Diplomarbeit

Development of an Infrastructure for Agent-based Intelligent Information Retrieval

Technische Universität Berlin

Fakultät IV - Elektrotechnik und Informatik
Institut für quantitative Methoden
Fachgebiet Systemanalyse und EDV

Betreuer:
Prof. Dr. Herrmann Krallmann
Dr.-Ing. Sahin Albayrak

Daniel Kirsch
Matrikel-Nummer: 178195

22. Oktober 2001

Zusammenfassung

Information Retrieval ist ein Begriff, der in unserer Informationsgesellschaft zunehmend an Bedeutung gewinnt. Die Verwendung von Methoden des Data Minings für Textanalysen, die über bloßes Text-Matching hinaus gehen, kann dazu beitragen, die enorme Informationsflut, der jedermann heutzutage ausgesetzt ist, zu ordnen.

Eine solche Methode, die die Latent Semantic Analysis (LSA) mit einer Self-Organizing Map (SOM) kombiniert, wird in dieser Diplomarbeit angewendet. Dabei werden semantische Ähnlichkeiten zwischen Dokumenten in Form einer Karte visualisiert.

Diese Diplomarbeit beschreibt die Implementierung eines Multi-Agenten-Systems auf der Basis der Agenten-Plattform JIAC IV, welches die Infrastruktur für ein solches visuelles Information-Retrieval-System bereitstellt. Dabei werden die besonderen Vorzüge von Multi-Agenten-Systemen ausgenutzt, ihrer Fähigkeiten zur selbständigen Kooperation und die Möglichkeit, zeitaufwändige und speicherintensive Probleme leicht zu verteilen. Im Mittelpunkt der prototypischen Realisierung ARISE (Agent-based Readjustable Intelligent Search Engine) steht zum einen die Skalierbarkeit des Systems, die durch ein verteiltes System gut gewährleistet werden kann. Zum anderen spielt die Wiederverwendbarkeit eine große Rolle, zu diesem Zweck wurden viele Teile des Systems so gestaltet, daß sie in anderen Anwendungen Eingang finden können.

Abstract

Information Retrieval is a term with increasing importance in our information society. The usage of complex data mining methods for text analyses is required to organize the large amounts of information everyone is exposed to.

One such method, a combination of Latent Semantic Analysis (LSA) and a Self-Organizing Map (SOM), is used in this diploma thesis. This approach allows the visualization of semantical similarities between documents in a map.

The diploma thesis describes the implementation of a multi-agent system based on the agent platform JIAC IV which provides the infrastructure for such a visual information retrieval system. It utilizes distinctive features of multi-agent systems, in particular the ability to cooperate autonomously and to distribute time-consuming and memory-consuming problems. This prototypical implementation called ARISE (Agent-based Readjustable Intelligent Search Engine) is focused on the scalability of the system provided by a distributed approach. Since reusability is an important factor many parts of the system are developed for simple inclusion in other applications.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the last decades one has observed a dramatic increase in the use of information-based technology. In 1994 the so-called Bangemann Report[1] introduces the term "information society"[Ban94] to describe the growing accessibility of this technology and its broad acceptance in our society. This very accessibility resulted in a demand for always more speed and storage capacity, leading to an overflow of stored and highly unorganized information. At this point the need arised for efficient information retrieval from huge databases, resulting in the development of a variety of search engines. The most illustrative example of this situation is the Internet: a so complex and heterogeneous database, that its contents can only be accessed efficiently through WWW search engines. But even well-assorted databases almost always have search engines to speed-up researches. The large quantities of textual data are a continuing challenge for applications like information retrieval systems.

Most of the existing search engines, like the popular web search engine Google, use text-matching techniques to find pages, sometimes in combination with other algorithms to improve the search results: "Google combines PageRank with sophisticated text-matching techniques to find pages that are both important and relevant to your search"[Goo01]. The input of a few keywords logically linked by boolean operations (NOT, AND, OR) results in a list of documents containing one or more of these keywords.

However, text-matching techniques are not context-sensitive and thus unable to recognize content similarity among documents. With text-matching the abbreviation "SOM", for example, could lead to such different results as "School Of Metaphysics", "Self-Organizing Map", and "Soil organic matter". One recognizes the need for a search engine able to analyze documents semantically, compare them with other documents or complex search requests,

[1] of the European Commissioner Martin Bangemann

and (graphically) display relations among them.

This diploma thesis with the title "Development of an Infrastructure for Agent-Based Intelligent Information Retrieval" provides an infrastructure based on an agent technology for such a text retrieval system. To this purpose it uses algorithms from the artificial intelligence (AI) implemented and tested by Richard Cissée in another thesis with the title "Neural Information Processing Methods for Agent-Based Intelligent Information Retrieval" [Cis01]. Most of these methods are well-known and well-studied but often only used for small information retrieval problems with low dimensionality. With the increasing of computing power there is now the possibility to use them for larger problems like text mining and information retrieval.

1.1 Objectives

The main purpose of this diploma thesis is the implementation of a new information retrieval system, which is able to analyze documents and recognize content similarities between them. In order to accomplish this, a scalable and reusable software architecture has to be developed. This architecture must provide functionality for the necessary special preprocessing of the documents after extracting the plain text, the creation of a database access, the implemention of administration tools, the embedding of the AI-methods, and finally the development of a graphical user interface for search requests to test the results. A concept for simple adaptation of the system to new document formats has to be developed. Due to the large complexity of the task the implementation of the algorithms itself was realized separately in a second diploma thesis [Cis01].

Since AI methods are very time-consuming and memory-consuming a distributed system needs to be implemented for a scalable architecture. Agent-oriented technology, being an approach which provides distribution, is chosen in the implementation. Another objective is the verification that the agent-based approach is well suited for the implementation of a scalable information retrieval system.

With the infrastructure and the AI-algorithms an agent system called ARISE (for Agent-based Readjustable Intelligent Search Engine) has to be implemented. This agent system, running on the JIAC IV environment [Dai01], can be part of complexer systems, and will deliver a concept of relations between textual information. However, it has to be possible to use the system as a stand-alone search engine with its own user-interface.

1.2 Structure of the Thesis

The structure of the thesis is as follows: In Chapter 2 the approach, including the used techniques is introduced. For some reasons (described in the following chapters) an agent-based implementation seems to be a good solution for the infrastructure. Therefore in Chapter 3 an overview of agent-based systems is given.

The next three chapters describe the implementation of the three parts of the text retrieval system. Chapter 4 deals with all processes necessary to prepare the text mining. In the following Chapter 5 the creation of the map, which represents the similarity relations between documents, is described. In Chapter 6 the handling of queries is explained and the Graphical User-Interface of the implementation is introduced.

In Chapter 7 the approach and the implementation are evaluated with a special focus at the scalability, reusabilty and performance of the implementation. Finally, in Chapter 8 the results are concluded and an outlook is given. Subsequent to the conclusion are the appendices.

Chapter 2

Approach

In this chapter the basic concepts of information retrieval as well as the methods of the artificial intelligence used for this Information Retrieval (IR) system are introduced. The explanation of the AI methods is only brief, since their implementation is part of the diploma thesis of Richard Cissée [Cis01]. Nevertheless, a basic understanding of the algorithms is necessary for the comprehension of the next chapters.

2.1 Information Retrieval - Text Retrieval

Information Retrieval (IR) describes a sub-field of Data Mining dealing with "the representation, storage, organization of, and access to information items" [1] [Rij79]. Text retrieval is the part of information retrieval concerning textual information items, i.e. documents. In the following three different models of text retrieval are represented.

Boolean Retrieval

Boolean retrieval is the most common technique used for text retrieval. A number of keywords is matched with those contained in the documents. The keywords are linked by logical operators like AND, OR and NOT, therefore this model is called boolean retrieval. It has no weighting, only the presence or not of a keyword in the document is relevant. Therefore it is impossible to consider the frequency of words or to access their importance for the content.

[1] In the context of text retrieval, these information items are documents.

Probabilistic Models

The main problem of boolean retrieval is the fact that most queries are "unprecise". When information about a special topic is searched, the user is often not familiar with all vocabulary associated to it. The probabilistic model and the vector space model are two approaches which consider this fact.

Probabilistic models determine the probability of a document's relevance with regard to the query. Most of these techniques are part of the supervised machine learning, i.e. they must be trained by a human, and sometimes the results are influenced by the personal view of the supervisor. An example of probabilistic models is the naive Bayes algorithm, an overview is given in [MN98].

Vector Space Models

The third model type is the vector space model. Vector space models use rules to create a vector representing every document. Content differences between documents are measured as distances between the vectors representing them. The approach used in this thesis is such a vector space model.

The output of the information retrieval system developed here is a graphical representation of documents in a 2-dimensional map. Adjacent documents are more similar to each other than documents lying at greater distances. Interdisciplinary documents are ideally placed between the clusters of their disciplines. Queries are treated as documents and visualized as points in the map. Starting at any point other documents can be searched. In this way relationships among documents and queries are easily visualized.

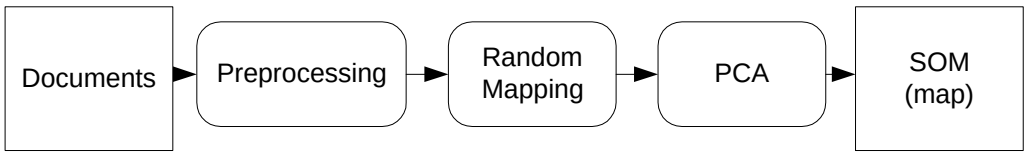

Figure 2.1: The ARISE text mining process

To analyze text documents a technique called Latent Semantic Analysis is used. "Latent Semantic Analysis (LSA) is a statistical model of word usage that permits comparisons of semantic similarity between pieces of textual information" [Fol96]. This technique was first introduced in 1990 by Deerwester, Dumais, Landauer, Furnas and Harshman [DDL+90]. After analyzing all documents using the Latent Semantic Analysis (LSA), a so called Self-Organizing Map (SOM), a special case of a neural network, is trained

with its results. This neural network is then represented as a 2-dimensional map.

However, before using the LSA and SOM algorithms it is necessary to preprocess the documents to enhance the results of the analysis (Figure 2.1).

2.2 Preprocessing

Since we are interested in comparing document content instead of merely comparing the words contained in a document, it makes sense to explicitly disconsider filler words like articles, pronouns, prepositions, etc. After the elimination of such words from the document text, the remaining words are used to transform the document in a vector, this is the first part of the LSA. There are different approaches for this transformation.

Word Frequency

The first one is to consider each different word in the document as one dimension of a vector space Vector components in each direction correspond to the frequency of the words associated with them. This simple approach has one important handicap. Two texts with the same words in completely different sequences have the same vector. Table 2.1 shows the document vector of the phrase "to be or not to be" with this approach[2].

Word	Frequency
to	2
be	2
or	1
not	1

Table 2.1: Frequency of words - example vector

Sliding Window

The second transformation uses a "sliding window" over consecutive characters while ignoring spaces. Every (different) combination of n consecutive characters[3] is one dimension in the vector space. By increasing n the window can reach whole word groups (so that texts with the same words in different

[2] without filler words removal

[3] n is the size of the window.

sequences will result in different vectors). Table 2.2 shows the document vector of the phrase "to be or not to be" with a window size of 4.

Character Combination	Frequency
tobe	2
obeo	1
beor	1
eorn	1
orno	1
rnot	1
noto	1
otob	1

Table 2.2: Sliding window - example vector

Sliding Window over Words

The third approach is to combine both previously mentioned techniques in a sliding window over words. In this case single as well as double words, consisting of two consecutive words, are stored. The details of all three variants and the complete discussion of their advantages and disadvantages is to be found in [Cis01]. Table 2.3 shows the document vector of the phrase "to be or not to be" with this approach[4]. In this thesis, we chose the third variant.

Word Combination	Frequency
to	2
be	2
tobe	2
or	1
beor	1
not	1
ornot	1
notto	1

Table 2.3: Sliding window over words - example vector

[4] window size 2

2.3 Dimensional Reduction

After determining the corresponding vector to each document, they span a m-dimensional space, where m is the number of all different words found in all considered documents. In general, m is very large, increasing with document number. The Latent Semantic Analysis (LSA) now reduces this high-dimensional space with minimal information loss. Several techniques are possible: Principal Component Analysis (PCA) is used to create vectors with lower dimensionality, containing nearly the same information as the higher-dimensional vectors. Because of its high complexity the PCA is very time-consuming. Therefore, before using the PCA, we reduce dimensionality by another method: Random Mapping. This relatively new technique was introduced by Kaski[5] in 1998. It can be used to reduce high-dimensional vectors to a dimensionality of a few hundred with small information loss. The dimensionality of the resulting vectors is subsequently reduced again with the PCA to a very low dimensionality because the information loss for low dimensionalities is smaller (as with Random Mapping).

Dimensional reduction is not the only reason for executing the PCA. In one of the following chapters is shown, that the compression of information with PCA can improve the results of a query.

