Fortschritt-Berichte VDI

VDI

Reihe 10

Informatik/
Kommunikation

Dipl.-Inf. Jan Werrmann,
Stuttgart

Nr. 849

AIRS - Advanced Ontology-based Information Retrieval System

AIRS – Advanced Ontology-based Information Retrieval System

Dissertation
zur Erlangung des akademischen Grades
DOKTOR RER. NAT.

der Fakultät für
Mathematik und Informatik
der FernUniversität
in Hagen

von
Jan Werrmann
geb. in Borna

Hagen 2016

Fortschritt-Berichte VDI

Reihe 10

Informatik/
Kommunikation

Dipl.-Inf. Jan Werrmann,
Stuttgart

Nr. 849

AIRS - Advanced Ontology-based Information Retrieval System

VDI verlag

Werrmann, Jan
AIRS – Advanced Ontology-based Information Retrieval System
Fortschr.-Ber. VDI Reihe 10 Nr. 849. Düsseldorf: VDI Verlag 2016.
174 Seiten, 49 Bilder, 5 Tabellen.
ISBN 978-3-18-384910-9, ISSN 0178-9627,
€ 62,00/VDI-Mitgliederpreis € 55,80.

Keywords: Information Retrieval – Ontology Development – Knowledge Representation – Advanced Search Technologies – Heterogeneous Document Landscapes

Obtaining the right information at the right time is one of the main challenges for modern societies. This holds especially for companies that must handle complex business processes that require case dependent information. Unfortunately, case dependent and relevant information is often widespread over different document systems. Users must interact with various applications and search for semantically related (and helpful) documents without any, or with only little, support by the disparate retrieval systems. In this work, a system called Advanced ontology-based Information Retrieval System (AIRS) is introduced that includes methods of state-of-the-art enterprise search technology and combines them with an ontology called AIRS Knowledge Base (AIRSKB). AIRS is deeply integrated with advanced information retrieval technologies to make search processes in large heterogeneous document landscapes more effective and increase the quality of search results.

Bibliographische Information der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliographie; detaillierte bibliographische Daten sind im Internet unter http://dnb.ddb.de abrufbar.

Bibliographic information published by the Deutsche Bibliothek
(German National Library)
The Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliographie (German National Bibliography); detailed bibliographic data is available via Internet at http://dnb.ddb.de.

ISSN 0178-9627
ISBN 978-3-18-384910-9

Acknowledgments

The following work was developed during my employment at the Global Service & Parts (GSP) department of Daimler AG. Daimler started a research project for the optimization of workshop literature access and I am thankful that Daimler gave me the opportunity to participate in it.

I came in touch with Professor Bernd J. Krämer from FernUniversität in Hagen who later became my supervisor. I would like to thank Professor Krämer very much for his support from the start of my research project until the end in all facets. He encouraged me to add a collective intelligence approach into my research project.

I also would like to thank Professor Gerhard Heyer from Universität Leipzig where I received my degree in computer science. He was the supervisor of my diploma thesis and through him I came in touch with Daimler. At Daimler, I would like to thank all my colleagues who supported this work.

Very special thanks to my family who supported me in writing my thesis. Especially to my wife Natalie Werrmann and to my parents Dr. Angela Werrmann and Udo Werrmann who gave me the motivation and strength. Last but not least, I want to thank all my friends for just giving me time to finish my thesis.

For my grandfather Johannes Ludwig.

Thank you for your math lessons when I was a lazy child ...

Contents

Abstract

Obtaining the right information at the right time is one of the main challenges for modern societies. This holds especially for companies that must handle complex business processes. These business processes require case dependent information. But relevant information is often widespread over different document systems. Enterprise search is a field of research that focuses on the challenges of information access. Unfortunately, a deep integration of knowledge networks as well as relationships between index documents is not sufficiently supported by enterprise search systems. Often, good retrieval results depend on these relationships, because the documents of the systems correlate somehow to each other regarding a special business case.

This presents a heterogeneous information system and document landscape to the employees who must find the right piece of information they need from different retrieval systems. In the end, an employee must interact with various desktop applications and search for semantically related (and helpful) documents without any, or with only little, support by the disparate retrieval systems.

The main motivation for this work can be summarized in the following research questions:

1. Can a single systems view be provided for all of the case-related documents kept in different retrieval systems?

2. Can seamless and guided access across these disparate and disconnected retrieval systems be designed?

3. Can the quality of retrieval results and the effectiveness of the retrieval process be improved by exploiting user feedback?

4. Can a technical solution be developed that is accepted by users in the field?

To address these research questions, enterprise search technologies were combined with knowledge representation techniques based on ontologies and user feedback processing. For this approach, a system called Advanced ontology-based Information Retrieval System (AIRS) was developed. It includes methods of state-of-the-art enterprise search technology and combines them with an ontology called AIRS Knowledge Base (AIRSKB). AIRSKB provides an overarching knowledge structure modeling documents, document sources and explicit or deduced relationships between them. This knowledge structure now represents a homogeneous and coherent search space. It is deeply integrated with advanced information retrieval technologies to make search processes in large heterogeneous document landscapes more effective and increase the quality of search results. AIRSKB also serves to capture the collective intelligence of knowledge producers and knowledge users, i.e., employees. To demonstrate the feasibility of the approach and evaluate its innovative capabilities and its usability, a prototype system, called AIRS Prototype, was designed, implemented, and tested in a domain of maintenance, service and repair of cars in car workshops. These field tests included:

1. A comparison of current workshop retrieval systems with AIRS Prototype and
2. system tests along predefined domain-specific test scenarios.

The field tests were carried out with workshop employees exhibiting many years of professional experience in workshop services. They showed that the new ontology-based retrieval is superior to the existing retrieval technology and that the collective feedback of workshop experts enables the automatic and valid reconstruction of hidden document relationships.

Zusammenfassung

Die richtigen Informationen zum richtigen Zeitpunkt zu bekommen ist eine der größten Herausforderungen moderner Gesellschaften. Das gilt speziell für Firmen, die mit komplexen Prozessen hantieren müssen. Diese Prozesse verlangen Fall-abhängige Informationen für spezielle Arbeitsschritte. Allerdings sind die Informationen oft über verschiedene Dokumentensysteme verteilt. Enterprise Search ist ein Forschungsgebiet, das sich auf die Herausforderung des Informationszugangs spezialisiert hat. Leider wird die tiefe Integration von Wissensnetzwerken oder Relationen zwischen Indexdokumenten von Enterprise Search-Systemen nur ungenügend unterstützt. Oft hängen gute Suchergebnisse von diesen Relationen ab, weil Dokumente der Systeme im Bezug zu einem Anwendungsfall miteinander in Verbindung stehen.

Insgesamt gesehen stellt sich für Mitarbeiter von Firmen oft eine heterogene System- und Dokumentenlandschaft dar, in der nach erforderlichen Informationen in verschiedenen Systemen recherchiert werden muss. Das Resultat ist, dass der Mitarbeiter mit verschiedenen Applikationen hantieren und nach semantisch verbundenen (und hilfreichen) Dokumenten ohne oder mit wenig Unterstützung der unterschiedlichen Recherchesysteme suchen muss.

Die hauptsächliche Motivation für diese Arbeit kann in den folgenden Forschungsfragen zusammengefasst werden:

1. Kann eine einheitliche Sicht auf fallrelevante Dokumente der verschiedenen Recherchesysteme hergestellt werden?

2. Kann ein nahtloser Zugriff auf diese ungleichen und nicht verbundenen Recherchesysteme bereitgestellt werden?

3. Wie kann die Qualität der Rechercheergebnisse und die Wirksamkeit der Recherchesysteme durch die Verwendung des Feedbacks von Systemnutzern verbessert werden?

4. Kann eine prototypische Lösung entwickelt werden, die Akzeptanz bei den Systemnutzern findet?

Um diese Forschungsfragen zu beantworten, wurden Enterprise-Suchtechnologien mit Techniken der Wissensrepräsentation (basierend auf Ontologien) und automatischer Feedback-Verarbeitung verbunden. Für den Ansatz wurde das System Advanced ontology-based Information Retrieval System (AIRS) entwickelt, das Methoden von Enterprise-Suchtechnologien nach aktuellem Stand der Technik verwendet und mit einer Ontologie, AIRS Knowledge Base (AIRSKB), verbindet. AIRSKB stellt eine übergreifende Wissensstruktur dar, welche Dokumente, Quellen sowie feste und adaptive Relationen zwischen den Dokumenten und Quellen beinhaltet. Dadurch fungiert die Wissensstruktur als homogener und kohärenter Suchraum. Sie ist tief integriert in erweiterte Information-Retrieval-Technologien um Suchprozesse in großen heterogenen Dokumentenlandschaften effektiver zu gestalten und die Qualität der Suchergebnisse zu verbessern. AIRSKB dient auch dazu,

die kollektive Intelligenz der Wissensproduzenten und Wissensnutzer, d.h. der Mitarbeiter zu erfassen. Um die Durchführbarkeit des Ansatzes zu demonstrieren und seine Innovationskraft und Benutzerfreundlichkeit zu bewerten, wurde ein Prototyp, AIRS-Prototype, entwickelt und in der komplexen Business-Domäne von Service, Wartung und Reparatur von Fahrzeugen in Kfz-Werkstätten verprobt. Diese Feldtests beinhalteten:

1. Einen Vergleich der Dokumentenrecherche in aktuell existierenden Werkstattrecherchesystemen mit der Dokumentenrecherche von AIRS-Prototype und

2. Systemtests, basierend auf vordefinierten domänenspezifischen Test-Szenarien.

Die Feldtests wurden mit Werkstattmitarbeitern durchgeführt, die jeweils über langjährige Berufserfahrung im Werkstattumfeld verfügten. Die Tests zeigten, dass der ontologiebasierte Suchansatz bessere Ergebnisse erzielt als die existierenden Recherchesysteme. Ebenfalls zeigte sich, dass kollektives Feedback der Werkstattexperten es ermöglicht, versteckte Dokumentenbeziehungen automatisch und valide zu rekonstruieren.

1 Introduction

Obtaining the right information at the right time is one of the main challenges for modern societies, especially for companies that must handle complex business processes. To execute a particular business process, employees must perform different tasks. For example, support agents of a company that provides customer software support must perform tasks that require searching for remedies for software problems. Customers report error messages of the application and agents search for remedies that match the error messages in corresponding retrieval systems. This makes finding relevant information one of the most important challenges in solving complex business processes. Other processes that include tasks of finding relevant information are similar. Almost every company has to face the challenge of information search. Examples are:

- **Maintenance, service and repair of cars in workshops.** Imagine a case where a customer's car needs a service in a workshop, after an accident, for example. Furthermore, let the task of the case be clear: some parts of the customer's car are broken. In a simplified workshop process description, an employee needs to open a new service case in the workshop's management system. Therefore, the workshop employee needs to categorize the given case with the help of a standardized symptom taxonomy: the employee selects some case-describing symptom taxonomy nodes. After the case is opened, the workshop employee now needs replacement parts for a car repair. Furthermore, the employee needs a repair instruction to find information about the installation of the replacement parts. After a workshop employee has repaired the car, work units of a work unit catalog must be selected for the customer's invoice. To summarize, all of the replacement parts, the case-describing symptoms, the work units and the repair instruction were used in the car repair process. Work units come from a work unit catalog and case-describing symptoms are categorized in a symptom taxonomy tree. Furthermore, replacement parts are available in the electronic replacement parts catalog. Instead, repair instructions are stored and accessible via the workshop information system, etc.

  Workshop employees need to find necessary information hidden in different document systems to deal with the entire workshop process. This information is mostly contained in isolated retrieval systems that mainly act independently of each other (see [93]). The data stored in these retrieval systems are semantically related to each other. Most of these relationships are not available as computer readable links between documents; rather, the workshop employees have to search for references to information in other retrieval systems and follow them by accessing different systems one by one. The workshop employees have to search for pieces of information in each of these retrieval systems. Additionally, long-term running systems are used for these critical business cases. These systems often do not provide the possibilities of modern information retrieval.

The example shows that workshop employees need to easily access information that is widespread across different and mostly isolated retrieval systems. Therefore, the users must handle various desktop applications on their devices. Furthermore, they need to access information that is related to both the given task and documents that have already been identified as useful to solve tasks that are similar to the given task. This means that the employees should benefit from other employees who have already solved similar business cases. Companies have to face these challenges to provide solutions to their employees, supporting them in solving complex business processes.

- **Intranet search.** Intranets as introduced in [18] or [11] are "*private, internal networks*" based upon They are "*restricted to organizational participants only*".

  Intranets provide employees the "*ability to access a wide variety of information sources*". These sources are accessible via an access-restricted website. Companies often use portal technology for this approach to make the sources accessible via the company's Intranet pages. Product examples for portal servers include the IBM WebSphere[1], SAP NetWeaver[2] or Red Hat JBoss Portal[3]. In contrast to ordinary web pages, portals are "*tailored according to the users' need*" (see [3]). Therefore, portals provide a well-known and stable technology for the integration of information in web pages.

  The contents of the sources are diverse. As explained in [52], they offer access to publications (Level 1), provide a platform for cooperation (Level 2) and serve as an application platform (Level 3). A higher level means more complexity in requirements. Furthermore, the included information sources are used for different business processes.

  The contents of Intranets can also be categorized along application areas (functional blocks). The application areas of Intranets are also diverse: "*News & Content*", "*Collaboration & Communication*", "*Employee profiles & Networking*", "*General Processes & Applications*" and "*Special Processes & Applications*" (see [64]). All application areas require overarching functions. A search across all contents of the functional blocks is such a requirement. This makes the search across all services and information sources a key benefit of Intranets. An Intranet search differs from an ordinary web search in the case that employees need to search for information that helps them to solve a particular business task. Additionally, users look for services and applications rather than simply information about a topic. Therefore, companies need to solve the challenge of representing different information types in a search engine. Furthermore, companies must implement the possibility of suggesting relevant information and services regarding the information need of a user.

  For example, let there be a case where an employee must set up a telephone conference call for a meeting. For this approach, the employee needs to ascertain how to apply for a telephone conference number that meeting members can dial. He also needs information about how to set up an online presentation and how to prepare his hardware for this approach. The employee can now search in the company's Intranet

[1] See http://www.ibm.com/software/websphere, last visited Sept. 18, 2016.
[2] See http://scn.sap.com/community/netweaver, last visited Sept. 18, 2016.
[3] See http://www.jboss.org/products/portal/overview, last visited Sept. 18, 2016.

for all of the necessary information and services. Companies often use state-of-the-art enterprise search technologies to establish a search across all information resources. The employees can subsequently use all of the search features provided by the search technology. For example, a similar search based upon the content of a search result page, suggestion of search terms, spelling correction, faceted search along a category, etc. If different departments of the company are responsible for telephone conference number management, online presentations and hardware, the users must search for every piece of information again. The reason is that departments often publish information about the services that they provide on their own Intranet page. This means that the search for necessary information in Intranets is a complicated task, especially when employees need to find semantically-related information for a given task of a business process. This raises the possibility that the users do not find the right piece of information and thus the search process becomes costly. One solution is to manually link semantically-related information of Intranet pages beyond the borders of departments' responsibilities. However, this must be undertaken for all business processes. Furthermore, the links must be updated due to constantly changing processes and rapidly changing information sources, which is a very costly process. Additionally, the users should benefit from other users who have already collected information and services that were necessary to solve similar business cases. This cannot be performed by manual linking.

- **Search for academic literature.** The search for academic literature is the basis for scientific research activities. The current solutions are usually based upon state-of-the-art search technologies. Examples include Google Scholar[4], Pub Med[5], Web of Science[6] or Scopus[7] (see [23]). Because academic literature is a special domain, searching for academic literature differs from an ordinary web search because it mostly depends on users' special research interests. This means that most of the users are interested in literature about a special scientific research area or a combination of research areas. This applies to both scientists from departments of an educational domain and employees from a research & development department of a company.

  Companies as well as public facilities provide their employees access to a wide field of publication databases. The databases share the common feature of free text search within the search activity. Furthermore, the databases themselves provide special domain-specific search features to their users.

  For example, the main features of Google Scholar are searching for publication time ranges, patents and citations, citation export and a special search result page adapted to domain interests (file access, author information, "cited by" information or similar article search). As another example, Springer Link[8] provides a special faceted navigation of search results by using the categories "Content Type", "Discipline" and "Subdiscipline".

[4]See https://scholar.google.de, last visited Sept. 18, 2016.
[5]See http://www.ncbi.nlm.nih.gov/pubmed, last visited Sept. 18, 2016.
[6]See http://wokinfo.com, last visited Sept. 18, 2016.
[7]See http://www.scopus.com, last visited Sept. 18, 2016.
[8]See http://link.springer.com, last visited Sept. 18, 2016.

Researchers are particularly interested in literature about a very special research field. Because the search only depends on free text search and categorization, it is not easy to find the right publications among the jungle of academic literature. Therefore, state-of-the-art search technologies (like Google Scholar) provide a similar search based upon the content of a publication. A semantic search based upon the concepts of the publications' content usually only exists in a simplified form. Accordingly, many research approaches try to increase the quality of natural text retrieval by using concepts (not only for academic literature search); for example, through different approaches of semantic query expansion (see [90], [61], [35], [36], [42], [55] or [24]). Many of these approaches increase the retrieval quality, although they do not match the main requirements of academic literature search. One main requirement is to find publications that are semantically related to each other in the context of a special research interest. Research interests may be diverse, whereby a researcher might be interested in publications about a topic or sub-topic. He might also be interested in articles of similar topics or publications about different topics. New research areas are often developed by a combination of different topics. Therefore, the research interest is a context of search and all of the publications that match the context are related to each other. This relationship between publications may be based upon their text content, which is accessible through similar search queries, although this is not necessarily the case. This makes academic literature search a difficult task. Additionally, researchers should benefit from other researchers with similar research interests and similar search contexts who have already collected publications of a particular research interest. This can be derived indirectly from the references at the end of a publication, although this only applies to the given publication. Furthermore, this represents an immutable list of references. Another possible solution consists of an expansion of private search contexts to a public knowledge network of search contexts over system borders as suggested in POSUKO (see [37]), that is developed as part of eleed[9].

A further challenge is the diversity of literature databases. This provides a heterogeneous literature database landscape to the users. The result is that users can only find publications if they search in the right database. This means that researchers must handle different literature databases. Furthermore, every database brings its own search methods. To summarize, another requirement is to provide a homogeneous access structure for a literature search that can suggest publications of similar research interests.

A possible solution is to develop software tools that provide a search function connected to various literature databases. Current reference management software tools like JabRef[10] or Citavi[11] often include connectivity to foreign literature databases. However, they provide a single point of database access rather than a homogeneous search infrastructure to the users.

The challenge is to develop solutions that break down the barrier of information search that is restricted through the limitations described above: information search

[9]Eleed is an open-access web journal, see https://eleed.campussource.de/, last visited Sept. 18, 2016.
[10]See http://www.jabref.org, last visited Sept. 18, 2016.
[11]See http://www.citavi.com, last visited Sept. 18, 2016.

across different publication databases, establishing relationships between publications that allow dynamic modeling of research interests by a set of related publications and the access of such a model of a research interest.

- **Service requests in Call Centers.** Support and marketing has changed from the 1980s to present. Call Centers have evolved from telephone marketing to complex process workflow platforms in the field of customer service (see [29], [15]). They are often the first customer contact for a product or service of a company. Therefore, Call Centers are centralized service points for customer support and they provide a single point of customer contact. Today, most processes are inbound (the customer calls) and the tasks that the agents must handle are different, including information requests (hotline, product information), customer services (product support, ticket services) or emergency calls (car breakdown services). The customers dial a number and enter into contact with a support agent of a Center. Different systems are used in Call Centers to connect the user call to a specific customer service process. These processes require task-specific access to different information sources of the company. The information contained in the sources is subsequently used to solve the customer request. Additionally, the agents need to answer the users' questions very quickly. Often, the customers describe a problem or service request in natural language and the agents need to find the right piece of information in the retrieval systems of the companies. This means that an agent must search in different systems by using the customers' descriptions. CRM[12] software systems are often used in Call Centers to manage the whole customer process and research.

  Many questions of clients are similar. The Call Center agents need to perform similar searches in the retrieval systems. One approach is to extend the current solutions with new search functions that deal with similar client requests and heterogeneous data landscapes (see [46], for example). However, they should also benefit from solutions of similar service requests. Key requirements include access to a wide field of information sources through a homogeneous access structure and an information search solution where Call Center agents can benefit from known solutions of previously-solved tasks.

- **Access project documentation.** A strong body of documentation and many status reports are usually produced during projects. For different business tasks, this project documentation must be searched for the right piece of information. For example, project managers need to prepare management reports and software developers need to search for requirements and specifications. Most of the project documentation is stored in project shares, Microsoft SharePoint[13] directories, project Intranet pages, special software development tools (like JIRA[14] or Confluence[15]) or other software tools like Alfresco[16]. The search for documents takes place at various points, including the operation system's search across directories and document contents,

[12]Customer Relationship Management, see [39] for an introduction in the topic of customer relationship management.
[13]See https://products.office.com/de-de/SharePoint/collaboration, last visited Sept. 18, 2016.
[14]See https://de.atlassian.com/software/jira, last visited Sept. 18, 2016.
[15]See https://de.atlassian.com/software/confluence, last visited Sept. 18, 2016.
[16]See https://www.alfresco.com, last visited Sept. 18, 2016.

a Microsoft SharePoint search or by using search functions of software development tools. The weakness is that project members must search for pieces of information in each of the project documentation storage databases again. Often, they need to perform different searches for related information because related documents are not accessible by default in the search. State-of-the-art software development tools provide features to easily link related documentation, although these links must be placed manually in the corresponding documents. One challenge is to optimize the access to project documentation for project members via a homogeneous access structure. Furthermore, the retrieval for project documentation should include the suggestion of documents based upon relationships between documents. These relationships between documents may be based upon their content, although this is not necessarily the case. This makes the search for related documents of documents difficult. Current retrieval systems often provide features to search for similar documents, albeit only based upon their retrieval algorithms.

- **Processing service tickets in product support.** Product support is a difficult task, given that users have specific issues with the use of an application and need support to solve their problem. Often, business applications provide a support channel directly in the application, producing a service ticket in the product support chain. The employee who works on the ticket often derives a user description of unusual application behavior in natural language. Sometimes the ticket comes with an error code produced by the application. The employees now need to search across the software documentation to produce solutions that help the users. Therefore, the support agents need help in searching for remedies to similar problems. They should benefit from previous searches of support agent colleagues who have already solved similar problems. Similar cases mean similar searches and this means similar remedies.

These examples are simply a selection of business cases that require case-dependent information searches across heterogeneous data sources. Other examples are similar; for instance, brokers must overlook various systems looking for similar information about courses. Employees of sales departments need to search for products and services in various data sources. Employees of human resources department need to search for skills of employees and candidates for employment to assemble project teams. Employees of travel agencies must search for travel offers in different systems to provide individualized travel deals to their clients. By looking at the examples described above, the main disadvantages of the current solutions are:

1. Users must perform recurrent searches in disparate databases and document management systems using different retrieval systems. This is similar for all of the examples described above. Users of an Intranet search must perform different searches to collect all of the necessary information. Employees of a workshop need to search in different systems for all of the information that they need for a car service or repair processes.

2. Documents can only be found if users include the respective retrieval system in their search. One example is the academic literature search, because the publications are distributed across many publication databases.

3. The search for semantically-related documents that may contain useful information is not supported by the retrieval systems. For example, software engineers need to search for software requirements that they need to implement, while employees need to search for semantically-related information (workshop literature search, Intranet search).

4. Users cannot benefit from successful searches of other users with similar information needs because search contexts are not maintained. This is similar for all of the examples described above, especially for a workshop literature search, Intranet search or scientific literature search.

This raises the possibility of errors, is costly and increases the time span that the users need for the business process. To summarize, business processes require case-dependent information. The information is necessary for tasks of processes as well as supporting the interaction with the customer. Companies produce various documents, database entries and other storage containers to maintain this information. Unfortunately, this information is often widespread across different systems. Therefore, companies use document retrieval systems as well as database technologies to make this information accessible. The main challenge is to enable retrieval systems to increase the effectiveness of the customer service processes and increase the prediction quality for service requests.

1.1 Challenges for Information Retrieval in Heterogeneous Domains

Companies often use long-term running systems for their critical business cases because modernization of a system landscape is very expensive and well-known systems and technologies are stable ("never change a running system"). Database systems serve as an error-prone yet mature technology for storage and retrieval of structured data. Their strengths lie in representing relational data models and performing complex join operations on data records. Moreover, they are also used for document retrieval. Unfortunately, databases are not optimized for the search in unstructured text resources.

In contrast to database systems, enterprise search systems are optimized for retrieval in text resources. Their strengths lie in a natural language-dependent search using complex retrieval and ranking algorithms. However, they are neither designed for join operations on data records nor for the representation of relationships and data models.

A technology is necessary that includes the strength of database technology in information retrieval technologies. The following main research challenges must be solved:

- **Information access** is a challenge for companies with a heterogeneous process, document and data landscape. One reason is that data stored in file systems, databases, Internet, Intranet, content management systems, knowledge bases and document information systems is constantly growing. Most of these data comprise unstructured text. Enterprise search systems face this challenge by establishing a cooperation-wide or process-dependent search across multiple data sources. These search technologies are now increasingly used to establish cooperation-wide information access or enhance the quality of software products.

Additionally, business intelligence reporting tools use state-of-the-art search technologies to combine and present aspects of the data to generate a new view on unknown or hidden information. For this purpose, Big Data analysis often serves as a pre-process that consumes the data from the different sources and prepares it for information exploration or indexing tasks (see [10] or [57] for more information about Big Data). This Big Data analysis often combines the power of distributed data storage based upon the concept of GFS (Google File System as described in [27]), Apache Hadoop's HDFS (see [5] or [97]), with algorithms based upon MapReduce (see [16]). During the parallel processed MapReduce steps, text mining technologies are often used to analyze unstructured text[17] and data mining technologies are used to harvest unstructured raw data. The result is either an indexed text corpus for search engines or structured data used to generate reports by using business intelligence reporting tools.

In the case of using a search engine index, several challenges need to be solved by firms besides the choice of the right search technology, which are also important for this work because they are relevant for a successful system development. These challenges include:

- **Information targeting**. Identifying information pieces in the original systems that can serve as search targets for the retrieval (index documents of the information retrieval systems).
- **Information gathering**. Handling information from various data sources to support an overall information search. The definition of a generic and overall valid data model is particularly important.
- **Information transfer**. Dealing with information extraction processes to read and analyze the data from the various data sources.
- **Information relevance**. Calculating the relevance of search results.
- **Information suggestion**. Suggesting information that could be relevant for the business task, even without a user-triggered search.
- **Information routing**. Choosing the right set of business case-relevant information for indexing.
- **Information quality**. As explained in [49], information quality describes several requirements that have to be considered by developers of information systems: "*ensure consistency*", "*avoid redundancy*", "*control access*" and "*strive for relevance for the recipients of information*".

These challenges are difficult to perform because they often require the knowledge of data scientists in combination with domain experts who perform the business cases.

- **Information search** is the challenge of finding relevant information in a collection of text data. The technology of information retrieval as explained in basic literature by Manning, Raghavan and Schütze (see [62]), van Rijsbergen (see [87]), Stock (see

[17] Text mining technologies can be used for various text processing tasks, such as automated language detection, part of speech tagging, spelling correction, named-entity recognition and co-occurrence-based text analysis. In summary, these computer-based methods are necessary for the semantic analysis of text resources and automated or semi-automated text structuring (see [41]).

[79]) and Singhal (see [74]) provide the concepts and functions used by various search engines based upon vector space retrieval in combination with Boolean retrieval.

Information retrieval technologies are the basis of search engines and enterprise search solutions. Search engines are software products that provide an information search in heterogeneous and domain-specific unstructured data sources. Apache Solr – which was built on top of Apache Lucene – is an example of an open source software that provides such retrieval algorithms (see [63] and [75]). Lucidworks developed an enterprise search server product based upon the open source solution Solr (Lucidworks Enterprise Search[18]). Other companies also provide enterprise search products, although most of them are closed source. Examples of these products include HPE IDOL by Hewlett Packard Enterprise[19], Oracle Secure Enterprise Search[20], Coveo Intelligent Search Platform[21], Search Solutions by Swiftyp[22] and Google Search Appliance[23]. Enterprise search comes with a wide range of features that enable knowledge-driven retrieval (see [86]): federated search (simultaneous search on multiple sources), integrated search (indexing and searching of data from foreign sources), concept search (extending index documents with related terms), faceted search (clustering of search results), semantic indexing (relationships between index words are kept in an external database, whereby the system can automatically offer alternatives to the search results), as well as other features that extend the content of a document or query. Additionally, enterprise search products often include technologies that extend the natural language-based vector space retrieval:

- Lightweight ontologies (taxonomies, synonym sets) are used to improve the quality of natural language search technologies.
- Similarity search is used to find documents similar to a result document.
- Automatic spelling correction of search terms helps to increase the search quality.
- Search term suggestions support the user in formulating the retrieval request.

All such approaches somehow result in an extension of index documents or search queries. Unfortunately, semantic knowledge about document relationships cannot be used or updated dynamically.

- **Information linking** describes the challenge of using semantic relationships between data records. Information retrieval systems are easily accessible by end users because they can formulate their search queries in natural language. Moreover, it is also possible for advanced users to formulate a more complex search query by Boolean linking of search terms, a weighted field search or wildcard search. Accordingly, systems can be developed to support both end users in general search environments as well as expert users dealing with business tasks in expert systems. Additionally,

[18] See https://lucidworks.com/solutions/use-case-enterprise-search/, last visited Sept. 18, 2016.
[19] See http://www8.hp.com/us/en/software-solutions/information-data-analytics-idol, last visited Sept. 18, 2016.
[20] See www.oracle.com/technetwork/search/oses/overview/, last visited Sept. 18, 2016.
[21] See http://www.coveo.com/en/platform, last visited Sept. 18, 2016.
[22] See https://swiftype.com, last visited Sept. 18, 2016.
[23] See https://www.google.de/work/search/products/gsa.html, last visited Sept. 18, 2016.

Information Retrieval Systems can be used in various environments owing to their technological diversity (IBM Watson technology serves as an example[24]).

Besides the retrieval of documents, critical business processes often need to combine the knowledge of documents from different sources. Therefore, the linkage of information across system borders is necessary for successful information access. This is a challenge for both the technology used as well as the business architects who support and develop the retrieval systems. Search technologies have their strength in finding information quickly. However, they are weak in the search for information stored in complex and adaptive knowledge networks, as well as representing them in a business model.

Other technology approaches concentrate on linking and merging information (or documents) from different resources. Firms often use these approaches to represent knowledge of business critical processes; for example, they combine information from different sources by establishing a product taxonomy. Several technologies can be used to enable data discovery and link documents semantically across system borders. For example:

- Enriching documents manually with link information to other documents.
- Setting up databases that hold document linkage information.
- Building universal ordering criteria associated with the documents (a product taxonomy, for example).
- Placing web services on top of a service-oriented architecture based upon the concept of service-oriented computing (see [69]) can be used to combine the information of various systems into a distributed information system.
- Setting up federated database architecture to combine data from different sources.
- Using semantic web technologies for annotating entities in documents.
- Integrating information in other systems (like indexing in an information retrieval system).
- Enabling data normalization (standardizing document authoring processes based upon terminology databases and a controlled vocabulary).

As stated in [86], enterprise search engines "*... must be able to regard the entire business environment as a source and should not be bound by a certain product repository or platform*". Many different sources (databases, intranet, email, etc.) should be included in the enterprise search index to face the requirements of a business task.

In [38], Hawking names five scenarios of enterprise search ("*external visitor to enterprise website*", "*Intranet search*", "*internal multi-source search*", "*searching for other than documents*", "*task-integrated corporate memory search*" and "*forensic search*"). Internal multi-core search describes the challenge of finding, ranking and accessing data of different sources and formats. For example, a project manager who needs information about a

[24]For details about IBM Watson visit http://www.ibm.com/smarterplanet/us/en/ibmwatson, last visited Sept. 18, 2016.

running project must search in different sources, including the Intranet, file shares, PowerPoints, emails, project reports, etc.

This challenge can be extended with the objective of finding data from a wide range of sources that is necessary to solve a special and well-defined business task. For example, the aforementioned project manager must prepare a management report about the project's current status. Therefore, he needs to search for all of the information about the project.

Additionally, seven open research problems in the field of enterprise search are named in [38], including "*effective ranking across heterogeneous collections characteristic of enterprises*". This addresses the retrieval across heterogeneous document types and retrieval systems. The documents differ in their types and content, which causes challenges in document result ranking and result presentation.

An enterprise search faces the challenge of obtaining the right information at the right time for the right process. This includes all of the semantic knowledge that can be found in internal sources and that helps to solve a given problem (or task). Therefore, current information retrieval systems (especially an enterprise search) must face the given limitations:

- A search across heterogeneous document types requires different retrieval strategies and result presentation.

- Retrieval is only based upon search technologies or database retrieval (joins across tables of relational databases). The links between documents have only limited influence on the case-dependent retrieval (besides the PageRank algorithm, see [68] for more information about PageRank algorithm). Furthermore, these links are not semantic relationships.

- Relationships between the documents cannot be represented in an index of search results (information retrieval systems are not designed for this approach).

- The automatic collected feedback of the users is not collected and processed for an automatic optimization of the retrieval algorithm. This means that previous search results and user interactions are not analyzed by the systems. The hypothesis is that this automatic collected user feedback (Precision-Recall-based[25]) can be used to improve the retrieval system's quality. A possible solution to this challenge is recommendation techniques provided by collaborative filtering approaches (see [81] or [22]). Nonetheless, they do not provide adaptive relationships between different kinds of documents nor pathways of related information in a network of documents.

To summarize, an enterprise search based upon information retrieval technologies serves as a basis for an intelligent information search of future business cases. Unfortunately, enterprise search systems are not optimized for knowledge access based upon the linkage of documents. Furthermore, information from the search will not be re-used in future research.

[25]For example, see [73] and [83] for more information about evaluation and measurement in information retrieval.

1.2 Research Questions and Methods

The disadvantages of current technologies facing the research challenges (information access, information search and information linking) are that they require data manipulation steps and are complicated to use with existing search technologies. In addition, many of these approaches are static in information linking: they do not enable weighed or adaptive knowledge networks of document linkage.

