25] Derek Gerstmann. Advanced visual interfaces for hierarchical structures. In *Human Computer Interaction*, March 2001.

[26] James A. Gosling. *Algebraic Constraints.* PhD thesis, Carnegie-Mellon University, 1983.

[27] Winfried Graf. Constraint-based graphical layout of multimodal presentations. In *Advanced Visual Interfaces*, pages 356–387, 1992.

[28] Alan C. Kay. The early history of smalltalk. In *HOPL Preprints*, pages 69–95, 1993.

[29] Ying K. Leung and Mark D. Apperley. A review and taxonomy of distortion-oriented presentation techniques. *ACM Trans. Comput.-Hum. Interact.*, pages 126–160, 1994.

[30] John M. McQuillan and David C. Walden. The arpa network design decisions. *Computer Networks*, 1:243–289, 1977.

[31] Colin Moock. *Essential ActionScript 2.0.* O'Reilly Media, 2004.

[32] Gary R. Morriwon, Steven M.Ross, and Jerrold E. Kemp. *Designing Effective Instruction.* John Wiley and Sons, 2004.

[33] B.A. Myers and et al. The garnet toolkit reference manuals: Support for highly-interactive, graphical user interface in lisp. Technical report, Carnegie Mellon University, Computer Science Department, 1990.

[34] David G. Myers. *Psychology.* Worth Publishers, 2004.

[35] Greg Nelson. Juno, a constraint-based graphics system. In *SIGGRAPH*, pages 235–243, 1985.

[36] D. Oberle, A. Ankolekar, P. Hitzler, P. Cimiano, M. Sintek, M. Kiesel, B. Mougouie, S. Vembu, S. Baumann, M. Romanelli, P. Buitelaar, R. Engel, D. Sonntag, N. Reithinger, and et al. Dolce ergo sumo: On foundational and domain models in swinto (smartweb integrated ontology). Technical report, AIFB, University of Karlsruhe, July 2006.

[37] R. Oppermann, B. Murchner, H. Reiterer, and M. Kock. *Softwareergonomische Evaluation. Der Leitfaden EVADIS II.* de Gruyter Verlag, Berlin, 1992.

[38] Sharon Oviatt. Ten myths of multimodal interaction. In *Communications of the ACM*, pages 74–81, November 1999.

[39] E. Pietriga. Isaviz: A visual authoring tool for rdf. http://www.w3.org/2001/11/IsaViz/, 2001-2006.

[40] E. Pietriga. Isaviz, a visual environment for browsing and authoring rdf models. In *WWW 2002 - The 11th World Wide Web Conference (Developer's day)*, May 2002.

[41] Emmanuel Pietriga, Christian Bizer, David Karger, and Ryan Lee. Fresnel: A browser-independent presentation vocabulary for rdf. In *International Semantic Web Conference*, pages 158–171, 2006.

[42] J. Preece. *Human-Computer Interaction*, chapter 27: Prototyping, pages 537–565. Addison-Wesley, 1994.

[43] Dennis Quan, David Huynh, and David R. Karger. Haystack: A platform for authoring end user semantic web applications. In *International Semantic Web Conference*, pages 738–753, 2003.

[44] Norbert Reithinger, Simon Bergweiler, Ralf Engel, Gerd Herzog, Norbert Pfleger, Massimo Romanelli, and Daniel Sonntag. A look under the hood: design and development of the first smartweb system demonstrator. In *ICMI*, pages 159–166, 2005.

[45] Norbert Reithinger and Daniel Sonntag. An integration framework for a mobile multimodal dialogue system accessing the semantic web. In *Interspeech 2005*, 2005.

[46] J. Rudd, K. Stern, and S. Isensee. Low vs. high fidelity prototyping debate. *ACM Press*, pages 76–85, 1996.

[47] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prentice Hall, 2002.

[48] Michael Sannella, John Maloney, Bjørn N. Freeman-Benson, and Alan Borning. Multi-way versus one-way constraints in user interfaces: Experience with the deltablue algorithm. *Softw., Pract. Exper.*, 23:529–566, 1993.

[49] Manojit Sarkar and Marc H. Brown. Graphical fisheye views of graphs. In *CHI*, pages 83–91, 1992.

[50] R. Storey, M.A.and Lintern, N.A. Ernst, and et al. Visualization and protégé. In *7th International Protege Conference*, 2004.

[51] Ivan E. Sutherland. Sketchpad: A man-machine graphical communication system. In *Spring Joint Computer Conference*, pages 329–345, 1963.

[52] R. Tamassia and I. F. Cruz. Graph drawing tutorial. http://www.cs.brown.edu/ rt/papers/gd-tutorial/gd-constraints.pdf.

[53] Joshua Tauberer. *What Is RDF.* http://www.xml.com/pub/a/2001/01/24/rdf.html, July 2006.

[54] E. Tsang. Foundations of constraint satisfaction. *Academic Press, New York*, 1993.

[55] Wolfgang Wahlster. Smartweb: Mobile applications of the semantic web. In *KI 2004: Advances in Artificial Intelligence*, pages 50–51. Springer, 2004.