2.4 SOM

With the resulting low dimensional vectors a neural network is trained, a so called Self-Organizing Map (SOM). This technique was introduced by Kohonen in 1990 [Koh90], therefore these maps are often called Kohonen maps. The graphical representation for such an Self-Organizing Map (SOM) is shown in Figure 2.2. In this example the training vectors are vectors with either only positive values as components or only negative values (randomly generated). Both groups are divided by the SOM during training. The vectors are represented by black points. Each color field represents one node of the neural network, the color represents the mean distance of this node to all neighbour nodes. The darker the color the greater the distance between this node and its neighbours (and the greater the distance between vectors associated with this nodes).

Although the LSA is over ten years old, there are only a few tools and projects using this technique, for example the WEBSOM project [WEB01], probably because it is more complex and resource expensive than text matching algorithms.

[5] based on the Johnson Lindenstrauss Lemma, see [JL84]

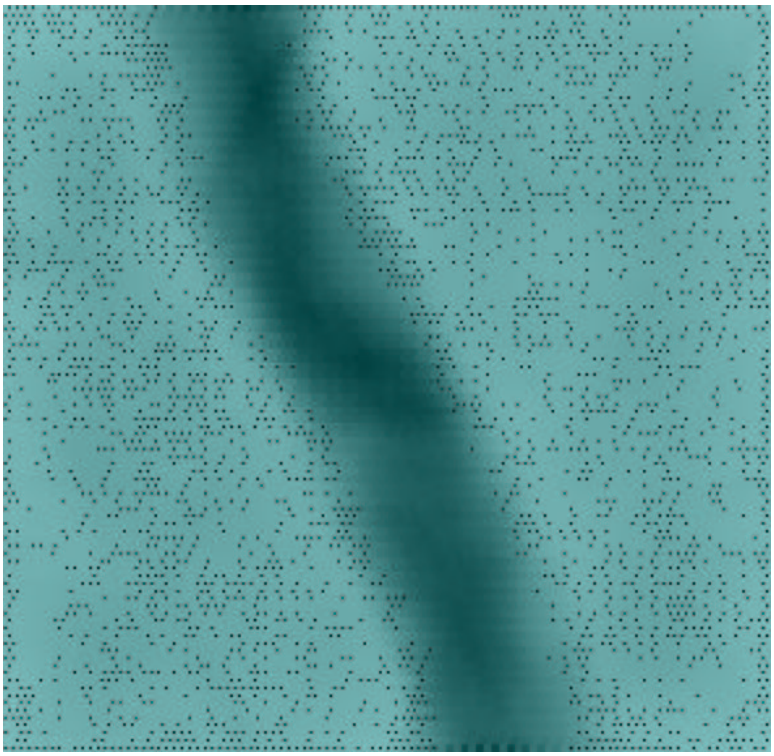

Figure 2.2: An example for a self-organizing map

2.5 Distributed Systems

The complexity of the techniques introduced in the previous sections requires a scalable architecture. We decided to implement the infrastructure as a distributed system in order to divide the complexity and execute time consuming processes in parallel. We have used a multi-agent system because it is easy scalable and allows easier implementation of distributed systems. In addition the AI capabilities make it easier to include text extraction for new document types (see Chapter 4).

The text retrieval process can be divided in three parts:

- **Preprocessing:** First the documents are preprocessed. The resulting data can be used for other purposes (for example simple text matching search), so this part is independent from both others.
- **Mapping:** Then the vector dimensionality is reduced and the Self-Organizing Map is trained.
- **Querying:** The last part is the handling of queries with a Graphical User Interface (GUI).

These three parts can run nearly independent from each other and are easy to distribute over many computers.

After a short introduction in agent-based systems, which is necessary for the following chapters, it is explained how the three parts will work for themselves independently and together.

Chapter 3

Agent-based Systems

In this chapter agent-based systems are introduced. This technology has aquired an increasing importance in the recent years, and it seems to be a good solution for complex problems like the text retrieval algorithms introduced in the previous chapter.

Agent-oriented techniques have their origin in distributed artificial intelligence, although they have been greatly influenced by artificial intelligence (knowledge-based systems), by the object oriented techniques, by the decision theory and the (tele-)communication.

Due to so many contributions there is no undisputable definition for "agents" (from latin: agens - the impelling force). However, there are some features common to the majority of them. An agent, according to Yoav Shoham is "an entity whose state is viewed as consisting of mental components such as beliefs, capabilities, choices, and commitments" [Sho91]. Later a second meaning of "agent" was introduced by telecommunication. This meaning has nothing to do with intelligent agents. According to Sahin Albayrak, "...the term Agent is used for two qualities of software, one is at least related to AI and the other is not"[Alb98]. It follows that a program must have the following attributes to be called an agent:

- **Autonomy**: i.e. the agent can control its own activities and invoke actions without human user interaction.
- **Interactivity**: the ability to interact with its environment. For this reason agents need to be able to communicate with their environment (and other agents).
- **Reactivity**: i.e. the agent can respond to events from the environment.
- **Goal-orientation**: "The term goal is used in a more metaphorical sense to describe the fact that agents pursue a given task until it is

finished and may delay or even reject other tasks in the process."

- **Mobility**: the ability of an agent to transport its code and information from a network place to another.
- **Adaptivity**: is a feature provided by artificial intelligence. Agents should have the ability to learn from mistakes and to adapt themselves to changing environment conditions.
- **Planning capabilities**: i.e. agents must be able to coordinate their long term plans and goals.
- **Reflection**: an agent has to represent its mental states internally.
- **Cooperation**: an agent is able to execute its task by interacting with other agents.

This is a very restrictive definition of agents and not all of these properties are considered common by all experts (see [Bra97] [WC01]). Common to all agent definitions seem only to be autonomy and interactivity.

Agent-based systems are distributed system allowing to use multiple resources for one problem, therefore they are often considered as a part of the Distributed Artificial Intelligence (DAI). They are a good solution for time-consuming problems which can be divided in many tasks. The most important difference between conventional distributed systems and the Distributed Artificial Intelligence (DAI) is the ability to handle situations and scenarios which are not conceived during the design of the system.

Speech Acts

The most popular form of communication between agents are speech acts, which give every data transmission a denotation (for example: questions, explanations etc.). There are two important types: Knowledge Query and Manipulation Language (KQML) [F+93] [FFMM94] and Foundation for Intelligent Physical Agent - Agent Communication Language (FIPA-ACL) [FIP97]. Both were devised to help software engineers develop multi-agent systems. While FIPA-ACL is an standard based upon philosophical science terms, KQML is an ad hoc standard which does not use the correct scientific terms.

FIPA-ACL provides the basic interaction mechanisms and primitive communication acts to let agents interact with each other. Complex communications in FIPA-ACL have to be organized in conversation protocols.

Belief, Desire, Intention

The concept of Belief, Desire, Intention (BDI) was introduced by Bratman in 1987 [Bra87]. BDI architectures are examples of practical reasoning - the process of deciding, which action to perform in the furtherance of the goals. The following example explains the three terms:

- I *believe* that if I work hard I will pass the exam.
- I *desire* to pass the exam.
- So I *intend* to work hard.

Belief and desire shape the intentions that agents adopt. BDI agents use two important processes: Deliberation, i.e. deciding what goal is to be achieved, and determination of means, i.e. deciding how to achieve the goal.

Agent Toolkits

In recent years a large number of platforms and toolkits for agent-oriented programming have been developed. One of the first was "Telescript" [Whi94], an object-oriented and communication-centered scripting language. It was developed by a consortium named General Magic founded by many well-known firms like AT&T, Motorola, Sony, Toshiba, Fujitsu, Philips and Apple. "Telescript" was concepted as an supplementation to existing programming languages. It provides the basis for agent-oriented software development. Standard concepts like TCP/IP are used for communication. But concepts like cooperation are not supported, they must be implemented by the agents themselves.

Beside "Telescript" SUN's object-oriented language Java[Jdk01] became important for agent-oriented platforms. Since Java is platform-independent it is easy to run Java-based programs (like agents) on different operating systems and hardware platforms. Java ME (MicroEdition) for many mobile systems like Palm OS or Windows CE is very interesting for mobile applications for 3rd generation networks.

"Grasshopper"[BBCM98] is one of these Java-based mobile agent platforms. It is built on top of a distributed processing environment. It allows different variants of communication like CORBA, Java Remote Method Invocation (RMI) and socket programming. "Grasshopper" is very communication oriented.

3.1 JIAC IV

The Java Intelligent Agent Componentware IV (JIAC IV)[1] [Dai01] is an agent development toolkit based on a Java class library, an Application Program(ming) Interface (API) for the development of agent oriented systems (see [AW99] and [FBK+01]). All agents run in an environment called marketplace (see Figure 3.1). The pure Java realization allows the deployment of agents and marketplaces within any conceivable hardware context. A central idea of the JIAC architecture is the inherent mobility support for any kind of agents (although many of them remain stationary), i.e. agents can migrate from one marketplace to another, travelling away with all their sources and information they store.

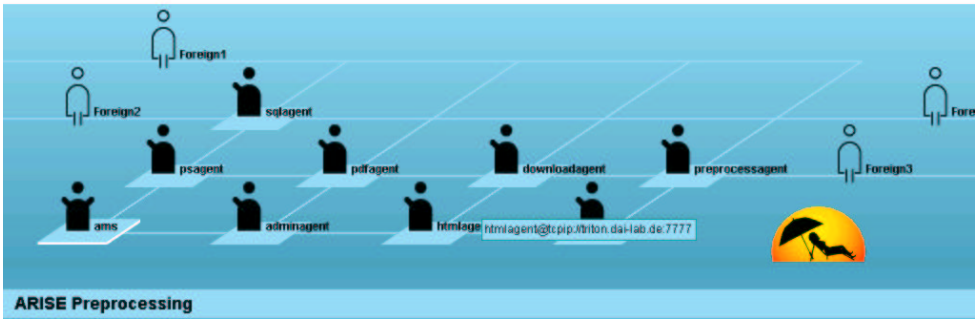

Figure 3.1: A JIAC IV marketplace with multiple agents

With JIAC IV it is easy to create distributed agent-oriented systems. Since it is Java-based, the most functionalities and classes of Java can be used for agents if writing new components. Java-based JIAC components are the most common way to implement functionality in JIAC IV. The Artificial Intelligence (AI) methods implemented by Richard Cissée are written in Java and can be easily integrated.

Ontologies, Plan Elements, Goals and Services

JIAC IV received important influence from knowledge-based systems. Agents have knowledge organized in ontologies. For communication two agents have to know the same ontology.

The abilities of JIAC IV agents are described with plan elements, which specify the tasks[2] an agent can execute. Every plan element has precondi-

[1] developed at the DAI-Lab
[2] i.e. plans

tions and effects, which describe the plan element in Java Agent Description Language (JADL) in first-order predicate logic. On the basis of these specifications the agents can later use these plan elements, if the effects satisfy their intentions and the preconditions are fulfilled.

Additionally every agent has the ability to set up goals, if it cannot satisfy its intentions by using its own plan elements, these goals are matched with services provided by other agents. Services are plan elements, which one agent provides for others. All these features are the basis for an intelligent cooperation between agents. So cooperation does not has to be implemented by the agent like in "Telescript", it is an elementary part of the system. This fact has played an important role for the choice of JIAC IV for the implementation of the Agent-based Readjustable Intelligent Search Engine (ARISE) system.

Communication between agents is uniformly realized by means of speech acts, based on the FIPA-ACL standard. These speech acts are distributed between agents on different marketplaces via Transmission Control Protocol/Internet Protocol (TCP/IP), while agents on the same Virtual Machine are using the Java Virtual Machine (JVM) communication. Security features are also supported.

JIAC IV is service-oriented, which means that agents interact with each other only through service usages. An agent (or a human) can use services from other agents by setting up a goal and sending it to the agent system. All services of other agents are registered at a central administration, the so called Directory Facilitator, and so a service can be selected which fulfills the goal (although the service provider may refuse the execution of a service) [SB01a] [SB01b]. With protocols the agents can determine the details of a service usage.

Human Services

To allow humans to use agent services, so called human services have to be executed. To access these services, a program called DAI Navigator can be used. It uses a Graphical User Interface (GUI) provided by a so called alter-ego-agent (which works as server for this GUI). After establishing the GUI, the Navigator can communicate with the alter-ego-agent, which can use the services of other agents.

Another possibility for human users to interact with the agent system is the Multi-Access Point (MAP), which allows the use of services via HTML- or WML-interfaces (see [ABF01] for details). However, this technology is still in development, and in its current state it is not possible to use it for complex interactions.