The need for a simplification of search for business relevant information causes the need for a research activity, whereby the main research questions can be formulated as follows:

1. Can a single systems view be provided for all of the case-related documents kept in different retrieval systems?

2. Can seamless and guided access across these disparate and disconnected retrieval systems be designed?

3. Can the quality of retrieval results and the effectiveness of the retrieval process be improved by exploiting user feedback?

4. Can a technical solution be developed that is accepted by users in the field?

The findings should answer the question of how a knowledge network can be deeply integrated in information retrieval technologies to support both heterogeneous document landscapes in large company domains as well as collective intelligence interaction between knowledge producers and knowledge users[26].

To find answers to these research questions, it is necessary to develop a system that enables a single systems view on disparate retrieval systems. Furthermore, this system should be able to process users' feedback to increase the quality of future searches. A prototype implementation of the system must be evaluated in real situations to obtain information about the acceptance of the system.

This means that a search across all types of documents must be established and a homogeneous access structure for the heterogeneous document landscape must be developed. Enterprise search engines provide state-of-the-art search technology. Information linking describes the challenge of establishing semantic relationships between data records. These semantic relationships are used to retrieve a homogeneous access structure from the heterogeneous document landscape. A homogeneous access structure provides a semantic knowledge network to the users. This can be achieved by various methods and technologies of information linking, as named in Section 1.1. In order to make the structure of the knowledge network accessible and shareable, it should be represented by an ontology (regardless which method or language is used to describe concepts of the ontology). The ontology itself is a model of the heterogeneous documents landscape and enables cross-document relationships. The document relationships can be used later to include the knowledge of users' feedback to raise the quality of document retrieval. This ontology again can be accessed through a knowledge representation framework. Examples of these frameworks include Ontopia[27] (Topic Maps) or Apache Jena[28] (OWL). The following

[26] In the current work, the two roles of knowledge producers and knowledge users are represented through the authors of the documents as well as through the employees who use the documents.

[27] See http://www.ontopia.net, last visited Aug. 18, 2016.

[28] See http://jena.apache.org, last visited Aug. 18, 2016.

steps include the elements of retrieval for documents and knowledge representation. They are suggested in this work, with the hypothesis that they provide a workflow of modern knowledge network-based information retrieval.

1. Design an overarching generic model that represents all documents, sources and relationships in a given business or organizational context. This model should support adaptive document relationships to reflect user feedback.

2. Build ontological individuals that represent the original documents and their sources in a uniform way to abstract irrelevant diversities. This is necessary to model the structure of a heterogeneous document landscape.

3. Analyze the source data and construct index documents for a state-of-the-art retrieval system. The indexing process is necessary to prepare the documents for searching.

4. Link each index document to the corresponding ontological individual. This is necessary to synchronize the index documents with the corresponding ontological individuals.

5. Map hard-coded document links to document relationships represented in the ontology. This is necessary to represent real existing document relationships in the ontology because they can be used in the retrieval to search for related documents.

6. Develop an evolutionary process for analyzing user behavior and use the corresponding feedback to adapt document relationships over time. Documents that have been used in the same business case are related to each other. These relationships can be represented in the ontology as relationships between the ontological representations of the documents. Furthermore, these relationships between the ontological individuals can be used later to search for related documents.

These suggested steps should enable developing retrieval algorithms beyond state-of-the-art natural language text retrieval due to a combination of an ontological model of documents with a corresponding index document structure in the retrieval process. Algorithms can be combined with technologies from the field of knowledge representation to include document relationship information in the entire retrieval process. The approach of combining knowledge representation through ontologies with enterprise search technology and feedback processing was chosen to answer the research questions. Therefore, this approach served as the basis for the research presented in this work. Companies can use this approach as a blueprint and apply it to their processes and documents landscapes. This work relies on existing technologies for knowledge representation through ontologies and state-of-the-art search technologies. The main scientific contributions of this thesis include:

- Combining multiple technologies to enable overall documents search: a system called Advanced ontology-based Information Retrieval System (AIRS) was developed, which includes methods of state-of-the-art enterprise search technology and combines them with an ontology called AIRS Knowledge Base (AIRSKB). AIRSKB models the heterogeneous document landscape and was developed to enable an intelligent overall retrieval and adaptive document relationships. The search algorithms

use document relationships, which are updated by an automated user feedback analysis. This work explains the concept, basic architecture and the indexing, retrieval and feedback workflows of AIRS.

- Design of an ontology that models the entire document landscape, provides a holistic search space and serves as a blueprint for ontologies that enable an overall document search in a heterogeneous document landscape. AIRSKB acts as a homogeneous access structure for a heterogeneous document landscape. This work explains AIRSKB's application context, conceptualization, theory and inference rules as well as its usage in the context of AIRS.

- Development of collective retrieval strategies and feedback analysis algorithms. For example, a search approach called Suggest Cluster Algorithm was developed during the research activities and is suggested in this work. This algorithm updates a knowledge network of document relationships by analyzing user feedback.

- Empirical study testing the effectiveness and quality of an ontology-based retrieval system and evaluating its usability and acceptance. The evaluation showed that a work environment based upon AIRS helps to optimize processes, while reducing the time that an employee needs to search for business case-relevant documents.

Furthermore, this work describes an exemplary implementation of AIRS developed during the research: AIRS Prototype. The AIRS Prototype was used to validate the methods and benefits for the workshop business cases by performing field tests. The hypotheses underlying these goals of these field tests were as follows:

- The workshop employees perceive the search for relevant documents using AIRS Prototype as helpful and easy to use.

- A system based upon the AIRS Prototype helps to optimize the entire workshop process by reducing the time that a workshop employee needs to search for business case-relevant documents.

To achieve this, the field tests include two main aspects: a "before–after" comparison of existing workshop retrieval systems with the AIRS Prototype and system tests regarding domain-specific test scenarios. The field tests were carried out in five workshops. Workshop employees with many years of professional experiences tested the system in their roles as domain experts. All workshop employees provided very valuable feedback, highlighting that a system based upon the AIRS retrieval principle could increase the workshop processes. The field tests showed that overall retrieval in a heterogeneous document landscape best supports the workshop's business cases and that collective intelligence feedback enables the automatic building of valid document relationships.

One finding of this work is the answer to the question of how a knowledge network can be deeply integrated in information retrieval technologies to support both heterogeneous document landscapes in large business domains as well as the collective intelligence interaction between knowledge producers and knowledge users. AIRS showed that such a deep integration must be performed by including an ontology that represents the heterogeneous document landscape. Moreover, the ontology should be used during the retrieval for representing document relationships as well as searching for related documents. AIRS

and AIRSKB provide a frame for modeling a heterogeneous document landscape and a feedback processing methodology to improve the retrieval quality.

1.3 About This Work

Parts of the research findings have already been presented in articles and discussed at conferences. This work enriches these findings with a description of a prototype implementation. The theoretical findings are updated in this work to reflect the current state of research. In addition, workshop experts with many years of technical experience have evaluated the prototype implementation of the new system in different workshops. This work includes parts of the following publications by the author of this work:

- **Modellieren im Kontext: Ontologie-basiertes Information Retrieval (in German)**. This technical report (see [93]) focuses on the theoretical background of information retrieval and knowledge representation through ontologies. It introduces the principle of the Application Context and its role in ontology engineering. Moreover, the principle of the Advanced Ontology-based Retrieval System (AIRS) was presented and equations for the navigation of ontology networks were given.

- **Harvesting Domain-Specific Data Resources for Enhanced After-Sales Intelligence in the Car Industry**. This article was published in the conference proceedings from the COLLIN 2011 conference (see [95]). It includes an explanation of the principle of AIRS and knowledge representation through ontologies. Furthermore, a case study is presented where the workshop document landscape and systems were examined to derive valid statements about the linking potential between various documents of different retrieval systems.

- **An Advanced Approach to User-based and System-centered Evaluation for the Improvement of Business-oriented Document Retrieval Systems**. This paper was first presented and discussed at the SDPS 2012 Berlin conference and it was later published through the conference documentation (see [94]). The content of the paper describes the challenges of developing the AIRS system and introduces the principles of internal and external evaluation. The outcome was that an iterative process called internal evaluation helps to improve the retrieval quality and an external evaluation should be conducted to obtain good statements about the retrieval system's quality. The external evaluation should also be conducted over time during the system's operating time to obtain information about the retrieval system's *acceptance* and *goodness*. Moreover, current information retrieval system evaluation methodologies were discussed and a suggestion for the evaluation of business-triggered retrieval systems was provided.

- **Workshop Process Optimization Based on the Collective Intelligence of Workshop Employees Involved in After-Sales Intelligence of Mercedes-Benz Cars**. This article was published in the *International Journal of Cooperative Information Systems* in 2013 and explains how the collective intelligence of workshop employees can be used to improve the system's quality by using updates on the document relationships stored in the AIRS ontology ([96]). In this publication, the AIRS ontology was first introduced as the AIRS Knowledge Base (AIRSKB).

The work is structured as follows. Chapter 2 presents exemplary research findings in the field of defining the concept ontology (Section 2.1), approaches of ontology engineering (Section 2.2), structuring of ontologies and their usage in different contexts (Section 2.3). Chapter 3 introduces AIRSKB as an adaptive document network and explains the role of the Application Context (Section 3.1), the Conceptualization (Section 3.2) and the importance of Inference Rules (Section 3.3). The chapter concludes with a summary of AIRSKB development steps in Section 3.4.

Chapter 4 starts with a definition of the concept "Heterogeneous Document Landscape". Section 4.1 subsequently explains and defines the concepts of a heterogeneous document landscape that exists in different domains and compares these concepts with AIRSKB elements. Furthermore, Section 4.2 introduces the concept of the Advanced Ontology-based Information Retrieval System (AIRS) and Section 4.3 provides an overview of the basic system architecture.

Chapter 5 explains how the document indexing as well as the retrieval work in detail and provides an introduction to the Suggest Cluster Algorithm. Therefore, Section 5.1 introduces the indexing of documents and Section 5.2 introduces the general retrieval and feedback workflow of AIRS. The Related Documents Search (see Section 5.3) and the Suggest Cluster Algorithm are subsequently explained (see Section 5.4 and Section 5.5). The role of collective intelligence in AIRS' feedback processing and knowledge network adaption is explained in detail in Chapter 6. Section 6.1 explains how the feedback of employees can be collected through feedback processing. Section 6.2 suggests methods to calculate the relevance of automatic collected feedback. The chapter concludes with a brief summary of AIRS and collective intelligence in Section 6.3.

Chapter 7 focuses on the AIRS Prototype, which is a prototype implementation of AIRS. AIRS Prototype is implemented as a rich web application that uses various frameworks for retrieval and knowledge representation. During AIRS Prototype development, a number of frameworks and components were implemented that take the lead of special activities. Section 7.1 presents a solution for properties management that was also developed. In Section 7.2, the AIRS Index & Search Framework component is presented, which is responsible for the document retrieval core. In Section 7.3, the AIRSKB Framework component is introduced, which is responsible for all knowledge network operations. Section 7.4 introduces the component that takes care of the inclusion of new document sources in AIRS (AIRS Include Sources). Section 7.5 introduces retrieval and feedback-processing algorithms of AIRS. Section 7.6 concludes the chapter with an explanation of the prototype environment. As stated before, the AIRS Prototype was evaluated in the workshops by using real-world use cases and workshop employees as technical workshop experts (system users). Chapter 8 proceeds in detail regarding the evaluation process. First, Section 8.1 introduces the domain of the field test. Subsequently, Section 8.2 introduces the AIRS Prototype user interface. Section 8.3 presents the experimental setup of the user tests and Section 8.4 explains how the tests were performed. The field test results are discussed in Section 8.5. Connected to this chapter are Appendices A.1, A.2 and A.3, which show examples of surveys and a user test description used in the field test. Appendices A.1 and A.2 present two surveys conducted during the field tests to derive a "before–after" comparison of workshop document retrieval with and without using the AIRS Prototype. Appendix A.3 shows workshop case scenarios used in the field tests.

Chapter 9 concludes with a summary of AIRS research (see Section 9.1) and provides an outlook of possible future research activities in the field of AIRS (see Section 9.2).

2 Ontologies in Computer Science

Ontologies have become key elements in computer science, being widely used to represent different kinds of knowledge. For example, they are used to:

- build a vocabulary to describe a particular domain;
- make intended meanings of terms explicit; or
- define constraints for a modeled domain.

The use of ontologies depends on the underlying application scenario, which describes a design goal for ontology development. In [93], several design goals are named. For example, possible design goals for the development of an ontology are:

- Defining a world of entities and relationships between them to derive a model of domain knowledge. An example is the top-level ontology OpenCyc[1]
- Establishing a lexicographic knowledge base that can be used for text analysis. For example, WordNet serves as a lexical database of English language[2].

The objectives are to derive a clear and shareable picture of a particular domain and make this picture accessible by computers. In order to clarify the term ontology and engineering processes of ontologies, this chapter starts with exemplary research findings in the field of ontological knowledge representation and engineering approaches. Furthermore, it presents scientific findings about the structuring and usage of ontologies in different contexts.

2.1 Concept Formation

An ontology is at least a knowledge base that helps to store, use and even re-use information about *things* in a given domain. These "things" can be representations of both physical objects like *cars* or non-physical entities like *accidents*. Both are *concepts* that represent elements of a domain. Finding all concepts of a domain is an elementary task for knowledge base development. After the set of all concepts that describe a domain is defined, the knowledge that comes in this concept set is by an ordering criterion. This ordering criterion comprises semantic relationships, which relate the concepts to each other. In this sense, a taxonomic hierarchy can be established as a top-level criterion to order concepts from the most generic to the most specific. This taxonomy hierarchy subsequently serves as a categorization of concepts. For example, a taxonomic relationship can be established between the two concepts *company* and *car producer* with the company as a root node and

[1] See [26] and http://www.opencyc.org, last visited Sept. 18, 2016.
[2] See http://wordnet.princeton.edu, last visited Sept. 18, 2016.

the car producer as a leaf node. This means that the concept of *company* is more generic than the concept of *car producer*, although *generic* means different things. On the one hand, one concept is more generic than another if the set of extensional interpretations describing the concept includes more single entities[3]. For example, a car producer produces and sells cars, whereas a company manufactures products, provides services and performs any other business. The products can be cars, as well as other things. Again, business is a more general description than selling cars. Therefore, the set of entities (extensional interpretations) that belong to the concept of a car producer is much smaller than the set of entities that belong to the concept of a company (including car producers). Accordingly, the car producer concept includes entities like Daimler AG and BMW AG but not Apple Inc. By contrast, the company concept includes all three entities. On the other hand, a concept is more generic than another one if at least the intentional setting contains fewer attributes than the more specific one.

For example, the company concept can be described through the following set of attributes: {*is-company* and *do-business*}. By contrast, the set of attributes that describes the car producer concept contains an additional attribute that makes the concept more specific: {*is-company*, *do-business* and *produce-cars*}. The most specific concepts are comparable due to the same set of attributes. They should be identified as ontological individuals in the ontology since these individuals are the current *instances* of the modeled domain knowledge. All other concepts should be modeled as ontological classes. Following the example of the company and car producer concepts above, *Daimler AG* should be modeled as an ontological individual of the ontological class *car producer*. Ontological classes inherit attributes (intentional settings) along the hierarchy from the most generic to the most specific concepts. Therefore, ontological individuals are also *instances* of the most generic ontological class as well as the more generic classes along the same hierarchy. In addition, all ontological individuals that belong to the same ontological class have the same set off attributes but can differ in their attribute values. Ontology design is a difficult engineering process, especially in the differentiation between ontological classes and individuals (see [67]).

Other semantic relationships should also be placed in the ontology, besides the taxonomic categorization. An example of this is a *has-a* relationship that links *cars* to *accidents*. This relationship brings sense into the knowledge network, which contains the concepts of *cars* and *accidents* (as long as the relationship *has-a* is clearly defined).

In the field of knowledge representation, a clear view of an ontology is necessary to enable applications to work with this domain knowledge. Unfortunately, the general description of what an ontology is can be interpreted in various ways. However, knowledge representation using ontologies is a wide field of research initiated a long time ago. The term ontology comes from the Greek language and means science of being. More than 2,000 years after the philosophical research in the field of ontologies began, Gruber answered the question in [30] concerning what an ontology for modern knowledge representation should be. He named an ontology as the "*explicit specification of a conceptualization*". A conceptualization is an abstract and simplified view of a domain that is described through a set of objects and a set of relationships between these objects.

In their work [33], Guarino and Giarette discuss the topics of ontology and knowledge

[3] Further information about the extensional and intentional description of concepts and semantic relationships can be found in [80], [76] and [50].

representation in detail. They stated that the meaning of the word *ontology* tends to remain a little vague from their perspective. To underline this statement, they summarize and analyze different interpretations of the term ontology. Accordingly, an ontology can mean a "*philosophy discipline*", an "*informal conceptual system*", a "*formal semantic account*", a "*representation of a conceptual system via a logical theory*", a "*vocabulary used by a logical theory*", a "*(meta-level) specification of a logical theory*" and (as described in [30]) a "*specification of a conceptualization*". Guarino and Giarette stated that the first interpretation differs from the other ones since it describes a philosophical discipline. By contrast, they argued that the others have more a technical meaning of the word ontology. Furthermore, they name the term "*conceptualization to denote a semantic structure that reflects a particular conceptual system ... and ontological theory to denote a logical theory intended to express ontological knowledge*". Moreover, "*conceptualizations ... are the semantical counterparts of ontological theories*". By doing so, they try to clarify the technical interpretation of the term *ontology*. They claimed that Gruber's definition of an ontology does not match the criteria and that the definition needed to be modified to reflect that "*an ontology is only a partial account of a conceptualization*". They finally provided their own definition of a technical ontology that fits their interpretation: an ontology in one sense is "*a logical theory which gives an explicit, partial account of a conceptualization*" and in another sense it is a "*synonym of conceptualization*". In [32], Guarino suggests the following definition: "*an ontology is a logical theory that constrains the intended models of a logical language*". To summarize, various definitions of the term ontology exist in literature, which can differ from each other.

The representation of domain knowledge through concepts and relationships is the core statement of almost every definition. The concept of Knowledge Representation in Context presented in [93] extends this core statement and names three elementary components of an ontology: an Application Context in combination with Conceptualization and Inference Rules. Following this, a matching definition of the term ontology for this work is:

An **ontology** is a knowledge base that is described by an application context and defined through a conceptualization in combination with inference rules.

2.2 Approaches of Ontology Engineering

Ontology engineering is a difficult task. Different approaches of methodologies building ontologies are introduced in the literature. In [85], Uschuld and King discuss a methodology for building ontologies. They explain that ontology development includes the stages of "*identifying the purpose*", "*building the ontology (capture, coding and integration)*", "*evaluation*" and "*documentation*". *Identifying the purpose* is close to the Application Context due to the importance of questions regarding "... *why the ontology is being built and what its intended uses are*". Moreover, for their ontology development, Uschuld and King focus on Gruber's definition of ontology.

In [56], López summarizes methodologies for building ontologies and compares them with the IEEE Standard 1074-1995 for software development. Furthermore, he explains the criteria for analyzing the different methodologies. For example, one criterion is the choice

of a strategy for building the ontology, which can be *application-dependent*, *application-semidependent* or *application-independent*.

In 2001, Noy and McGuinness wrote in [67] about their experiences in ontology engineering. The authors suggested the seven steps "*determine the domain and scope*", "*consider reusing existing ontologies*", "*enumerate important terms in the ontology*", "*define classes and the class hierarchy*", "*define properties of classes*", "*define the facets of the properties*" and finally "*create instances*". Furthermore, they discuss the reason why someone should develop an ontology. They explained that the main reasons are sharing, re-use of domain knowledge and making domain assumptions explicit. Ontologies are developed to separate domain knowledge from operational knowledge as well as analyzing the domain knowledge.

According to Hosapple and Soshi, five approaches ("*Inspiration*", "*Induction*", "*Deduction*", "*Synthesis*" and "*Collaboration*") of *ontology-design* are necessary to build an ontology (see [44]). Additionally, in 2007, Braun, Schmidt and Walter named five steps of ontology engineering using a collaborative approach: "*emergence of ideas*", "*consolidation in communities*", "*formalization*" and "*axiomatization*" (see [7]). In 2001, Staab et al. stated that ontologies are the appropriate answer to the question of how to build some kind of organizational memory for IT-supported knowledge management solutions in enterprises (see [77]). They introduced a meta-process for ontology engineering that includes two steps. The first step is the *feasibility study* phase, which comprises *project-setting* activities like *identifying the problem*. The second step is the ontology development itself, which is again divided into four tasks: *ontology kickoff*, *refinement*, *evaluation* and *maintenance*. An ontology kickoff task focuses on early development steps that concentrate on typical questions for the goal, domain and scope or usage scenarios. Therefore, the output is an ontology requirements specification that should guide the ontology engineer in further development steps. In the refinement phase, "*a mature and application-oriented ontology*" should be built upon the base of the ontology requirements specification built in the ontology kickoff phase. In the evaluation phase, the ontology will be – inter alia – tested and analyzed through beta tests. The maintenance phase concentrates on the life cycle of the ontology. The authors argue that an ontology has to be "*maintained frequently like other parts of software*". The authors depict that the three phases of refinement, evaluation and maintenance should form an iterative process during the ontology life-cycle.

Later in 2002, Gruninger and Lee presented their findings about the use of ontologies and ontology engineering (see [31]). They stated that ontology engineering "*considers the entire ontology life cycle*", which again comprises "*design, evaluation, validation, revision and deployment*". Furthermore, they identified two kinds of extremes in ontology design: small lightweight ontologies, which were built to share and merge knowledge; and heavyweight formal ontology, which were mostly built and developed by "*consortia and standards organizations*". Fernández-López and Gómez-Pérez stated in 2002 that no completely mature proposal for building ontologies exists (see [25]). The authors introduced methodologies for building ontologies from three different groups and compared them with the IEEE 1074-1995 standard for software development that comprises the following steps: *project management phase*, *software development-oriented processes*, (*pre-development processes*, *development processes*, *post-development processes*) and *integral processes*. The three groups of methodologies for building ontologies were:

1. methodologies for *building ontologies from scratch or reusing them without transforming them*: Cyc, Uschold and King, Grüninger and Fox, KACTUS, METHON-

TOLOGY, SENSUS.

2. A *methodology for re-engineering ontologies* that comprises the three steps: *reverse engineering*, *restructuring* and *forward engineering*.

3. *Methodologies for cooperative construction of ontologies*: $CO4$ and $(KA)^2$.

The authors found that none of the methodologies for building ontologies completely match the IEEE process, "*since there are some ... techniques that are missing in all these methodologies*". Pinto and Martins provided an overview of ontology building in 2004 (see [71]). They describe the development life-cycle of ontologies that comprises *specification*, *conceptualization*, *formalization*, *implementation* and *maintenance*. Additionally, *knowledge acquisition*, *evaluation* and *documentation* should be the activities that accompany the whole development of life cycles. The authors also describe and analyze methodologies to build ontologies from scratch (TOVE, ENTERPRISE and METHONTOLOGY). They came to the conclusion that ontology building remains "*more a craft than an engineering task*".

2.3 Structuring and Using Ontologies

The objective of almost every ontology-based knowledge engineering process is a storage structure that holds the ontological elements. Such a storage structure can be informal, such as natural language text files or described by a database schema. The Web Ontology Working Group of the W3C developed a knowledge structure that was based upon RDF/RDFS to store and share ontological knowledge. Therefore, in 2004, the W3C gave a recommendation for the *Web Ontology Language (OWL)* (see various web-articles at [12]). Before that, Horrocks, Patel-Schneider and van Harmelen wrote an article about the rise of OWL in 2003 (see [45]). In the same year, Bouquet et al. presented their work in [6], where the authors extended the OWL language (called C-OWL) to allow the representation of contextual ontologies. Contextual ontologies are again "*local (... not shared) models that encode a party's view of a domain*".

Ontologies can be categorized along different attributes. Therefore, many researchers focus on the usage and importance of ontologies for different areas. For example, the differences of database schema evolution and ontology evolution were discussed by Noy and Klein in an article in 2004 (see [66]). Furthermore, the authors summarized the evolution of ontology research in the field of computer science. They stated that the research interests started with "*defining what a formal ontology is*" and subsequently the researcher focused on representation languages for ontologies. Later on, the research shifted to the "*vision of widespread and reuse of ontologies*". After this, the development of content ontologies (like OpenCyc) emerged as the most important research field. Having developed a large set of ontologies, researchers focus on the topics of merging ontologies. Finally, the authors state that evolution and versioning of ontologies are current research interests in the field of ontology research. Benjamins et al. discuss in [4] knowledge management through ontologies. They introduce a methodology of annotating HTML pages with ontological facts to enable retrieval across a large community of similar domain knowledge. They argue that this kind of knowledge management is based upon ontologies that can be specifically used in companies. Furthermore, they distinguish between two types of

knowledge management systems: "*vertical and horizontal systems*". Vertical knowledge management systems are "*often developed inside a company and are highly situation specific*". By contrast, horizontal knowledge management systems can be "*applied to a variety of business situations*".

In [9], Chandrasekaran, Josephson and Benjamins write about ontologies and the role that they play in information systems and AI. They explain that ontologies belong to the content theory about "*sorts of objects, properties of objects, and relations between objects that are possible in a specified domain of knowledge*". In their work, the authors focus on the different aspects of understanding ontologies and the conceptualization by natural language terms and relationships. Representation vocabularies are sets of terms that describe facts of the given domain, while the body of knowledge is a collection of facts about the domain. Moreover, they explain different usages of ontologies – for instance – that are used in information retrieval systems, explaining that Internet search engines "*need domain ontologies to organize information and direct the search processes*".

When the semantic web emerged in 2000, ontologies were named as knowledge structures that could help to bring an understandable order and reasoning to a semantic machine in the chaos of the growing world wide web. Researchers focus on semantic web technologies and the use of ontologies in this field. For example, in 2001, Maedche and Staab presented their findings about learning ontologies and their use for the semantic web in [59]. They argued that the success of the semantic web depends – for example – on ontologies and the requirement for their fast and easy engineering. Ontology learning itself "*facilitates the construction of ontologies by an ontology engineer*". They presented their ontology engineering workbench *OntoEdit* that helps to create ontologies for the semantic web, which should include the import of existing ontologies. Modeling and information extraction of existing source documents (web pages that help to build ontological knowledge). These documents can be structured (databases), semi-structured (schema-information) or free text. Moreover, the authors consider a process of ontology learning that combines knowledge acquisition with machine learning. For the authors, an ontology comprises a number of sets of concepts, relations, lexical entries and links between these entries. Other authors postulated similar ontology elements over the years. For example, in 2006, Sure, Ehring and Studer (see [82]) argued that an ontology should include a lexicon that is a terminological mapping of the domain, concepts, semantic relations and rule-based links between the concepts. Maedche et al. (see [59]) explained the relations between concepts are divided into a set of taxonomy relations, a set of non-taxonomy relations between concepts and relations that relate concepts and relations to their lexical entries. Finally, a set of axioms "*that describe additional constraints of the ontology*" should belong to an ontology.

In 2001, Wache et al. characterize in their article (see [91]) the three main architecture approaches: "*single ontology approaches, multiple ontology approaches, and hybrid approaches*". While single ontology approaches use a global ontology to describe all of the information sources, multiple ontology approaches describe every information source via its own ontology (local ontology). Hybrid ontology approaches, instead, establish a global shared vocabulary over the local ontology to make the local ontologies comparable.

Maedche and Staab stated in 2002 that there is a need for measuring similarity between ontologies. This need is founded on the requirements of extending, adapting or comparing ontologies with other existing ontologies. Therefore, they focus their research on the measurement of similarity between ontologies (see [60]). They introduce methods for measuring the similarity between two different ontological levels: the lexical and the conceptual

level. At the lexical level, a measure based upon the Levenshtein's edit distance (see [54] for more information about Levenshtein distance) is used to calculate the string similarity of lexical entries of the two ontologies. At the conceptual level, concepts (hierarchical ranges) and relations (domains and ranges) of the two ontologies were compared. Furthermore, the authors performed an evaluation study at an ontology engineering seminar. They wanted to determine the *quality of their measures* and "*get an intuition about how similar ontologies of the same domain that have been modeled by different persons*". They let the users first build a taxonomy with a given top level structure. Second, they build a taxonomy (including relations) with a given set of concept-describing terms. Third, they build relations over a given taxonomy. According to the authors' results, it was interpreted that lexical entries referring to concept "*have a considerably higher agreement*" than ones referring to relations. Furthermore, they state that users who have to model a taxonomy "*tend to agree or disagree on taxonomies irrespective of the amount of material being predefined*". Additionally, they ascertained that users "*found it easy to use a predefined lexicon but extremely difficult to continue modeling given a predefined taxonomy*". The authors' summary about their measures is that they can be applied in different application fields to measure the similarity between two given ontologies. The use of Levenshtein distance seems to be a valid model for comparing the taxonomic lexical entries of two concepts. This is important for ontologies that represent domain-specific snippets of the world that can be described through natural language. However, the approach can also be used for ontologies that describe similar sets of objects (documents and sources). Measuring the similarity of documents is important due to the task of updating document information: documents can change over time and can be added or deleted. The measurement presented in [60] can be used to compare attributes values of given documents.

The research in the field of ontologies remains ongoing and has become increasingly relevant in different fields of computer science. However, not all of it holds relevance for the given research work. For example, Roman et al. describe in their article (see [72]) a web service modeling ontology (WSMO) that provides a formal language for describing relevant aspects of web services "*in order to facilitate the automatization of discovering, combining, and invoking electronic services over the Web*". Wang et al. presented in 2004 their work on context modeling and their reasoning was using OWL in [92]. Ontologies for enterprise knowledge management are discussed by Meadche et al. in [58]. The authors present their research on enterprise knowledge management architecture to "*support multiple ontologies*" and "*manage ontology evolution*". In 2006, Noy et al. introduced a framework for ontology evolution in collaborative environments (see [65]). They analyze and categorize "*different scenarios for ontology maintenance and evolution*", develop a solution in a "*single unified framework*" and implement the solution as a set of plugins for the Protégé[4] software. In 2007, Ding et al. provided an overview of findings in the field of using ontologies in the semantic web and semantic web languages such as RDF/RDFS, DAML+OIL and OWL (see [20]). Accordingly, the semantic web language is relevant for knowledge storing, including for this work. Furthermore, the authors discuss different web ontology tools: ontology editors (Protégé, SWOOP), ontology repositories (like DAML Ontology Library or Swoogle) and ontology language processors for reasoning (like FaCT++, Racer and Pellet).

[4]Protégé is an open source editor for modeling ontologies and available at http://protege.stanford.edu, last visited Sept. 18, 2016.

3 AIRS Knowledge Base

The concept of Knowledge Representation in Context introduces the Application Context as a frame that helps to understand a domain as well as the need for a knowledge base containing domain information. The domain of solving complex business tasks and the requirement to access documents that help to solve the tasks result in a need for such a knowledge base. This knowledge base comes in the form of an ontology.

This chapter describes an ontology that was developed to represent documents, storage containers of documents, attributes of documents, restrictions of document validations and relationships between documents. This ontology is called AIRS Knowledge Base (AIRSKB) and contains ontological elements (classes, individuals and relationships) to model an adaptive knowledge network of documents. In this chapter, the Conceptualization of AIRSKB is explained and Inference Rules as a frame for navigation through the ontology are introduced.

As stated above, AIRSKB is an ontology that combines a model of documents, sources of documents and document restrictions of document validations with the users' understandings of related information. Related information means different kinds of relationships between documents between the documents and their relationship strange can change over time depending on a rating of relevance. AIRSKB is an adaptive knowledge network that was developed during the research phase of this work. By its nature, AIRSKB is an ontology with classes, individuals and weighted relations between the individuals (see [30], [9] and [34] for a detailed introduction to ontologies). According to the characterization of ontologies presented in [91], AIRSKB is based upon a single ontology approach because each of the sources is modeled through use of the same domain-specific ontology. To reach this, an approximated and general valid model across documents, sources and document relationships was developed. This model can be used to represent any kinds of new document sources as long as the sources fit a set of restrictions of the ontological model: the source must contain documents, documents comprise segments and segment values and documents can be related to each other. Moreover, AIRSKB is not a global ontology; rather, it is highly domain-specifically built for approaches that need document representations.

The development of AIRSKB followed an application-dependent strategy (see [56]) due to the process of abstraction of documents and sources and their use for given business dependent retrieval. It was built by using the concept of Knowledge Representation in Context, which is presented in [93] and [95]. It comprises three parts of ontology development:

- Application Context provides an answer to the following questions: why is an ontology necessary, how should it look like and for what should it be made?[1]

[1]The ontology kickoff task, presented in [77], is similar to the Application Context ontology-engineering phase where the scope of an ontology is identified.

- Conceptualization is a complex model that comprises classes, individuals and weighted semantic relations between the individuals.

- Inference Rules are necessary to navigate through the ontology.

While developing AIRSKB, the current understanding of *what an ontology for this work is* and the approaches discussed in literature served as frame for the ontology engineering steps. Therefore, existing approaches (see previous section) were adapted to the following design steps:

1. *Initial activities.* Focusing on the questions that the Application Context defines. This helps to understand the domain and the need for an ontology (see Section 3.1).

2. *Performing conceptualization.* The step defines classes and relationships (see Section 3.2).

3. *Defining inference rules.* The theoretical frame for the knowledge acquisition (see Section 3.3) means defining the mathematical background. Axioms and equations enable ontological reasoning.

4. *Performing indexing.* This step involves the process of the ontological individuals that match the frame of the conceptualization (the real-world entities that belong to the ontology). This is presented in Section 5.1.

5. *Ontology representation.* Defining an ontology representation framework or language that can be used to share the ontology among applications and people. This also involves the technology concerning how the ontology can be used in applications.

6. *Performing application usage and ontology evaluation.* This step comprises the usage of the ontology in an explicit application environment. This can be seen as a second evaluation step, which guarantees that the ontology is at least usable. Developing and testing a prototype solution that uses the ontology is similar to this step (see Chapter 8).