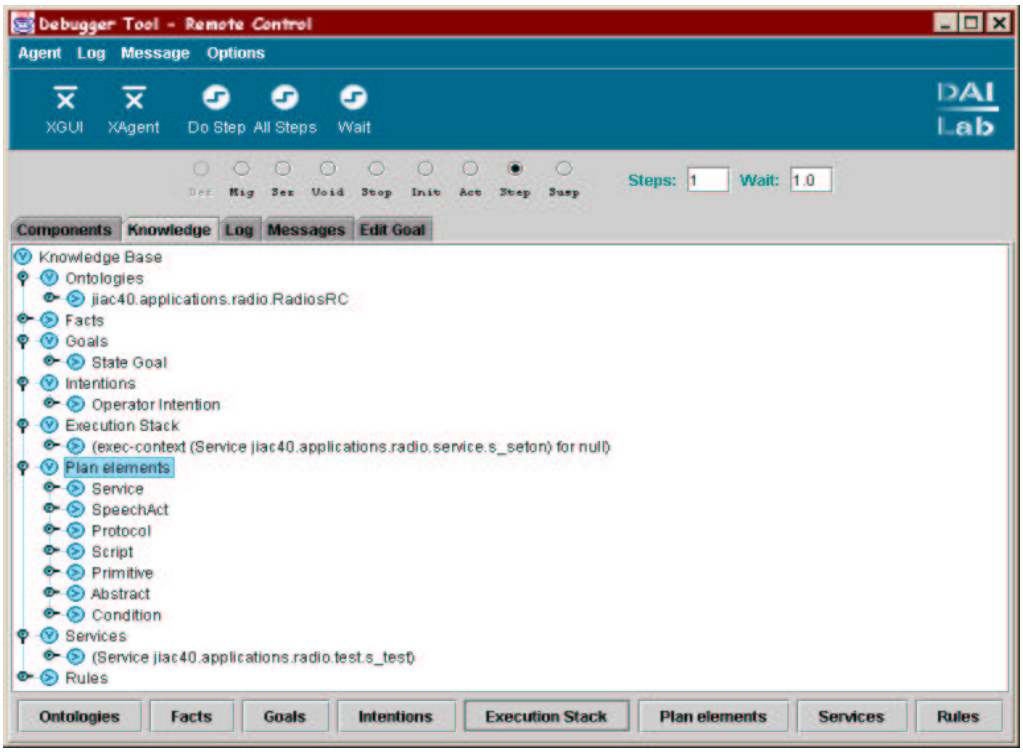

Figure 3.2: The debugger tool: Here it shows ontologies, plans, services and goals of an agent

JIAC Tools

JIAC IV supports the development of new agent systems with a number of tools. Apart from the runtime environment, the parsing tools and the ontology compiler this includes a debugging tool (see Figure 3.2) which allows to view the knowledge base of an agent during runtime. The ontology builder allows creation and manipulation of ontologies, which can be developed visually. The Agent Development Environment (ADE) is an agent builder, which makes it possible to configure new agents by adding ontologies, components, plans and facts in a simple Graphical User Interface (GUI).

3.2 Design

Since agent-based systems have a great complexity, there is no general concept for the design of a multi-agent system. So it is impossible to use the standard software engineering method like Fusion for the design of the ARISE system.

It seems logical to implement the three independent parts (preprocessing, mapping, querying) mentioned in the previous chapter as three different marketplaces. Since JIAC IV is service-oriented, the determination of the agents results from the determination of the necessary tasks, i.e. the services. Each agent performs at least one service, some agents have to perform more (related) services.

The following three chapters describe the three marketplaces "preprocessing", "mapping", and "querying". The details of the marketplaces, the agents, and the interaction between them are explained.

Chapter 4

Preprocessing

This chapter explains the preprocessing of the documents. Before textmining the documents have to be preprocessed to improve the results of the text retrieval process. The preprocessing consists of four phases: supplying the documents, extracting the plain text, analyzing it, and finally storing the analyzed data in a database. These tasks are logically modelled as four different agent types which work like an assembly line.

Independent of the data source (for example a database or the WWW) the first agent, the Download agent, has to deliver the documents in an uniform data format. This data format will be interpreted by a second agent, which extract the plain text from the document. There may be different agents for different document formats (Postscript, PDF, HTML, etc.). All of these agents send their results (as plain text) to a third agent, the Preprocessor agent, which analyzes their structure, create meta data about the documents and send database statements to a database agent. Then the database agent updates the meta database. The meta data may be used by other agents to do the Latent Semantic Analysis, but it is also possible to use it for other purposes (like text matching search).

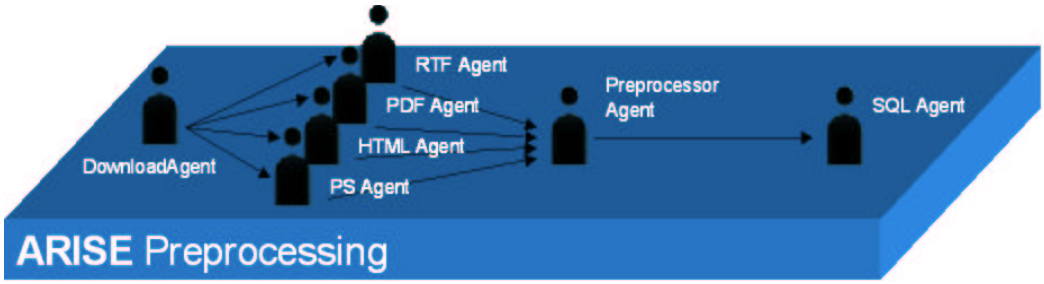

Figure 4.1: The ARISE preprocessing architecture

In our implementation the assembly line is situated on one marketplace and consists of seven agents (see Figure 4.1): The Download agent retrieves the documents from the WWW. Four agents, namely HTML agent, PS agent, PDF agent, and RTF agent, can extract the text from the most popular document formats of the web. The Preprocessor agent performs the text analysis and creates SQL statements which the SQL agent executes on the database. Additionally to these agents the marketplace has the obligatory manager and an administration agent for human control of all agents.

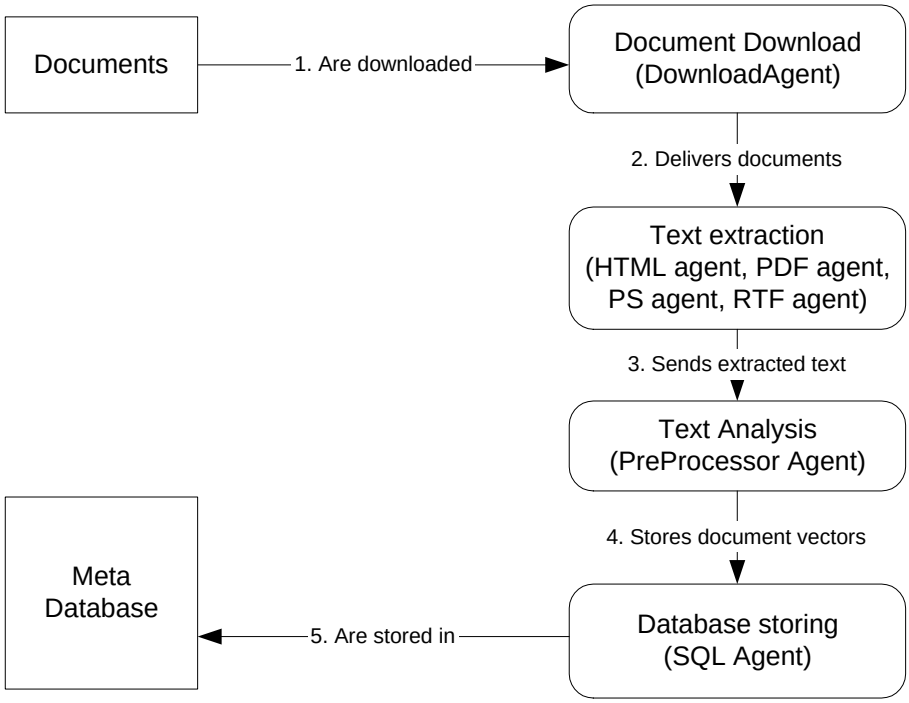

Figure 4.2: Data flow of the preprocessing marketplace

The preprocessing is nearly independent from the LSA and SOM, and can run in parallel on many CPUs (with one or more database servers). So it is possible to create many preprocessing marketplaces, for example for different content providers like libraries and universities. The data of all these marketplaces will be used later by a central marketplace to create the map (see Chapter 5).

4.1 Supplying Documents

There are many possible sources for documents, for example news tickers, news groups or databases. We have implemented an agent which retrieves

the documents from the probably most popular source: the World Wide Web (WWW). To start the download process with the Download agent one has to specify a text file containing a list of Uniform Resource Locators (URLs) which are to be downloaded. The agent verifies the URLs and starts downloading the first document. Every 2-5 seconds an new document is downloaded. The content of each document is stored in an array and - together with the URL - sent to the agents which extract the plain text. The Download agent sets up a goal containing all information and the content type of the document. On the basis of this content type the document processing agents can decide to accept or refuse this goal. If no agent can extract the text from a specified content type the goal will be rejected (and the document will not be stored in the database). The Download agent does not have to know which kinds of documents can be processed.

This approach demonstrates a great advantage of the agent-based technology because it is possible to add (or remove) text extracting agents for other document types without recompiling or reconfiguring the system. Document types which cannot be processed will be abolished and if a new agent can process one of them, its service will be used automatically.

If many Download agents are used in parallel for increasing the number of documents which can be downloaded, it may be useful to give them lists with documents from different servers because too many requests can decrease the performance of the server which provides the documents.

The ARISE system uses a collection of 1400 documents, mainly abstracts of theses and articles. The majority of these documents are Hyper Text Markup Language (HTML)-documents, because the performance of preprocessing was slowed down by other formats[1].

Sometimes documents of the same source but with different contents and subjects are considered as similar by the ARISE text retrieval system because these documents have large headers (proportional to their text length). The "similarity" of these documents consists only of these headers. A good example are the dissertation abstracts of the Free University of Berlin [DAR01]. The English abstracts have always a German header and that seems to be the reason, why documents with different subjects are clustered together in this case. The only way to solve this problem is the creation of special Download agents, which can filter headers. Unfortunately it is necessary to rewrite such an agent for every different website, web database, or content provider.

In the following sections the preprocessing technique will be explained with a small example. The following little HTML-document is delivered by the Download agent. It contains a title, an image, some text, and a table.

[1] text extraction of RTF, PDF, and PS is more complicated than of HTML

```
<html>
 <head>
  <title>News - German Chancellor in Washington</title>
 </head>
 <body>
   <h1>
      The German chancellor Schr&ouml;der arrives at
      5 o'clock.
   </h1>
   <table><tr>
       <td>
         <img src="picture.jpg">
       </td>
       <td>
         The German chancellor.
       </td>
   </tr></table>
 </body>
</html>
```

4.2 Extracting Plain Text

Only a few documents are in a plain text format, the majority (especially in the World Wide Web) exists in styled formats like Hyper Text Markup Language (HTML), Rich Text Format (RTF), MS Word, Postscript (PS) or Adobe's Portable Document Format (PDF). Some of them use tags, include pictures and tables (like HTML), and some can even be compressed like PDF.

We have implemented four agents which accept HTML, Rich Text Format (RTF), Postscript and PDF. These agents run on the same marketplace like the other agents, but in some cases they should preferable run on an own marketplace (on their own computer). Since each format is processed by a different agent, the preprocessing can be configured depending on the frequency of document formats. For example: the Postscript agent uses an third-party application[2] to extract plain text from the document. For every Postscript file to preprocess a script will be started once. Since it is time-consuming, the agent should run on its own marketplace, if many Postscript documents

[2]AFPL GhostScript 7.0 with the script "ps2ascii.ps"

have to be processed. Another very time-consuming text-extracting agent would be an agent for Optical Character Recognition (OCR).

Because JIAC IV allows the inclusion of Java components, existing third-party Java class libraries can be used to handle text and file formats. The PDF agent of ARISE uses such a library [Ety01].

Text extraction makes another advantage of JIAC IV (and Java) obvious: it is platform-independent. Some text-extracting algorithms run only on particular operating systems, for example the conversion of Word documents in plain text with Word macros. So it is necessary and possible to run different marketplaces on different operating systems to solve such problems.

The greatest advantage of the service-oriented structure of JIAC IV is that the agents can organize the text extracting process autonomously. The following scenario is possible: The Download agent gets a ZIP-Archive. Now the ZIP agent extracts the documents of the archive which will be processed by document agents (like a HTML Agent). These agents may extract the text and may ask an OCR Agent for extracting text from included pictures. So long as all qualified agents necessary to process these documents exist, they will cooperate to solve the problem of extracting the text from a document together.

In the example (see section 4.1) the HTML-agent would accept the goal (because the document has document type "text/HTML") and extract the plain text. HTML-tags for special characters have to be replaced by the characters, like the "ö" with an ö. Only the following text is extracted:

```
News - German Chancellor in Washington

The German chancellor Schröder arrives at 5 o'clock.

The German Chancellor.
```

The title is now a part of the text. The table is removed, but its containing text is still present. Images are ignored in our implementation, although it may be useful to analyze them with an OCR, because often the headlines of HTML-documents are designed as images.

4.3 Text Analysis

After extracting the plain text from documents these texts must be transformed into vectors. First all unimportant characters have to be removed. In our implementation this includes all punctuation marks (which are replaced with blanks) and even digits. Some special characters are replaced, like ä,

ö and ü with a, o and u. Apostrophes and hyphens have to be removed accordingly. The search should not be case sensitive, so all characters are transformed to lower cases.

In some cases it is important to keep digits, for example if the database has many documents about chemistry, because most chemical formulas include digits and would be corrupted without them. In our implementation only the 26 alphabetic characters are left after character removal. In the example, the number five (of the time), one hyphen, and one apostroph disappear. The "ö" in Schr"oder is replaced by "o". The example looks now like this:

```
news german chancellor in washington the german
chancellor schroder arrives at oclock the german
chancellor
```

Next the unimportant words have to be deleted. "Unimportant" are all words of a so called stop list, like filler words, articles, pronouns, prepositions etc. The list of unimportant words is called stop list and depends on the language. It is impossible to use the same stop list for different languages and therefore it is necessary to develop a language detection algorithm if different languages have to be supported. A suggestion for such an algorithm is made in section 8.1.

The creation of a stop list allows the reduction of the dimensionality of vectors and the amount of data, since they do not hold any information. But it is difficult to say which word is important and which is not. Most times the word "well" is only a filler word, but in another context it means the same like "fountain". The vocabulary of the stop list depends on the documents stored in the database. With the ARISE system it is possible to define own stop lists for every different application. The example looks like this after removing all unimportant words with the standard stop list (English) of the ARISE system:

```
news german chancellor washington german
chancellor schroder arrives oclock german
chancellor
```

The words "in", "the"(twice) and "at" have been removed. Then the frequency of all different words is counted. The concatenation of every two consecutive words is created and their frequency is counted too. A vector with all words and double words as components is created (see Table 4.1). Then the agent creates database statements to store these vectors in a database and sends these statements to the SQL agent.

Dimension	Frequency
german	3
chancellor	3
germanchancellor	3
news	1
newsgerman	1
washington	1
chancellorwashington	1
washingtongerman	1
schroder	1
chancellorschroder	1
arrives	1
schroderarrives	1
oclock	1
arrivesoclock	1
oclockgerman	1

Table 4.1: Preprocessing - example vector

Due to the agent-based technology it is easy to choose another approach (see Chapter 2) for creating vectors from a document. One have only to remove or disable the Preprocessor agent and replace it with another agent able to perform the desired transformation.

Stemming

The preprocessing algorithm is unable to detect inflections. For the ARISE system the word "computer" and its plural variant "computers" are different things. The same problem exists with verbs, "send" and "sent" are not the same for the algorithm. The results of the Latent Semantic Analysis can be enhanced by reducing all words to their basic form. This process is called stemming.

The best solution for this problem is the usage of a tool able to detect such inflections, a morphology analyzer like the German Canoo [Can01]. However, even these tools often suggest several alternatives, especially in languages like German. Of course a morphology analyzer makes the LSA more language dependent, because it can be used only for languages such a tool supports.

An alternative method is rule-based stemming. Prefixes and suffixes are truncated by using rules like: replace suffix "ator" with "ate" [3]. The best al-

[3] for example *navigator* and *navigate*

gorithm for English language is the heuristic stemming algorithm introduced by Porter [Por80] [Por00]. For some other languages similar algorithms exist (a Dutch version for example is introduced in[KP94]), but for languages with a very complex grammar, like German, these algorithms are more complicated [Cau99].

In addition it makes sense to replace acronyms and abbreviations with the term they stand for. If one text contains the phrase "Latent Semantic Analysis" and another one contains the acronym "LSA" the LSA is probably not able to recognize that both means the same. The problem is, that there are different meanings for many acronyms. For the abbreviation "SOM" the Acronym Finder [Acr01] delivers 26 different meanings. The only way to handle this problem is to compare all results with the text. In most cases the author uses the full term of the acronym at least once (to explain the acronym) and so the right term can be found.

Due to the large complexity and the language dependency the ARISE system has no stemming algorithms or acronym replacement. One possible way to integrate a stemming feature would be the creation of a new agent, which provides the stemming as a service for other agents. If it gets words in a language it does not support, it only returns the word.

4.4 Database

The large amount of data produced by the preprocessing needs to be stored in a database. From this database other agents can read the vectors to analyze them.

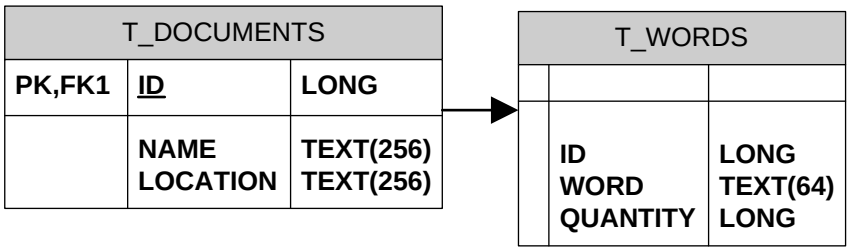

Figure 4.3: Preprocessing data model

4.4.1 The Data Model

The meta database is a relational database with a very simple structure (see Figure 4.3). It consists of two tables: *T_DOCUMENTS* and *T_WORDS*.

T_DOCUMENTS contains information about the documents, i.e. the title and the location together with a unique document ID as primary key. The second table, *T_WORDS*, contains for each different word of each document how often it appears in this document. The document ID is the foreign key and the same as in *T_DOCUMENTS*.

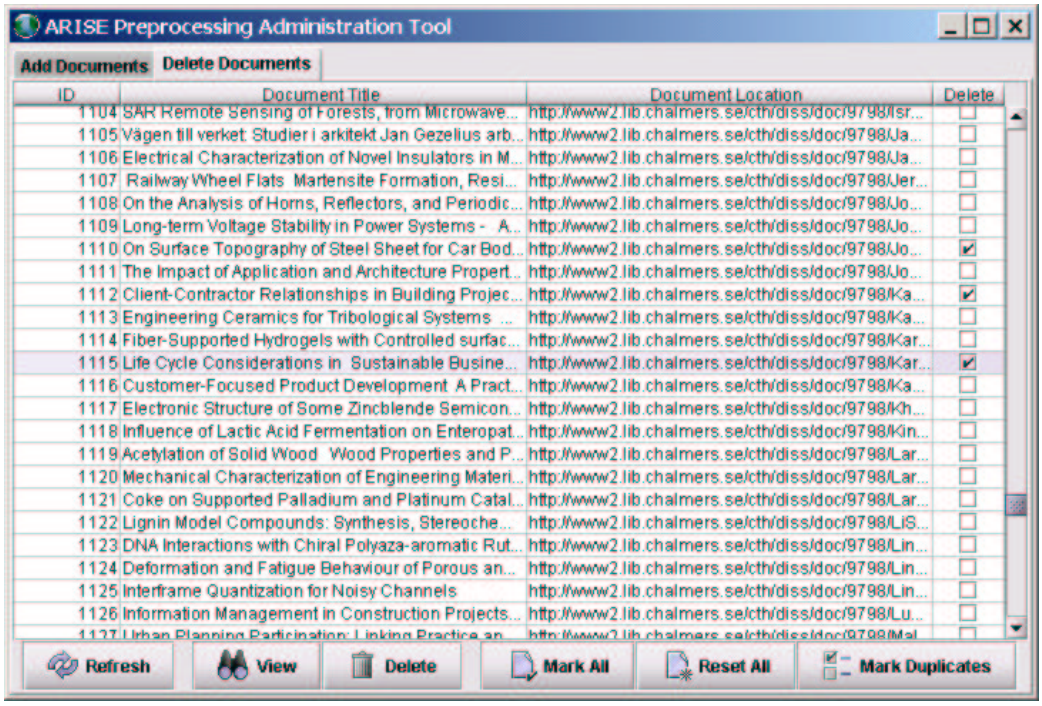

Figure 4.4: The admininstration console of the preprocessing marketplace

4.4.2 The SQL Agent

To access databases, the agents of the ARISE system can use the so called Structured Query Language (SQL) agent. SQL, the Structured Query Language, allows users to access data in relational database management systems, such as Oracle, Sybase, Informix, Microsoft SQL Server, Access, and others by allowing users to describe the data the user wishes to see or update. The SQL agent allows sending of simple updates and queries (represented by Java String objects) to the database. The connection between this agent and the database is established using SUN's Java Database Connectivity API (Java Data Base Connectivity (JDBC)). The JDBC technology allows the use of a wide range of SQL databases. Since only simple functions from the JDBC API are used for the SQL agent, it works with any JDBC-1.0-driver.

The results of a query are serialized[4] and send to the agent sending the query[5]. Since the creation of result objects from a database query is very time-consuming, the SQL agent should - for queries - run on a separate computer to use the full advantage of a distributed system.

4.5 Administration

To handle the preprocessing a special administration agent has been implemented. It allows the start of the preprocessing assembly line and the handling of documents stored in the meta database (see Figure 4.4).

It is possible to view documents stored in the database and to delete some (or all) of them. As a special feature, documents with the same URL can be searched. In this case, only the older document is marked for deleting. This mechanism allows a simple update of documents by preprocessing documents again and deleting the old versions afterwards.

[4] i.e. to bring objects in a sequential form (for communication or storing)

[5] in a special self created Java class: the *SerializableResultSet*

Chapter 5

Mapping

In this chapter the mapping marketplace is introduced. The mapping marketplace is the heart of the ARISE system (see Figure 5.1). It is named "mapping" because it contains the functionality to create the Self-Organizing Map (SOM). During the preprocessing a vector was created from every document. The vector space is very high dimensional and has to be dimensionally reduced by the Random Mapping (RM) Agent.

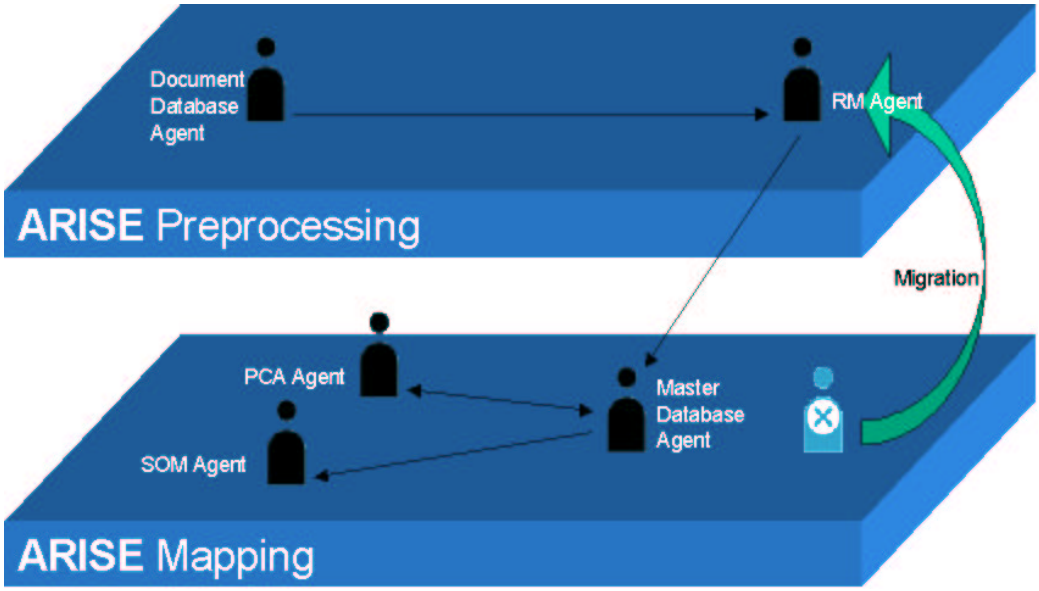

Figure 5.1: The ARISE mapping marketplace

Then the Principal Component Analysis (PCA) is used to identify an optimal low-dimensional[1] subspace in the dimensionally reduced vector space

[1] e.g. 10 dimensions

(PCA Agent). Details about the PCA are described in the section 5.2. Finally, the output vectors of the PCA are used to train the Self-Organizing Map of the SOM Agent. Every agent (except the SOM Agent) stores its results in a central database (see Figure 5.2).