The engineering approach presented in [59] (see previous section) differs from the approach that was used to build AIRSKB due to the different application contexts: AIRSKB is an ontology that describes a world of documents rather than describing the world of concepts based upon the content of documents. However, the engineering processes differ (AIRSKB's construction does not necessarily contain knowledge acquisition algorithms like text-mining technologies to identify concepts and semantic relations[2]), although the understanding of ontological elements is similar: AIRSKB comprises concepts and relations between these concepts and the Inference Rules build constraints that include axiomatic knowledge about the domain.

[2]Except advanced usages of AIRSKB's architecture like the Suggest Cluster Algorithm, as described in Section 5.4.

3.1 Application Context

Because an ontology is mostly designed for a specific scenario, understanding the Application Context should be the starting point for every ontology-engineering process. Just because there are different types of design goals (ontology as world of entities, ontology as lexicographic knowledge base and ontology as modeling paradigm, see [93] and [95]), several ontologies may differ in terms of both semantics and structure. Following this, an Application Context is a use-case-triggered engineering paradigm that has an impact on both the conceptualization as well as the theoretical and mathematical background. However, what is the best way to deal with it? The quintessence of understanding the Application Context is a clear view of the domain, the Conceptualization and the need for Inference Rules later in the ontology development process. The question is: how can work steps be defined for the design of an ontology and what must be considered? Following these findings, facing the Application Context means to exploring the underlying domain and answering a catalog of questions (see [93]):

a) *Is there finally a use case for an ontology?* The main focus here is the question of "why". Regarding the use case, is there really a need for an ontology? If yes, what are the key benefits of using an ontology?

b) *Is it possible to generate or define relations between ontological concepts?* Relationships are necessary for every ontology. They are the glue that holds together the concepts. Moreover, relations bring sense and semantic knowledge into an ontology. To summarize: without relations, an ontology seems quite useless for many scenarios. Therefore, it is very important to have a clear understanding about the essence and the possibility of finding or defining relations early in the ontology engineering process.

c) *How to perform the conceptualization?* At least, the domain needs to be modeled: it must be clear which kinds of concepts and relations take part and how they can be mapped to a specific syntax of an ontology language or any other kind of storage. One question is: which kinds of concepts and relations are good representations for both the domain and the underlying use case? The main focus here is the underlying use cases' impact on the conceptualization.

d) *Are there additional influences?* A conceptualization should paint a clear picture of the world to be modeled. Often additional influences like conditional relationships between concepts are necessary to paint such a clear and believable picture of the domain. It is necessary to validate the concepts and relations to existing real-world use cases to locate conditions. Therefore, the demand of conditional relationships can be formulated. The task is to bring these conditions into the ontology (if they exist).

With respect to a top-down analysis of use cases in the domain of business case triggered document search, these general questions were adapted as follows:

a) *Why use an ontology for retrieval of documents?* For the use case of overall retrieval, it is necessary to define pathways across system boarders. Additionally, a structure is necessary that can represent, adapt and learn knowledge about document relationships. An ontology as knowledge structure can guarantee all of this.

b) *How is it possible to realize relationships between documents of different document locations?* Documents stored in different locations (for example, isolated retrieval systems) are not connected to each other. Unfortunately, an overall view of all documents and all of the relationships between them is mandatory for various use cases. By harvesting the domain, two ways were found to set or indicate relationships between documents (also across system borders): relations can be defined by editorial processes as well as approximated by harvesting the use cases where documents of several document systems were used together.

c) *How to perform the conceptualization of documents and different document locations?* In the case of a different document sources and business case triggered search, documents and even the locations where they appear (sources) can be seen as central concepts. The next section discusses AIRSKB's conceptualization in detail.

d) *What are existing conditions for document relationships?* The question is: what is a condition of a relation at all and where does it come from? For AIRSKB, this can be answered by Example 1 that explains the concepts of different document locations and processes of car workshop processes. It depicts a relationship between a repair instruction and a set of replacement parts necessary for the car repair under the condition of a special car model. Documents are concepts that can be related to each other. Again, conditions should also be modeled as concepts and they must be linked to the documents to which they belong. Accordingly, a conditional relationship between documents exists, if at least all related documents belong to the same set of conditions. Chapter 4.1 explains the concepts of AIRSKB more in detail.

To summarize, the Application Context provides a clear view of the conceptualization and it can be seen as some kind of legitimization for later ontology development processes. Moreover, it depicts the role of the ontology in a component landscape. For an approach of business case triggered document search, this means that the ontology helps to build a semantic knowledge network. Accordingly, connections can be realized between different document locations. A general definition of a document can be established to enable an overall view of documents. This overall view is subsequently used for document retrieval across system borders. Somehow, a concept is an idea of what a document is should be defined in the conceptualization. The Application Context also helps to identify these relations between documents. Furthermore, it helps to understand that those relationships also can arise from editorial processes or by evaluating old use cases. It also helps to understand how conditions in document relationships can appear and whether they have an influence on the relationships. The next step of AIRSKB engineering process is the Conceptualization itself and later the definition of Inference Rules where best context-sensitive pathways are defined for the overall document retrieval.

By looking at repair instructions and replacement parts, it is clear that these kinds of documents can only be used for a special car model type or a limited group of similar car model types. This means that a workshop employee only needs those repair instructions and replacement parts that fit both the current service task and the customer's car type.

Example 1: Understanding a conditional relationship that exists in car workshop processes between different kinds of documents.

3.2 Conceptualization

Conceptualization comprises ontology engineering where concepts and relationships of a given domain are identified and modeled for a particular use case defined by the Application Context. This section describes the Conceptualization of AIRSKB in detail. Section 3.3 subsequently introduces the corresponding formal definitions of the same AIRSKB elements. The following notation is valid for all elements of AIRSKB and helps to distinguish between ontological elements (classes and individuals) and the objects that they represent:

DOCUMENT (written in small capitals) means both an ontological class or an ontological individual of the class DOCUMENT. For example: a class is meant by formulations like " ... the class DOCUMENT ...", " ... of class DOCUMENT" or "... of concept DOCUMENT". Typical elements include: D for class Document or S for class Source. Instead, an ontological individual of the class DOCUMENT is meant for formulations like " ... a DOCUMENT ... " or " ... a DOCUMENT has". Several ontological individuals are meant by formulations like " ... DOCUMENTS of ... " or " ... concepts of DOCUMENT". Typical elements include: d, d_1, d_2, ..., d_n.

A document (written in lowercase) means the object that is represented by the ontological individual DOCUMENT (see definition of *document* in Section 4.1).

The same statements apply also to the other elements of AIRSKB (source versus SOURCE, for example). Therefore, typical elements for SOURCE individuals include: s, s_1, s_2, ..., s_n. By harvesting both use cases and a retrieval system landscape, the following findings about a domain of document search were made (see [93]):

- The contents of each document location (isolated retrieval systems, for example) are documents.
- These documents are attributed or segmented.
- Documents of the same document locations are attributed or segmented in a similar way.
- Documents can be related to other documents (multiple documents were used together in a single case, for example).
- Relations can be restricted by validity conditions (conditional relationships).

- These conditions correlate with the information needs of the users.

This means that concepts for documents and document locations must be modeled. These documents are attributed or segmented and can have multiple validity conditions. Relations providing information about related documents should also be represented in the ontology. These relationships build a network of related documents and it should be possible to navigate through this network from any document to all of the related documents. In this sense, pathways must be described and mathematical formulas for pathway calculations must be established (see Section 3.3). Therefore, ontological elements for all these requirements are necessary.

Replacement part information of an electronic parts catalog, case describing symptoms of a symptom taxonomy, work units of a work unit catalog and the repair instruction documents are elements that are used together in a car repair process by a workshop employee. Each of them is a single element of information. In AIRSKB, these elements are represented through concepts called DOCUMENTS.

Example 2: Understanding the concept DOCUMENT by example of a car workshop process.

Documents, Sources and Document-Attributes

The concept DOCUMENT is the core entity of AIRSKB and represents a unit of semantically grouped data where the groups comprise different attributes or segments. A group again serves as information literal that is used to solve a particular problem of a business case. Real world examples for the concept DOCUMENT are repair instruction documents, Intranet pages, emails, technical reports or a requirements document.

As written in Example 2, work units come from a work unit catalog and case-describing symptoms are categorized in a symptom taxonomy tree. Furthermore, replacement parts are available in the electronic replacement parts catalog. Instead, repair instructions are stored and accessible via the workshop information system, etc. Each of these systems is modeled through a SOURCE concept.

Example 3: Understanding the concept SOURCE by example of a car workshop process.

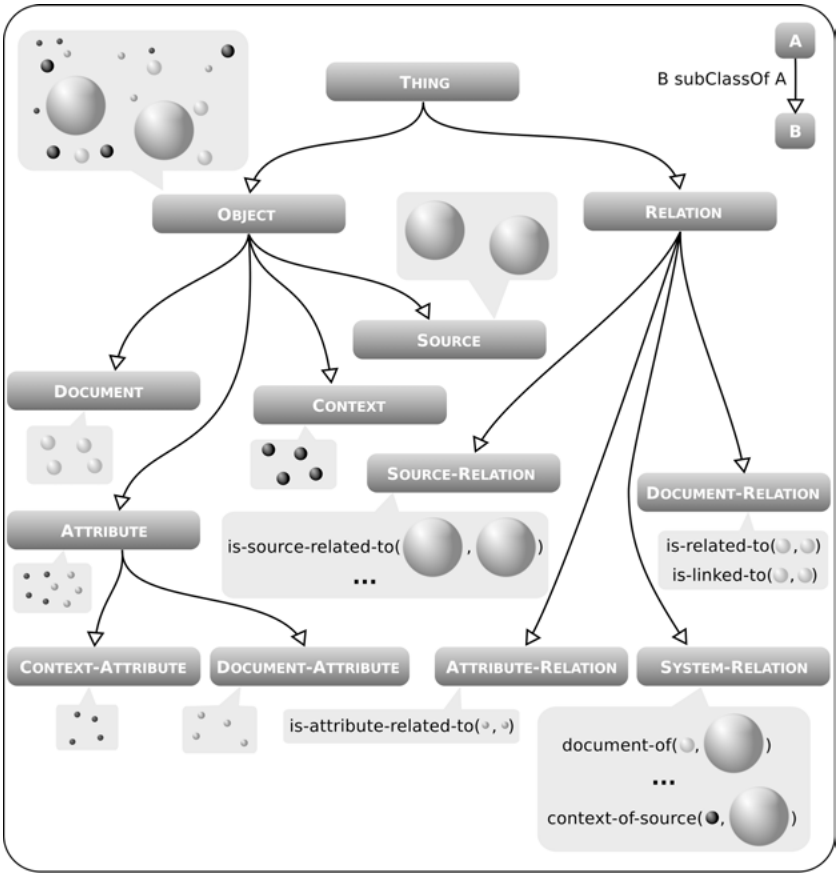


Figure 3.1: AIRSKB class hierarchy and visual representation of typical elements.

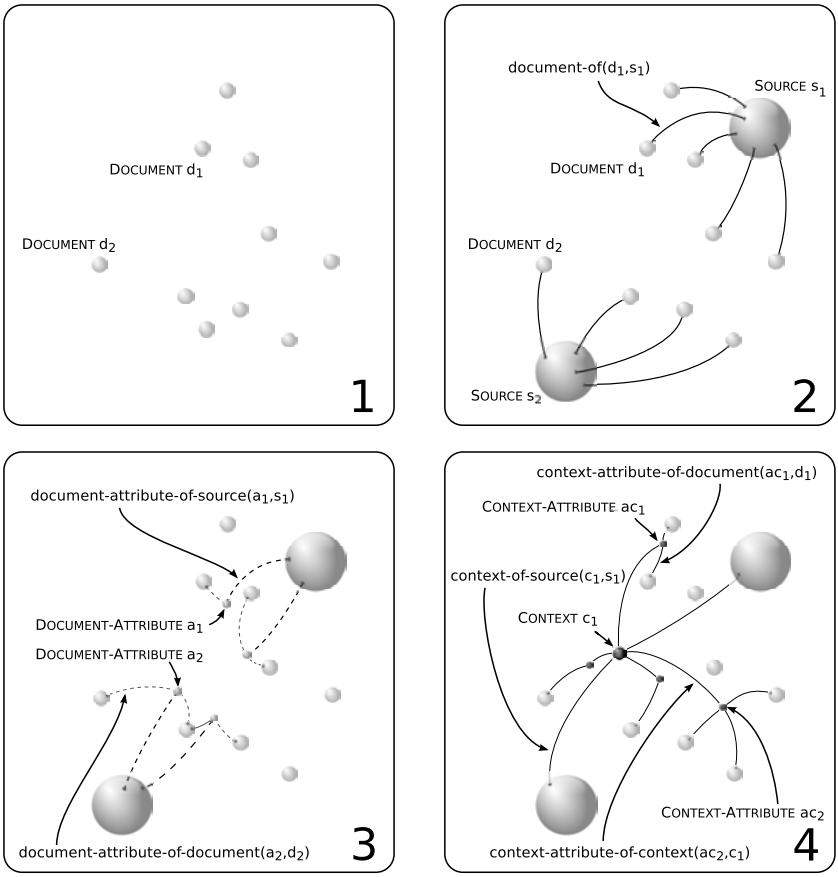


Figure 3.2: DOCUMENTS (Box 1) related to "their" SOURCES (Box 2). Box 3 shows DOCUMENT-ATTRIBUTES, DOCUMENTS and SOURCES. All DOCUMENT-ATTRIBUTES that a DOCUMENT can have are linked to the SOURCE. Only the DOCUMENT-ATTRIBUTES that a document has are linked to a DOCUMENT. Box 4 shows CONTEXTS, CONTEXT-ATTRIBUTES, DOCUMENTS and SOURCES. CONTEXTS are linked to SOURCES. CONTEXT-ATTRIBUTE (values of the contexts) are linked to the DOCUMENTS.

Example 2 describes the idea of the concept DOCUMENT in detail by means of a workshop use case. In other words, a DOCUMENT concept represents a single element of an information set that is used somehow in processes. Figure 3.2, Box 1 visualizes the idea of DOCUMENT concepts as small balls. A SOURCE concept stands for the document location (an isolated retrieval system, for example), where similar DOCUMENT entities come from. Example 3 describes the concept SOURCE.

Figure 3.1 shows the class hierarchy of AIRSKB and depicts individuals that belong to the classes in the speech bubbles. In the Figure, the two main classes OBJECT and RELATION describe the two distinct entity sets that are part of AIRSKB, namely the objects and semantic relationships between them. OBJECT is the super class of the classes DOCUMENT, SOURCE, ATTRIBUTE and CONTEXT. Entities that belong to the class DOCUMENT represent single information carrier that are used in processes (for example, PDF documents containing repair instructions).

The SOURCE class contains all of the representations of document locations. For example, isolated retrieval systems that contain documents are represented through SOURCE entities. CONTEXT is the class that represents a validity condition imposed upon the document of a document location. For instance: one entity or individual of the class CONTEXT can be the representation of product types. This means that documents of an isolated retrieval system can be valid for special product types only. ATTRIBUTE is the super class of all attributes. DOCUMENT-ATTRIBUTE is more specific than ATTRIBUTE because it represents a segmentation a document has as. By contrast, CONTEXT-ATTRIBUTE describes a special validity condition that belongs to a document. The class RELATION is the super class of all relations that are part of the AIRSKB ontology. DOCUMENT-RELATION represents all relationships that can appear between DOCUMENT individuals: is-related-to and is-linked-to relationships. Additionally, the class SOURCE-RELATION represents all relationships between SOURCES: is-source-related-to and is-source-linked-to. Furthermore, the ATTRIBUTE-RELATION class represents all of the is-attribute-related-to relationships between DOCUMENT-ATTRIBUTES of AIRSKB. By contrast, the SYSTEM-RELATION class represents all of the relationships that build the structure of the document network. Examples are the document-of relationships that link the DOCUMENTS to "their" SOURCES. Table 3.1 provides an overview of all relationships of AIRSKB. As stated above and illustrated in Figure 3.2 (Box 2), DOCUMENTS are related to the SOURCE to which they belong via a document-of relationship. This type of relationship depicts occurrence of documents in document locations. Both the concept name and the concept id are properties of the symptom node.

Replacement parts are described through a set of attributes; for example, a replacement part has a name, a description, a size and a price. All these attributes ("name", "description", "size" and "price") are modeled as DOCUMENT-ATTRIBUTES. Another kind of DOCUMENT-ATTRIBUTE interpretations are segments of a technical information document that contain different information. For example, these documents have different text segments for "complaint", "remedy" and "symptoms". Each of these segments is represented via a DOCUMENT-ATTRIBUTE in AIRSKB.

Example 4: Understanding the concept DOCUMENT-ATTRIBUTE by example.

The concept of a Document-Attribute represents an attribute or a segment of a document. Because Documents of the same Source comprise similar attributes or segments, Document-Attributes also belong to Sources: if a Document-Attribute belongs to a Source, the Document of this Source comprises the segmentation or attribution that is represented through the Document-Attribute. Since an attribute or segment of the "real"-world document is a key-value pair, a segment can be empty or the attributes have a null value. Accordingly, only real-world document segments or attributes with no null values are relevant for AIRSKB. Document-Attributes only provide information about not-null-value segments or attributes of a document. In this sense, Document-Attributes belong to Sources via a document-attribute-of-source relation and through a document-attribute-of-document relationship with Documents. Figure 3.2, Box 3 shows Document-Attributes and their relationships. Example 4 describes the concept of Document-Attribute.

Contexts and Context-Attributes

Validity of documents can be limited by several conditions. These conditions are constraints for the usage of documents in a given use case. Example 5 describes in detail what such a document condition is. Following this, the concept Context is a representation for a group of similar validity conditions and a condition itself is represented through use of the concept Context-Attribute. The relationship context-attribute-of-context binds a condition (Context-Attribute) to the group to which it belongs (Context). As shown in Figure 3.2 (Box 4), Contexts can be linked to Source concepts and Context-Attributes can be linked to Document concepts, if the Context-Attribute's Context is linked to the Document's Source. This means that Contexts provide information about possible Document validity conditions (Context-Attributes) for all Sources to which the Context is linked. The link between Contexts and Sources is described by via the context-of-source relation and the link between Context-Attributes and Documents is modeled with the context-attribute-of-document relationship.

By looking at Example 2, various documents are used in the car repair. However, these documents only apply to the customer's vehicle. For another vehicle with a similar damage, the workshop employee needs other kinds of documents. As stated in Example 1, replacement parts fit only one car type or a limited set of similar car types. These document constraints are validity conditions. Following this, "car type" is a generic term for special kinds of validity conditions (class of particular validity conditions) and special kinds of car models are particular validity conditions. In AIRSKB, special classes of validity conditions are modeled as Context concepts and particular validity conditions are represented through Context-Attribute concepts.

Example 5: Understanding the concepts Context and Context-Attributes by example.

System-Relations

These kinds of relationships represent the structure of the heterogeneous document landscape. They relate Document individuals to "their" Source individuals or Context-Attribute individuals to "their" Context individuals. Examples are document-of, context-attribute-of-context or document-attribute-of-document. For example, Box 2 of Figure 3.3 shows a visualization of a document-of relationships between Document $d1$ and Sources $s1$. Boxes 3 and 4 show different kinds of System-Relations: document-attribute-of-source, document-attribute-of-document, context-attribute-of-source, context-attribute-of-document and context-attribute-of-context.

Document-Relations, Source-Relations and Attribute-Relations

Example 2 describes how documents of various sources can be part of the same car repair case. In general, all these documents are (somehow) related to each other. More specifically, Example 2 depicts a relationship between replacement parts and a repair instruction for the parts installation. This seems to be a case-independent relationship. Moreover, between symptom nodes of a symptom taxonomy exist as a hyponymy relationship. These kinds of relationships are modeled in AIRSKB within is-related-to for adaptive relations and is-linked-to for constant relationships (assuming that nodes of a symptom taxonomy are modeled as documents). The following restrictions apply to Document-Relations represented in AIRSKB:

1. Document-Relations are homogeneous relations across the set of all Documents.
2. Document-Relations are symmetric. Relations between Documents of AIRSKB are non-directional because AIRSKB is a simplified model of document relationships. The relations represent only the information that documents are related to each other. This information is not directed.
3. Document-Relations are strict (irreflexive). A relationship of relevance between the same document constitutes a tautology. Representing this information in the ontology as a Document-Relation between Documents causes cycles in paths and thereby infinite loops when calculating pathways.
4. Only one Document-Relation between the same two Documents can exist (see Section 3.3). The reason is because AIRSKB only represents the information that documents are related to each other.

Additionally, Section 3.3 explains AIRSKB relationships more in detail. Figure 3.3 shows a visualization of Documents, Sources and relationships between them. Box 1 shows a group of related Documents and Figure 3.3, Box 3 demonstrates what a hyponymy relationship between Documents of a Source can look alike. Relationships between Sources cannot exist without at least one relationship between two Documents of both Sources; see Figure 3.3, Box 2. Source relationships can be adaptive (is-source-related-to) or constant (is-source-linked-to). In turn, this makes a Source relationship an indicator for Document relationships. Additionally, the same restrictions for Document relations also apply to Source relationships.

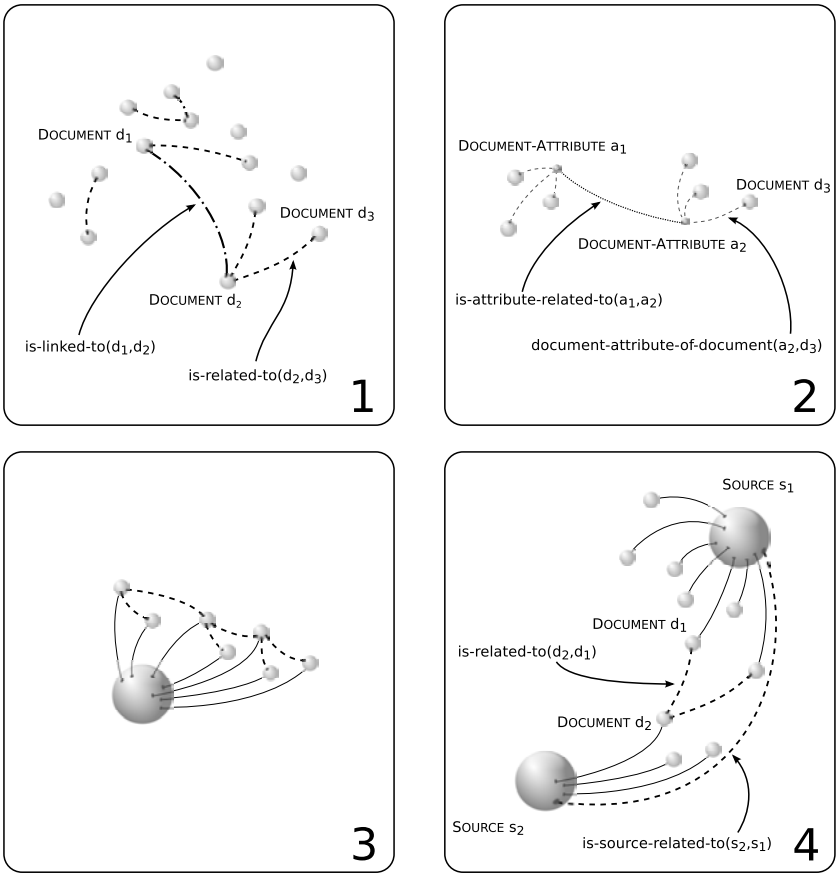


Figure 3.3: DOCUMENTS can be related to each other via is-related-to or is-linked-to relation (Box 1). DOCUMENT-ATTRIBUTES can be related to other DOCUMENT-ATTRIBUTES (Box 2). A hyponymy relation between DOCUMENTS of the same SOURCE can be modeled using AIRSKB, as shown in Box 3. SOURCES are related to each other if at least one relationship between two DOCUMENTS of both SOURCES exists (Box 4).

Document-Attributes can be related to other Document-Attributes within the adaptive is-attribute-related-to relationships that describe *how* Documents can be related to each other (via an attribute or a segment). Figure 3.3, Box 3 displays such a relation. Relationships between attributes or segments of different documents are represented through the Attribute-Relations is-attribute-related-to. These relationships can exist between two Document-Attribute individuals.

Pathways Through the Document Network

AIRSKB's pathways follow the path definition of graph theory. This means that AIRSKB can be interpreted as a kind of undirected graph with Documents as nodes and Document-Relations as edges. The graph is undirected because Document-Relations are symmetric. A pathway is subsequently a trail of distinct Documents (nodes in the graph theory), where any two consecutive Documents are part of a Document-Relation (edges in the graph theory). Section 3.3 explains AIRSKB pathways more in detail for all elements of AIRSKB. In general, Document pathways represented in AIRSKB comprise a set of n Documents and a set of $n-1$ Document-Relations between the n Documents:

1. One Document (as start node) is a defined as a start point of the pathway and another one is the defined end point (as end node). Start Documents and end Documents are different. This means that Document pathways are cycle-free.

2. Edges between Documents are represented by Document-Relations. The start Document as well the end Document are both part of only one Document-Relation of the Document-Relations set. The start point Document as well as the end point Document are part of the same Document-Relation if both are the only ones in the Document set (a path of only two Documents). All other Documents of the Document set are part of exact two Document-Relations of the Document-Relations set.

As mentioned in Example 2, a workshop employee needs to select some case describing symptom nodes to open a new service case in the workshop's management system. While selecting the symptoms with the aid of AIRSKB, useful replacement part assumptions can be made. This is because an information need can be "carried" through a heterogeneous document landscape. Now let the information need be the search for the symptom nodes in the symptom taxonomy. Furthermore, let the symptom nodes found be related to some technical information documents of the technical information and problem-solving system. If at least these technical information documents are related to some replacement parts of the electronic parts catalog, the employee's information need can be carried from one source to another and a replacement parts assumption can be made, even during the service case opening.

Example 6: Understanding hidden knowledge and pathways by example.

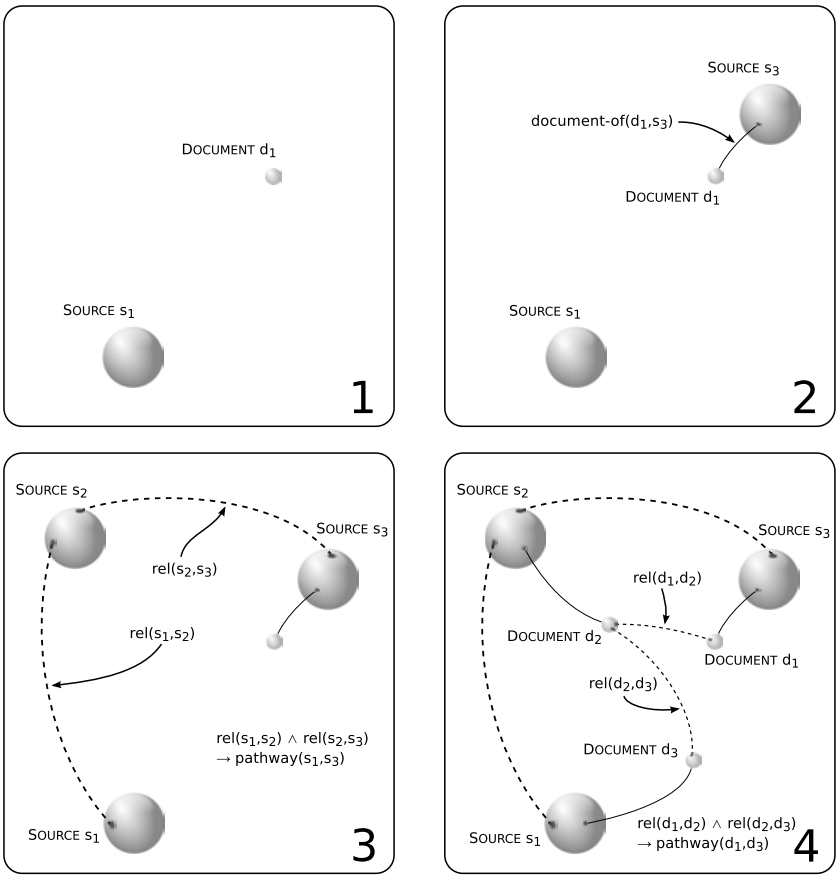


Figure 3.4: AIRSKB pathways: how to find related DOCUMENTS of a target SOURCE for a given DOCUMENT. Task: find all DOCUMENTS of the given SOURCE s_1 that are related to the given DOCUMENT d_1 (Box 1). First step (Box 2): get the SOURCE of the given DOCUMENT d_1, using the document-of relationship: SOURCE s_3. Second step (Box 3): find the best SOURCE pathway from SOURCE s_3 to target SOURCE s_1 (pathway over SOURCE s_2). Third step (Box 4): follow the best SOURCE pathway at DOCUMENT level (if exists) to the target DOCUMENTS.

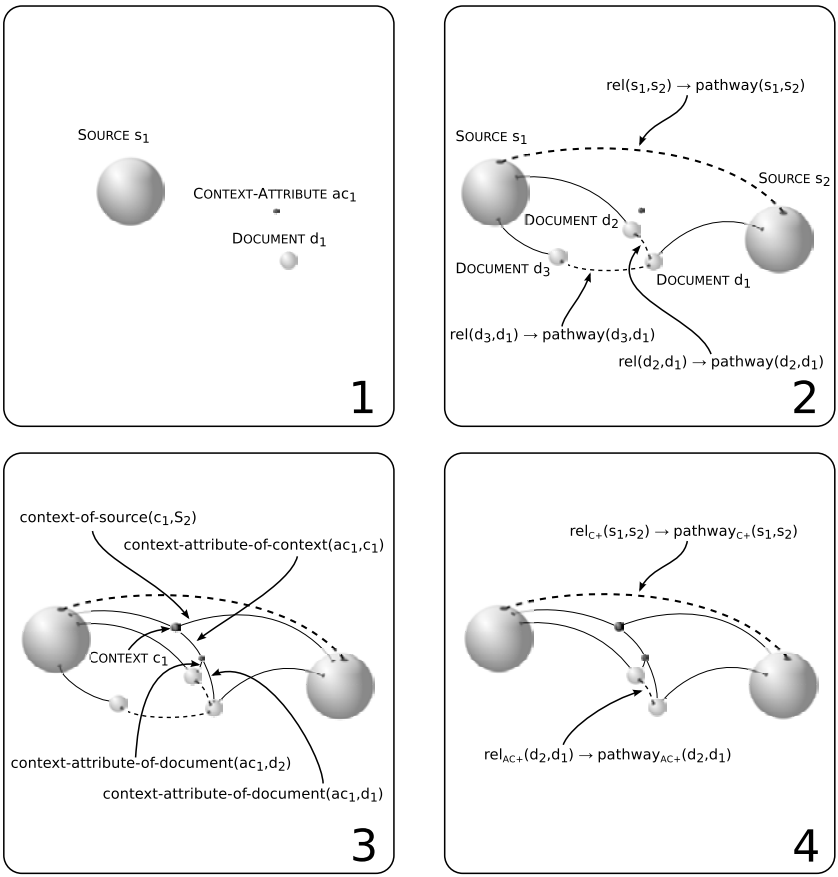


Figure 3.5: Task: find all DOCUMENTS of the given SOURCE s_1 that are related to the given DOCUMENT d_1 regarding a given CONTEXT-ATTRIBUTE ac_1 (Box 1). First step (Box 2): find related DOCUMENTS of the given DOCUMENT d_1 (see Figure 3.4). Second step (Box 3): Validate the source pathways found whether they match CONTEXT c_1 and the resulting document pathways whether they match the given CONTEXT-ATTRIBUTE ac_1. Result (Box 4): Only pathways are valid, where the DOCUMENTS fit the given CONTEXT-ATTRIBUTE ac_1.

A relation between two Documents – whether it is an is-related-to or an is-linked-to relation – is the shortest possible pathway. In general, pathways describe ways to *swing* from one Document to another, using relations as lianas. The same pathway definition and pathway conditions for Document pathways also appear for Source pathways. The only difference is that possible relations for Document pathways are is-related-to and is-linked-to relations and is-source-related-to as well as is-source-linked-to for Source pathways. For AIRSKB, finding new – previously unknown – knowledge means detecting relationships (or pathways) between Documents that are apparently hidden on first look. Following this, one possible use case can be the question for related Documents of a given Source regarding an also given Document (of another Source). Example 6 shows such a use case in the workshop process. Establishing pathways in the ontology (at the Document or Source level) means defining a framework for the knowledge network. Inference Rules are used to describe the best pathways through the ontology to make predictions about the relevance of Document relationships.

Figure 3.4 describes in detail how related Documents of a Source can be found by following pathways: in the first step (Box 2), the Source of the given Document should be found via the document-of relationship. With the help of next step (Box 3), the Source pathways must be found from the Document's Source to the target Source. The next step (Box 4) is to *follow* Document relationships starting at the given Document to Documents from the target Source. Finding Source pathways to the target Source first significantly reduces the calculation effort. Finally, one can rate the Document pathways to find the best related Documents (see Section 3.3).

Figure 3.5 provides an insight into context-sensitives pathways. Therefore, only Documents appear in the pathway's Document set that belong to a given Context-Attribute or a set of Context-Attributes. Figure 3.5 depicts in (Box 3) where the Document pathways found are reduced to those that match the given Context-Attribute.

To sum it up, AIRSKB comprises various concepts that represent documents and source locations. These concepts are summarized in Table 3.2, where the concept name and the concept scope are described. Furthermore, all AIRSKB relations are summarized in Table 3.1, including with the relation's name and scope. These kinds of relationships were designed to fill AIRSKB up with life and regarding the application context, to support various use cases like the relevance learning capability or the definition of context-sensitive pathways.

Table 3.1: AIRSKB's relationships. System-Relations of AIRSKB represent the structure of a document landscape. Document-Relations and Source-Relations instead represent the linkage between the elements of the document landscape.