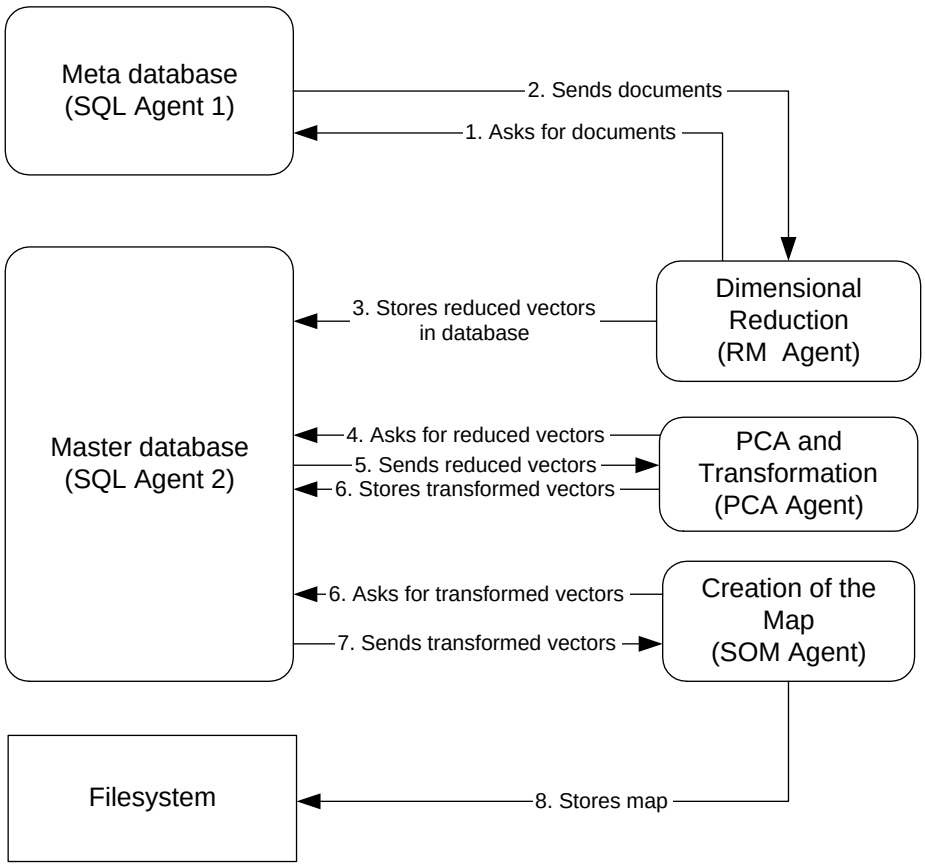

Figure 5.2: Data flow of the mapping marketplace

The ARISE system contains two different types of database: a document database and a master database. The first one contains the preprocessed vectors (see section 4.4). There may be many databases of this type, because documents can be preprocessed in parallel on several marketplaces. The master database can exist only once in the ARISE system. It stores the vectors after dimensional reduction. Since the PCA needs all vectors to analyze the principal components, they have to be stored in the master database. The mapping market contains at least two database agents for the handling of both databases.

Although the mapping marketplace is one logical unit, it is possible to split it physically, especially the PCA agent and the SOM agent should be

installed on different JIAC IV marketplaces (and engines), if the number of documents is very high, or exceeds the hardware capacity[2].

In the ARISE system the similarity of two documents is measured by calculating the (usually Euclidean) distance between the two vectors representing them. If only the absolute frequency of the words in a document corresponds to the components of its vector, document vectors created from small documents containing only few words (e.g. small abstracts, queries) are close to the null vector, and are therefore dissimilar to document vectors of larger documents, although they may be similar in content. Therefore all values are divided by the total number of words, i.e. the relative frequency of each word in the document is calculated (see example in Table 5.1). This weighting causes the components of each document vector to add up to one, which results in document vectors having more similar lengths, independently of the document length.

Term	**Frequency**	**Relative Frequency**
german	3	0.1428
chancellor	3	0.1428
germanchancellor	3	0.1428
news	1	0.0476
newsgerman	1	0.0476
washington	1	0.0476
chancellorwashington	1	0.0476
washingtongerman	1	0.0476
schroder	1	0.0476
chancellorschroder	1	0.0476
arrives	1	0.0476
schroderarrives	1	0.0476
oclock	1	0.0476
arrivesoclock	1	0.0476
oclockgerman	1	0.0476

Table 5.1: Calculating the relative frequencies

5.1 Random Mapping

Now the dimensionality of the document vectors has to be reduced, even if the document has only some words. Although the example from Chapter 4

[2]e.g. 10000 documents may be to much for an 1-GHz-PC with 256 MB RAM

seems to have only 15 components it has in fact an almost infinite number of components, but most of them have the value zero. For a better understanding consider the following example: The example vector from Chapter 4 (vector 1) has 15 dimensions and the document vector of the phrase "latent semantic analysis" (vector 2) has 5. But if both vectors have to be in the same space, this space has 20 dimensions, although some components are "empty" in each vector (see Table 5.2).

Dimension	**vector 1**	**vector 2**
german	3	0
chancellor	3	0
germanchancellor	3	0
news	1	0
newsgerman	1	0
washington	1	0
chancellorwashington	1	0
washingtongerman	1	0
schroder	1	0
chancellorschroder	1	0
arrives	1	0
schroderarrives	1	0
oclock	1	0
arrivesoclock	1	0
oclockgerman	1	0
latent	0	1
semantic	0	1
latentsemantic	0	1
analysis	0	1
semanticanalysis	0	1

Table 5.2: Two vectors - more components

Random Mapping is one technique to reduce the dimensionality of the space containing the vectors. It is also possible to use the Principal Component Analysis but Random Mapping has two important advantages. It is faster and can be used online (i.e. without knowing all other vectors). The second advantage is very important, if preprocessing occurs distributed among many computers, because the Random Mapping can reduce vectors in parallel without considering other vectors.

The RM Agent - a Mobile Agent

As described in Chapter 4 the preprocessing can occur on different marketplaces running on different systems. The Random Mapping reduces the space dimensionality (the information), so it makes sense to place the RM agents (RM for Random Mapping) on the preprocessing marketplaces. This design has two advantages: parallel processing of the documents is much faster then central processing at the mapping marketplace and the dimension reduction decreases the network traffic, too.

In some cases these advantages may become unimportant, for example if the central marketplace runs on a very powerful environment, only one preprocessing marketplace exists, and/or the network is very fast. In this cases the agent should be central placed on the mapping market.

The most flexible solution is a mobile agent which can be sent to any marketplace. The preprocessing marketplace can allow the migration of a RM agent from the central marketplace, if it trusts this marketplace. The meta data can be used by different search engines (not only for LSA) and a content provider can make its repository public, allowing agents like the RM agent to migrate and use their databases. If the content provider does not allow migration (for example because the capacity of the runtime environment is not powerful enough to run so many agents) the meta data can be read from the central marketplace of the search engine.

ARISE allows the administrator to send the RM agent to any marketplace he wishes. He specifies the meta database and the URL of the target marketplace. Then the agent migrates, reduces the dimensionality of all vectors, sends them to the database of the central marketplace and travels back to its home marketplace. It is possible to specify a start ID, so that only documents with a greater ID are reduced and sent. This is an effective way to make an update of new documents from a meta database, it is not necessary to reduce preexistent document vectors again.

5.2 Principal Component Analysis

The objective of the Principal Component Analysis (PCA) is to reduce the dimensionality of vectors and to identify new meaningful underlying dimensions (variables). The idea is to find a lower-dimensional subspace of the high-dimensional vector space where all document vectors can be projected onto with a minimum of information loss. Figure 5.3 shows the geometrical interpretation of an example: Two dimensions X and Y show a relation (Figure 5.3 a), with an increasing of X there is an increasing of Y, so the Principal Component Analysis identifies a one-dimensional subspace where

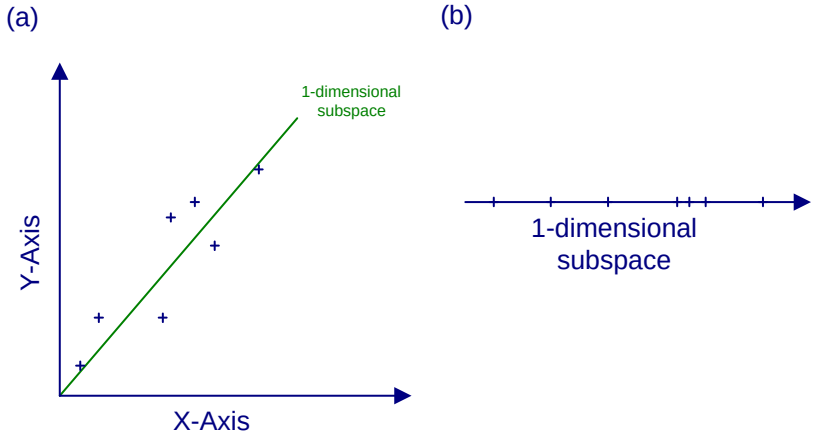

Figure 5.3: An PCA example: a 2-dimensional data set is projected in an 1-dimensional subspace

all points can be projected onto (Figure 5.3 b).

For the LSA that means very simplified worded: If the PCA recognizes, that in many documents an increasing of the quantity of one word causes the increasing of the quantity of another one, both are "merged" to one dimension. This is the intention of the LSA. Words with relations between them, like "botany" and "plants" have to be set to one dimension. So two texts would be similar, if one often contains the first word and the other the latter.

It is clear that this technique cannot be implemented as an online algorithm, because if relations between components of vectors have to be found, these vectors must all be known. So the agent which performs this analysis has to be placed on the central marketplace.

The PCA Implementation

Since the theory of the PCA and the most details of the implementation are discussed in the diploma thesis of Richard Cissée [Cis01] (who implemented the algorithm), only the important and interesting details about the PCA agent in the ARISE system are discussed in this subsection.

The first step of the PCA is the creation of the so-called covariance matrix. Since all vectors have to be considered the creation is very memory consuming. In the ARISE system the PCA agent gets all vectors successively from the database, so memory is saved but the process is rather time-consuming, because every vector has to be read a second time from the database at the

end of the PCA. The covariance matrix is stored in a text file, because it is needed again for the querying marketplace.

The next task is determining the eigenvalues of the covariance matrix. There a two possible ways, a numerical one and an approach with an feed-forward single-layer neural network. These networks are much faster with high-dimensional input vectors, but because the dimensionality is already reduced by Random Mapping, the first technique is sufficient, in fact it is faster than the neural network for small (lesser than 1000) dimensions.

After determining the eigenvalues of the covariance matrix the new vectors can be created. For this every input vector has to be read a second time (because they are not stored at the agent). The output vectors are stored in the database, although it would be possible to send them directly to the SOM agent. But especially if some parameters have to be tested[3] it is useful to have the vectors of the PCA stored, because then there is no need to start this process a second time. Another reason is, that the very memory-expensive SOM agent can be started stand-alone if there are memory lacks.

5.3 Self-Organizing Map

After the vectors are reduced and the principal components are determined the creation of the Self-Organizing Map starts. The SOM is a neural network which is trained with the output vectors from the PCA.

The implementation of the algorithm is part of the diploma thesis of Richard Cissée [Cis01]. We decided to store the final Self-Organizing Map in an Java serialized object, so that other agents may use the results of the Latent Semantic Analysis without having access to a database server. A typical scenario is the querying marketplace as described in the next chapter. The serialized objects contain not only the network, but also they hold information about every document (title and URL).

The database could be theoretically erased after the creation of the map, but in most cases it is useful to keep at least the vectors which are reduced with Random Mapping, because they can be reused for updates. In some cases all data can be deleted, for example in news systems, which have to update articles every day.

[3] like the neighborhood radius of the SOM

5.4 Database

The large amount of data that arises during the mapping process makes it advisable to store intermediate results. Especially the reduced vectors after being processed by Random Mapping have no dependencies from other vectors and do not have to be recalculated each time after an update with new documents. To access databases, the agents can use the same agent as on the preprocessing marketplace: the SQL agent.

5.4.1 Data Model

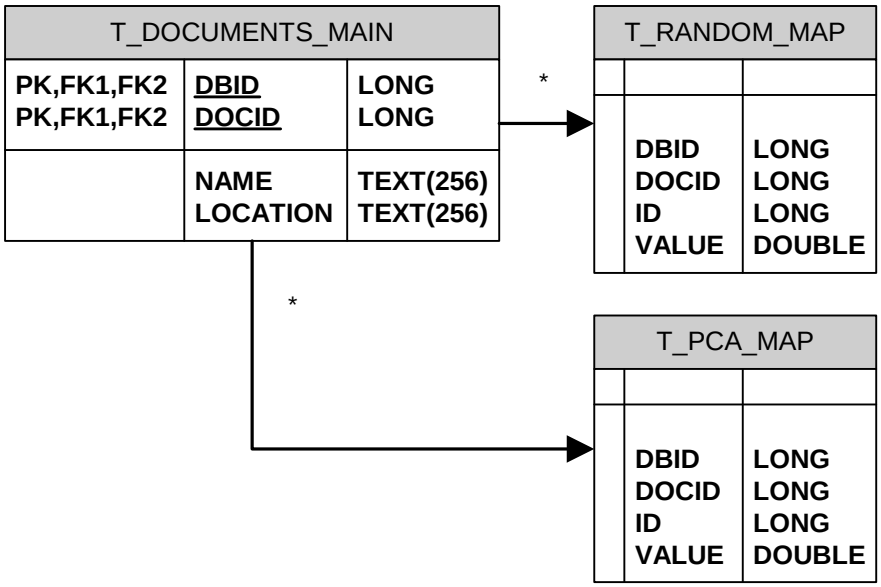

Figure 5.4: Master data model

The data model of the ARISE master database consists of the following three tables: *T_DOCUMENTS_MAIN*, *T_RANDOM_MAP* and *T_PCA_MAP*. The table *T_DOCUMENTS_MAIN* contains information about the documents, i.e. the title and the location, the ID of the document database (zero, if only one document database exist) and an unique document ID. The table *T_RANDOM_MAP* contains for every document the values in each dimension (*ID*) after the Random Mapping process. The database ID and the document ID are the same as in *T_DOCUMENTS_MAIN*. This ID consisting of two parts is necessary to allow the parallel work of several RM agents without synchronisation. It guarantees that different documents never get the same

ID, even if they have the same one on different preprocessing marketplaces. The database ID have to be set by the administrator of the ARISE system. The table *T_PCA_MAP* contains the document vectors after the PCA.