Relation Name	Type	Description
document-of	System-Relation	A relationship between a DOCUMENT and a SOURCE. One can say that a DOCUMENT is contained in a SOURCE.
document-attribute-of-source	System-Relation	A relation between a DOCUMENT-ATTRIBUTE and a SOURCE. Information about a segment or an attribute that a document of a source can have.
document-attribute-of-document	System-Relation	A relation between a DOCUMENT-ATTRIBUTE and a DOCUMENT. Information about a segment or an attribute that a given document really has.
context-of-source	System-Relation	Relationship between a CONTEXT and a SOURCE. The relationship gives information about possible validity conditions (CONTEXT-ATTRIBUTES) for documents *of the* source.
context-attribute-of-document	System-Relation	Relation between a CONTEXT-ATTRIBUTE and a DOCUMENT. In addition to the context-attribute-of-source relationship, this stands for a validity condition a given document *of the* source really has. The relation implies a context-attribute-of-context relationship between the CONTEXT-ATTRIBUTE and a CONTEXT. The same CONTEXT subsequently has a context-of-source relationship with the given DOCUMENT's SOURCE.
context-attribute-of-context	System-Relation	Relation between a CONTEXT-ATTRIBUTE and a CONTEXT
is-related-to	Document-Relation	Relationship between two DOCUMENTS whose intensity may change over the time.
is-linked-to	Document-Relation	Another type of a DOCUMENT relationship. The difference from is-related-to is that these kinds of relationships have constant weights.
is-source-related-to	Source-Relation	A relationship between two SOURCES whose intensity may also change over time.
is-source-linked-to	Source-Relation	A constant relationship between two SOURCES.
is-attribute-related-to	Attribute-Relation	A relationship between two DOCUMENT-ATTRIBUTES of two DOCUMENTS.

Table 3.2: AIRSKB's concepts.

Concept Name	Description
DOCUMENT	A representation of a single information element known as a document used in a case-triggered process. Such an information element comprises segments or attributes. Examples of information elements are replacement parts or repair instructions used in a car repair process. An attribute of a replacement part – for example – is the part number.
SOURCE	A model of a document location (retrieval systems, for example) containing the documents. SOURCES contain a "group" of similar documents.
DOCUMENT-ATTRIBUTE	A representation of a document's segment or attribute.
CONTEXT-ATTRIBUTE	Model for a validity condition of a document. A validity condition exists if an information element is valid for only one or at least a limited group of use-cases. Examples are replacement parts that are "valid" for a special car type only.
CONTEXT	A group of similar validity conditions represented through CONTEXT-ATTRIBUTES. For example, "*car type*" is the name a group that includes all possible car types.

3.3 Theory and Inference Rules

The objective of Inference Rules is to define mathematical calculations for pathways through use of the ontological model. The ontological model comprises concepts and relationships defined in the conceptualization of ontology development. Therefore, Inference Rules work on weighted relations between documents of AIRSKB ([96]). This means that a computable basis and a formal definition of AIRSKB's elements are necessary.

The following section starts with a mathematical definition of the concepts and relationships of AIRSKB as presented in the previous section through of the use of graph theory. After this, these definitions are extended to represent context-sensitive relationships between concepts of AIRSKB. The section concludes with the mathematical definition of best context-sensitives pathways between documents of sources. These definitions can be used later to calculate best context-sensitive pathways between a document and documents of a given source in a particular use case. A particular use case can be a path-finding function of a knowledge network search framework based upon AIRSKB. This search framework can include calculation algorithms for the best pathways and correlating documents. Furthermore, it could optimize the relation weights between DOCUMENTS of AIRSKB automatically[3]. Moreover, new relations between DOCUMENT individuals can be created for the retrieval process. AIRSKB relationships between DOCUMENTS and the DOCUMENTS' SOURCES focus on three different kinds of entity relationships:

[3]Collective intelligence approaches can be used to establish adaptive DOCUMENT relationships. An example is the use of automated feedback processing, which has a direct influence on the weight of DOCUMENT relationships (see Section 6).

1. *Static relationships* - Kinds of relationships that do not change over time and represent constant relationships between documents and sources defined by domain experts or by system-indicated links.

2. *Adaptive relationships* - Kinds of relationships that change over time and represent user-generated knowledge about relationships between documents and documents' sources. These relations are the core of the AIRSKB's knowledge network.

3. *System-generated relationships* - Kinds of relationships that represent system-generated links between AIRSKB elements that cannot change. These relationships include all System-Relations. Examples are the relationships "document-of", "document-attribute-of-document" and "context-of-source".

Documents, Sources, Contexts and Attributes

At the bottom, AIRSKB follows the Graph definition. This means that AIRSKB is an ordered pair of nodes and edges[4]. For AIRSKB, nodes are objects O and edges are relations R.

$$AIRSKB = (O, R) \tag{3.1}$$

O is a set that contains all entities of AIRSKB: Sources, Documents, Document-Attributes and Contexts:

$$O = S \cup D \cup A \cup C \tag{3.2}$$

The elements of O are defined as follows: let S be a set of n Source entities,

$$S = \{s_1, s_2, \ldots, s_n\} \tag{3.3}$$

D be a set of m Document entities,

$$D = \{d_1, d_2, \ldots, d_m\} \tag{3.4}$$

C be a set of l Context entities,

$$C = \{c_1, c_2, \ldots, c_l\} \tag{3.5}$$

A be a set of $j + k$ Attribute entities, A_D be a set of j Document-Attribute entities and A_C be a set of k Context-Attribute entities, with

$$A_D = \{a_1, a_2, \ldots, a_j\} \tag{3.6}$$

$$A_C = \{ac_1, ac_2, \ldots, ac_k\} \tag{3.7}$$

$$A = A_D \cup A_C \text{ and } A_D \cap A_C = \emptyset \tag{3.8}$$

The AIRSKB elements sets are pairwise disjoint:

[4] A graph is a pair of disjoint sets $G = (V, E)$, with $E \subseteq [V]^2$ (see [19]).

$$\forall M_i, M_j \in \{S, D, A, C\}, i \neq j : M_i \cap M_j = \emptyset \tag{3.9}$$

A relation is a pair of elements of O. Let R be a set of all relations represented through AIRSKB, with:

$$R \subseteq \{\{o_i, o_j\} | 1 \leq i, j \leq n, i \neq j, o_i, o_j \in O\} \tag{3.10}$$

R contains all kinds of relations of AIRSKB: Source-Relations, Document-Relations, Attribute-Relations and System-Relations.

$$R = R_{\text{Source-Relation}} \cup R_{\text{Document-Relation}} \cup R_{\text{Attribute-Relation}} \cup R_{\text{System-Relation}} \tag{3.11}$$

The relations of AIRSKB are used to model the structure of the ontology and represent relationships between Documents and Sources. Structure of the ontology is represented trough system-generated relationships $R_{\text{System-Relation}}$ like document-of or context-attribute-of-context. $R_{\text{Document-Relation}}$ and $R_{\text{Source-Relation}}$ represent static or adaptive relationships between Documents and Sources.

Document, Document-Attribute and Source Relations

Let document-of $\in R_{\text{System-Relation}}$ be a system-generated relationship between a Document d and a Source s.

$$\text{document-of}\,(d, s) \tag{3.12}$$

Every Document d belongs to exactly one Source s:

$$\forall d \in D\ \exists! s \in S : \text{document-of}\,(d, s) \tag{3.13}$$

As explained in previous section, Document-Attributes represent the attribution (or segmentation) of documents of a source. A Source belongs to a super-set of all Document-Attributes that a Document of this Source can have. By contrast, a Document is only related to the Document-Attributes that appear in the document. In this sense, let document-attribute-of-source $\in R_{\text{System-Relation}}$ be a system-generated relationship between a Document-Attribute a and a Source s.

$$\text{document-attribute-of-source}\,(a, s) \tag{3.14}$$

Accordingly, let document-attribute-of-document $\in R_{\text{System-Relation}}$ also be a system-generated relationship between a Document-Attribute a and a Document d.

$$\text{document-attribute-of-document}\,(a, d) \tag{3.15}$$

To summarize, a document-attribute-of-source relationship between a Document-Attribute a and a Source s is an indicator for a possible document-attribute-of-document relationship between the same Document-Attribute a and a Document d:

$$\text{document-attribute-of-source}\,(a, s) \rightarrow \exists d \in D : \text{document-of}\,(d, s) \wedge \text{document-attribute-of-document}\,(a, d) \tag{3.16}$$

Additionally, a Document-Attribute a has a document-attribute-of-source relationship with exactly one Source s. Documents that have a document-attribute-of-document relationship with the same Document-Attribute a must also belong to the same Source s:

$$\text{document-attribute-of-source}\,(a, s_k) \wedge \text{document-attribute-of-source}\,(a, s_l) \rightarrow s_k = s_l \tag{3.17}$$

$$\text{document-attribute-of-document}\,(a, d_k) \wedge \text{document-attribute-of-document}\,(a, d_l) \rightarrow \text{document-of}\,(d_k, s) \wedge \text{document-of}\,(d_l, s) \wedge d_k \neq d_l \vee d_k = d_l \tag{3.18}$$

Context and Context-Attribute Relations

As stated before in previous section, Context-Attributes are ontological elements representing specific validity conditions that a document has. Let context-attribute-of-context $\in R_{\text{System-Relation}}$ be a system-generated relationship between a Context-Attribute ca and a Context c:

$$\text{context-attribute-of-context}\,(ac, c) \tag{3.19}$$

Since more than one type of devices can exist, many Context-Attributes can belong to the same Context. However, each Context-Attribute ac belongs to exactly one Context c:

$$\forall ac \in A_C \; \exists! c \in C : \text{context-attribute-of-context}\,(ac, c) \tag{3.20}$$

Contexts belong to Sources and provide information about possible validity conditions that the documents of the source can have. Instead, a Context-Attribute is a representation of a specific validity condition that a document of this source already has. A Document is called context-sensitive regarding the Context-Attribute ac. Since a document can be valid for more than one validity condition, aDocument can be related to more than one Context-Attribute. Let Contexts belong to Sources through the system-generated relationship context-of-source $\in R_{\text{System-Relation}}$ and let context-attributes belong to Documents, including via a system-generated relationship called context-attribute-of-document $\in R_{\text{System-Relation}}$:

$$\text{context-of-source}\,(c, s) \tag{3.21}$$

$$\text{context-attribute-of-document}\,(ac, d) \tag{3.22}$$

If a Context-Attribute ac belongs to a Document d, the Context c that belongs to the Context-Attribute ac must also have a relationship with Source s. The Source s again must have a document-of relationship with the Document d:

$$\begin{aligned} \text{context-attribute-of-document}\,(ac, d) \;\rightarrow\; & \text{context-attribute-of-context}\,(ac, c) \\ & \wedge\, \text{context-of-source}\,(c, s) \;\wedge \\ & \text{document-of}\,(d, s) \end{aligned} \tag{3.23}$$

Document and Source Relationships

As stated above, the most important relationships for AIRSKB are adaptive relationships as well as static relationships between Documents and Sources. These relationships can help to find related documents across system. They can also be used to make hidden knowledge of document relationships visible in the context of document search. A formal definition for these kinds of relations is necessary to define a mathematical basis for related documents as well as the best pathways through the ontology.

There are two types of relationships that take part in $R_{\text{Document-Relation}}$: is-related-to and is-linked-to. Additionally, there are also two types of relationships that take part in $R_{\text{Source-Relation}}$: is-source-related-to and is-source-linked-to. The first two relationships are Document relationships, whereas by contrast the others are Source relationships.

Let $R_{\text{is-related-to}} \subseteq R_{\text{Document-Relation}}$ be the set of all is-related-to relationships between two Documents. Furthermore, let is-related-to $\in R_{\text{is-related-to}}$ be an adaptive relationship between two Documents d_p and d_q, with $d_p \neq d_q$:

$$\text{is-related-to}\,(d_p, d_q) \tag{3.24}$$

Let *hasWeight* be a function that maps is-related-to relations to value called *weight*:

$$\begin{gathered} \text{hasWeight} : R_{\text{is-related-to}} \rightarrow \{x | 0.1 \leq x \leq 1, x \in \mathbb{R}\}, \\ \text{is-related-to}\,(d_p, d_q) \mapsto \mathit{weight} \end{gathered} \tag{3.25}$$

Only is-related-to relationships of a significance threshold ≥ 0.1 are represented into AIRSKB. Let $R_{\text{is-linked-to}} \subseteq R_{\text{Document-Relation}}$ be the set of all is-linked-to relationships between two Documents. Furthermore, let is-linked-to $\in R_{\text{is-linked-to}}$ be a static relationship between two Documents d_p and d_q, with $d_p \neq d_q$:

$$\text{is-linked-to}\,(d_p, d_q) \tag{3.26}$$

Let *hasWeight* be a function that maps is-linked-to relations to a relation weight of 1:

$$\text{hasWeight} : R_{\text{is-linked-to}} \rightarrow \{1\},\ \text{is-linked-to}\,(d_p, d_q) \mapsto 1 \tag{3.27}$$

1 means that a confirmed and constant relation between Document d_p and Document d_q exists. $\text{HasWeight}\,(\text{is-linked-to}\,(d_p, d_q)) = 1$ is semantically similar to $\text{hasWeight}\,(\text{is-related-to}\,(d_p, d_q)) = 1$ but these kinds of relations will never change over time. Only one kind of relation can exist between two Documents at the same time: either is-related-to or is-linked-to:

$$\text{is-related-to}\,(d_p, d_q) \;\oplus\; \text{is-linked-to}\,(d_p, d_q) \tag{3.28}$$

Let $R_{\text{is-souce-related-to}} \subseteq R_{\text{Source-Relation}}$ be the set of all is-source-related-to relationships between two Sources. Furthermore, let is-source-related-to $\in R_{\text{is-souce-related-to}}$ be an adaptive relationship between two Sources s_p and s_q, with $s_p \neq s_q$:

$$\text{is-source-related-to}\,(s_p, s_q) \tag{3.29}$$

Let *hasWeight* be a function that maps is-source-related-to relations to a specific weight:

$$\begin{gathered}\text{hasWeight} : R_{\text{is-souce-related-to}} \rightarrow \{x | 0.1 \leq x \leq 1, x \in \mathbb{R}\}, \\ \text{is-source-related-to}\,(d_p, d_q) \mapsto weight\end{gathered} \tag{3.30}$$

Only is-source-related-to relationships of a significance threshold ≥ 0.1 are represented in AIRSKB. Additionally, let $R_{\text{is-source-linked-to}} \subseteq R_{\text{Source-Relation}}$ be the set of all is-source-linked-to relationships between two Sources. Furthermore, let is-source-linked-to $\in R_{\text{is-source-linked-to}}$ be a static relationship between two Sources s_p and s_q, with $s_p \neq s_q$:

$$\text{is-source-linked-to}\,(s_p, s_q) \tag{3.31}$$

Let *hasWeight* be a function that maps is-linked-to relations to a relation weight of 1:

$$\text{hasWeight} : R_{\text{is-source-linked-to}} \rightarrow \{1\},\ \text{is-source-linked-to}\,(s_p, s_q) \mapsto 1 \tag{3.32}$$

Sources s_p and s_q are related to each other via is-source-related-to relation if at least one is-related-to relationship between two Documents of both Sources exists, but no is-linked-to relationship between the Documents. For $s_p \neq s_q$:

$$\begin{gathered}\text{is-source-related-to}\,(s_p, s_q) \rightarrow \\ \exists d_i \in D : \text{document-of}\,(d_i, s_p) \ \wedge\ \exists d_j \in D : \text{document-of}\,(d_j, s_q) \\ \wedge\ \text{is-related-to}\,(d_i, d_j) \ \wedge \\ \nexists d_s \in D : \text{document-of}\,(d_s, s_p) \ \wedge\ \nexists d_t \in D : \text{document-of}\,(d_t, s_q) \\ \wedge\ \text{is-linked-to}\,(d_s, d_t)\end{gathered} \tag{3.33}$$

Sources s_p and s_q are related to each other via is-source-linked-to relation if at least one is-linked-to relationship between two Documents of both Sources exists, with $s_p \neq s_q$:

$$\begin{gathered}\text{is-source-linked-to}\,(s_p, s_q) \rightarrow \\ \exists d_p \in D : \text{document-of}\,(d_p, s_p) \ \wedge\ \exists d_q \in D : \text{document-of}\,(d_q, s_q) \\ \wedge\ \text{is-linked-to}\,(d_p, d_q)\end{gathered} \tag{3.34}$$

Let is-attribute-related-to $\subseteq R_{\text{Attribute-Relation}}$ be the set of all is-attribute-related-to relationships between two Document-Attributes. Furthermore, let is-attribute-related-to $\in R_{\text{Attribute-Relation}}$ be an adaptive relationship between two Document-Attributes a_p and a_q, with $a_p \neq a_q$:

$$\text{is-attribute-related-to}\,(a_p, a_q) \tag{3.35}$$

Masking Relations

As written before, a weight of 1 for an is-related-to relationship is semantically similar to a weight of 1 for an is-linked-to relation. Therefore, it is not necessary for the pathway calculation to know whether is-related-to or is-linked-to or a mixture of the two relations took part in the pathway. Masked relations are primarily used for the pathway calculation. They are designed to hide the information concerning whether the relation that took part in a pathway is an is-related-to or an is-linked-to relation. Relation rel also masks relationships between SOURCES: is-source-related-to or is-source-linked-to. Therefore, a masking relation takes all of the properties of the relation that it masks. Including the value of the *hasWeight* function. Accordingly, let $\mathrm{rel}\,(o_p, o_q)$ be a masking relationship between two DOCUMENTS o_p and o_q or two SOURCES o_p and o_q, with:

$$\mathrm{rel}\,(o_p, o_q) = \begin{cases} \text{is-related-to}\,(o_p, o_q)\,, & \text{rel masks is-related-to.} \\ \text{is-linked-to}\,(o_p, o_q)\,, & \text{rel masks is-linked-to.} \\ \text{is-source-related-to}\,(o_p, o_q)\,, & \text{rel masks is-source-related-to.} \\ \text{is-source-linked-to}\,(o_p, o_q)\,, & \text{rel masks is-source-linked-to.} \end{cases} \tag{3.36}$$

Let R_{mask} be the set of all masking relations, for every $\mathrm{rel}\,(o_p, o_q) \in R_{\text{mask}}$ a relation of R must exist, which is masked by $\mathrm{rel}\,(o_p, o_q)$:

$$\begin{gathered} |R_{\text{mask}}| = |R| \ \wedge\ \forall \mathrm{rel}\,(o_p, o_q) \in R_{\text{mask}}\ \exists! \text{is-related-to}\,(o_p, o_q) \in R_{\text{is-related-to}} \\ \oplus\ \exists! \text{is-linked-to}\,(o_p, o_q) \in R_{\text{is-linked-to}}\ \oplus\ \exists! \text{is-source-related-to}\,(o_p, o_q) \in R_{\text{is-source-reltated-to}} \\ \oplus\ \exists! \text{is-source-linked-to}\,(o_p, o_q) \in R_{\text{is-source-linked-to}} \end{gathered} \tag{3.37}$$

A weight of a masked relation is also calculated through the *hasWeight* function. R_{mask} is the set of all masking relations, whereby the *hasWeight* function maps rel relations to the results of the *hasWeight* function from the masked relations:

$$\begin{gathered} \text{hasWeight} : R_{mask} \to \{x | 0.1 \leq x \leq 1, x \in \mathbb{R}\}, \\ \mathrm{rel}\,(o_p, o_q) \mapsto \begin{cases} \text{hasWeight}\Big(\text{is-linked-to}\,(o_p, o_q)\Big), & \text{rel masks is-linked-to} \\ \text{hasWeight}\Big(\text{is-related-to}\,(o_p, o_q)\Big), & \text{rel masks is-related-to} \\ \text{hasWeight}\Big(\text{is-source-linked-to}\,(o_p, o_q)\Big), & \text{rel masks is-source-linked-to} \\ \text{hasWeight}\Big(\text{is-source-related-to}\,(o_p, o_q)\Big), & \text{rel masks is-source-related-to} \end{cases} \end{gathered} \tag{3.38}$$

Context-Sensitive Relations

A relation at DOCUMENT level is context-sensitive regarding a CONTEXT-ATTRIBUTE ac or a set of CONTEXT-ATTRIBUTES $A_{C+} \subseteq A_C$, if both DOCUMENT o_p and DOCUMENT o_q have a context-attribute-of-document relationship with a CONTEXT-ATTRIBUTE ac or all CONTEXT-ATTRIBUTES of A_{C+}:

$$\begin{aligned} \mathrm{rel}\,(o_p, o_q)_{A_{C+}} \ \rightarrow\ & \mathrm{rel}\,(o_p, o_q) \ \wedge \\ & \forall ac \in A_{C+} : \text{context-attribute-of-document}\,(ac, o_p) \ \wedge \\ & \forall ac \in A_{C+} : \text{context-attribute-of-document}\,(ac, o_q) \end{aligned} \tag{3.39}$$

Additionally, a relation at the SOURCE level is context-sensitive regarding a CONTEXT c or a set of CONTEXTS $C_+ \subseteq C$, if both SOURCE o_p and SOURCE o_q have a context-of-source relationship with a CONTEXT c or all CONTEXTS of C_+:

$$\begin{aligned} \text{rel}\,(o_p, o_q)_{C_+} \quad \rightarrow \quad & \text{rel}\,(o_p, o_q) \ \wedge \\ & \forall c \in C_+ : \text{context-of-source}\,(c, o_p) \ \wedge \\ & \forall c \in C_+ : \text{context-of-source}\,(c, o_q) \end{aligned} \tag{3.40}$$

Context-Sensitive Pathways

A pathway is a valid concatenation of DOCUMENTS or SOURCES individuals of AIRSKB. More specifically, a pathway is a set of relations, where one can "follow" DOCUMENT relationships or SOURCE relationships in a way whereby one DOCUMENT or SOURCE (o_1) is defined as the starting point of the pathway and another DOCUMENT or SOURCE (o_n) is the end point of the pathway. Both DOCUMENTS or SOURCES appear only once in the pathway (no cycles are allowed) and all other DOCUMENTS (or SOURCES) appear twice in the pathway. Ultimately, a pathway is a relation between two DOCUMENTS or two SOURCES (o_1 and o_n), with

$$\begin{aligned} \text{pathway}\,(o_1, o_n) \quad := \quad & \text{rel}\,(o_1, o_2) \circ \text{rel}\,(o_2, o_3) \\ & \circ \ldots \circ \text{rel}\,(o_{n-1}, o_n) \end{aligned} \tag{3.41}$$

Therefore, the path definition follows the graph theory: a path is a trail of distinct nodes. For AIRSKB, this means that a path is a sequence $w = (o_1, o_2, \ldots, o_n)$ of n nodes, with:

1. $O^+ \subseteq D \oplus O^+ \subseteq S$
2. $R^+ \subseteq R_{\text{SOURCE-RELATION}} \oplus R^+ \subseteq R_{\text{DOCUMENT-RELATION}}$
3. $O^+ \subseteq D \rightarrow R^+ \subseteq R_{\text{DOCUMENT-RELATION}}$ (in the case of a DOCUMENT pathway)
4. $O^+ \subseteq S \rightarrow R^+ \subseteq R_{\text{SOURCE-RELATION}}$ (in the case of a SOURCE pathway)
5. $\forall o_i \text{ of } w \ : o_i \in \{o_1, o_2, \ldots, o_n\} = O^+$
6. $\forall \{o_i, o_{i+1}\} \subseteq O^+ : \exists! \text{rel}\,(o_i, o_{i+1}) \in R^+, 1 \leq i \leq n-1$
7. $i \neq j \rightarrow o_i \neq o_j, 1 \leq i, j \leq n$

Like relations, pathways can also be context-sensitive. This means that all relations of the pathway belong to the same CONTEXT-ATTRIBUTE ca or set of CONTEXT-ATTRIBUTES A_{C+} and all relations of the pathway at SOURCE level belong to the same CONTEXT c or set of CONTEXTS C_+

$$\begin{aligned} \text{pathway}\,(o_1, o_n)_{A_{C+}} \quad := \quad & \text{rel}\,(o_1, o_2)_{A_{C+}} \circ \text{rel}\,(o_2, o_3)_{A_{C+}} \\ & \circ \ldots \circ \text{rel}\,(o_{n-1}, o_n)_{A_{C+}} \end{aligned} \tag{3.42}$$

$$\begin{aligned} \text{pathway}(o_1, o_n)_{C_+} \quad := \quad & \text{rel}\,(o_1, o_2)_{C_+} \circ \text{rel}\,(o_2, o_3)_{C_+} \\ & \circ \ldots \circ \text{rel}\,(o_{n-1}, o_n)_{C_+} \end{aligned} \tag{3.43}$$

Since pathways are cycle-free concatenations of relations, multiple possible pathways between DOCUMENT o_1 and DOCUMENT o_n can exist. The same applies to SOURCE pathways. Accordingly, let $P_{\text{pathway}(o_1,o_n)}$ be the set of all pathways from DOCUMENT o_1 to DOCUMENT o_n or SOURCE o_1 to SOURCE o_n:

$$\begin{aligned} P_{\text{pathway}(o_1,o_n)} \quad := \quad & \{\text{pathway}\,(o_1,o_n) \,|\, \forall\, \text{rel}\,(o_p,o_q) \in \text{pathway}\,(o_1,o_n) : \\ & \text{rel}\,(o_p,o_q) \in R_{mask} \text{ and } o_p,o_q \in D \text{ or } o_p,o_q \in S\} \end{aligned} \tag{3.44}$$

Weight of Pathways

Mapping a pathway to a weight is calculated by a function called *weight* with all of the relation's weights:

$$\begin{aligned} \text{weight}\Big(\text{pathway}\,(o_1,o_n)\Big) \quad := \quad & \text{hasWeight}\Big(\text{rel}\,(o_1,o_2)\Big) \\ & \circ\, \text{hasWeight}\Big(\text{rel}\,(o_2,o_3)\Big) \\ & \circ \ldots \circ\, \text{hasWeight}\Big(\text{rel}\,(o_{n-1},o_n)\Big) \end{aligned} \tag{3.45}$$

The AIRSKB Framework suggests four implementations of the *weight* function:

$$\text{weight}\Big(\text{pathway}\,(o_1,o_n)\Big)_{\text{baseline}} := \prod_{k=1}^{n-1} \text{hasWeight}\Big(\text{rel}\,(o_k,o_{k+1})\Big) \tag{3.46}$$

$$\text{weight}\Big(\text{pathway}\,(o_1,o_n)\Big)_{\text{arithmetic mean}} := \frac{\sum_{k=1}^{n-1} \text{hasWeight}\Big(\text{rel}\,(o_k,o_{k+1})\Big)}{n} \tag{3.47}$$

$$\text{weight}\Big(\text{pathway}\,(o_1,o_n)\Big)_{\text{geometric mean}} := \sqrt[n]{\prod_{k=1}^{n-1} \text{hasWeight}\Big(\text{rel}\,(o_k,o_{k+1})\Big)} \tag{3.48}$$

$$\text{weight}\Big(\text{pathway}\,(o_1,o_n)\Big)_{\text{harmonic mean}} := \frac{n}{\sum_{k=1}^{n-1} \frac{1}{\text{hasWeight}\Big(\text{rel}(o_k,o_{k+1})\Big)}} \tag{3.49}$$

Best Context-Sensitive Pathways

Best pathways are the basis for advanced retrieval algorithms in the case that DOCUMENT relationships bring semantic and previously-hidden knowledge of retrieval processes. Best pathways mean semantically valid DOCUMENT connections. In this case, the information need of the retrieval system's user can be carried from one DOCUMENT to another beyond (or additional to) the system's relevance calculation algorithm. In AIRSKB, the best pathways are those with the highest weight.

$$\begin{aligned} \text{pathway}\,(o_1,o_n)_{\text{best}} \quad := \quad & \max\Big(\Big\{x | x = \text{weight}\Big(\text{pathway}\,(d_1,d_n)\Big), \\ & \forall\, \text{pathway}\,(o_1,o_n) \in P_{\text{pathway}(o_1,o_n)}\Big\}\Big) \end{aligned} \tag{3.50}$$

Finding the best context-sensitive pathways is one task for applications using AIRSKB:

$$\begin{aligned} \text{pathway}\,(o_1, o_n)_{A_{C+}\,\text{best}} \;:=\; & \max\Big(\Big\{x | x = \text{weight}\Big(\text{pathway}\,(o_1, o_n)\Big), \\ & \forall\,\text{pathway}\,(o_1, o_n)_{A_{C+}} \in P_{\text{pathway}(o_1, o_n)}\Big\}\Big) \end{aligned} \tag{3.51}$$

$$\begin{aligned} \text{pathway}\,(o_1, o_n)_{C_+\,\text{best}} \;:=\; & \max\Big(\Big\{x | x = \text{weight}\Big(\text{pathway}\,(o_1, o_n)\Big), \\ & \forall\,\text{pathway}\,(o_1, o_n)_{C_+} \in P_{\text{pathway}(o_1, o_n)}\Big\}\Big) \end{aligned} \tag{3.52}$$

Often it is necessary to know which DOCUMENTS of SOURCE s are relevant for the current information need represented by DOCUMENT o_1:

$$\text{pathway}\,(o_1, s)_{\text{best}} := \{\text{pathway}\,(o_1, o_n)_{\text{best}}\,|\text{document-of}\,(o_n, s)\} \tag{3.53}$$

In AIRS, best context-sensitive pathways for document search are defined as follows:

$$\text{pathway}\,(o_1, s)_{A_{C+}\,\text{best}} := \{\text{pathway}\,(o_1, o_n)_{A_{C+}\,\text{best}}\,|\text{document-of}\,(o_n, s)\} \tag{3.54}$$

3.4 Summary of AIRSKB Development

In summary, necessary steps towards the ontology design of AIRSKB development have been provided in this chapter. These steps include:

1. *Initial activities.* Focusing on the questions that the Application Context defines. This helped to understand the domain and the need for an ontology (see Section 3.1). Elements of the ontology have been defined (see definition for documents, sources, relationships, conditional relationships, pathways and context-sensitive pathways of Section 3.2 and Section 3.3). This is necessary for a common understanding of the domain. Additionally, a case study was performed to examine how the documents of the workshop document systems match the findings of the application context description. Furthermore, it was checked whether the systems correlate in a semantic way. After these initial activities, the picture of the ontology that needs to be built was very clear. This enormously helped to perform the conceptualization because concepts and attributes and requirements of the ontology were well known in advance.

2. *Performing conceptualization.* The step comprises defining classes and relationships (see Section 3.2). The ontology can be used for retrieval across document landscapes. AIRSKB includes concepts for documents, sources, relationships between the document and attributes that can be used to describe validity conditions of documents. The class model of the AIRSKB was built and attributes of the individuals were described whereby the individuals match the previously defined classes.

3. *Defining inference rules.* The theoretical frame for the knowledge acquisition (see Section 3.3) means defining the mathematical background. Axioms and equations ensure ontological reasoning. For AIRS, this is presented in Section 5.1.

4. *Ontology representation.* Defining an ontology representation framework or language that can be used to share the ontology among applications and people. This also involves the technology concerning how the ontology can be used in applications.

It is quite difficult to find an optimal implementation for AIRSKB because knowledge representation through ontologies is a wide field of research ([96]). Various technologies for representing and querying ontologies exist, which are specialized for certain application areas. Ontology concepts differ in their formalism, expressivity and domain range. As stated in [96], examples of ontology concepts and languages are RDF/RDFS[5], OWL[6], Topic Maps (see [70]), even NoSQL databases (see [8] and [53]), among many others. Some examples of ontology frameworks are: Ontopia[7] for Topic Maps, Apache Jena[8] for OWL and also NoSQL databases like the graph database Neo4j[9]. Moreover, it is possible to build one's own ontology-triggered storage model and querying framework by using XML format taxonomy, standardized technologies like relational databases or even document-oriented databases like MongoDB[10], or Apache CouchDB[11]. The next chapter provides an insight into the concept of ontology-based information retrieval, which uses the AIRSKB ontology.

[5]See http://www.w3.org/standards/techs/rdf, last visited Sept. 18, 2016.
[6]See http://www.w3.org/standards/techs/owl, last visited Sept. 18, 2016.
[7]See http://www.ontopia.net, last visited Sept. 18, 2016.
[8]See http://jena.apache.org, last visited Sept. 18, 2016.
[9]See http://neo4j.org, last visited Sept. 18, 2016.
[10]See http://www.mongodb.org, last visited Sept. 18, 2016.
[11]See http://couchdb.apache.org, last visited Sept. 18, 2016.

4 Ontology-based Retrieval Across Heterogeneous Document Landscapes

In [95], a case study examining three workshop document systems of an after-sales domain of a car workshop is presented. These document systems were: the workshop help-system, the symptom taxonomy and a subset of an electronic parts catalog. The findings suggest that these systems correlate in a semantic way, whereby technical information and problem-solving system documents contain symptom descriptions and can contain replacement parts information. According to this, semantic relationships between the information elements were identified (see [95]):

- Symptom nodes are organized in a taxonomy tree. Accordingly, a taxonomic hyponymy exists between the taxonomy concepts from the superordinate term to the specific term.
- Technical information and problem-solving system documents contain sections that explicitly name replacement parts.
- Technical information and problem-solving system documents include labeled fields with symptom information. This information relates to symptom nodes of the symptom taxonomy.

Furthermore, an ontology was built based upon both real-world workshop systems and documents. Figure 4.1 displays a small section of the newly-built ontology. The dots represent DOCUMENT concepts and the lines between the dots represent is-related-to relations. All other elements of AIRSKB were not included in the picture (SOURCES, DOCUMENT-ATTRIBUTES, etc.). The goal was to expose the potential of semantic relationships between uniform interpretations of information elements (DOCUMENTS) without the limitations of any system borders. Figure 4.1, Area A shows the structure of the symptom taxonomy. The root node of the taxonomy is located somewhere in the middle of the figure. All other nodes have been automatically arranged whereby they can be displayed well. Figure 4.1, Area B shows the links between the technical problem-solving documents and Area C shows the links between the technical problem-solving documents and the replacement parts. Following this, one can also recognize a structure that comprises three different levels that cross system borders (nodes from Area A are connected to nodes from Area B that are again connected to nodes from Area C). This is even more evident in Figure 4.2: Area A shows the symptom taxonomy, Area B depicts the documents of the technical problem solving system and Area C again shows replacement parts. By following these levels, one can establish a semantic link between symptom nodes and replacement parts that is not given in the original landscape.