5.5 Administration

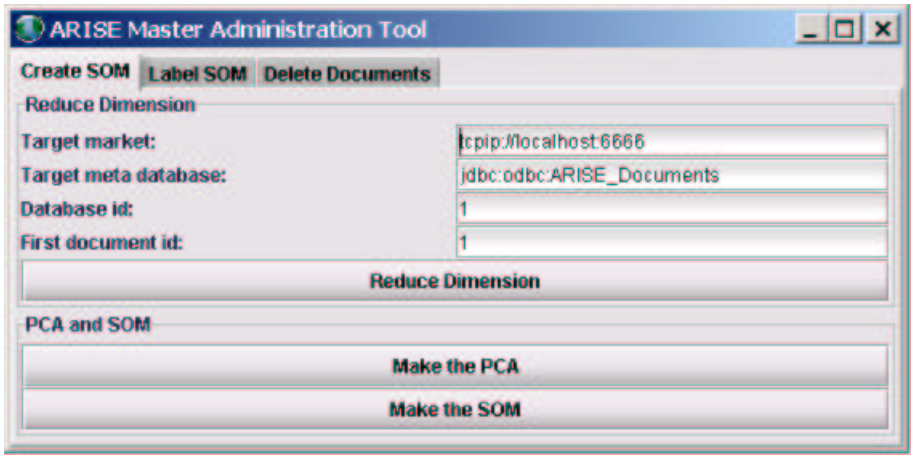

Figure 5.5: The administration agent

To handle the creation of SOMs a special administration agent was implemented (see Figure 5.5).

For Random Mapping the target market and the target meta database have to be set, the administator has to choose a database ID and to determine the start ID. Only documents with a higher ID will be reduced (see section 5.1 for details). The parameters of the PCA and the SOM (dimensionality) cannot be altered without a restart of the system because any change during the run of system can endanger the system integrity of ARISE.

Finally, it is possible to view documents stored in the database and to delete some (or all) of them, like in the Administration agent of the preprocessing market. Documents with the same URL can be searched.

Labeling the SOM

Labels are tools which facilitate the interpretation of a map by a human user by adding some descriptive words. These words summarize aspects of the area containing the label and act as landmarks. In this way, an easy orientation is provided.

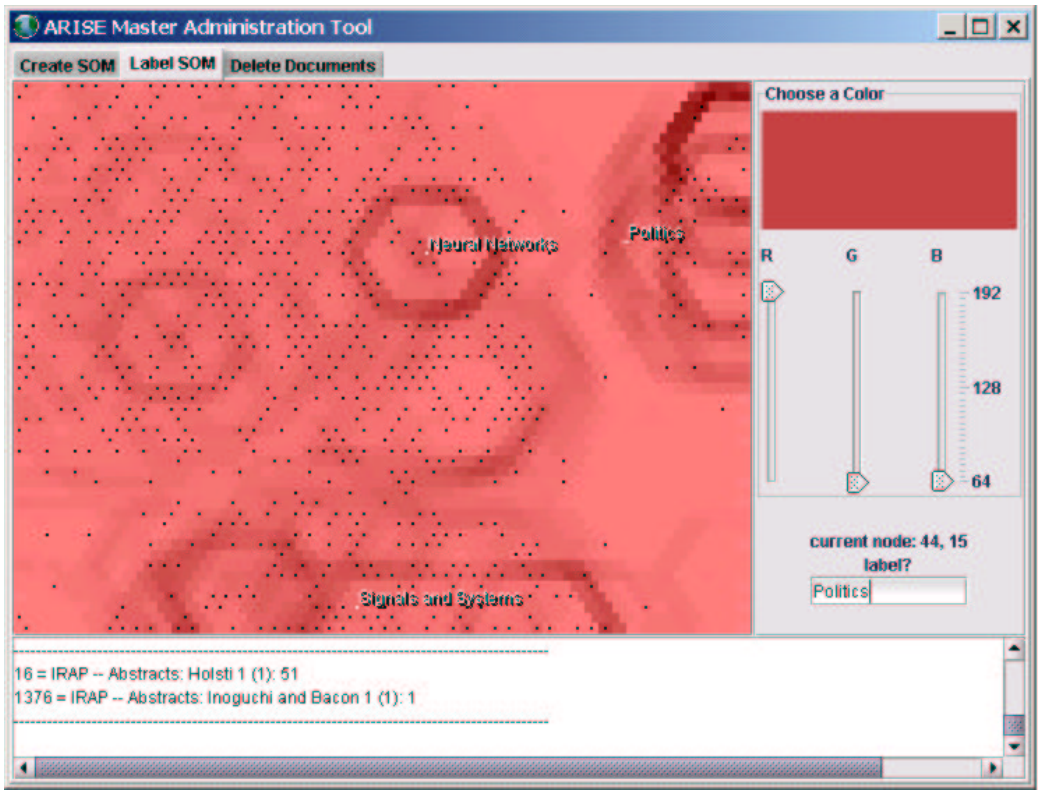

Figure 5.6: Creation of labels with the administration agent.

With the Administration agent the administrator can set labels for the current SOM. The graphical user interface allows him to view all documents and set new labels at prominent points (see Figure 5.6).

Of course there exist some techniques of automatic keyword labeling. One was introduced by Kaski and Lagus [LK99] in 1999. These technique allows the creation of keywords for clusters (implying a cluster structure exists) or map areas. A labeling is only necessary for human users, for most agent applications (see section 8.1) it is not necessary because only the agents will use the maps. Due to this and to the fact, that the techniques for automatic labeling are expensive, a manual labeling system was considered to be sufficient for the purposes of the ARISE system.

Chapter 6

Querying

After preprocessing the documents and calculating the map the user must be provided with the possibility to send queries to the system and view the search results. A third marketplace executes this service. In order to do this, an interface is needed to allow human users to interact with the multi-agent system. Such an interface is introduced with the DAI Navigator in section 6.2.

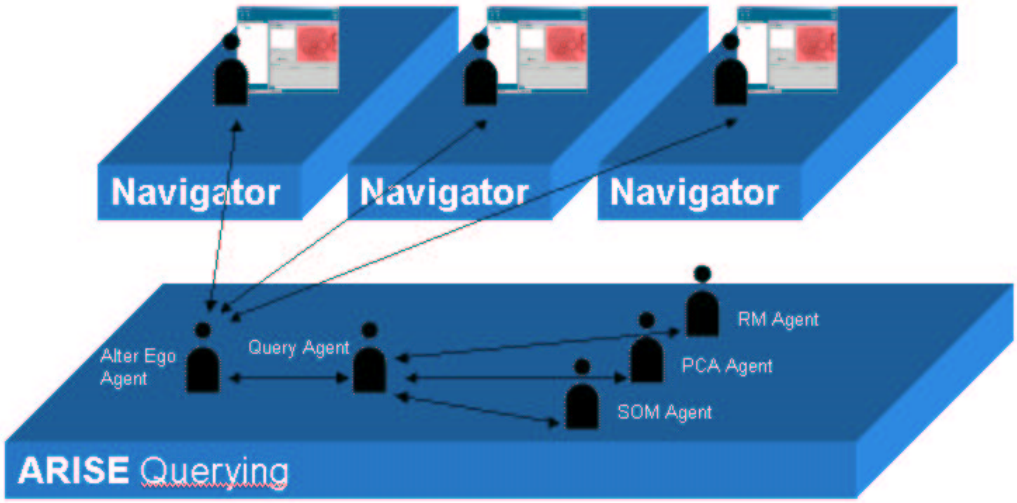

Figure 6.1: The ARISE querying marketplace

On the querying marketplace there are three already known agents: the SOM agent, the PCA agent, and the RM agent (see Chapter 5). Every query is handled as a new document (with the exception that it is not stored in the database). Additionally, there are two new agents on this marketplace. The

query agent is responsible for managing the queries. The alter ego agent acts as an interface between the DAI Navigator and the agent system and will be explained in section 6.2.

Since all relevant information is stored in two files (covariance matrix and SOM) this marketplace can run without a database server. It is therefore possible to create a very small server (with low memory and CPU capacity) without the other marketplaces (preprocessing and mapping). As a consequence, the files may be even downloaded by a mobile communication device like a PDA and then used locally.

Updates of the querying market are very simple. It is possible to overwrite the PCA matrix and SOM files with newer versions even when the system is running. Whenever the agents recognize a change on these files, they update their data. The simple handling of updates is important for systems which often generate new SOMs, for example in systems generating newsletters every hour (see section 8.1 for details).

6.1 Processing of a User Query

The probably most interesting agent of the querying marketplace is the Query agent, which controls the handling of user queries. It generates a vector from the query string in a similar way as described in section 4.3 (it is not necessary to extract the plain text before because the queries are already written in plain text). Then it sends this vector to the RM agent, PCA agent and SOM agent (in this sequence) and gets the Best-Matching Unit (BMU) as shown in figure 6.2. The BMU is the node of the SOM neural network, which is most similar to the document vector.

6.2 DAI Navigator and Multi-Access-Point

The handling of queries is organized in a client-server-architecture with a central server storing the maps and calculating the queries and a graphical user-interface as client. There are two possible approaches for such an architecture: The "thin client" is a low-cost, centrally-managed client component. The whole functionality is provided by the server, which handles all interactions. The greatest advantage of this architecture is, that it has minor system requirements on the client side. For example, web interfaces (without Java or Javascript) are "thin clients" allowing a user to access programs with a simple web browser. The "thin client", however, has three main disadvantages. It produces a large amount of network traffic between server and client since every function called by the user results in client-server-communication.

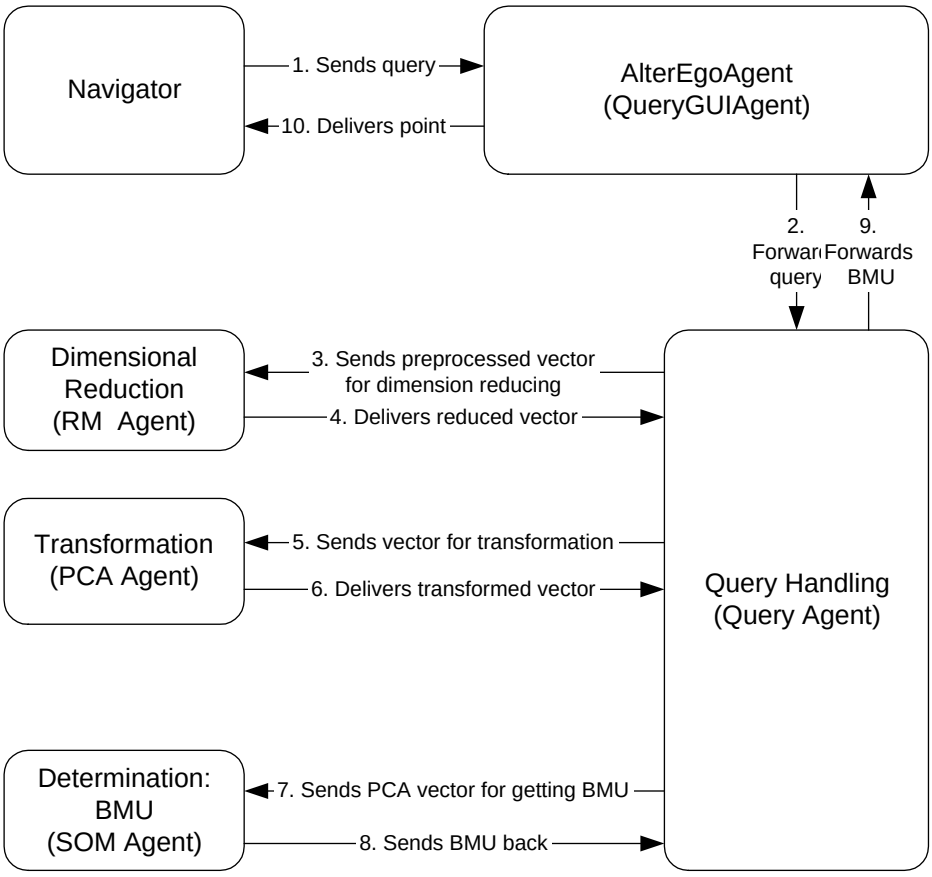

Figure 6.2: Data flow of the querying marketplace

In addition, this very communication makes this approach slow, especially if used with low-performance communication channels. Finally, thin clients cannot provide rich functionality, for example, HTML-Forms provide only some simple input components.

The other approach is called “thin server”. As the name indicates, this architecture has a more complex client and the major part of functionality is on client side. Often they are only started by the server and run stand-alone (like applets). They are faster as “thin clients”, because simple interactions do not cause communication with the server. They do not overload the server and can offer all possible user input components. But “thin server” architectures have disadvantages, too. The most striking one is the larger amount of system requirements on the client side. This includes additional non-standard software and hardware components. Often the client has to be installed locally before using it, otherwise the invoking of the clients would be very time-consuming.

The DAI Navigator is a hybrid of both architectures. Technically it is an agent which allows a human to use services through a Java Swing interface (see Figure 6.3). It was developed at the DAI-Lab[1] for invoking complete GUIs by downloading the GUI files[2] from the server. So it is possible to use multiple instances of GUI services without installing the files of the service provider locally.