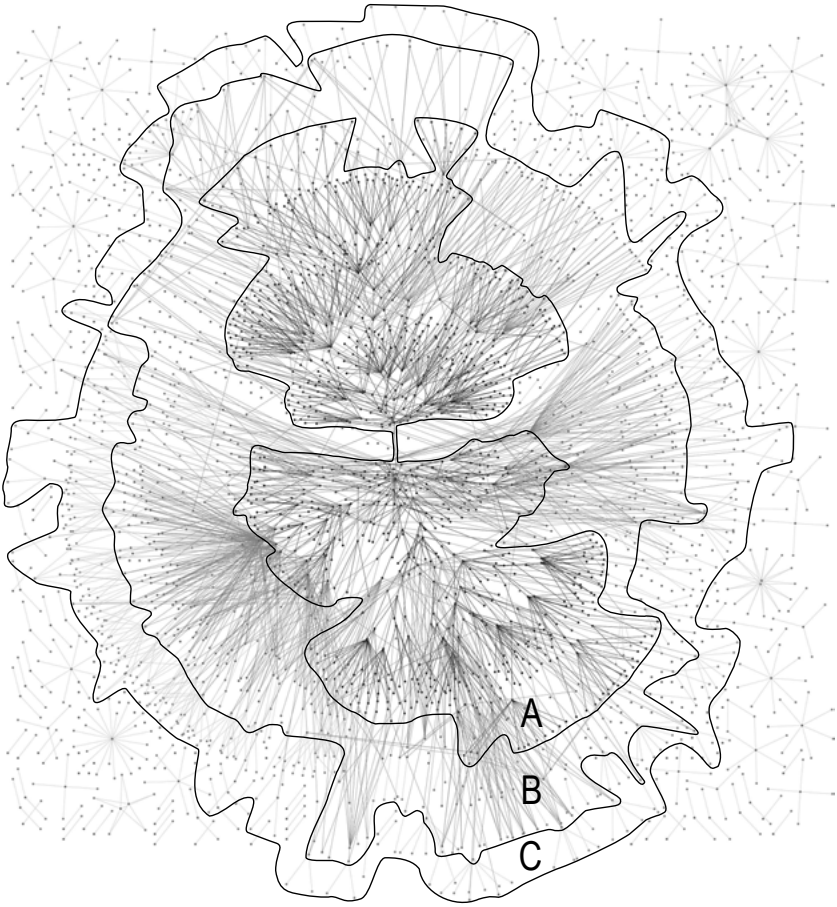


Figure 4.1: The after-sales symptom taxonomy extracted from the AIRSKB ontology. Figure adapted from [95].

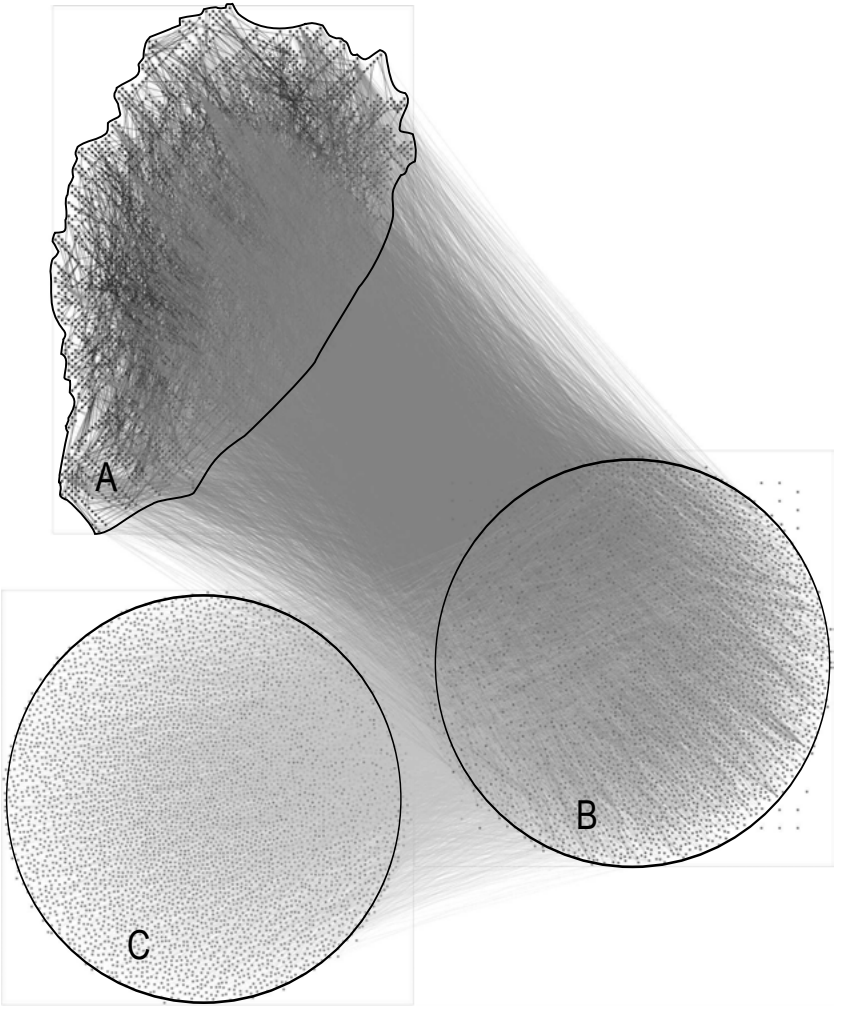


Figure 4.2: Is-related-to relationships between DOCUMENTS of three SOURCES extracted from AIRSKB. Figure adapted from [95].

AIRSKB is an ontology that was designed to represent different kinds of documents, sources and relationships between them. In domains of solving complex business tasks as described above, these documents and the sources where they appear form a heterogeneous landscape. Concepts of a heterogeneous document landscape must be identified and described to verify whether they can be represented through ontological elements of AIRSKB. For this purpose, a definition of the concept "Heterogeneous Document Landscape" itself is necessary:

A **Heterogeneous Document Landscape** is domain where documents that can appear in different source locations (databases, isolated retrieval systems, file systems and others) are used to solve particular problems.

Through use of this definition, the concepts, relationships and restrictions of the heterogeneous document can be compared with existing ontological elements of the AIRSKB. This section describes the concepts of a heterogeneous landscape documents through the use of sample documents. Furthermore, this section introduces advanced ontology-based information retrieval where the AIRSKB is used to implement advanced document search technologies. This advanced technology extends ordinary vector space retrieval with the document knowledge network provided by AIRSKB. Therefore, new search approach and retrieval strategies can be developed. A system architecture called Advanced Ontology-based Information Retrieval System (AIRS) was developed during the research and is introduced in this section.

4.1 Concepts of a Heterogeneous Document Landscape

Employees often need information stored in different locations to solve complex business tasks. Examples of these tasks include the service and repair of cars in workshops, Intranet searches, service requests in call centers or service tickets in product support (see Chapter 1). All information that is contained in different locations such as retrieval systems form a network, representing the knowledge that is part of a company. However, this network also includes hidden knowledge that exists in a potential semantical linkage. Example 7 explains the concept of hidden knowledge in further detail.

Let there be a service case in a car workshop: a customer brings a car to the workshop with a specific service request. The receptionist asks the customer for the service request's reason. The customer subsequently reports about unusual car behavior; for example, a rattling noise coming from the left rear wheel. This noise describes a symptom that appears while driving. One data source in the after-sales domain can be a standardized symptom taxonomy where symptoms of unusual car behavior are described. The workshop help documents are correlated to these symptoms. These workshop help documents can contain information about replacement parts that are necessary for a car repair. Altogether, simply by recognizing the customer's unusual car behavior and mapping it to the symptom taxonomy, possible replacement parts for a car repair can be found.

Example 7: Hidden knowledge in heterogeneous document landscape.

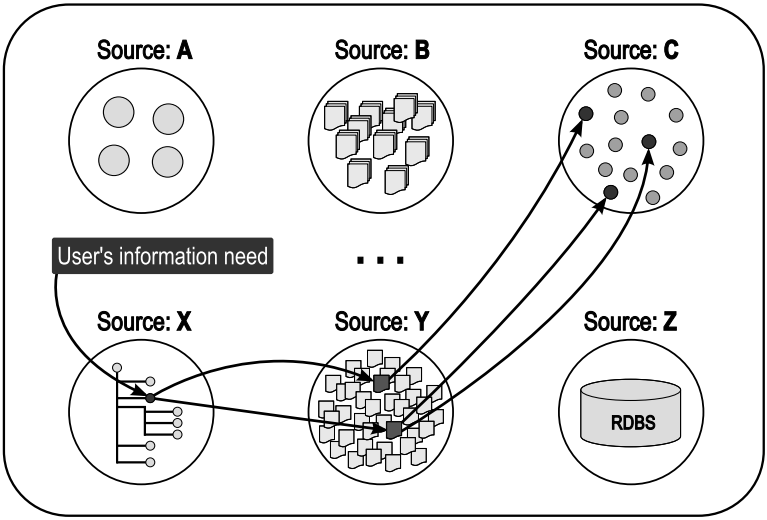


Figure 4.3: An employee needs to deal with various information fragments hidden in different retrieval systems (Source A-Z). These document retrieval systems act independently of each other, whereby the employee needs to carry the information search manually from one retrieval system to another. The figure was first shown in [93].

Figure 4.3 shows a part of the heterogeneous landscape, which comprises different information systems. It also shows how an information need of a user can be carried from one system to another.

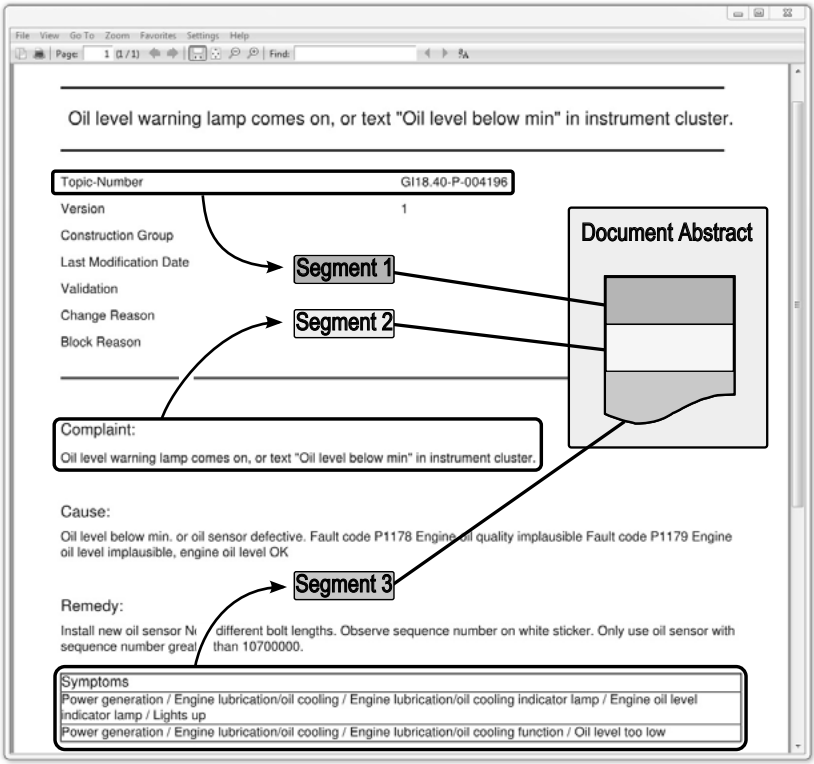


Figure 4.4: Example repair instruction document as shown in [96]. The original document is shown in the background, black covered areas ("Segment 1", "Segment 2", "Segment 3") mark the segments used to build the document abstract.

The information need must be carried from one system to another to find business case-relevant documentation. For this approach, the idea of what a document is in the context of search must be clear. An overall definition is necessary concerning what a document is, because a document can be many things: a node of a symptom-taxonomy, a service manual, a replacement part description or a complex diagnosis tree.

Werrmann considers in [96] a **document** as a single piece of information an employee uses for a business case, an object or piece of information that could have attributes and is connected to only one retrieval system.

Furthermore, all of the segments of a document that are relevant for the retrieval must be identified. This means identifying all elements of a document that users want to search for. Therefore, an abstracted version of a document is necessary that contains all of the relevant segments of a document. A segment itself is a defined part of a document that contains certain information.

Therefore, it is necessary to build an abstracted form of each document stored in the original retrieval systems. This is necessary due to the variability of documents in the different document locations.

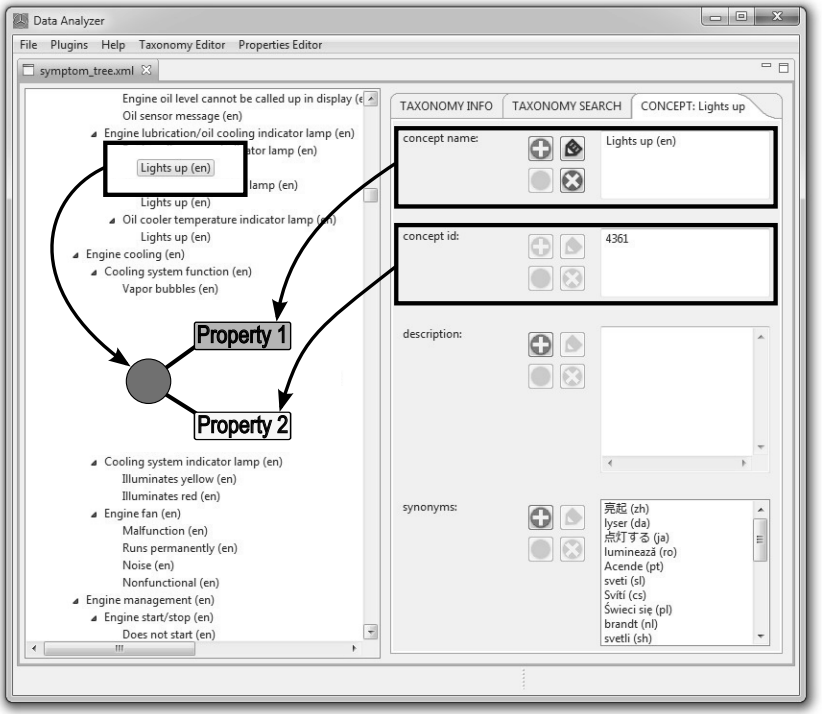


Figure 4.5: Example symptom tree node document. The original system to manage the symptom tree is shown in the background. Black covered areas mark the taxonomy node "Lights up" and the elements that are interpreted as properties ("concept name" and "concept id").

Figure 4.4 shows a PDF document that contains a complaint-based and symptom-based repair instruction for a special car model, which is used in workshop processes for the service. It demonstrates the idea of how to make abstracted versions of original documents. As stated above, the abstracted version must include all business case-relevant information

that is necessary for the search. Additionally, the document abstract should include all attributes that offer linkage potentials. Figure 4.5 depicts a node from the symptom taxonomy. The symptom taxonomy is used in reception processes of car workshops for the categorization of customer service requests. It shows kinds of information elements, namely the "segments" (black-bordered rectangles). The segments are now *properties* or *attributes*.

Documents are stored in different retrieval systems or other kinds of data storage systems. For example, Figure 4.3 shows different data storage systems (Source A-Z). The Storage systems serve as sources for different kinds of documents. Therefore, a source is also a concept of a heterogeneous document landscape that needs to be defined:

Source stands for a document location, a container of documents with similar attributes (databases, isolated retrieval systems, file systems, for example).

Figure 4.3 also shows that a link between documents can exist. These links can be hard-coded links as shown in Segment 3 of Figure 4.5, where a repair instruction document names a set of symptoms from a symptom taxonomy (shown in Figure 4.4)[1]. However, they can also be assumptions of relevance according a particular business case:

A **relationship** between documents or sources is a semantic linkage between these documents or sources. Computer accessible links between documents or sources indicate semantic relationships between them.

A relationship between two documents for a specific business case may be restricted. A restriction comes in the form of validity conditions for documents. For example, two documents are only related to each other if both belong to the same set of attributes. The same applies to sources. Such a restricted relationship between two documents or sources is a conditional relationship:

A **conditional relationship** is a semantic relationship between documents or sources that is restricted by a set of validity conditions.

Figure 4.3 shows a pathway from documents of "Source: X" to documents of "Source: Z". Pathways and context-sensitive pathways through the heterogeneous document landscape are defined as follows:

Pathway is a pairwise semantic linkage of documents or sources (relationships), where one document (or source) is the defined start point and another is the defined end point. This outlines a connected graph without loops where nodes are documents (or sources) and edges are relationships.

A **context-sensitive pathway** is a pathway where the semantic linkages between the documents or sources are conditional relationships.

All of these concepts of a heterogeneous document landscape can be represented through elements of AIRSKB. Table 4.1 shows a comparison of AIRSKB classes, individuals and relations with concepts of the heterogeneous document landscape.

[1]This implies that symptom taxonomy nodes are interpreted as documents.

Table 4.1: Correlation between AIRSKB elements and concepts of heterogeneous document landscape.

AIRSKB class, individual or relation	Concept of heterogeneous document landscape
DOCUMENT	document, document abstract
SOURCE	source
DOCUMENT-ATTRIBUTE	segment, property
CONTEXT-ATTRIBUTE	validity condition
CONTEXT	validity conditions type
$\mathrm{rel}\,(o_p, o_q)$	relationship
$\mathrm{rel}\,(o_p, o_q)_{A_{C+}}$	conditional relationship
$\mathrm{pathway}\,(o_1, o_n)$	pathway
$\mathrm{pathway}\,(o_1, o_n)_{A_{C+}}$	context-sensitive pathway

4.2 Advanced Ontology-based Information Retrieval System (AIRS)

Advanced ontology-based Information Retrieval System (AIRS) is an approach to extend search technologies with ontological knowledge to support the business case triggered retrieval in heterogeneous document landscapes. The special aspect is that this ontological knowledge is not used to optimize the text retrieval itself by trying to close the gap between syntax and semantic of understanding natural language[2]. Rather, the ontological knowledge is used to look at the text itself in the upper context: whole documents rather than terms or text fragments are interpreted as concepts that can be related to each other. The idea is that this approach enables the access to heterogeneous document landscapes and implements a network view that allows making statements about adaptive document relationships. These adaptive document relationships can be used to improve the document retrieval.

By its nature, AIRS describes a document retrieval system that includes the knowledge about a document landscape in the form of an ontology in the retrieval process. For this purpose, AIRS includes the information stored in different retrieval systems in a search index and combines the retrieval with the capability of an ontology network that models the heterogeneous document world. The AIRSKB ontology is used for the retrieval itself as well as the representation of domain-specific knowledge. This knowledge can also be updated over time through the use of AIRS. This means that AIRS deals with users, domains, feedback and document landscape knowledge. This new knowledge is subsequently re-used in AIRS' retrieval. For this approach, AIRSKB provides the possibility of including the system users' feedback in the retrieval process to optimize the search for case-relevant documents.

AIRS combines the search functionality of a state-of-the-art information retrieval system with AIRSKB through use of a component based architecture that includes a search index, databases and interfaces to various document landscapes. AIRS can be implemented as a

[2]As stated in Chapter 1, enterprise search technologies include features (using taxonomies or synonym sets) to improve the quality of natural language search. These features focus on the extension of text with conceptual knowledge about the underlying domain.

Rich Internet Application that uses a standard Internet browser and a search engine-like web page as a user interface.

Chapter 5 explains document indexing processes and search as well as feedback processing of AIRS in further detail. Furthermore, Chapter 7 presents an exemplary implementation of AIRS that can serve as blueprint for the development of systems. The next section introduces the basic system architecture of AIRS and its corresponding components.

4.3 Conceptual Architecture of AIRS

During the development of the AIRS architecture, many aspects had to be considered, including the gold standard of information representation and retrieval as well as the further development of the after-sales systems. The first version of AIRS' architecture was presented in [93] and [95]. In this early architecture version, the primary focus lay in ETL processes[3], the knowledge representation technologies and the retrieval components. The final version of AIRS' architecture was presented in [96] and showed that the main focus of the architecture shifted to retrieval approaches that include the feedback of system users in the retrieval process.

Figure 4.6 provides a brief overview of the AIRS backend architecture at the server side. The two main components of AIRS are the Indexing Component and the Core Component. The Indexing Component manages the Indexing Workflow that comprises all ETL processes that are necessary to load the data from each retrieval system into the Information Retrieval System (IRS) index of AIRS. In addition to the ETL process, part of the Indexing Workflow is a task responsible for AIRSKB, namely the ontology-populating task. After ontology engineering, the Indexing Component translates each document abstract into its ontology entity representation and stores it as an individual in AIRSKB. For this task, it is necessary to generate good document abstracts for every allocated document of each source of the heterogeneous document landscape.

The Core Component is responsible for the Retrieval and Feedback Workflow. For the retrieval, the Core Component uses all data stored in the IRS index and AIRSKB to generate retrieval strategies for a business process triggered-document search and manages the search task. AIRS also includes an automated evaluation, which is used for a bootstrapping adaption of the retrieval process.

As the Core Component is involved in the entire search task, it is part of the evaluation workflow. Moreover, it manages the communication with end users and foreign data interfaces. The Indexing Component and the Core Component use three specialized components in AIRS for the Indexing Workflow and an extended bootstrapping retrieval:

- The IRS Component brings the search functionality of a state-of-the-art information retrieval system into AIRS and it manages the Search Index.

- The Ontology Component is responsible for the Ontology Storage and is used to generate retrieval strategies and find document relationships.

[3]Extract-Transform-Load processes are responsible for data extraction from sources, data transportation to a special processing unit, data transformation and cleaning processes and data loading into a target system (see [88]).

- The Statistics Component is used for the automated feedback processing. It controls the Statistics DB.

The Core Component manages the interaction between the Ontology Component and IRS Component during the retrieval, as well as the interaction between Statistics Component and Ontology Component during the feedback processing.

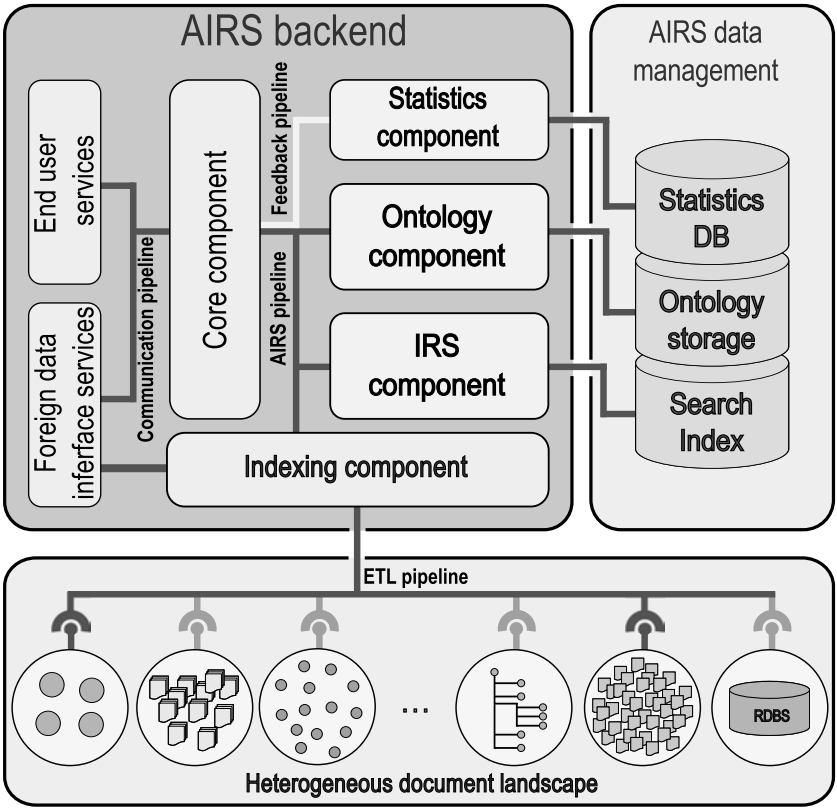


Figure 4.6: AIRS architecture as shown in [96].

5 Indexing and Retrieval for Advanced Ontology-based Information Retrieval

Natural language-based retrieval systems are the gold standard for modern information search. IRS enable a reliable and fast search across large sets of documents. These documents may contain structured, semi-structured or unstructured text data. To enable fast search for information, an index of the document's content must be built first. Accordingly, search technologies based upon information retrieval comprises two workflows that act independently to each other (both workflows contain highly language dependent processing steps, which must be implement for each language again):

- Indexing of documents is the final step necessary to build an index that is used later for the retrieval. Therefore, the content of the documents is analyzed and transformed to extract information used to build corresponding index documents. The indexing process is highly complex and contains different text processing algorithm. For example, elementary steps of indexing documents for a Boolean or vector space retrieval are (see [62]): collecting documents to be indexed, tokenize text of documents (list of terms), undertake linguistic pre-processing of tokenized text (stemming of terms, for example), create inverted index of terms that comprises a dictionary, posting lists (documents where the terms appear) and additional information for result ranking (term frequency and inverse document frequency). Furthermore, index terms can be annotated with additional information like a set of synonym terms. In case of vector space model[1], indexed documents are represented through n-dimensional vectors, where n is the number of index terms. Enterprise search technologies often contain complex workflow engines that support indexing documents of different types and languages.

- Natural language-based document retrieval uses the information of the previously-built index for the retrieval function. It contains two elementary functions: a retrieval model and a ranking function. The retrieval model is mostly based upon a combination of vector space retrieval in combination with Boolean retrieval and special search operators. Through use of this retrieval model, all of the documents are identified that match the criteria of the search query. In the case of vector space retrieval, queries as well as documents are represented through term vectors. The relevance of documents (according to a query) is calculated by measuring the similarity between the vectors. A ranking function try to rank the relevant documents based upon a retrieval model and a set of different criteria (term frequency and inverse document frequency or PageRank).

Different approaches try to increase the process of indexing by analyzing the contents of the documents; for example, through Latent Semantic Analysis approaches (see [17] or

[1] See [84] for more information about research in the field of vector space models.

[43]). However, an IRS index is a static collection of indexed data. All of the information that users want to search for must be indexed in advance. Additionally, if documents are updated, they must be indexed again and the index must be updated. The result is that search technology based on information retrieval is not designed for information that rapidly changes. A relationship of documents is a kind of rapidly-changing information. Therefore, AIRS includes an ontology (AIRSKB) in its search approach. AIRSKB represents the document relationships that can rapidly change and the index of AIRS contains information that is necessary for the search. The indexing process of AIRS differs from ordinary indexing approaches of search technologies in the way in which documents must be indexed in the AIRS index and corresponding ontological individuals of AIRSKB must be built.

Two main workflows can be determined at AIRS' server side: Indexing Workflow and Retrieval and Feedback Workflow. As usual in IRS, these two workflows act independently of each other. This section starts with an introduction to Indexing Workflow of AIRS, where the indexing of different document types is introduced. Furthermore, the Retrieval and Feedback Workflow is introduced in this section.

5.1 Indexing Workflow

In [96], Indexing Workflow is described as part of the system initialization. It is the kind of workflow, where all documents of the heterogeneous document landscape are loaded into the IRS index and where AIRSKB is populated with ontological individuals. Furthermore, its first implementation is performed between the system development and the system's regular operating time. Later on, it should be undertaken in a repetitive manner to update the index because documents can change over time. Additionally, the document authors can provide new documents and other documents can be marked as no longer being valid. The current challenge is to keep both the AIRS index and AIRSKB up-to-date. More specifically, it is important to synchronize the state of AIRS' index documents with the state of the corresponding AIRSKB individuals and the original documents stored in the document's source location (of the heterogeneous document landscape).

As stated in Section 4.2, AIRS works with document abstracts that have been built from the original documents. These Document Abstracts contain the necessary information for the retrieval. The document abstracts are subsequently transformed to AIRS index documents.

Index document fields of state-of-the-art retrieval systems[2]. The keys are the field names and the values are stored in the corresponding fields. An information retrieval index differs from relational database tables especially in the way in which information retrieval system indexes can be seen as a one-dimensional search index. A search index at least looks like just one table of a relational database. This causes some challenges in building an index for retrieval.

In [96], the challenge is stated as it could be very difficult to transform multi-dimensional data into a one-dimensional search index without losing any necessary information or data restrictions. In a few cases of multi-dimensional or otherwise complex data, it is even

[2]Apache Solr can be seen as an example as an information retrieval system that uses a configuration file for index description. In solrconfig.xml, fields and field values that are used for the retrieval are described (for details visit https://wiki.apache.org/solr/SolrConfigXml, last visited Sept. 18, 2016.).

impossible. Therefore, by looking at the underlying business case, only data with search capability and linkage potential should be extracted. For example, regarding a relational database as source, not every field of a database table contains business case-relevance information. It is not necessary to make a model that is a full semantic copy of the original document. In addition, the source information must be stored in the index document and documents of the same source need to be built in an equal way.

On the other hand, all of the segments of the original document that are necessary for the retrieval must be translated into Document Abstracts. The Document Abstracts are subsequently transformed into index document fields. The Index Workflow causes a process that requires decisions regarding which information of the original documents is necessary for the retrieval and which can be neglected. This decision can be made by domain experts or data scientists[3].

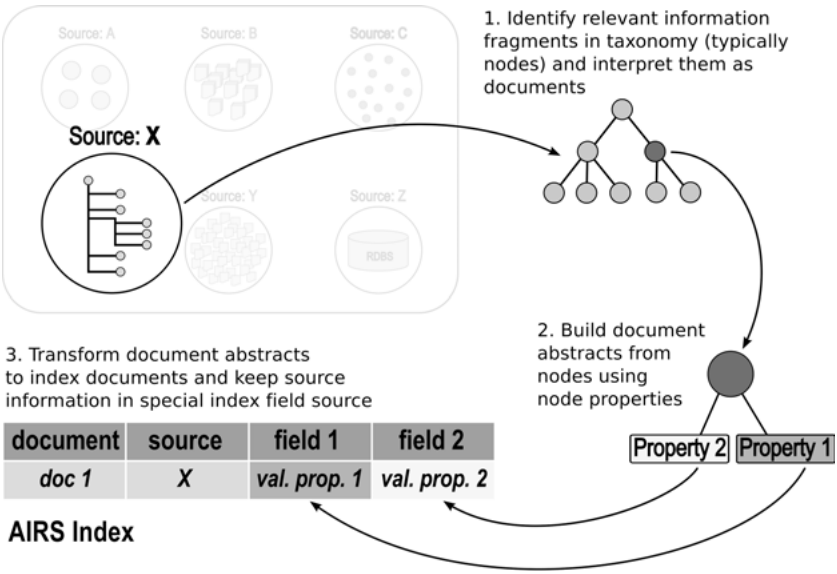


Figure 5.1: Keeping taxonomy node information in a search index using document abstracts.

Given that AIRS also includes an ontology in the retrieval process including all of the documents of the heterogeneous document landscape, AIRSKB must be populated with document individuals. Unlike AIRS index documents, AIRSKB Document individuals do not contain the value information of the document abstract key-value pairs. More specifically, it is not necessary because the values are simply used for the vector-space retrieval offered by the underlying search technology. By contrast, the keys are used to implement a link between AIRS index documents and AIRSKB Document individuals.

[3]See [13] for more information about the job profile data scientist.

For instance, in the case of a car workshop domain, repair instruction documents comprise segments that contain different information (see Figure 4.4). One such a segment is "complaint", which contains information about unusual behavior of a car, resulting in an error. Data scientists should decide whether a segment is necessary for the retrieval in terms of the underlying business case. Segment "complaint" contains such information because the workshop employee should be able to search for the complaint to find the corresponding remedy ("remedy" is also a segment of repair instruction documents). Therefore, the segment "complaint" must be later transformed in an index field. The AIRSKB instead represents the repair instruction document and its segments. Therefore, AIRSKB includes information about the ontological individual that has a segment called "complaint". This information is represented by a DOCUMENT-ATTRIBUTE that belongs to the "repair instruction" DOCUMENT (see Section 3.2).

To summarize, all document abstracts need to be placed in the AIRS index as index documents and as AIRSKB individuals in the AIRSKB ontology. Moreover, the key-value pairs must be transformed in fields and field values of the corresponding AIRS index documents. Additionally, the keys must be implemented as DOCUMENT-ATTRIBUTE individuals and correlated to the corresponding DOCUMENT individuals of AIRSKB. The following figures depict how this transformation can be carried out during the Indexing Workflow.

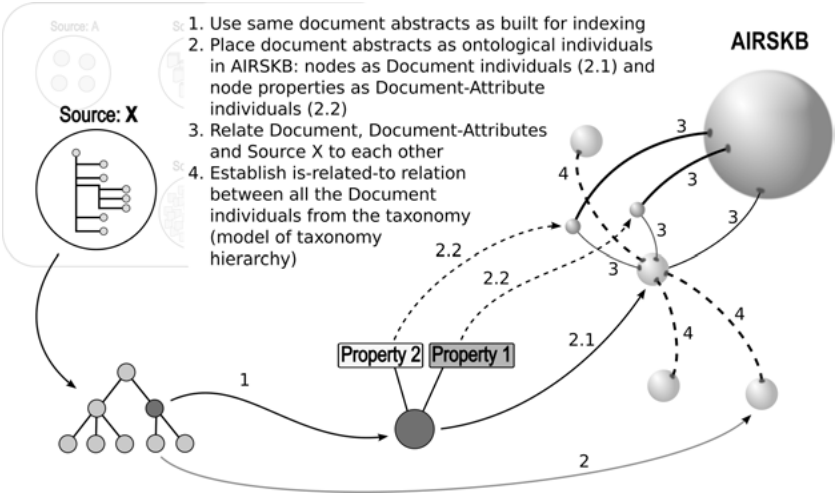


Figure 5.2: Transformation of taxonomy node information into corresponding AIRSKB individuals.

Figure 5.1 illustrates how a document abstract made from a taxonomy node can be loaded into an IRS index (Step 1). A taxonomy node typically is included in a taxonomic hierarchy and contains multiple attributes. One such example is the lexicographic taxonomy, which contains a hierarchical order of symptoms and symptom places of car defects. Of course, each symptom or symptom place has a name, an identification number and

synonym descriptions in multiple languages. All of these attributes are taxonomy node properties represented through key-value pairs. A Document Aabstract subsequently contains selected properties and values (Step 2). The Document Abstract will be placed in the AIRS index as shown in Figure 5.1: a source field is placed in the index that contains the taxonomy name as source information. Multiple index fields are implemented, which represent the properties and contain the properties information (Step 3). Later on, these fields can be used for the retrieval across all of the stored taxonomy nodes. Again, the taxonomic hierarchy is not placed in the AIRS index because all relationship information is represented through the AIRSKB ontology. As stated before, the Document Abstracts need to be modeled as DOCUMENT individuals in the AIRSKB ontology.