Since it needs the installation of a Java Runtime Environment and the JIAC IV platform the DAI Navigator is no real "thin client". However, the using of Java and Swing allows the creation of complex GUIs and functionalities without communication with the server. It is therefore logical to develop a "thin server" architecture when implementing a Navigator interface.

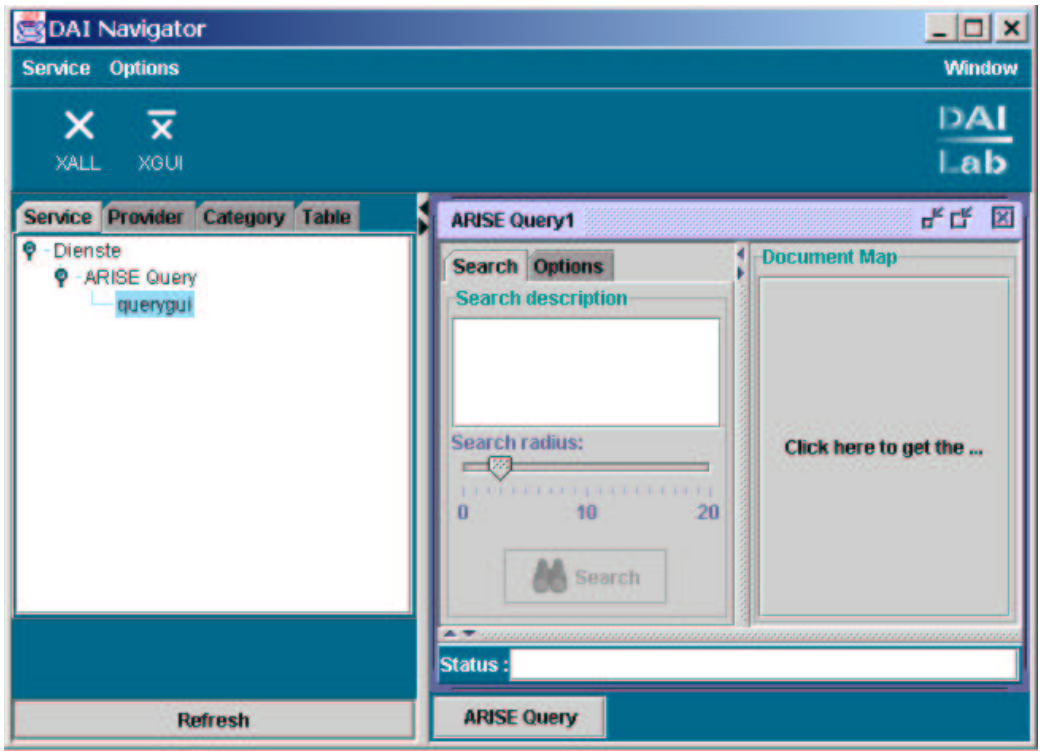

Figure 6.3: The DAI Navigator

Currently a new interface for human-to-agent communication with JIAC is being developed. With the Multi-Access-Point (MAP) it is possible to create a specification for a graphical user interface with XML [ABF01]. With this specification the Multi-Access-Point creates the interface in HTML or

[1]DAI - Distributed Artificial Intelligence
[2]Java classes

WML, depending on the browser[3] preferred by the user. With the MAP a "thin client" architecture is necessary since only simple input-output-relations are possible.

Since this MAP is highly experimental and unstable our implementation is based on the DAI Navigator, although it would be nice to use the ARISE system via webbrowser. To use the full advantages of the DAI Navigator the ARISE query client is implemented in a "thin server" architecture.

6.3 The Graphical User Interface

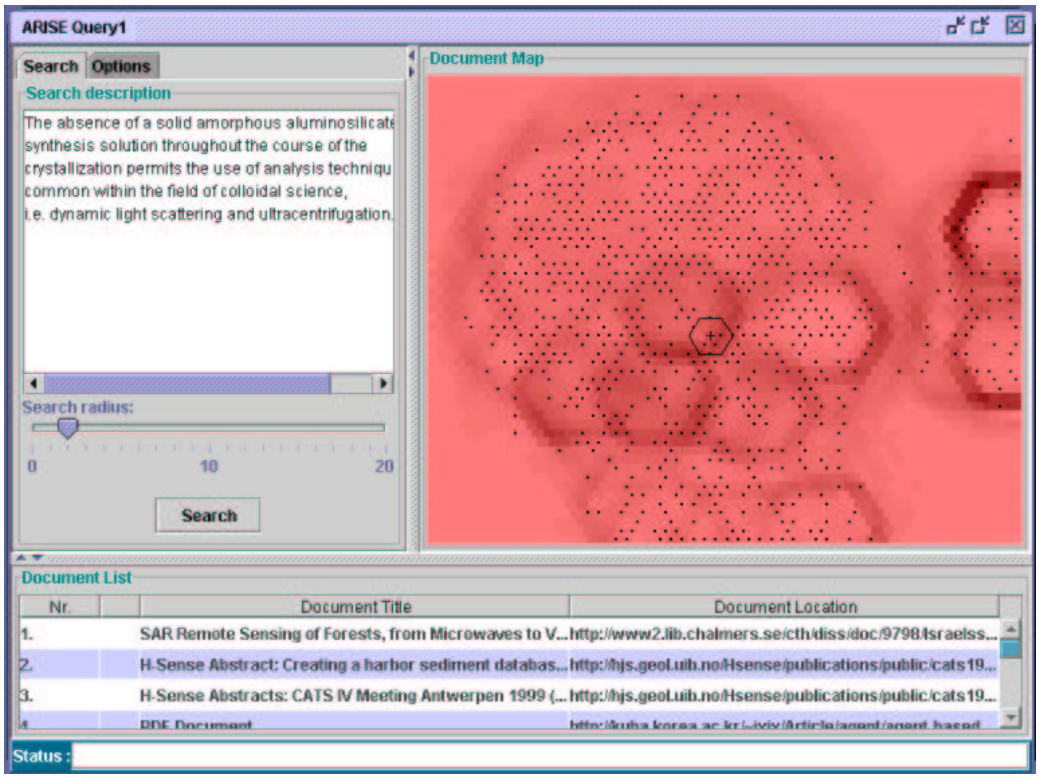

Figure 6.4: The ARISE graphical user interface

The Graphical User Interface of the ARISE system is very easy to use. One panel contains the map (SOM), which has to be loaded from the alter-ego-agent. At this panel the user can choose a position with the mouse. As a result documents in the neighboring area will be presented in a table at the bottom of the GUI. With a double click at the documents of the table a

[3]any webbrowser or WAP device is possible

document viewer is started and shows the document content. It is possible to control the number of shown documents indirectly with a search radius slider. The presentation of the document list is very quick (only some milliseconds) because all information is stored on client side.

Everytime a query has to be processed, it is sent to the agent system, which returns the coordinates in the map. On the left side there is a panel for user queries, it is possible to paste text from the system clipboard, so that complex texts can be copied from other locations.

Every instance of the GUI gets the current version of the SOM. The DAI-Navigator does not provide a concept for session handling, so it is not possible to inform active clients about new SOMs.

Chapter 7

Evaluation

In this chapter the results of the implemented approach are evaluated with a special focus at the scalability, reusabilty and performance of the implementation.

7.1 Reusability

During the implementation phase of the ARISE system a main focus was set to the reusability of the system and its components, services and agents. The JIAC components contain Java classes which can be reused for other purposes.

The agents can be easily integrated in other projects. Especially some infrastructure agents can be reused in new JIAC IV applications. This includes the SQL agent for updating and querying a database and the download agent for getting documents from the WWW. All parameters of the agents of the ARISE system can be controlled by agent configuration files, so no recompilation is necessary in order to use them in another context. For example, by changing a few lines in the configuration file of the SQL agent it can be used for every database (provided it exists an applicable driver):

```
de.dailab.jiac41a.arise.sql.bean.SQLBean.driver=
                          sun.jdbc.odbc.JdbcOdbcDriver
de.dailab.jiac41a.arise.sql.bean.SQLBean.database=
                          jdbc:odbc:ARISE_Documents
```

The agents of the information retrieval system (SOM agent, RM agent and PCA agent) can also be reused, although it is not likely that they really be used for really other purposes. However, the AI methods can be

used for other purposes, Principal Component Analysis (PCA) and SOM are techniques, which can be used for various forms of data mining. The Random Mapping is well suited for dimension reduction and can be used for information compression.

Finally, the whole ARISE system can be embedded in another agent application. A suggestion for an application, a personal newsletter system, is made in section 8.1.

7.2 Scalability

One important aspect of every implementation is the possibility to scale it depending on the complexity of the problem to solve. For the ARISE system a multi-agent platform was used to handle scalability. As described in the previous chapters, most parts of the LSA can be distributed. The JIAC IV multi-agent platform is well suited for the distribution of the ARISE system, because all communication between agents and components is organized by JIAC IV, no further implementation is needed.

The costs of preprocessing increase linearly with the number of documents one has to preprocess. The tests with the ARISE implementation use a collection of 1400 documents from different sources. Preprocessing can be executed in parallel on different marketplaces (on different computers), so the performance of the system is increased with the addition of new preprocessing marketplaces. If several marketplaces overextend the database server, it is even possible to use several databases. One possible bottleneck in the ARISE system is the download of documents, especially if the network has only a slow internet connection. If some agents are too slow, it is possible to place them on their own marketplace on another computer (for further details see Chapter 4).

The mapping marketplace has not such a good scalability as the preprocessing marketplace. As described in Chapter 5 it is possible to run the Random Mapping (RM) agent in parallel in the same way as the preprocessing agents. But the Principal Component Analysis (PCA) and the creation of the Self-Organizing Map (SOM) cannot easily run distributed. Especially the costs for the SOM increase rapidly with an increasing number of documents. One possible solution to decrease the costs of this algorithm is a so called Hierarchical Self-Organizing Map (HSOM), which will be explained in section 8.1. The PCA costs increase nearly linearly, but since it cannot be computed in parallel, the current hardware is the bottleneck for this algorithm. Admittedly it is possible to compute the covariance matrices (see section 5.2) in a distributed way and add them centrally, this may decrease the execution

time of this part of the PCA slightly. But even with these modifications both techniques (PCA and SOM) are the most time-consuming parts of the ARISE system. The SOM agent needs almost half an hour for the creation of the map for a collection of 1400 documents[1] by an input vector dimensionality of 10.

For the querying marketplace the complexity of the document space is almost unimportant. Only the transmission of the serialized SOM following each service request increases, but this problem can be solved with the use of a Hierarchical Self-Organizing Map (HSOM) (second layer SOMs can be loaded dynamically only if they are needed). The time for searching the Best Matching Unit (BMU) increases but is still very small. It is possible to install several querying marketplaces to handle a large amount of queries, provided that every marketplace has the same SOM. The distribution of queries can either be handled by the users (which choose among the service providers) or by an agent, which distributes the queries over the marketplaces, depending on their workload. Such an agent has still to be implemented.

Multiple Users

The query service is available for several users. With increasing number of users the time lag[2] increases, too. The most simple way to handle this problem is invoking new instances of the querying marketplaces (i.e. increasing the number of query service providers). So the user can choose a provider and if it is to slow, he can try another one. An more elegant possibility is to invoke only one alter-ego-agent, and enhance its capabilities so that it can choose a provider which can handle this query. So it can distribute the load of queries and the user does not have to choose a provider manually.

7.3 Performance

Although the performance of Java has increased in the recent years, Java-based applications are still slower than platform-dependent programs written in C/C++ [Sos99]. Of course there are some possiblities to increase the performance which are actually used in the ARISE implementation (see [Jav01]). This lower performance is the price for platform independency, i.e. the possiblity to use the same application on different systems and architectures. Since JIAC IV bases, like some other multi-agent systems, on Java, its performance is not very high.

[1] Hardware: PC, Athlon 1 GHz, 256 MB RAM

[2] i.e. the time difference between user query and the incoming of the query result

The collection of 1400 documents used in this thesis can be processed with the single CPU system described in the previous section in 2.5 hours. Since JIAC IV applications can be implemented as distributed systems it is possible to increase the performance by running some tasks in parallel and computing time-consuming processes on different CPUs.

Chapter 8

Conclusion

As a result of this thesis the ARISE multi-agent system was implemented. It is a text retrieval system using AI methods which allows the searching, visualization and exploration of document collections. It analyzes the document contents and uses maps to visualize the relations among the documents. Due to the complexity of the text retrieval algorithms the implementation relies on system distribution.

In this thesis it has been shown that the multi-agent platform JIAC IV is an effective solution for problems needing a distributed architecture. Due to the distribution the implemented solution is very scalable. The loss of performance caused by the use of a Java-based agent toolkit can be compensated by running time-consuming parts of the system in parallel.

Furthermore JIAC IV agents are well suited for searching and handling of documents of various types due to the inclusion of Java components[1], and their basic problem solving mechanisms. These mechanisms allow the nearly self-organized text-extraction from documents.

The ARISE system is easy to reuse, it can be included in more complex applications (see 8.1) or used stand-alone. Furthermore it provides some generic functionality, like database access, document processing, and download from the World Wide Web (WWW) that are reusable for various applications.