Figure 5.2 depicts how a document abstract (Step 1) can be transformed in an AIRSKB DOCUMENT individual (Step 2.1) and AIRSKB DOCUMENT-ATTRIBUTE individuals: properties are modeled as DOCUMENT-ATTRIBUTES (Step 2.2) that are linked to the DOCUMENT individual (Step 3). DOCUMENT individuals (stands for the taxonomy nodes) are related to each other like the original taxonomy nodes (Step 4). Each AIRS index document and each AIRSKB individual have a special attribute called AIRSKB_ID. The unique AIRSKB_ID number ensures the identification of each AIRS index document and AIRSKB individual: an AIRS index document and an AIRSKB individual that have the same AIRSKB_ID belong to the same original document. The taxonomic hierarchy is modeled as relationships between DOCUMENT individuals: after all taxonomy nodes are modeled as DOCUMENT individuals, relationships between all of the DOCUMENT individuals are added to the AIRSKB ontology (Step 4 of Figure 5.2).

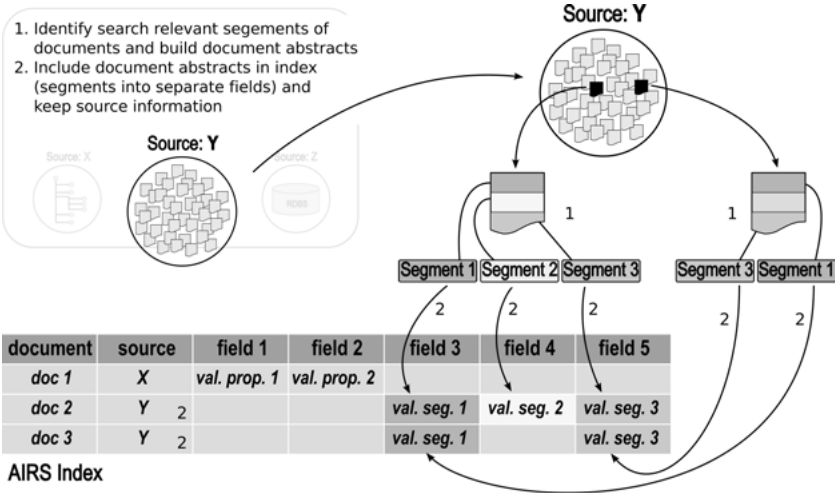


Figure 5.3: Keeping documents (PDF, HTML, WORD, et cetera) in a search index using document abstraction; figure first shown in [96].

Figure 5.3 depicts how documents that have a structure based upon segments can be placed in a search index. Examples of such documents include repair instructions (as explained in Section 4.2 and shown in Figure 4.4), replacement parts as well as other types of documents such as web pages. The important aspect is that similar documents must belong to the same source. A source again can be a retrieval system (like a replacement parts catalog) or a well-defined container of similar documents (for example, an internet domain). Documents of the same source comprise a super-set of segments. A segment of a document comprises a key-value pair (like the properties discussed in the taxonomy transformation example above). Figure 5.3 shows two documents of "Source Y". The left document comprises three segments ("Segment 1", "Segment 2" and "Segment 3"), while the right comprises two segments: "Segment 1" and "Segment 3". The document abstract indicates which of the segments are necessary for the retrieval and must be placed in the index (Step 1). In Figure 5.3, all of the segments of the left document have been chosen and placed in the index document "doc 2" (Step 2). The processed segment values are placed in corresponding fields ("field 3", "field 4" and "field 5"). During the retrieval, each field can be addressed separately to enable retrieval features like faceted search. Additionally, these fields can be used to search only in a special field for document values. For example, one can search for documents that belong to a given replacement part number in the index documents' field "replacement part number". The source information needs to be added to each of the index documents, as shown in the figure (see index field "source").

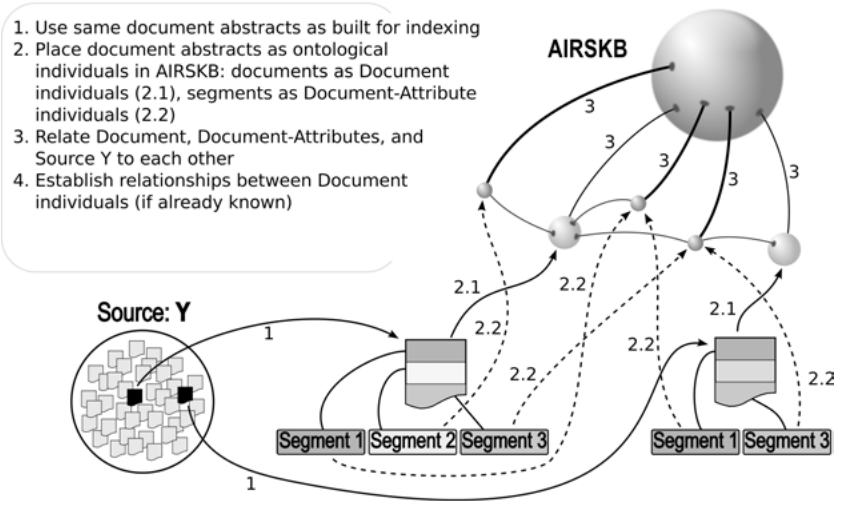


Figure 5.4: Transformation of documents (PDF, HTML, WORD, et cetera) into corresponding AIRSKB individuals.

Figure 5.4 shows how documents that comprise segments can be modeled as ontological elements and placed in AIRSKB as described before. The AIRSKB individuals are

built the same way as explained in the taxonomy example above: a DOCUMENT individual is placed in the AIRSKB ontology for each document abstract that is transformed in an AIRS index document (Steps 1 - 2.1). AIRS index fields are placed as DOCUMENT-ATTRIBUTES in AIRSKB (Step 2.2) and they are linked the to the DOCUMENT where they appear through use of a document-attribute-of-document relation. The DOCUMENTS and DOCUMENT-ATTRIBUTES subsequently need to be linked to the corresponding SOURCE of AIRSKB (Step 3) through use of the SYSTEM-RELATIONS document-of and document-attribute-of-source. In Step 4, relationships between the DOCUMENT individuals can be established (not shown in the figure). Documents that comprise segments are often based upon PDF, HTML, CSV or any other semi-structured format. These documents are difficult to transform in AIRS index documents and AIRSKB individuals because the structure is often unclear and segments are not easy to identify. As previously mentioned, this is a difficult task that needs the knowledge of domain experts or data scientists.

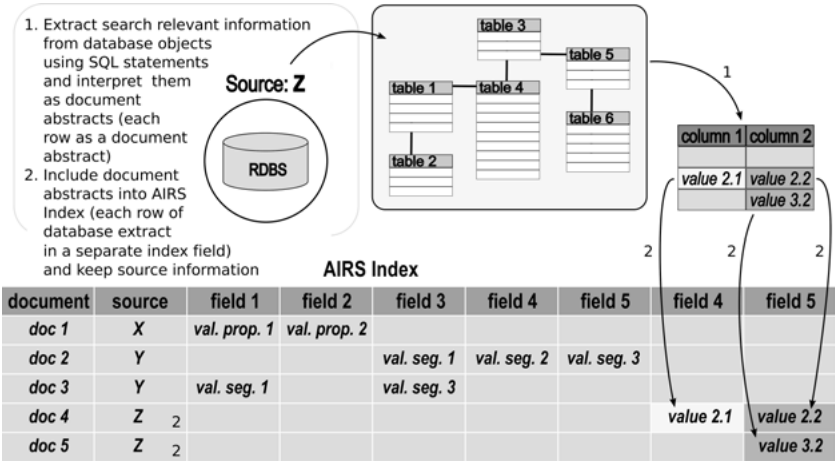


Figure 5.5: Keeping search relevant information of database objects in a search index using document abstracts.

Often information is stored in databases and a document is built dynamically after requesting it. Figure 5.5 shows how a database SQL query can be used to build documents that are dynamically stored in a database schema (Step 1). The database response forms the Document Abstract, which is later used to include the document information in the AIRS index. All of the rows of the response containing necessary information for the retrieval later in AIRS' retrieval workflow can be selected to build index fields (Step 2). In the case a value exists, they can be included as field values (Step 2).

Figure 5.6 shows the process to add AIRSKB individuals to the AIRSKB ontology from document abstracts built from a database. The same database query is used to identify documents in the database schema. As stated above, the result of a SQL query belongs to a document abstract (Step 1) and should be modeled as an AIRSKB DOCUMENT

individual (Step 2.1). Column names are the Document-Attributes that were added to the AIRSKB ontology (Step 2.2). Later they are linked to the corresponding Source individual (Step 3). In the case of values of the column in the corresponding row not being null, the Document-Attributes are linked to the Document where they appear (also Step 3). In Figure 5.6, two Documents are included in the AIRSKB ontology. One of them is linked to two Document-Attributes and the other is linked to only one of Document-Attributes (because the corresponding value of "column 1" is null). To summarize, a Document of the Source individual representing "Source: Z" can have two Document-Attributes representing "column 1" and "column 2".

Figure 5.7 shows how a relationship can be established between documents that are kept in the index and represented in the AIRSKB. As stated before, AIRSKB can be used for the task of establishing relationships between documents even across system borders. A link between the two documents "doc 1" and "doc 2" is not represented in the AIRS index (because they have to change dynamically). Therefore, a relation between to documents is represented through a Document relationship in AIRSKB (Step 1). This relationship can even be carried to index field level: a relationship between Document-Attributes in AIRSKB (Step 2). If two Documents are related to each other, this also applies to "their" Sources (Step 3).

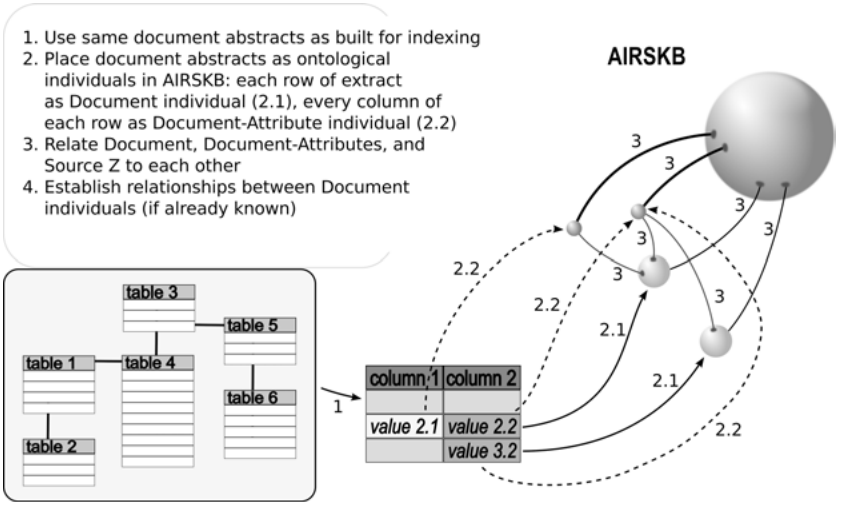


Figure 5.6: Transformation of search relevant database objects information into corresponding AIRSKB individuals.

For this work, only relationships are carried in the AIRSKB during the Indexing Workflow that already exist in the original retrieval systems. These links between documents were subsequently placed in the AIRSKB ontology as Document relationships. Nonetheless, these kinds of existing relations remain very rare in the original documents.

Section 6 describes a kind of relationship between DOCUMENTS that is established during the retrieval and bases automated feedback processing.

Nonetheless, how can new, previously-unknown relationships between documents be established during the Indexing Workflow? Relations can be found or approximated through an algorithmic calculation to measure the similarity of text fragments, for example. Another kind of relation between documents can be established via text mining algorithms to find corporate language entities. For example, by using an entity-recognition algorithm, replacement parts number can be found in the free text of repair instructions. Later on, the repair instruction document can be automatically linked to the replacement parts found in the free text.

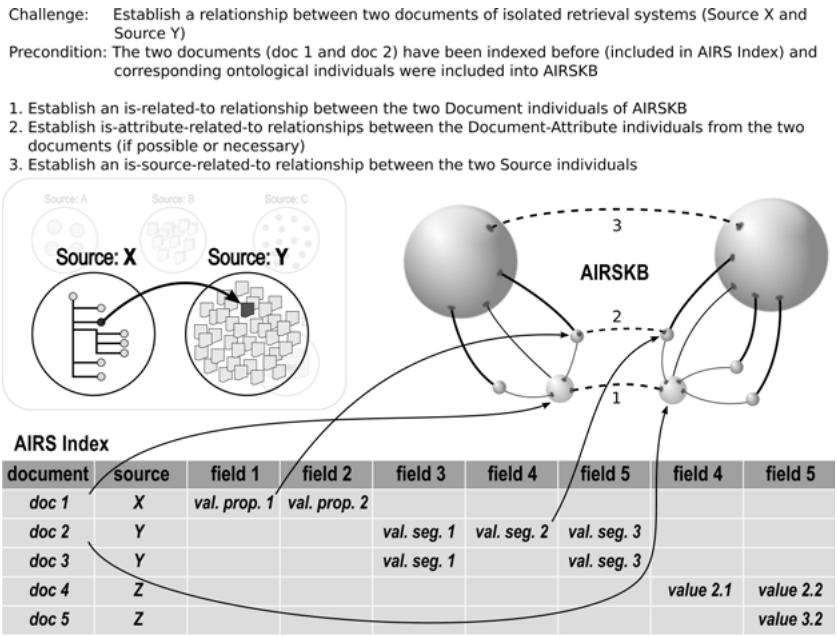


Figure 5.7: Establish relationships between two indexed documents using AIRSKB relationships; figure first shown in [96].

5.2 General Retrieval and Feedback Workflow

The Core Component of AIRS takes control of the Retrieval and Feedback Workflow. It defines the retrieval strategy for each search process and business case with the support of the Ontology Component. The Core Component coordinates the interaction between the Ontology Component and the IRS Component to deliver good sets of answers to

the user. The user types in a search query and the AIRS client-side application build a search instance object around it and send this object to the AIRS server where the Core Component takes control over it.

Figure 5.8 shows how multiple AIRS client applications communicate with the AIRS backend server application with the help of a search instance object. Therefore, each search instance object has a unique identifier to enable the specific interaction of each AIRS client application with the AIRS server.

At backend servers-side again, several retrieval algorithms process the user's query and combine the capability of both ordinary document retrieval as well as ontology-based document network retrieval. After collecting all relevant information and documents, the AIRS backend server extends the search instance object with multiple search result objects and a feedback objects. A search result object again is a container for search results that belong to the same kind of information: documents that are grouped by sources or a set of related documents based upon the Suggest Cluster Algorithm (see Section 5.4). Each search result object contains:

- a relevance-sorted set of answers AIRS responds to according to the user's information requirement,
- metadata like the total number of found answers,
- the number of answers the user gets,
- a search result explanation and more.

By contrast, the feedback object is used to collect feedback to identify previously-unknown knowledge about document relationships and include this knowledge in the AIRSKB ontology. The Statistics Component takes care of this process and the feedback object collects all of the necessary information:

- query time stamp,
- ranked list of answers,
- the original user query,
- the system-modified query, and
- search metadata (properties like user context information, search language, etc.).

The feedback process starts at the AIRS backend server side, where the Core Component sends the search instance object back to the AIRS client where the query came from. At the client side, the search result is visualized according to the answer type. After the user closes the business task, it is important to collect user feedback data such as all documents that the user included in the business case[4]. The client's Statistics Component adds all this information to the search instance object and sends it back to the server. Thereafter

[4]The user adds all relevant documents that are necessary to solve the business task to a container (like a cart in various web shops).

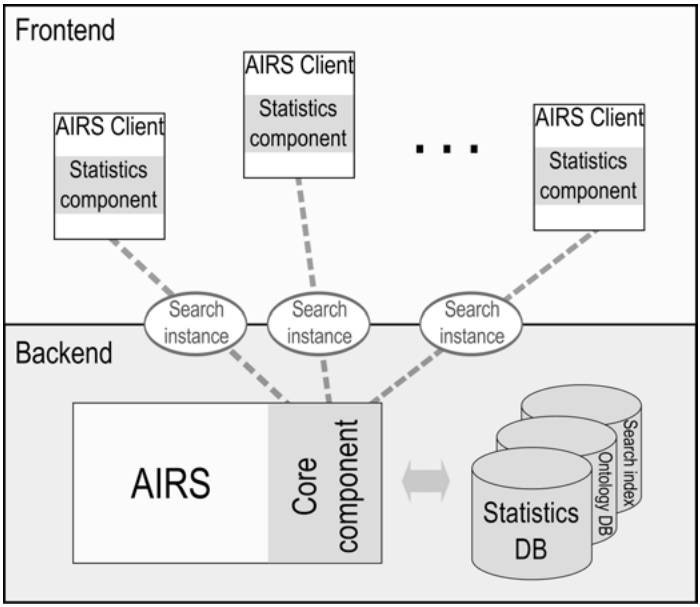


Figure 5.8: Client-server interaction with the help of the search instance object that is used for both the retrieval for documents as well as for automated feedback procession. The figure is adapted from the version shown in [94].

at the server side, the Statistics Component interacts with the Ontology Component to ascertain whether new document relationships or updated weights of relationships flow back into the AIRSKB ontology. Because AIRSKB is used to generate retrieval strategies and find document relationships during the retrieval process, the feedback of the users is collected and processed automatically and flows back into the retrieval process for future searches. Section 6 explores the feedback process in further detail and explains how this feedback information is associated with the collective intelligence of users and document authors. Section 5.5 instead explains how the AIRS feedback-processing infrastructure is used to generate Suggest Clusters that are used in the "Suggest Cluster Algorithm" (see Section 5.4).

After the user specifies the context-dependent parameters and types in the information need description in the form of a natural language query, AIRS starts the search for every kind of information that supports the user in solving the business task. Therefore, AIRS includes different algorithms for the information seeking process and combines the capability of its three main components that are used for the retrieval. Figure 5.9 shows the search graph of AIRS and the interaction between the three components: Core Component, IRS Component and Ontology Component. At least these components support two independent search algorithms that are involved in the information gathering process. Both algorithms need a query processing: the IRS Component at the backend servers side validates the user's query regarding spelling errors and combines possible spelling correc-

tions with suggestions for similar search terms (these functions only exist in dependence from the underlying information retrieval system). In Figure 5.9, the first three steps show the query preparation for one of the two different retrieval algorithms. Figure 5.9 Cluster A shows the extended information retrieval algorithm Suggest Cluster Algorithm. The Suggest Cluster Algorithm uses AIRS' infrastructure to combine knowledge from prior searches and automated user feedback to search for related documents. This kind of search for related documents is extended retrieval: the search results are based upon the feedback of the users. Section 5.4 presents the "Suggest Cluster Algorithm". In addition, Figure 5.9 Cluster B presents the steps towards an extended document search based upon state-of-the-art retrieval technology. However, AIRS' retrieval workflow for "Document Search" (see Cluster B of Figure 5.9) is much more complex than ordinary document retrieval. First of all, it extends the document retrieval in a way that it offers result clusters to the user. These result clusters were built automatically by the clustering of search results regarding attribute values. Ontology Component delivers the Source information stored in the AIRSKB ontology and AIRS divides the search into multiple sub-searches, one for each Source. The search result documents are again grouped by "their" Sources. To achieve this, AIRS enriches the search query automated for each source stored in the AIRS index with special attributes to match its retrieval characteristics (see Figure 5.9 Cluster B, Step B.1). Such characteristics are search properties like validity conditions, as described above (Context-Attributes). By looking at Figure 5.9 Cluster B, Step B.1 of the algorithm is processed through the Ontology Component: AIRS searches for all different Sources that are contained in AIRS. This is necessary because documents of different sources may need different retrieval strategies. These retrieval strategies address special index fields as well as sets of context dependencies (Context-Attributes). Once a Source is identified, the AIRS properties management searches in the AIRS properties files for information about special search characteristics that are necessary to search for documents of the given source. The Ontology Component builds a list of all Sources found in AIRSKB (Document Search Algorithm, Step B.2). Core Component iterates through this list (Step B.3) and makes a single search for each source (Step B.5) after it adapts the query with special characteristics, as previously described (Step B.4).

Finally, the IRS Component is used for document search (Step B.5) and each result is included in the result set (Steps 4-5). This means that this kind of retrieval supports many result classes and it is only content driven. Following this, it is very easy to include new sources in the AIRS document retrieval: only the index documents, the AIRSKB individuals and the special search characteristics in the AIRS properties file are necessary. Based upon the knowledge network capability of the AIRS document retrieval, it is also possible to search for related documents for each search result. Section 5.3 proceeds in further detail in a related-document search regarding a single document.

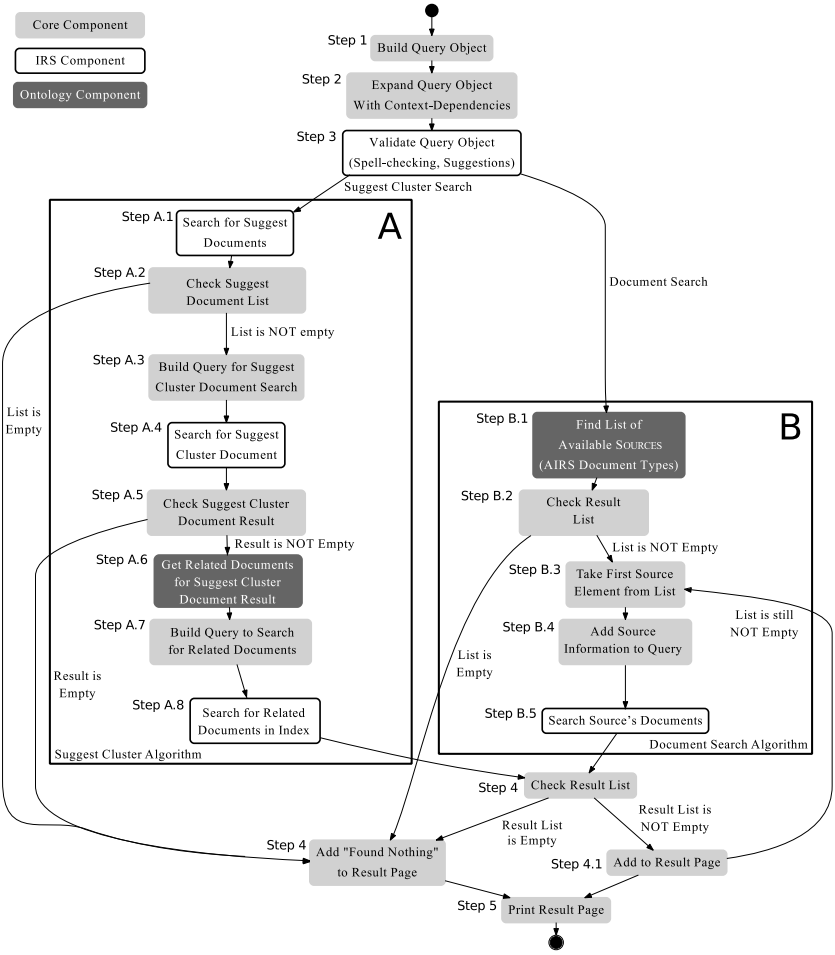


Figure 5.9: AIRS retrieval graph regarding a user search and the interaction between Core Component, Ontology Component and IRS Component. Cluster A shows the "Suggest Cluster Algorithm" and B shows the extended "Document Search Algorithm" of AIRS.

5.3 Related Documents for a Single Search Result

AIRS' retrieval capability offers both an extended document retrieval algorithm as shown in Figure 5.9 and the possibility to find related documents regarding a single document. Figure 5.10 depicts two functions of Ontology Component, which are used for this task: the search for direct related documents and the search for related documents for a target source. A single search result serves as the starting point: the search instance object (Figure 5.10, Step 1) contains the user's query (free text, concatenation of terms $t_A t_B t_C \dots t_N$) and a ranked list of result documents (doc b, doc c). For instance, a user could now be interested in documents that are related to one of the result documents (for more information about the results topic). Accordingly, the user clicks on a single result document (doc c, for example) and can trigger a new search for directly related documents. The system's response is a new search instance object that contains a set of related documents ranked by relationships strength (Figure 5.10, Step 2). The special aspect is that neither the client application nor the user knows where the result list comes from: AIRS routes queries automatically to the components that can answer the requests (free text search to IRS Component and related document requests to Ontology Component). Every answer looks the same for the client application.

The search for directly related documents addresses all of the knowledge that exists in a virtual way but knowledge that cannot be adapted easily in current retrieval system indexes. By means of virtual knowledge, the meaning of the knowledge networks that includes information about documents relationships of all users is addressed. For example, an employee of a car workshop has many years of experience in the workshop processes and knows exactly what kinds of workshop documents are necessary for a given use case. In other words, if a workshop employee knows what kind of maintenance a car of a customer needs, the employee might know which kinds of replacement parts and which kinds of repair instruction documents are necessary for this maintenance. This kind of knowledge is private and isolated from other colleagues. It is also not represented in the current retrieval systems. One can interpret the sum of all of the knowledge of a single workshop employee as a private knowledge network about workshop document relationships. The totality of knowledge networks of all workshop employees can be seen as a collective knowledge network of document relationships. Unfortunately, this knowledge is not represented in the current retrieval system. The collective intelligence of workshop employees is discussed more in detail in Section 6. Because the direct related documents are also derived from feedback, the user gains part of the collective knowledge network.

Figure 5.10, Step 2 shows the findRelatedDocuments function of the Ontology Component, which consumes an AIRS document ("doc c") and searches in AIRSKB for direct related DOCUMENTS. This function searches for all context-sensitive relations $\mathrm{rel}\,(o_p, o_q)_{A_{C+}}$ of a given DOCUMENT o_p in the AIRSKB (see Section 3.3). AIRS uses the related documents to build a search result object. For this approach, relation weights are used for result ranking rather than the IRS retrieval algorithm. As shown in Section 3.2 as well as in Section 3.3, AIRS can be used to navigate through the knowledge network and find related documents of a given target SOURCE regarding an also given DOCUMENT. The user can now select a search result document and choose a target source that the user is interested in and start a search for related documents[5].

[5]For example, a user can be interested in searching for repair instructions regarding a replacement part.

Figure 5.10, Step 3 now shows the findRelatedDocumentsOfSource function of the Ontology Component, which is responsible for this task. For this purpose, the pathway equations presented in Section 3.3 and in Section 3.3 are used to build the ranked-result set of documents. Accordingly, best context-sensitive pathways pathway $(o_1, s)_{A_{C+}\,\text{best}}$ are calculated for the given DOCUMENT o_1 and also given SOURCE s. Section 6 shows how new relations between AIRSKB documents can be established.

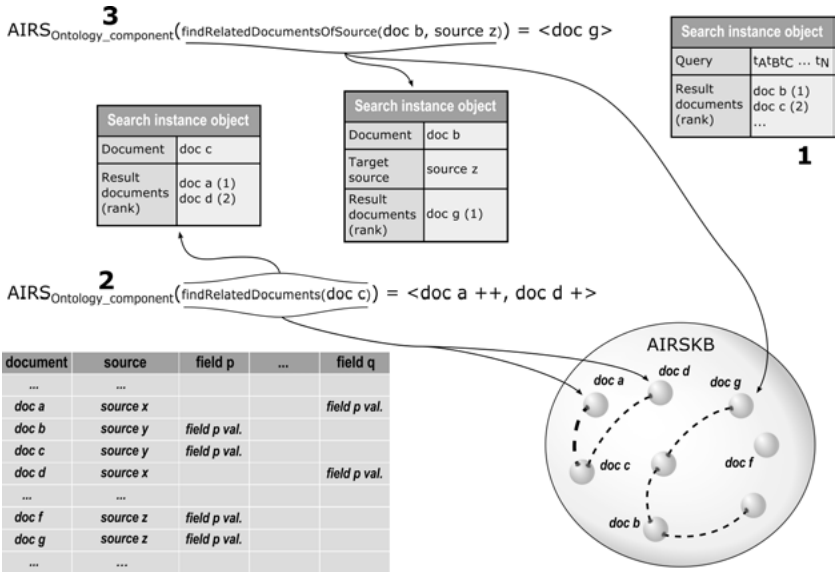


Figure 5.10: Finding related documents regarding a single result document. The starting point is a ranked result list that the AIRS application offers the user (Step 1). A user can select a single document and trigger a new search for related documents (Step 2) or related documents of a given target source (Step 3).

5.4 Document Search Using Suggest Cluster Algorithm

AIRS combines state-of-the-art document retrieval with knowledge network technologies offered by an ontology. Section 5.2 explained how the retrieval of AIRS works and Figure 5.9 depicts the algorithmic structure of the search concerning how the IRS Component and Ontology Component work together under the control of the Core Component. By looking at the "Documents Search" algorithm (Figure 5.9, Cluster B), AIRS' document retrieval offers result sets grouped by the documents' sources. These documents set are built dynamically with the help of source information stored in the AIRSKB ontology. Section 5.3 explains how the users can search for related documents regarding a single search result

by using the power of document relationship knowledge stored in AIRSKB. However, the infrastructure of AIRS can also be used for new kinds of retrieval technologies that comprise a deeper integration of the ontological knowledge into the document retrieval. The algorithm shown in Graph 5.9, Cluster A is an example of an advanced usage of the AIRS infrastructure. It provides the Suggest Cluster Algorithm. The Suggest Cluster Algorithm extends the ordinary search with document sets (the so-called Suggest Clusters), which are linked to concept sets (set of so-called Suggests). A concept can be a description of a real-world activity, like "repairing something" or an entity like "car". The real-world activity is again represented through a set of terms that describe the activity. This term set can comprise synonyms (like "fixing" or "repairing" for activity "repairing something") or other semantic relationships that make sense for the given domain to build concepts. A concept is interpreted as a Suggest document. A Suggest Cluster is subsequently a combination of concepts. This combination subsequently builds a more specific concept. For example, the concept for the activity "repairing something" in combination with the concept of "car" builds the concept of "repairing cars". The idea behind this is that Suggest Cluster "repairing cars" can be described through different term combinations like "repairing vehicle", "fixing car" and others.

As stated above, the Suggest Cluster Algorithm uses the AIRS and AIRSKB infrastructure. In this sense, Suggest can be interpreted as a virtual system that holds all of the concepts. A document of this virtual system comprises a set of terms that describes the concept. Suggest Cluster can also be interpreted as virtual system that holds more specific concepts described through Suggest documents: a Suggest Cluster document is a concatenation of Suggest documents. For AIRSKB, this means that that both Suggest Cluster and Suggest serve as SOURCE individuals:

$$\{suggest\_cluster, suggest\} \subset S \tag{5.1}$$

Suggest Cluster and Suggest documents are interpreted in AIRSKB as DOCUMENT individuals. In general, Suggest Clusters are knowledge hot spots that bring together groups of concepts with a group of workshop documents. The special aspect is that this connection between concepts and documents is not based upon typical search technology. Rather, it based upon knowledge mapping between concepts and documents through using relevance feedback.

Graph 5.9, Cluster A shows how the Suggest Cluster Algorithm works in general and depicts the necessary steps in processing a user query towards the search result. The first steps (Steps 1 - 3) are necessary for processing the user's query to prepare it for later retrieval steps. The Core Component and IRS Component take care of these steps. The Suggest Cluster Algorithm itself starts with a search for Suggest documents (Step A.1) with the prepared query by using the IRS Component (Step A.1). If Suggest documents were found, a new request for a corresponding Suggest Cluster is prepared by Core Component (Steps A.2 - A.3). The query for the Suggest Cluster document comprises a concatenation of previously-found Suggest documents. The IRS Component subsequently takes care of the search for the Suggest Cluster document in the AIRS index (Step A.4). If a Suggest Cluster document has been found (Step A.5), the Ontology Component searches for related DOCUMENTS for theSuggest Cluster DOCUMENT found in the AIRSKB (Step A.6). If related DOCUMENTS exist, the Core Component builds a query to retrieve the corresponding document information from the AIRS index (Step A.7), because only

the AIRS index holds content information of document (in contrast to the AIRSKB ontology). Once the documents were found in the AIRS index (Step A.8), they are displayed to the user as a search result (Steps 4 - 5).

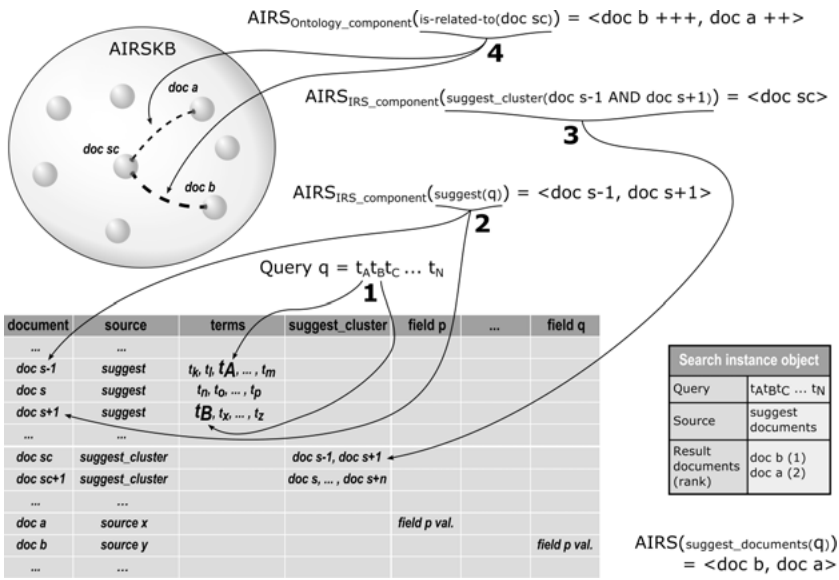


Figure 5.11: Searching for related documents using the Suggest Cluster Algorithm. First, the search term of a user's query is mapped against the Suggest documents (Step 1) using vector-space retrieval technologies offered by the IRS Component (Step 2). The Suggest document is represented by a set of terms in the AIRS index. For example, $\{t_B, t_X, \ldots, t_Z\}$ is the term set for Suggest document "doc s+1". The set of *found* Suggest documents is now used to build a new query to find the corresponding Suggest Cluster document (Step 3). The final step is a search for related documents in the Ontology Component regarding the *found* Suggest Cluster document (Step 4).