The Latent Semantic Analysis in combination with SOMs represents an important enhancement of the abilities of intelligent agents. Since agents are meant to reproduce human behavior and thinking to some extent, this information retrieval system is a natural step in this direction. The concept of "similarity" of texts is very important for agents which have to act in favour of humans.

[1] allowing the usage of various Java class libraries for different text formats

8.1 Outlook

Inspite of the very good results achieved with the ARISE system, there is still room for improvement. As described in the introduction, "normal" text matching search engines combine several algorithms to enhance their results.

Naive Bayes

Keyphrases are words or phrases which are very important for the meaning of the text. Usually authors add a litte list of keyphrases manually, but assigning keyphrases to existing documents is very laborious. For this there exist some approaches for keyphrase extraction based on the naive Bayes learning scheme [FPW+99]. This approach may be a good supplementation for the ARISE system, since it can be used for automatic labeling and to introduce a weighting scheme into the LSA.

Stemming

As described in section 4.3 the ARISE system is unable to detect inflections. There exist stemming tools for several languages. Some demos can be tested at LingSoft [Lin01]. These tools could be used to detect and replace inflections. The easiest way to integrate a stemming algorithm would be the Porter's algorithm, but there exist variants for a few languages only and it is an heuristic algorithm.

Hierarchical Self-Organizing Maps

In section 7.2 the problem of the increasing computing time for SOMs was discussed. One possible solution are multilayer or Hierarchical SOMs [Lam92]. In a multilayer SOM documents are organized in a hierarchical structure. The first layer is a SOM with only a few hundred nodes which separates the documents in many groups. Subsequently a second SOM is trained for each group. This means that on the first layer of the Hierarchical Self-Organizing Maps (HSOM) one obtains a rather rough representation of the input space but with descending hierarchy the granularity increases. For every layer the Best Matching Unit (BMU) has to be computed in case of a user query.

Such an approach is especially well suited for the representation of the contents of a document collection. The reason is that document collections are inherently hierarchically structured with respect to different subject matters (this is essentially how conventional libraries have been organized for centuries).

Multiple Languages

A problem is the handling of multiple languages. The ARISE system supports the usage of different stop lists (see section 4.3), but it is difficult to create a stop list usable for multiple languages simultaneously. It makes sense to first determine the language of a document and store documents different languages in different SOM's.

There are several methods to determine the used language, but it is also possible to use the ARISE system, if used in the following way: A SOM has to be trained with example documents from all languages (the same number of documents for every language). Because documents of one language are more similar than documents of different languages, the resulting clusters represent the languages. It is not necessary to initialize a stop list for this process, because "unimportant" words are characteristical for a language. With a cluster analysis it is possible to decide, to which cluster a new document belongs, i.e. in which language it is written. This is a special form of a multilayer SOM, because the first layer only represents the language, the second one the documents written in this language.

Of course, the implementation of a LSA-based text retrieval for multiple languages is a very complex problem. Beside creating SOMs, stoplists and stemming algorithms for every language it is necessary to implement a cluster analysis.

Web Interface

During the implementation of the ARISE system a new concept was developed at the DAI-Labor. The Multi Access Point (MAP), introduced in section 6.2, is highly experimental and some features needed for a search engine with a graphical representation are still not implemented, but it will soon be possible to implement a web interface for the ARISE query service.

Applications

Although the ARISE system can be used as a prototypical stand-alone search engine it was originally designed to enhance the abilities of JIAC IV agent systems. Before an agent can act for a human, it has to understand the wishes and intentions of its user. The ARISE system allows the agents to compare textual descriptions, an interesting ability for Filtering-by-example. Filtering-by-example is an information filtering method that enables a user to refine filtering results by indicating examples of what he or she wants (content-based filtering). Product descriptions for example can be matched with the queries of a consumer, so it is possible to give a specification of the

kind of book one is looking for and the agent delivers a list of books available for this subject (or it orders the books).

Another operational area is personal information management. The number of services allowing the user to specify personal newsletters is increasing. But most of them allow only the choice of some predefined categories. With the ARISE system it is possible to create own categories, containing textual descriptions (which can be enhanced at any time). These descriptions can be considered as documents and a vector can be created. So documents which have a position near to the positions of the category vector are added to the newspaper. The term "near" could be trained by the user by evaluating the chosen articles of recent newsletters.

"Personal News" is the name of a new internal project of the DAI-Lab. It uses large parts of the ARISE system. The user has the possibility to create own categories and to organize them in a tree-like structure. He or she can describe the categories textually. The "Personal News" system compares the vectors of these textual descriptions with news articles in a SOM generated every day. If some documents are in neighborhood of one of these vectors, the documents are added to the newsletter. The complete newsletter is sent to the user via e-mail. It makes sense to combine this method with the naive Bayes algorithm introduced before to enhance the accuracy of the results.

Appendix A

List of Abbreviations

AI	Artificial Intelligence
AOSE	Agent-Oriented Software Engineering
API	Application Program(ming) Interface
ARISE	Agent-based Readjustable Intelligent Search Engine
BMU	Best Matching Unit
DAI	Distributed Artificial Intelligence
FIPA-ACL	Foundation for Intelligent Physical Agent - Agent Communication Language
GUI	Graphical User Interface
HSOM	Hierarchical Self-Organizing Map
HTML	Hyper Text Markup Language
IR	Information Retrieval
JDBC	Java Data Base Connectivity
JIAC IV	Java Intelligent Agent Componentware IV
JVM	Java Virtual Machine
KQML	Knowledge Query and Manipulation Language
LSA	Latent Semantic Analysis
OCR	Optical Character Recognition

PCA Principal Component Analysis

PDF Portable Document Format

PS Postscript

RM Random Mapping

RTF Rich Text Format

SOM Self-Organizing Map

SQL Structured Query Language

TCP/IP Transmission Control Protocol/Internet Protocol

URL Uniform Resource Locator

WML Wireless Markup Language

WWW World Wide Web

XML eXtensible Markup Language

Bibliography

[ABF01] Sahin Albayrak, Joos-Hendrik Böse, and Sebastian Feuerstack. Multi Access Point für JIAC, 2001.

[Acr01] Acronym Finder: Look up 206,000+ acronyms/abbreviations & their definitions, 2001. http://www.acronymfinder.com/.

[Alb98] Sahin Albayrak. Introduction to Agent Oriented Technology for Telecommunications. In Sahin Albayrak, editor, *Intelligent Agents for Telecommunication Applications*, pages 1–18. IOS Press, Amsterdam, Berlin, Oxford, Tokyo, Washington DC, 1998.

[AW99] Sahin Albayrak and Dirk Wieczorek. JIAC - A Toolkit for Telecommunication Applications. In Sahin Albayrak, editor, *Intelligent Agents for Telecommunication Applications*, pages 1–18. Springer, Berlin, Heidelberg, New York, Barcelona, Hong Kong, London, Milan, Paris, Singapore, Tokyo, 1999.

[Ban94] Martin Bangemann. The Bangemann Report: Europe and the global information society. 1994, 1994.

[BBCM98] M. Breugst, I. Busse, S. Covaci, and T. Magedanz. Grasshopper – A Mobile Agent Platform for IN Based Service Environments. In *Proceedings of IEEE IN Workshop 1998*, pages 279–290, Bordeaux, France, 1998.

[Bra87] M. Bratman. *Intentions, plans and practical reason.* Harvard Univ. Press, Cambridge, MA (USA), 1987.

[Bra97] Jeffrey M. Bradshaw. *Software Agents.* AAAI Press, Mento Prak, USA, 1997.

[Can01] Canoo - Sprache lernen: interaktive Anwendungen und Community. Homepage, 2001. http://www.canoo.net.

[Cau99] Jorg Caumanns. A Fast and Simple Stemming Algorithm for German Words, 1999. http://www.inf.fu-berlin.de/inst/pubs/tr-b-99-16.abstract.html.

[Cis01] Richard Cissée. Neural Information Processing Methods for Agent-Based Intelligent Information Retrieval. Diploma thesis, Technische Universität Berlin, 2001.

[Dai01] JIAC - Intelligent Agent Componentware. Homepage, 2001. http://www.dai-lab.de/jiac/index_en.html.

[DAR01] DARWIN Digitale Naturwissenschaftliche Bibliothek der FU Berlin. Homepage, 2001. http://darwin.inf.fu-berlin.de/work/Main/.

[DDL+90] Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by Latent Semantic Analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.

[Ety01] Etymon Systems Inc.: PDF Products. Homepage, 2001. http://www.etymon.com/pdf_products.html.

[F+93] Tim Finin et al. Specification of the KQML Agent-Communication Language – plus example agent policies and architectures, 1993.

[FBK+01] Stefan Fricke, Karsten Bsufka, Jan Keiser, Torge Schmidt, Ralf Sesseler, and Sahin Albayrak. A Tool-kit for the Realization of Agent-based Telematic Services and Telecommunication Applications. *Communications of the ACM*, April, 2001.

[FFMM94] T. Finin, R. Fritzson, D. McKay, and R. McEntire. KQML as an Agent Communication Language. In *Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM'94)*, pages 456–463, Gaithersburg, Maryland, 1994. ACM Press.

[FIP97] FIPA. FIPA 97 Specification, Version 2.0, Agent Communication Language, 1997.

[Fol96] P. Foltz. Latent semantic analysis for text-based research. *Behavior Research Methods, Instruments, and Computers*, 28:197–202, 1996.

[FPW+99] Eibe Frank, Gordon W. Paynter, Ian H. Witten, Carl Gutwin, and Craig G. Nevill-Manning. Domain-Specific Keyphrase Extraction. In *IJCAI*, pages 668–673, 1999.

[Goo01] Google Search Technology - Our Search: Why Use Google. Homepage, 2001. http://www.google.com/technology/index.html.

[Jav01] Java Performance Tuning. Homepage, 2001. http://www.javaperformancetuning.com.

[Jdk01] java.sun.com - The Source for Java(TM) Technology. Homepage, 2001. http://java.sun.com.

[JL84] W. Johnson and J. Lindenstrauss. Extensions of Lipschitz maps into a Hilbert space. *Contemp. Math.*, 26:189–206, 1984.

[Koh90] Teuvo Kohonen. The Self-Organizing Map. In *New Concepts in Computer Science: Proc. Symp. in Honour of Jean-Claude Simon*, pages 181–190, Paris, France, 1990. AFCET.

[KP94] W. Kraaij and R. Pohlmann. Porter's stemming algorithm for Dutch. In L.G.M. Noordman and W.A.M. de Vroomen, editors, *Informatiewetenschap 1994: Wetenschappelijke bijdragen aan de derde STINFON Conferentie*, pages 167–180, 1994.

[Lam92] Jouko Lampinen. On Clustering Properties of Hierarchical Self-Organizing Maps. In I. Aleksander and J. Taylor, editors, *Artificial Neural Networks, 2*, volume II, pages 1219–1222, Amsterdam, Netherlands, 1992. North-Holland.

[Lin01] Lingsoft Demos - Lingsoft Language Sense. Homepage, 2001. http://www.lingsoft.fi/demos.html.

[LK99] Krista Lagus and Samuel Kaski. Keyword selection method for characterizing text document maps. In *Proceedings of ICANN99, Ninth International Conference on Artificial Neural Networks*, volume 1, pages 371–376. IEE, London, 1999.

[MN98] A. McCallum and K. Nigam. A comparison of event models for Naive Bayes text classification. In *AAAI-98 Workshop on Learning for Text Categorization*, pages 41–48. Madison, WI: AAAI Press, 1998.

[Por80] M. F. Porter. An Algorithm for Suffix Stripping. *Program*, 14(3):130–137, 1980.

[Por00] Modern Information Retrieval - Porter's algorithm. Homepage, 2000. http://sunsite.dcc.uchile.cl/irbook/porter.html.

[Rij79] C.J. Van Rijsbergen. *Information Retrieval.* Butterworths, London, 2nd edition, 1979.

[SB01a] Ralf Sesseler and Siegfried Ballmann. *JIAC IV Programmier-Handbuch.* DAI-Labor Berlin, 1.1 edition, 2001.

[SB01b] Ralf Sesseler and Siegfried Ballmann. *JIAC IV Technisches Handbuch.* DAI-Labor Berlin, 1st edition, 2001.

[Sho91] Y. Shoham. Agent-oriented programming. In *Proceedings of the 11th International Workshop on DAI*, pages 345–353, 1991.

[Sos99] Dennis M. Sosnoski. Java Performance Comparison with C/C++, 1999. http://www.sosnoski.com/Java/Compare.html.

[WC01] M. Wooldridge and P. Ciancarini. *Agent-Oriented Software Engineering: The State of the Art*, volume 1957 of *Lecture Notes in AI.* Springer, Berlin, Heidelberg, New York, Barcelona, Hong Kong, London, Milan, Paris, Singapore, Tokyo, 2001.

[WEB01] WEBSOM - A novel SOM-based approach to free-text mining. Homepage, 2001. http://websom.hut.fi/websom/.

[Whi94] J. E. White. Telescript Technology: The Foundation for the Electronic Marketplace. Technical report, 2465 Latham Street, Mountain View, CA 94040, 1994.