Figure 5.11 depicts how the Suggest Cluster Algorithm works in detail. The first step after the user types in the query is to map these queries to the index documents representing the concepts. These concept documents are called Suggest documents (the source information stored in the index is "suggest"). Each concept is represented through such an index document and for a concept description a set of terms is stored in a special index field (see documents "doc s-1", "doc s" and "doc s+1" of Figure 5.11, for example). This is shown in Steps 1 and 2 of Figure 5.11, where the query terms are mapped against terms of the concept using state-of-the-art vector-space retrieval offered by the underlying retrieval system (encapsulated through the IRS Component). Once the query was mapped to the AIRS index, the result set containing Suggest documents is used to build a new query, which is a conjugation of suggest-id components. This new query is now mapped against

the AIRS index to find the corresponding Suggest Cluster document (see Step 3 of Figure 5.11). Suggest Cluster documents in detail are special index documents that have index fields containing a set of concept ids. Moreover, each Suggest Cluster index document has a unique set of concept ids, whereby each Suggest document is a representation of only one concept cluster. A concept cluster again can represent a thing, an activity or a process description in the real world.

As stated above, Suggest documents are concepts that stand for an object representing a thing, an activity or a process in the real world. By looking at the workshop process, these things are concrete task-describing elements. For example, if a customer's car needs service because the front window is broken, whereby a replacement is necessary. The German word for front window is "Frontscheibe", although "Windschutzscheibe" is also a similar translation. These two words are synonyms. Together, they stand for the concept of the object front window. The set of terms {Frontscheibe, Windschutzscheibe} build the corresponding Suggest document. Considering that the front window must be replaced, another concept becomes part of the case description, namely the part replacement activity. To "replace something" can be translated as "ersetzen" in German. Additionally, the German words "tauschen" and "wechseln" are synonyms for "ersetzen". This means that the set {ersetzen, tauschen, wechseln} stands for the activity of "replacing something". This activity concept is also stored in the AIRS index as a Suggest document. The workshop employee now types the string "Windschutzscheibe aufgrund Riss ersetzen" (German for "Replace front window due to crack") in the search bar and the AIRS' IRS Component maps the query against the AIRS index. Because the terms of the query map both the Suggest document of "front window" (SD1) as well as the Suggest document of the activity "replace something" (SD2), the two Suggest documents appear in the result set. This result set now is used by the Core Component to build a new query to search for a Suggest Cluster document that matches the set of Suggest documents found. Given that Suggest documents are assigned to unique sets of Suggest document identifiers, the necessary Suggest Cluster document (SCD1) must belong to the Suggest document set {SD1,SD2}. The resulting query is "SD1 AND SD2". Once the Suggest Cluster document is found, the Core Component triggers a search in the Ontology Component for documents that are related to the Suggest Cluster document found: findRelatedDocuments(SCD1). If documents that are related to the Suggest Cluster document exist in AIRSKB, AIRS builds the result set containing these documents, using the weights of the relationships (equal weights cause equal ranks in the result set). In this sense, even documents that could not be necessarily determined by the underlying search engine algorithm can be found through the user's query.

Example 8: Example for the Suggest Cluster Algorithm.

Example 8 explains such a concept cluster reference to real world activities. If such a Suggest Cluster document was found in the AIRS index, the Ontology Component searches for the Suggest Cluster document representation in AIRSKB to find the Documents related to the Suggest Cluster Document. As Figure 5.11 Step 4 depicts, the weight of the relationships is used to build and rank the result set containing these related Documents.

After AIRS enriches the related DOCUMENTS with content information, this result set is submitted to the user as a search result.

5.5 Update Suggest Clusters for Suggest Cluster Algorithm

In the previous section, Suggest documents were introduced as AIRS' containers for concepts that represent objects, activities or processes of the real world. Furthermore, these concepts are described through a set of terms. These term sets can be interpreted as synonym clusters in the sense of a specific domain, namely the after-sales automotive market. The domain is a basic condition for describing concepts through terms, because terms are often ambiguous. A clear domain reduces this effect due to a clear vocabulary: some meanings of ambiguous terms often make no sense in a specific domain. An example is the German word "Bank", which actually has several meanings, including financial institution or bench. If the specific domain is about financial transactions, the word "bench" does not work as an interpretation for "Bank".

Returning to the Suggest documents, one question remains open: what seems to be a good natural language source for such a concept describing synonym sets? Good sources for such natural language resources are technical domain specific thesauri, taxonomies or even natural language databases used for controlled vocabulary processes. Additionally, text mining technologies can be used to build these natural language resources. In this sense, calculating co-occurrences of a higher order over a domain-specific text resource can produce sets of related words[6]. Even free accessible text resources can be used to build sets of synonyms. For example, the sub-project of the Wikimedia foundation Wiktionary[7] can be used to build concepts for objects, activities or processes. For AIRS, various natural language resources (in-house and external) were used to build the Suggest documents. All together AIRS contains more than $7,000$ unique Suggest documents. As explained in the previous Section 5.4, each Suggest Cluster document fits exactly one combination of Suggest documents. For example, a set containing only three Suggest documents (SD1, SD2 and SD3) causes seven possible Suggest Cluster documents: SCD1{SD1}, SCD2{SD2}, SCD3{SD3}, SCD4{SD1,SD2}, SCD5{SD1,SD3}, SCD6{SD2,SD3} and SCD7{SD1,SD2,SD3}.

Following this, the count of all possible Suggest Cluster documents for a Suggest document set is described through the power set of all of the Suggest documents without the empty set: $\mathcal{P}\left(\text{Suggest documents}\right) \setminus \emptyset$. It seemingly can be concluded that an index must at least contain $2^{|\text{Suggest documents}|} - 1$ Suggest Cluster documents. For AIRS, this means that the index must contain $2^{7,000} - 1$ different Suggest Cluster documents, which is an unreachable number. Fortunately, only a very few combinations of Suggest documents semantically make sense. An easy solution is that only Suggest Cluster documents flow in the AIRS index if the combination of Suggest documents was used in real-life situations. Therefore, the system investigates the relevant feedback object of a given workshop maintenance or car repair use case.

Figure 5.12 displays how dynamically adding Suggest Cluster documents regarding a

[6] See [41] for more information about co-occurrences of a higher order.
[7] See http://en.wiktionary.org, last visited Sept. 18, 2016.

feedback object works in detail. Steps 1 and 2 work the same way as the first two steps in Figure 5.11, albeit with the difference that the feedback object can contain multiple queries. Many queries are present because a user can conduct even more than one search for a document in a case. Altogether, a feedback object contains all of the documents that were included in the case solution (the feedback documents shown in the search instance object table of Figure 5.12) and all of the queries made to reach that solution. This means that Suggest Cluster documents are updated or new ones are established in AIRSKB for each query of the feedback object. Figure 5.12 shows this process only for a query q.

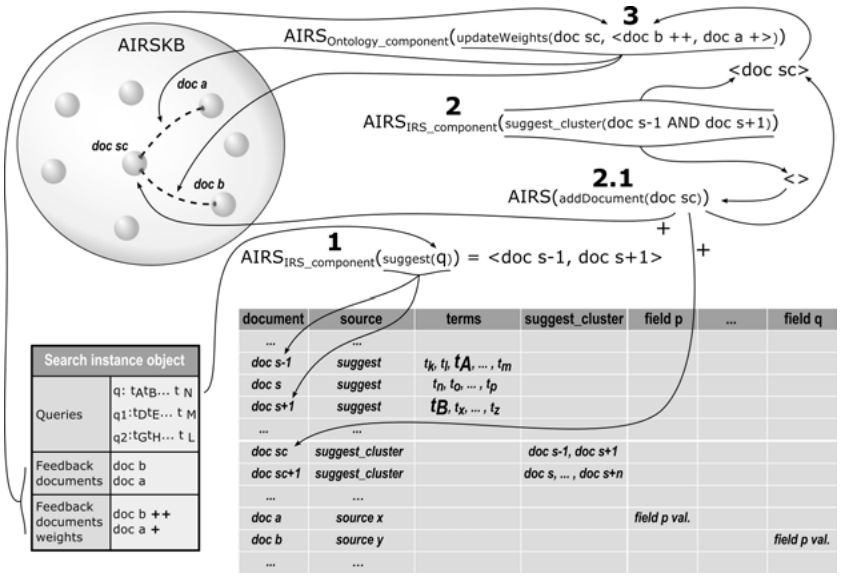


Figure 5.12: Updating Suggest Cluster documents of AIRSKB regarding feedback information. After feedback information is collected by the AIRS application, each query of the feedback is used to find the relevant Suggest documents (Step 1). The *found* Suggest documents ("doc s-1" and "doc s+1") are subsequently used to search for the corresponding Suggest Cluster document (Step 2). In the example, "doc sc" is the Suggest Cluster document found, because it is the index document that matches the query ("doc s-1 AND doc s+1"). Once no Suggest Cluster document exists, a new one that fits the Suggest documents is built and adds to the AIRS index as well as AIRSKB (Step 2.1). In this case, a new Suggest Cluster document representing the combination of the two Suggest documents ("doc s-1" and "doc s+1") is established in AIRS. Whether a new Suggest Cluster document has been built or an existing one has been found, the Ontology Component is used to update existing relationships between the DOCUMENTS of the feedback object or update existing relationships in AIRSKB (Step 3).

After a query q was mapped against the AIRS index (see Step 1), Step 2 is used to identify the correlating Suggest Cluster document. In the case that no Suggest Cluster document

exists, a new one needs to be established in the AIRS index as well as AIRSKB (see Step 2.1). The Suggest Cluster document as well as the feedback documents are used to either update existing relationships between the Suggest Cluster Document and the feedback Documents of AIRSKB (with the help of the relevant feedback documents' judgments) or establish new relationships in AIRSKB, using baseline weights for new relationships. This is shown in Step 3 of Figure 5.12, where the Ontology Component updateWeights is used for this task. Again, this needs to be performed for each query of the feedback object. The result of this is that multiple Suggest Cluster Documents of AIRSKB link to the same set of Documents ("doc a" and "doc b" in the Figure 5.12). Theoretically, various sets of concepts link to the same use case solution represented through a set of documents. In the case a Suggest Cluster Document (correlated to a user query across a set of Suggest documents) links to another solution (feedback documents), AIRS recognizes this and updates the weights of the relations between the one Suggest Cluster Document and all of the feedback Document of AIRSKB. Existing weights between the Suggest Cluster Document and other Documents of AIRSKB that were not included in the feedback object are reduced. Those weights of Document relationships where the documents were included in the feedback, are increased and new included Document relationships become baseline default values. The idea behind this is that similar Suggest document sets means similar use cases, although there is not only *one* solution for the use case. The collective intelligence of workshop employees also means that a dynamic knowledge base that can adapt itself to new findings is necessary. Even experts can fail in finding case solutions and other experts can find *better* solutions. The hypothesis is that the best solution – like in Darwin's evolution theory – prevails over time. The more workshop experts that match a Suggest Cluster document with a query and use the right documents in the case solution, the more relevant that these documents appear to the document cluster. This is measurable through changing document relationships over time where the right documents appear in higher ranked positions in the search result.

The only question left is when the update process of the Suggest Cluster document should be performed. For instance, it can be undertaken every time following a search or it can be conducted later, provided that the feedback objects are stored in a database for a subsequent analysis.

6 Sharing Knowledge through AIRS

This section presents and extends the findings of [96], discussing the AIRS system in connection with collective intelligence. In [2], collective intelligence is defined as – among other things – "*intelligence that is extracted out from the collective set of interactions and contributions made by your users*". Moreover, AIRS is a system that processes implicit intelligence and acts as a recommendation engine (see also [2]). Furthermore, the processed implicit intelligence is presented and persisted through an ontology, namely AIRSKB. As stated above, AIRSKB is a model that stands for the heterogeneous document landscape. One can interpret this model as a large network that holds knowledge concerning how documents across disparate sources are related to each other.

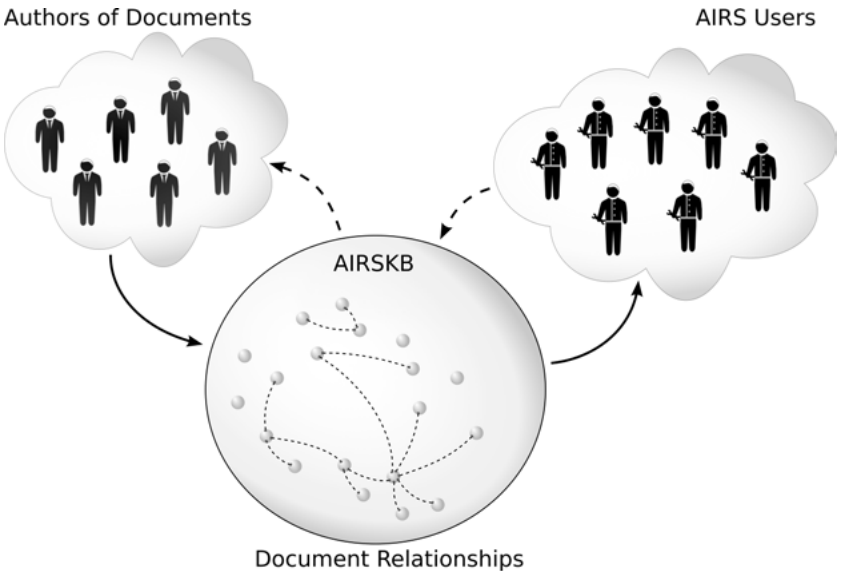


Figure 6.1: AIRS and collective intelligence as shown in [96].

In Section 3.1, the Application Context was named as one of the important elements of the ontology engineering process. This Application Context had a major influence on the ontology design and was one reason for an ontology model that is an abstracted network of document relationships without containing data, because the goal was to include the collective intelligence of users into the retrieval process in an easy way. Each AIRS user

has an own idea of document relationships. More specifically, they know which documents are helpful for a case solution. The challenge is to merge both the users' knowledge and AIRSKB's knowledge network. Figure 6.1 shows the goal of AIRSKB's design. The authors of documents gather all knowledge that can support the users in their case. Without AIRS, the users work with this knowledge, but they have no influence on the retrieval due to system limitations. The design of AIRSKB gives the possibility to collect the knowledge of users to optimize the retrieval algorithm. Figure 6.2 illustrates how the collective intelligence gathering process works in detail. An author creates a document, (Step 1) and correlates it to another document, (Step 2). Later on, this document is published in its retrieval system and is available to the users. AIRS takes this document and includes all necessary information about the document into its IRS index and AIRSKB (Step 3). If the user uses AIRS rather than the isolated retrieval system for the case, the user can find the document (Step 4) and includes it in the case solution (Step 5). If the user includes a second document in the case (Step 6), AIRS recognizes that there is a relationship between the two documents (Step 7) because both documents were necessary to solve the task. AIRS includes this information in its AIRSKB by building a new relation between the two documents (Step 8). The new document relation is available for future searches.

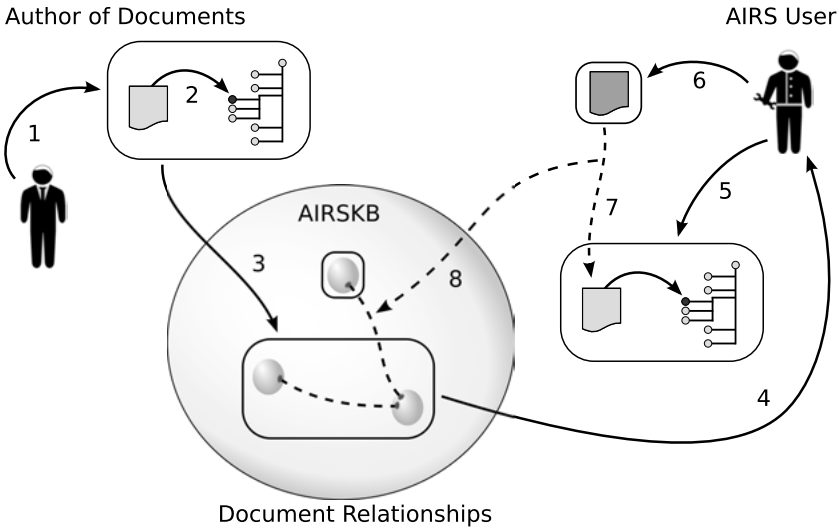


Figure 6.2: AIRS and collective intelligence in detail as shown in [96].

For this approach, three things must be considered: *why* is it necessary to collect feedback, *how* feedback can be collected and *what* is relevant to be collected. Therefore, the next section discusses in detail why feedback should be collected and how it can be collected through the Statistics Component. Furthermore, this chapter explains what relevant feedback is and how the relevance judgments from users can be approximated.

6.1 Collecting Feedback with the Statistics Component

Collecting feedback is a task of the Statistics Component. With feedback processing, entity relationships between the AIRSKB individuals can be updated or the AIRS' system quality can be measured. In [94], the evaluation of information retrieval systems is introduced as a process that "*requires finding a computable measure*". Accordingly, this measure is "*a generalization of many users' individual understandings of relevant information the system retrieves according to the information need each user defined before*". The outcome of this is that in the case of business-oriented retrieval, tasks (like those that can be found in the workshops) differ from other search tasks like an ordinary web search. The reason for this is that users perceive search results as instructions rather than recommendations. The focus here is on the business case itself and the straightforward process to solve it.

If the feedback is used to update AIRSKB, there exist two possible ways for the system to update AIRSKB's relationships. Just in time, meaning right after a user has finished the use case or after a specified time period. In this case, the feedback is stored for later analysis in the AIRS' statistics database. Additionally, collecting the feedback first and analyzing it later can be undertaken in a repetitive manner. Following this, a relation aging process can be implemented. Relation aging follows the principle of watering flowers: if flowers are watered, they are healthy and they will grow. Not watering the flowers makes them die very slowly. Darkening the window will let them die even faster. Having recognized that the flowers are dying because they are dry, watering lets them be healthy and grow again.

"Using relations" means including documents that are related to each other in the business case. Relations that are used often are "healthy", whereby they become more relevant for similar business cases (their weights are increased). Relations that are not used (or seldom used) are not "healthy" and lose relevance for a business case and their weights decrease over time. Not using relations in a given use case – even if the related documents have been suggested – is the same as darkening the window: the relation weights are reduced faster. To summarize, the feedback can be collected to support one of the following cases:

- Enabling relation-aging over time,
- Updating document relationships for both related documents search and Suggest Cluster search, as well as
- Getting information about the *acceptance* and *goodness* of AIRS.

However, all three of the cases need feedback information. Following this, several aspects need to be collected through the feedback process. Section 5.2 names elements of the feedback object and Section 6.2 describes how relevance judgments can be processed by an application.

The feedback needs to be collected on the client side. Feedback information is transported through the search instance object, as described in Section 5.2. For this task, the system can collect user-click stream data as well as search metadata. Responsible for collecting this information is the AIRS client's Statistics Component. After the feedback has beenn sent back to the AIRS server, the Statistics Component at the backend side analyzes the feedback and stores it in the AIRS statistics databases. Figure 6.3 shows the data schema of one of the databases responsible for storing the feedback: the feedback

table is used to correlate the feedback to a given use case performed with the AIRS client application. A use case itself was the process a workshop employee went through to process the maintenance, service or repairs request. As depicted in Figure 5.9, AIRS extends the user's query with context dependencies that are necessary for the retrieval. Additionally, AIRS splits the search into multiple sub-searches: one for each source that is contained in AIRSKB. The table feedback_contexts now links existing CONTEXT-ATTRIBUTES to the feedback stored in the feedback table. All of the CONTEXT-ATTRIBUTES, CONTEXTS, DOCUMENTS and SOURCES that exist in the AIRS environment are also stored in the Statistics Database of AIRS to link given feedback information to the documents.

Following the retrieval of the algorithm in AIRS (AIRS splits each search into different sub searches), the source_query table (Figure 6.3) contains information for each of the sub-searches undertaken by the AIRS retrieval algorithm. Search result information like the number of documents found or the current presented document results in a range for each of the sub-searches, which are stored in the document_result table (Figure 6.3). Documents of this range that were presented to the user are stored as entries in the presented_documents table (Figure 6.3). These documents are the search results embedded in a search result web page container. As stated before, obtaining relevance feedback from the users is a difficult process. Section 6.2 explains how this task can be performed by AIRS. AIRS collects all the all of the user feedback and stores it in the AIRS statistics database to the same feedback object of the search to which it belongs. This can be achieved after a user has finished the business task or while AIRS is analyzing the ongoing business task (recognizing all used documents). Therefore, AIRS observes the user's behavior while the user interacts with the result set. Regardless of whether the entire business case or only a single interaction of a user with a result document is analyzed, the feedback is processed by AIRS in the same way. Accordingly, if a user now clicks and views a document at the client's side, an AIRS feedback information is performed and stored in the AIRS statistics database as an entry in the document_actions table (Figure 6.3). Therefore, the feedback object in the feedback table is identified, the correlated source query entry in the source_query table and the correlated document result as an entry in the document_result table (Figure 6.3). If the user-selected document now is contained in the selected_documents table, a new document action is included in the document_action table (Figure 6.3). Otherwise, a new selected document is included in the selected_documents (Figure 6.3) table first. In this sense, a document action is everything that a user can do with result document. The user can:

- include the result document in the business case solution,
- delete a result document from the business case solution,
- simply view a result,
- start related-document navigation from a result document (the related document will be included as a new selected document in the AIRS statistics database) and many more.

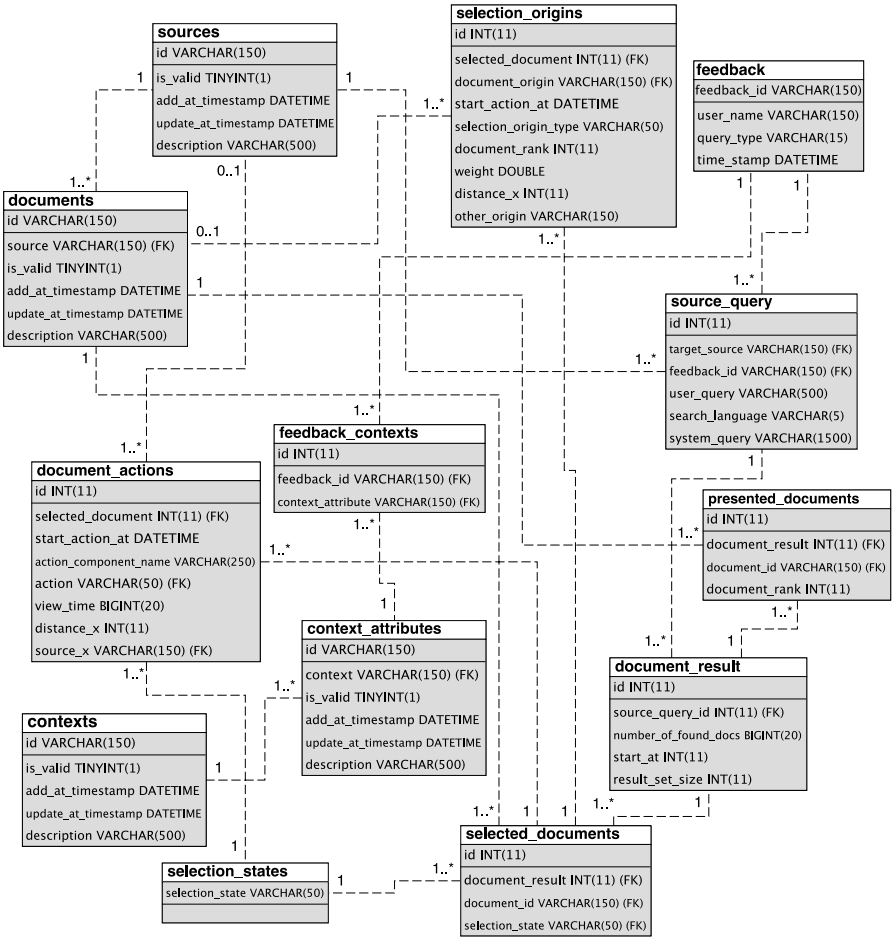


Figure 6.3: AIRS statistics database schema. Every feedback object is correlated to an entry in the feedback table. Additionally, the context of the search is stored in the table feedback_contexts. The schema contains a table for search result documentation (source_query, document_result, presented_documents and selected_documents) as well as for tracing the user interaction (selection_state, document_actions and selection_origins). All index entities are stored in tables and linked to the feedback (tables sources, documents, contexts and context_attributes).

All of these actions are stored as document actions and can be later used to approximate document relevance regarding a solved business case. These actions can also be used to update approximate relationships between documents. The data stored in the Statistics database can now also be analyzed through standard business intelligence reporting tools like Apache BIRT[1] as an example of an open source solution or any other product to calculate the acceptance and goodness (see [94]) of the AIRS retrieval system over time.

Therefore, getting result relevance judgments by the users is a task that is necessary to understand the systems acceptance, goodness and quality. The principles of *goodness* and *acceptance* were developed during the development of AIRS and first presented and discussed at the SDPS 2012 Berlin Conference. To measure the quality of a system or update document relationships (as used in AIRS), it is necessary to obtain relevance judgments about search results from the user. Since [94] explores the topic of how to obtain relevance judgments in further detail, the next section is a slightly modified extract from this publication.

6.2 Getting Relevance Judgments

This section is an adapted version of Section *Getting Relevance Judgments* first presented in [94]. Getting feedback from the user is a complex matter. For a good evaluation, it seems very important to obtain objective relevance judgments for every answer in the result set. Unfortunately, this is a major problem. A decision concerning whether an answer is relevant is always a subjective assessment made by a person. Good relevance judgments are obtained when a group of users does not know that they are judging.

In [94], relevance judgments strictly depend on the visualization of the results presented to the user. For example, by using a document list, one can put a *document is relevant* button next to each document. Regardless, this is ambiguous: the user could also think that he has to rate the quality of the document rather than the document's relevance. A solution is to approximate the document's relevance by making assumptions. For example:

- a document does not seem relevant if the user does not care for it (a brief summary of the document is shown for each result, whereby the user can choose whether he wants to see it in detail),
- a document seems relevant to the user if the user clicks on the document link,
- a document seems even more relevant to the user the longer time the user spends viewing it (how long does it take until the user closes the window after the user clicked on the document link),
- a document seems less relevant to the user the shorter time the user spends viewing it, and
- a document definitely seem relevant to the user if he downloads it (documents are also associated with their PDF-file version).

[1] See http://www.eclipse.org/birt, last visited Sept. 18, 2016.

By doing so, there is one open question left, namely how to ascertain whether the search result leads the user to the business task goal. The answer is not easy and always seems to be: *examine the business task*. Fortunately, it is possible to approximate this by answering questions like:

- What does the user do right after the result viewing (search again or close the business task)?
- Was there a support request (submitted by the user) linked to the user's business task (user cannot satisfy business task)?
- Does the business task solution belong to the AIRS answers?
- Does the user use the system for the business task?

For AIRS, the feedback documents were chosen under the given assumptions by harvesting the business case. These documents are subsequently included in a feedback processing chain. For example, Algorithm 4 uses the feedback documents to update the Suggest Cluster documents of AIRSKB (see Section 7.5).

6.3 Summary

A user needs to solve a given case. Accordingly, the user searches for documents and includes all relevant documents in the case solution. AIRS helps the user to collect these case-relevant documents. Users who need to solve similar cases must do the same. Each user performs the same search for relevant documents again. This means that for similar cases many sets of documents exist, which have been collected by many users. One can assume that the intersection of all sets of collected documents provides all of the documents that are necessary for the resolution of the case. The result is an adaptive network of document relationships that bases on system usage. How can this knowledge about these document relationships be collected to increase the quality of retrieval processes for future searches? The Statistics Component helps to collect user feedback, which is an approximation of the relevance of documents of a given case. This feedback is later used to improve the entire retrieval process of AIRS. For example, DOCUMENT relationships are updated in AIRSKB. These relationships are used in the Suggest Cluster Algorithm and for Related Documents Search. This means that the collective intelligence of many AIRS users help to improve retrieval processes and solve complex business cases over time.

7 Architecture and Functionality of a Prototype Implementation

The following section introduces the AIRS Prototype as an exemplary implementation for a software solution based upon AIRS. This exemplary implementation includes the architecture of components as well as suggestions for algorithms based upon the previously-explained Retrieval and Feedback Workflow of AIRS. The AIRS Prototype software is a complex module-based application landscape based upon Java technologies that comprises five main modules:

- Properties Management – A backend software module that coordinates all properties requirements of all of the AIRS modules.
- AIRS Index Search – A backend module that encapsulates the functionality of an underlying retrieval system.
- AIRSKB – A module that serves as a wrapper for underlying knowledge representation frameworks.
- AIRS Include Sources – A framework based upon concurrent programming, using the producer/consumer principle for the indexing process.
- AIRS Prototype – The core web-application that includes the backend and frontend modules.

AIRS Prototype is not related to any after-sales system architecture or style guide. It is simply an exemplary implementation for the AIRS architecture as presented in the previous Section 4.3. In addition, the AIRS Prototype is used for gold-standard technology presentations as well as for user tests within the help of workshop employees as presented in Section 8. Later on, the AIRS Prototype application can serve as a template for enterprise implementations of the AIRS principle. The AIRS Prototype backend components are implemented in Java. Therefore, the components are published as Java libraries with API documentations. These libraries combine various frameworks presented in this section (AIRS Index & Search Framework, for example). The AIRS Prototype web application is based upon the Web Toolkit provided by Google and released as open source[1]. Both the frontend and backend are implemented in plain Java. GWT later compiles the Java code for the frontend to JavaScript code, which can be run at the client side in a standard browser. Server side backend components run the Java byte code in a Java VM and include the methods and algorithms of the AIRS backend frameworks. Both frontend and backend codes are compiled to a Java servlet web application that can be run through a Java servlet container such as Apache Tomcat[2]. As stated above, the AIRS Prototype software

[1] GWT (formally known as Google Web Toolkit) can be found at http://www.gwtproject.org, last visited Sept. 18, 2016.

[2] Apache Tomcat can be found at http://tomcat.apache.org, last visited Sept. 18, 2016.

landscape includes various software products used in the AIRS framework. Examples are:

- MySQL databases – Use for statistics and evaluations in the Statistics Component.
- Apache Solr – An open source search server on top of Apache Lucene serves as an AIRS' search component and is used by the AIRS Index & Search Framework in the IRS Component.
- Apache Tomcat – An open source Java servlet container serves as the application server that provides the AIRS Prototype web application to a standard Internet browser.
- GWT – A software framework for the development of dynamic web applications provided by Google that is used for the development of the AIRS Prototype web application.
- MongoDB – A NoSQL database that is used as storage backend for the AIRSKB via an AIRSKB Framework wrapper in the Ontology Component.
- Ontopia – A Topic-Maps framework for DML[3] as well as DQL[4] operations. In the AIRS Prototype, an AIRSKB Framework wrapper is used for DQL operations using the TOLOG query language. As storage backend, Ontopia provides MySQL backend and XTM (XML-based file) storage for Topic-Maps. The AIRSKB wrapper has an implementation for both of them.
- Neo4j – Is a NoSQL database. More specifically, Neo4j is a graph database used via an AIRSKB wrapper as storage backend for the AIRSKB concepts and relations. The wrapper again provides an adapter for the CYPHER language for DQL operations against a Neo4j data storage.
- Apache Jena – A JAVA-based framework for RDF/RDFS, OWL and the SPARQL language. The AIRSKB Framework also includes a wrapper for OWL storage of the AIRSKB and queries against the storage by using SPARQL. For the OWL storage, the wrapper supports both the XML-based file format OWL as well as the MySQL backend provided by Apache Jena.

The AIRS Prototype framework implementation follows the AIRS backend architecture as suggested in Figure 4.6, although it was slightly modified as shown in Figure 7.1. For instance, no foreign date interfaces were used in the AIRS Prototype. At the server side, the main component responsible for the communication between the frontend and backend is the Core Component. As described in Section 7.5, the Core Component includes the highest abstraction layer for methods necessary to access the AIRS document retrieval functions.

At lower levels, the methods become more specific and go deeper into separate parts of the AIRS backend architecture and components. The Core Component itself is the control component responsible for the task of coordinating and routing functions to lower-level components. Thereby, the Core Component routes specific tasks to the three main

[3] **D**ata **M**anipulation **L**anguage.
[4] **D**ata **Q**uery **L**anguage.

components. They are responsible for the retrieval and feedback tasks: the IRS Component is responsible for the document retrieval, the Ontology Component is responsible for the AIRSKB retrieval and the Statistics Component is responsible for evaluation and the coordination of DML operations of the application life-cycle. These kinds of operations are executed by the IRS Component for index updates and the Ontology Component for AIRSKB updates. The IRS Component includes the AIRS Index & Search Framework, which is responsible for all IRS operations, while the Ontology Component includes the AIRSKB Framework, which is responsible for all AIRSKB operations. The following sections describe the modules in detail.

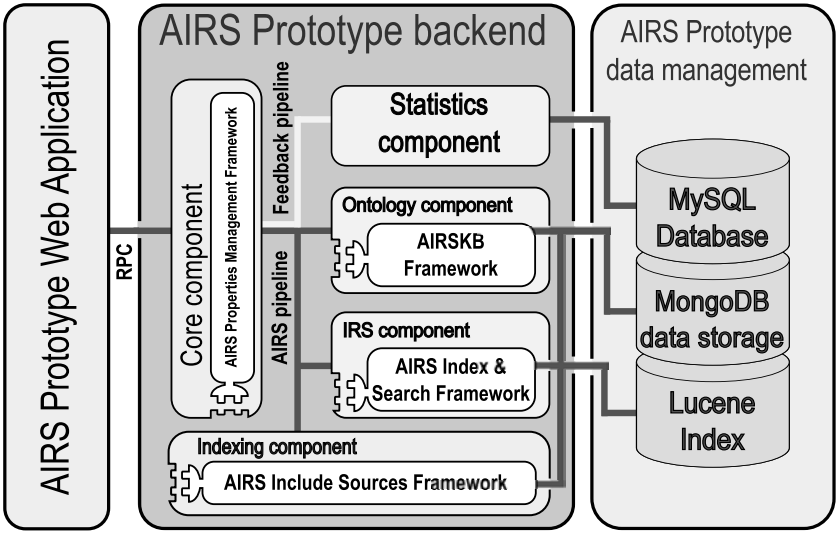


Figure 7.1: AIRS Prototype architecture overview.

7.1 Properties Management Using a Taxonomic Structure

Properties management is a difficult matter, especially in software development processes. Program logic should be separated from configuration: code lines that contain parameters are evidence of a poor programming style. Since the AIRS software landscape contains multiple wrapper modules, many parameters are necessary to configure the underlying framework. Therefore, the development of a framework for properties management was a logical decision for the help of configuration parameter management. Accordingly, the research for the AIRS Prototype application started with the development of the Properties Management component.

The AIRS program landscape is implemented in Java. With the Properties class[5], the

[5]See Java 7 documentation at http://docs.oracle.com/javase/7/docs/api/java/util/Properties.html, last visited Sept. 18, 2016.

Java library includes a mechanism that can be used to set properties outside the program code in a separate text file: the so-called Java properties file. Therefore, Java's own properties management is based upon a simple key-value storage mechanism where both key and value are strings. Additionally, the Java Properties class provides a method to store these key-value pairs in a well-defined XML properties file rather than simply a text file. Inside the Java program code, a method of the properties class can be called that consumes the key and returns the value stored in the properties file (as long as the entire properties file was loaded before). Unfortunately, configuration properties are not always as easy as simple key-value pairs. Often, they are much more complex. Programmers avoid this limitation through simple programming tricks. For example, if a list of property parameters is necessary, the value of the key-value pair is often formatted to include the list elements in just one string, separated by defined symbols. Later in the program code, this value string is split into a list along these symbols.

AIRS Prototype includes different wrappers for foreign libraries. This means that many complex configuration parameters exist. For example, the Apache Solr's configuration file contains multiple and complex parameters. Many of them could also be set in the program code[6]. Apache Solr itself is an open source enterprise search server used as the underlying information retrieval system under the AIRS Search Index wrapper. A more complex properties management was necessary for the AIRS program landscape environment. The following requirements were defined for a properties management:

1. Properties should be arranged in a logical hierarchical order: a taxonomic structure. This is necessary to match various properties levels given by the heterogeneous library landscape as well as through the module-based AIRS architecture.

2. Properties should be Java objects that are accessible by a name attribute and encapsulate a value container.

3. Value containers should have a type attribute. The type again provides information about the value container's content: a simple value or a list of values.

4. A value must have a value type attribute. It needs to encapsulate the value itself and it can be restricted through a value restriction type attribute. Additionally, if the value container holds a list of values, each value needs a key attribute to identify it.

5. A value restriction type should be a simple value range restriction or a set of valid values from which the value needs to be selected.

6. Even complex Java objects should be stored as properties values.

7. Property files need to be both humanly readable and machine processable.

To match this requirements catalog, a storage model was developed that was based upon a taxonomic structure. Standard XML was used as the file format.

[6] See Apache Solr documentation at http://wiki.apache.org/solr/SolrConfigXml, last visited Sept. 18, 2016.

```
<?xml version="1.0" encoding="UTF-8"?>
<properties_file name="properties.xml">
  <description>This is the description of the properties file.</description>
  <value_ranges>
    <range RANGE_ID="range_type_1" RESTRICTION_TYPE="STRING_RANGE">
      <val>VALUE 1</val>
      <val>VALUE 2</val>
    </range>
    <range RANGE_ID="range_type_2" RESTRICTION_TYPE="MAX_INTEGER">
      <maxVal>3</maxVal>
    </range>
  </value_ranges>
  <properties>
    <property name="node1">
      <description>This is a description of a property.</description>
      <children>
        <property name="node1.node11" is_property="true">
          <value VALUE_TYPE="SINGLE_VALUE">
            <val RANGE_IDREF="range_type_1" TYPE="STRING">VALUE 2</val>
          </value>
          <children>
            <property name="node1.node11.node111" is_property="true">
              <value VALUE_TYPE="SINGLE_VALUE">
                <val RANGE_IDREF="range_type_2" TYPE="INTEGER">1</val>
              </value>
            </property>
          </children>
        </property>
        <property name="node1.node12" is_property="true">
          <value VALUE_TYPE="LIST">
            <val key="KEY1" TYPE="STRING">String Value 1</val>
            <val key="KEY2" TYPE="DOUBLE">1.23</val>
            <val key="KEY3" TYPE="BOOLEAN">true</val>
          </value>
        </property>
      </children>
    </property>
    <property name="node2" is_property="true">
      <value VALUE_TYPE="SINGLE_VALUE">
        <val TYPE="URI">/home/airs/synonyms.txt</val>
      </value>
    </property>
  </properties>
</properties_file>
```

Listing 7.1: An example of AIRS properties file "properties.xml".

Subsequently, functional requirements were defined from the perspective of the programmers to enable a properties API call as easy as for the original Java Properties class:

1. A programmer should be able to load, change, validate and store property files using the Java API bridge.

2. After the programmer loads the properties file, he should be able to load a property value using a Java API method call via the property name or via a combination of property name with a value key in the case of selecting a list element. The method itself returns the property's value. The return value's type is the corresponding Java datatype, which matches the type defined in the properties file.

```
1  /* ### STEP 0 – load properties file ### */
2  PROPERTIES properties = new PROPERTIES();
3  properties.loadFromXML("properties.xml");
4
5  /* ### STEP 1 – load a property ### */
6  Double double_value = (Double) properties.getPropertyValue("node1.node12", "KEY2");
7
8  /* ### Step 2 – load a list of property values ### */
9  VALUE_CONTAINER value_container = properties.getPropertyValueContainer("node1.node12");
10 List<PROPERTY_VALUE> property_value_list = (List<PROPERTY_VALUE>) value_container.getPropertyValue();
11 /* iterate over all property values */
12 for(PROPERTY_VALUE property_value : property_value_list) {
13   if(property_value.getKey().equals("KEY3")) {
14     boolean boolean_value = (Boolean) property_value.getValue();
15   }
16 }
17
18 /* ### STEP 3 – build a new SINGLE_ELEMENT property string value ### */
19 /* define an integer upper border value restriction */
20 PROPERTY_VALUE_RANGE_RESTRICTION restriction = new PROPERTY_VALUE_RANGE_RESTRICTION(
21     TYPE.INTEGER, "range_type_3");
22 /* set a minimum value */
23 restriction.addMinInteger(4);
24 /* set a property value */
25 PROPERTY_VALUE property_value = new PROPERTY_VALUE(restriction);
26 property_value.setValue(2); /* causes an exception */
27 property_value.setValue(5); /* you are doing it right */
28 /* set a value container and add property value */
29 value_container = new VALUE_CONTAINER();
30 value_container.addValue(property_value);
31 /* build property and set value container to property */
32 PROPERTY property = new PROPERTY("node1.node13");
33 property.setPropertyValueContainer(value_container);
34 /* add property to properties (the method adds the property automatically to the right position at the properties file) */
35 properties.addProperty(property);
36
37 /* ### STEP 3 – update properties file ### */
38 properties.storeToXML();
```

Listing 7.2: An example how to use the properties library over an API call.

Listing 7.1 shows a simple example of a properties file. The properties.xml file contains five properties sorted in a taxonomic order. Property *node1* (Line 16) is the only one that is not bound to a property value. This kind of property exists only for the taxonomic structure, whereas the other ones have the attribute *is_property*. The properties file also

contains two kinds of value restrictions: a string range restriction (Lines 5 - 8) and a maximum integer value range restriction (Lines 9 - 11). The string range restriction defines a valid set of strings for a property value and the maximum integer restriction defines an upper barrier for possible integer values for an integer property value. The range restriction can be bound later to property values, as shown in Line 19 as well as Line 24.

The property names follow the taxonomic structure, using dots as a separator. Following the hyponymy relationship, the property with the name *node1.node11* (Line 17) is a child property of the property *node1* (Line 14). A property itself can encapsulate both property value (Lines 18 - 20) and children properties (Lines 21 - 27). A property value encapsulates a single value (Line 18) or a list of values (Line 30). Values have a type (Line 40, for example) and – in the case of a list – a key (Line 31, for example). After developing the properties file, a Java library was developed.

Listing 7.2 shows a snippet of the Java code that demonstrates the usage of the properties of the API library. A properties file can be loaded easily via a loadFromXML() method, which consumes the properties file location as a string value (Lines 1 - 3). After the XML is loaded, the properties library builds a taxonomic tree and stores it in the local machine's memory as a PROPERTIES object. Properties are translated into corresponding Java objects, which are easily accessible by the property's name and an optional key (Line 6). The code snippet in Lines 9 - 16 demonstrates how a list of property values can be accessed via the property API. A simple integer property value with a lower value border of four is created and shown in Lines 20 - 36. First, a restriction is implemented: an integer restriction with the id *range_type_3* (Lines 20 - 21). After this, the minimum integer value is defined (Line 23). According to this, a new property bound to this restriction (Line 25) can only be associated with integer values greater than four, otherwise an exception will be raised (Line 26). Later on, a value container provides the wrapper for the created single value property (Lines 29 - 30). Subsequently, a new property can be created, as shown in Line 32 and the value container can be bound to the property (Line 33). The addProperty() method of the PROPERTIES class ensures that the property is placed in the right position in the property taxonomy (Line 36). The storeToXML() method stores the PROPERTIES object back into the properties file. This kind of properties management makes the development of AIRS significantly easier. For example, if a properties file is loaded into a PROPERTIES object, this object can easily be distributed to various components and frameworks. In addition to the properties library, an unpublished Eclipse[7] plugin was developed to create and manipulate AIRS properties over an easy-to-use WYSIWYG[8] editor. Using this editor and the properties of the API library, 325 parameters (property values) were easily managed during the AIRS software landscape development.

7.2 AIRS Index & Search Framework

The AIRS Index & Search framework is a backend Java application that provides an abstraction layer across an underlying information retrieval system. It approximates the IRS's own index structure and index objects to an AIRS-conform index document, the

[7] Eclipse is an open source IDE especially for software development. More information can be found at http://www.eclipse.org, last visited Sept. 18, 2016.

[8] **W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et.

so-called AIRS documents. Additionally, it provides Java methods to add an AIRS index document to the IRS index methods as a wrapper across the IRS own search function. This AIRS search function consumes a search_instance object (as shown in Algorithm 1, see Section 7.5) and returns a search result for each source contained in the IRS index. As shown in Figure 7.2, the AIRS Index & Search Framework (Box A) comprises two layers: the AIRS Index & Search Layer and the IRS Wrapper Layer.

The AIRS Index & Search Layer provide state-of-the-art information retrieval system functions for document indexing (including document deletion and update) and document search. The difference between ordinary index and search frameworks and libraries like Apache Lucene is that the AIRS Index & Search Layer does not have any implementation of information retrieval algorithms. It simply serves as an API and defines objects and methods that consume document objects for indexing as well as search instance objects for a document search. The return value of the search method is also the same search instance object enriched with search result documents that match the user's query. The IRS Wrapper Layer again maps the methods of the AIRS Index & Search Layer to methods of an existing information-like retrieval library. By doing so, each underlying index and search framework must be implemented to match the AIRS Index & Search Layer to methods that the underlying framework provides. The IRS Wrapper Layer serves as a container for both such a wrapper and container-managing components. Box B of Figure 7.2 shows the Apache Solr Wrapper used to encapsulate Apache Solr. The wrapper comprises three different layers: a connector to the foreign framework (AIRS to Apache Solr Connector), the foreign framework itself (Apache Solr Framework) and the underlying storage container that is used by the foreign framework (Apache Lucene Index).

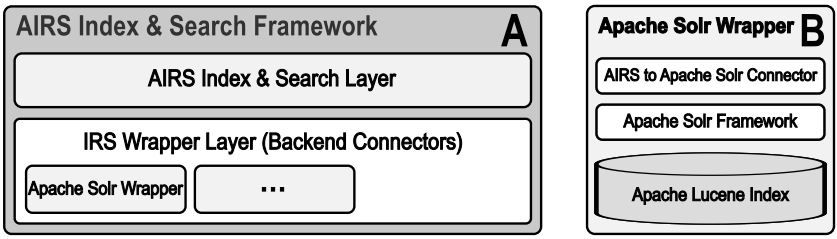


Figure 7.2: AIRS Index & Search Framework.

7.3 AIRSKB Framework

Like the AIRS Search Index framework, the AIRSKB Framework (Figure 7.3) is a backend Java application that serves as the wrapper for a knowledge-representation framework. Anyway, it comprises three different layers rather than two. From a top-down perspective, the wrappers serve as a Functional Layer, Concept Layer and Backend Layer (Box A of Figure 7.3). The Functional Layer is the top layer, providing advanced AIRSKB features like the methods that search for the best context-sensitive pathways through the ontology. It works with methods provided by the next level layer – the Concept Layer – where

ontology DML and DQL methods are implemented. Examples of such DML methods are addDocument(), addSource() or addRelation() as well as the corresponding deleting and update methods. The two top layer wrapper functions are the only ones that are accessible through the AIRSKB interface from foreign applications. Like in the AIRS Index & Search Framework (Section 7.2), a wrapper is necessary for existing underlying ontology storage framework. This is necessary because the AIRSKB Framework also has no implementation of knowledge representations storage and search functions. The Storage Layer provides such wrapper functionality. It serves as the backend connector as well as the wrapper container for existing underlying storage and ontology search engines like Apache Jena. Box B of Figure 7.3 introduces the MongoDB Wrapper of the AIRS Prototype.

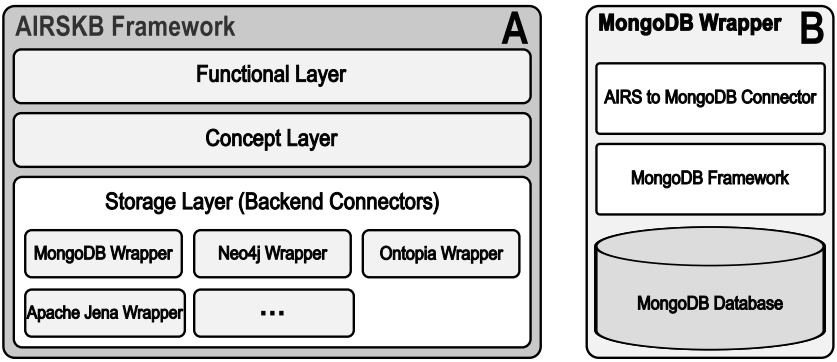


Figure 7.3: AIRSKB Framework.

7.4 AIRS Include Sources – Indexing Framework

The AIRS Includes Sources Framework is the ETL layer that helps to import the information of the original document retrieval systems into the AIRS software landscape. The AIRS Includes Sources Framework (Figure 7.4) is responsible for the DML operations between the AIRS search index and the AIRSKB, especially during the indexing process. It translates the individual documents of each source retrieval system to the AIRS documents. Additionally, the same AIRS documents are placed as ontological individuals in the AIRSKB and the framework's methods establish relationships between these concepts if real-world relationships between the documents exist. Similar to the AIRSKB Framework and the AIRS Index & Search Framework, the AIRS Includes Sources Framework provides a wrapper architecture: a wrapper must be implemented for each source whose documents need to by placed into the AIRS index and the AIRSKB. For example, Box B of Figure 7.4 shows the Source A Producer that serves as wrapper for the indexing process of documents from source A. It consumes documents of a connected source A (via a storage connector for the given retrieval system) and prepares them for indexing. In addition to the wrapper architecture, the AIRS Includes Sources Framework provides a process con-

trol flow component based upon the producer–consumer principle to manage the indexing process, even for large data sets like databases with millions of data entities. Therefore, the framework uses AIRS document units that are processed during the workflow. This works as follows:

1. Place an empty synchronized AIRS document list of a defined size in the middle.
2. Let a producer component write unique AIRS documents to the end of the AIRS documents list as long as the list is not full, whereas it waits if the list is full.
3. Let a consumer take the first AIRS document out of the list and write it into the AIRS index as well as the AIRSKB until the list is empty, whereas it waits if the list is empty.
4. Let the producer push a stop marker document through the pipeline and let the consumer stop if it processes the stop marker document.

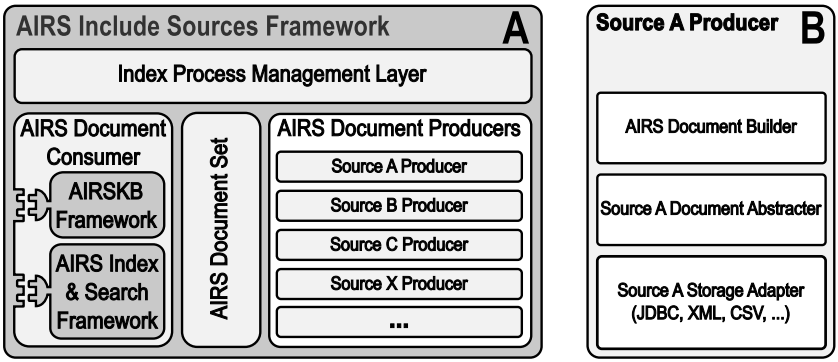


Figure 7.4: AIRS Includes Sources Framework.

7.5 Retrieval and Suggest Algorithms

In Section 4.3, three main AIRS backend components that are responsible for the document retrieval as well as the system feedback processing were named: IRS Component, Ontology Component and Statistics Component. Section 5.2 subsequently explains the retrieval and feedback processing in detail. The AIRS Prototype environment implements multiple algorithms to ensure the interaction between these components that are involved in retrieval as well as in AIRSKB knowledge expansion. It also provides a cascade of access functions to leave responsibilities for core functions in the components. For example, top-level functions are implemented in the Core Component. Such a function is the high-level search function that is implemented in the Core Component and encapsulates access to underlying components. Therefore, the search function is responsible for the communication between the user (via the frontend implementation) and the document retrieval of

AIRS: it consumes the user's query as well as some search metadata (language, context, dependencies) and processes this search request in its implementation. It combines the functions of IRS Component and Ontology Component to derive result documents that best match the user's request. IRS Component is designed to encapsulates core IR capability of an underlying information retrieval system like Apache Solr[9], HPE IDOL[10] or Elasticsearch[11] and others. Instead, Ontology Component encapsulates the functionality of standard knowledge representation and access frameworks, as named in Section 3.4. To ensure this abstraction, a common understanding of core objects is necessary. Section 4.2 named these common objects as "documents" and "sources". Four exemplary algorithms of AIRS' Core Component are presented in this section:

- Context-sensitive document search across all sources of the heterogeneous document landscape (including Suggest Cluster Algorithm, see Algorithm 1, Figure 5.9 and Section 5.4).
- Search algorithm for documents of a given source that are related to a given document across a document pathway (see Algorithm 2, Figure 5.9 and Section 5.3). The algorithm contains the context-sensitive search for related documents in the ontology, as presented in Figure 3.5.
- AIRS Prototype implementation of updating AIRSKB relationships using system automated feedback information in the form of used documents regarding a finished case (see Algorithm 3).
- AIRS Prototype implementation of updating or establishing Suggest Cluster documents for Suggest Cluster algorithm (see Algorithm 4 and Section 5.5).

As stated before, the algorithms use lower-level functions of other AIRS components. An example of a lower-level function call can be found in Line 6 of Algorithm 1 for context-sensitive document search (implemented in Core Component): ontology_component.getSources(*void*). The getSources() method is implemented in the Ontology Component and manages a high level access to the AIRSKB Framework (Functional Layer). In the next deeper level, the middleware implementation of the AIRSKB Framework (Concept Layer) guarantees a homogeneous access structure to the lowest level component of AIRSKB Framework (Storage Layer). This Storage Layer of AIRSKB Framework is the lowest level component that provides implementations to access different knowledge representation frameworks across a special wrapper that maps the AIRSKB's concepts and relations to the storage structures defined by the underlying frameworks. The outcome is a simple list of existing sources that needs to be requested at the top level. At the lowest level, knowledge storage structures hold the AIRSKB elements. Examples for these structures are OWL files, NoSQL databases and even XML files of a custom format. The middleware transports and translates the top-level request to the lower-level components and routes the response back to the top-level component.

[9] See http://lucene.apache.org/solr/, last visited Sept. 18, 2016.
[10] See http://www8.hp.com/us/en/software-solutions/information-data-analytics-idol, last visited Sept. 18, 2016.
[11] See https://www.elastic.co/products/elasticsearch, last visited Sept. 18, 2016.

Algorithm 1 shows how AIRS Prototype performs a context-sensitive search across different sources. The SEARCH() function of the algorithm is based upon the search approach presented in Figure 5.9: Document Search in combination with Suggest Cluster Algorithm.

Algorithm 1 High-level access to Document Search Algorithm and Suggest Cluster Algorithm.

```
1: ▷ The search function of AIRS' Core Component takes a search instance object that
2: ▷ contains the user's search query as well as search properties. The return value is the
3: ▷ same search instance object enriched with a set of search results.
4: function SEARCH(search_instance)
5:     ▷ Call Ontology Component to get a list of all available SOURCES.
6:     SOURCES ← ontology_component.GETSOURCES(void)
7:     ▷ Iterate over the SOURCE except the Suggest Cluster SOURCE.
8:     for all SOURCE of SOURCES \ Suggest Cluster SOURCE do
9:         ▷ Add context-dependencies and SOURCE information to query.
10:        Query ← ADDCONTEXTATTRIBUTESTOQUERY(void)
11:        Query ← ADDSOURCEINFORMATIONTOQUERY(SOURCE)
12:        ▷ Call IRS Component to find documents for the given SOURCE
13:        ▷ and keep search result
14:        SearchResult ← irs_component.SEARCH(Query)
15:        ▷ SearchResult consists of SUGGEST documents.
16:        ▷ Entry point for Suggest Cluster Algorithm.
17:        if SOURCE ! = Suggest SOURCE then
18:            ▷ Add search result to the end of the search instance's search result list.
19:            search_instance ← ADDSEARCHRESULT(search_result)
20:        else
21:            ▷ Build query for SUGGEST CLUSTER document search with the help of
22:            ▷ the found SUGGEST documents.
23:            Query ← irs_component.BUILDSUGGESTCLUSTERQUERY(SearchResult)
24:            Query ← ADDSOURCEINFORMATIONTOQUERY(Suggest Cluster SOURCE)
25:            ▷ Call IRS Component to find Suggest Cluster document (if exists).
26:            Suggest Cluster document ← irs_component.SEARCH(Query)
27:            if Suggest Cluster document is not null then
28:                ▷ Call Ontology Component to search for DOCUMENTS in AIRSKB that
29:                ▷ are related to Suggest Cluster document.
30:                DOCUMENTS ← ontology_component.FINDRELDOCS(document)
31:                ▷ Call IRS Component to derive content information
32:                ▷ for related DOCUMENTS
33:                Documents ← irs_component.GET(DOCUMENTS)
34:                ▷ Add documents as search result to search instance object
35:                search_instance ← ADDSEARCHRESULT(Documents)
36:            end if
37:        end if
38:    end for
39:    return search_instance ▷ Submit search instance object containing search results.
40: end function
```

The algorithm works whereby after a user submits the query, AIRS Prototype automatically builds a search instance object, which serves as a starting point for the SEARCH() function. This search instance object is the same as described in Section 5.2. During a user request, the search instance serves as container for search properties like user query, number of hits to return, search language and context dependencies.

On the other hand, Section 5.2 details the core content of the search instance response object. The first step of the Document Search Algorithm comprises a search for all available SOURCES of AIRSKB (Lines 5 - 8). Thereafter, a loop across all available SOURCES aside from the Suggest Cluster SOURCE is executed (Line 8). The next step comprises a search query expansion with context-dependencies and SOURCE information to match the right IRS index documents that belong to the current source (Lines 10 - 11). The search for documents of current source itself is performed in Line 14. The *SearchResult* object contains all information retrieved from the system, as a a ranked document result list that matches the query and search metadata. If the current source is not the Suggest source, the search result for each source is placed into the search instance (Line 19). By contrast, if the current source actually is the Suggest source, the Suggest Cluster Algorithm is performed (Lines 21-35), whereby a new search query for a unique Suggest Cluster document is built. Therefore, the query is adapted to match only a Suggest Cluster document (Lines 23 - 24). For example, function BUILDSUGGESTCLUSTERQUERY() (Line 23) is a call for lower-level processing steps of IRS Component and the implementation depends on the underlying retrieval system wrapper (IRS Wrapper Layer, see Section 7.2). For instance, in Apache Lucene, this can be carried out by a query expansion with a conjunction of search terms. Therefore, previously-found Suggest documents serve as search terms: $Query \leftarrow SuggestDocument_1\ AND\ SuggestDocument_2\ AND\ \ldots\ AND\ SuggestDocument_n$. After the query has been adapted, a new search for the Suggest Cluster document is performed (Lines 26). If such a Suggest Cluster document exists, the Ontology Component is called to obtain all DOCUMENTS that are direct related to the Suggest Cluster DOCUMENT found (Line 30). The function FINDRELDOCS() of Ontology Component is a high-level access to the AIRKB through use of the corresponding Functional Layer functions of AIRSKB Framework. Depending on which knowledge framework is used for the AIRSKB storage, Storage Layer of AIRSKB Framework translates the high-level request for related DOCUMENTS in the appropriate knowledge storage query language. The Ontology Component submits a list of DOCUMENTS based upon context-sensitive relations of AIRSKB. In detail, this triggers a search in the AIRSKB Framework for all context-sensitive relations $\text{rel}\,(o_p, o_q)_{A_{C+}}$, where o_p is the Suggest Cluster DOCUMENT and o_q is the related DOCUMENT. These related DOCUMENTS and weights of the relationships are used to build a search result later. Previously, the contents of the DOCUMENTS are delivered by the IRS Component (Line 33). The search result (Line 35) is subsequently passed to the application, which has called the SEARCH() function (Line 39).

Algorithm 2 depicts the search for DOCUMENTS of a given SOURCE that are related to an also given DOCUMENT. Therefore, the algorithm requests a collection of context-sensitive pathways of the given DOCUMENT to DOCUMENTS of the given SOURCE. Ontology Component provides the high-level access function FINDPATHWAYS() for this task (Line 6). This function calls AIRSKB Framework that again contains algorithms to calculate best context-sensitive pathways $\text{pathway}\,(o_1, s)_{A_{C+}\text{best}}$ from DOCUMENT o_1 to DOCUMENTS of SOURCE s (see Section 3.3). The algorithms of AIRSKB Framework

operate as described in Figure 3.5:

1. Search for all Sources contained in AIRSKB. As stated above, tasks that interact with AIRKB's current knowledge storage require implementing access for the query language used by the knowledge storage. As described in Section 7.3, a wrapper architecture is used to grantees this. For example: SPARQL is used in an Apache Jena Wrapper to retrieve the Sources. The SPARQL query subsequently looks like "SELECT ?source WHERE { ?source type:Source 'Source' }".

2. Locate the Source of the given Document and search for the best pathways at Source level to the target Source to obtain a clue about possible existing Document pathways. This is necessary because it reduces the steps of finding Document pathways enormously: the algorithm follows only Document relationships that have a corresponding Source relationship and where the Source relationship is part of a previously found Source pathway. Finding existing Source pathways can actually be calculated very quickly, because usually only a few different Sources exist. AIRSKB also contains far fewer Sources than Documents. Therefore, ordinary graph search algorithms like breadth-first search or depth-first search can be implemented. In the case of AIRS Prototype, an algorithm based upon the breadth-first search algorithm was used to find pathways between two given Sources.

3. The next step comprises finding best Document pathways from the given Document to Documents of the also given Source. As written in the previous step, the algorithm follows only those Document relationships whose Sources also have a corresponding relationship. This makes the algorithm of finding pathways cheaper because in the case of using an algorithm based upon breadth-first search, only those Documents of the same depth that belong to the "right" Source are investigated in the next step. Another possibility to reduce the number of Documents for the next processing step is to follow only Document relationships of a defined minimum weight. Additionally, one can also follow only those Documents that are connected to each other by a context-sensitive relationship $\text{rel}\,(o_p, o_q)_{A_{C+}}$ (see Section 3.3) to reduce the Document candidates for the next step. Single Documents do not have many relationships with other Documents. Therefore, this can be undertaken in the next processing step after the pathways have been found. AIRS Prototype contains an algorithm that bases on breadth-first search in combination of Document candidate reduction using valid Source pathways.

4. Once Document pathways have been found, the last step comprises two tasks: reduction of invalid pathways that does not match the context-sensitive pathway definition as described in Section 3.3 (if that has not already been done in the previous step) and ranking of pathways by their weights. Different functions can be used to calculate the weights of Document pathways. For example, Section 3.3 suggests four different functions: $\text{weight}\Big(\text{pathway}\,(o_1, o_n)\Big)_{\text{baseline}}$, $\text{weight}\Big(\text{pathway}\,(o_1, o_n)\Big)_{\text{arithmetic mean}}$, $\text{weight}\Big(\text{pathway}\,(o_1, o_n)\Big)_{\text{geometric mean}}$ and $\text{weight}\Big(\text{pathway}\,(o_1, o_n)\Big)_{\text{harmonic mean}}$. All four functions have been implemented in the Function Layer of AIRSKB Framework. The function to use can be selected via properties file.

Additionally, not all found pathways are returned: if too many existing pathways have been found, only the best ones are returned (the maximum number of pathways to return can be defined via properties file). All other steps are processed in the Ontology Component. By looking at Algorithm 2, a loop across all found pathways is processed (Line 7) and a search result object is built in the loop that serves as container for result documents (Line 8). After the target DOCUMENT is extracted from the pathway (Line 10), IRS Component is called to retrieve document content information from AIRS index (Line 12). The document and the weight of the pathway are added to the built search result object (Lines 13 - 18). The weight is later used in the client application for result ranking. The current loop ends after the search result object is added to the search instance object (Line 19). Finally, the search instance object is submitted to the client application if all pathways have been processed the same way.

Algorithm 2 Find DOCUMENTS of a given SOURCE that are related to an also given DOCUMENT.

```
1:  ▷ The high-level function of AIRS' Core Component takes a DOCUMENT as well as a
    SOURCE
2:  ▷ and returns a search instance object containing a ranked document result list.
3:  function findRelatedDocumentsOfSourceX(document, source)
4:      ▷ Call Ontology Component to get is a ranked list of context-sensitive pathways
5:      ▷ from the given document to documents of the given source.
6:      Pathways ← ontology_component.findPathways(document, source)
7:      for all Pathway of Pathways do                    ▷ For each found pathway.
8:          new object SearchResult                       ▷ Search result container.
9:          ▷ Get target DOCUMENT of current pathway
10:         DOCUMENT ← Pathway.getTargetDocument(void)
11:         ▷ Call IRS Component to content of DOCUMENT.
12:         document ← irs_component.getDocument(DOCUMENT)
13:         ▷ Place result document in search result.
14:         SearchResult ← addDocument(DOCUMENT)
15:         ▷ Use pathway weight for result document ranking.
16:         weight ← Pathway.getWeight(void)
17:         SearchResult ← addWeight(weight)
18:         ▷ Add search result to search instance object
19:         search_instance ← addSearchResult(SearchResult)
20:     end for
21:     return search_instance                  ▷ Contains a set of ranked result documents.
22: end function
```

On the other hand, Algorithm 3 shows a function that updates AIRSKB relationships through using feedback information. This can happen immediately after the user has finished the business case or after a defined time span. If the function is not called right after the business case, the feedback must be stored in the Statistics DB of the Statistics Component. Once the feedback data has been stored, it can be processed later. In both cases, the AIRS' Core Component calls the function processCartFeedback() that consumes a list of documents that were somehow included in the case by the user (see Example 2). The idea behind the algorithm is as follows: the documents that have been

used in the same business case are somehow related to each other because they were used in the same context. These relationships between the documents represent knowledge that is modeled in AIRSKB.

Algorithm 3 Update AIRSKB relationships using system feedback.

```
▷ The function processes feedback after the end of a use case, it consumes a list of
▷ documents that were used somehow in the workshop case.
function processCartFeedback(documents)
    ▷ Elements that buffer AIRSKB relations that needs to be updated.
    new object NewRelations
    new object RelationstoIncreaseWeight
    new object RelationstoDecreaseWeight
    ▷ For each document that that played a role in a use-case:
    for all documentFrom of documents do
        ▷ Call Ontology Component to get a list of all context-sensitive relations of the
        ▷ given document.
        Relations ← ontology_component.getRelations(documentFrom)
        ▷ First step: check if new relations need to be established in the AIRSKB.
        for all documentTo of documents \ {documentFrom} do
            if (documentFrom, documentTo) ∉ Relations then
                ▷ A new relation needs to be established in the AIRSKB
                new object Relation
                Relation ← addDocuments(documentFrom, documentTo)
                ▷ Place new relation in the buffer element.
                if NewRelations not contains Relation then
                    NewRelations ← add(Relation)
                end if
            end if
        end for
        ▷ Second step: check if weights of existing relations need to be updated.
        ▷ Idea: inspect EACH relation of the current feedback document whether
        ▷ another feedback document is involved. If so, then increase the relation's
        ▷ weight, decrease it otherwise.
        for all Relation of Relations do
            used ← false
            for all documentTo of documents \ {documentFrom} do
                if documentTo part-of Relation then
                    used ← true
                    break
                end if
            end for
            if used = true then                    ▷ Other feedback document is involved.
                if RelationstoIncreaseWeight not contains Relation then
                    RelationstoIncreaseWeight ← add(Relation)
                end if
```

Algorithm 3 Update AIRSKB relationships using system feedback – Part Two.

```
41:             else                                  ▷ Other feedback document is NOT involved.
42:                 if RelationstoDecreaseWeight not contains Relation then
43:                     RelationstoDecreaseWeight ← ADD(Relation)
44:                 end if
45:             end if
46:         end for
47:     end for
48:     ▷ Third step: update relations of AIRSKB.
49:     ▷ Call Ontology Component to add new relations of AIRSKB
50:     ontology_component.ADDRELATIONS(NewRelations)
51:     ▷ Call Ontology Component to update relations of AIRSKB
52:     for all Relation of RelationstoIncreaseWeight do
53:         Relation ← INCREASEWEIGHT(void)
54:         ontology_component.UPDATERELATION(Relation)
55:     end for
56:     for all Relation of RelationstoDecreaseWeight do
57:         Relation ← DECREASEWEIGHT(void)
58:         ontology_component.UPDATERELATION(Relation)
59:     end for
60: end function
```

Therefore, the algorithm checks whether all of the given documents are related to each other. If two documents are used together in the same business case, the strength of the relationship between them grows. By contrast, if only one document of two related documents are used in the same business case, the strength of the relationship between them shrinks. The algorithm now operates as follows: temporary lists of document relationships are built to buffer those that need to be established or updated in AIRSKB (Lines 5 - 7). Subsequently, a loop across all feedback documents is performed (Line 9). In the loop, the Core Component is called to derive all of the relationships of the current document (Line 12). These relationships are processed in later steps of the algorithm. First, the algorithm investigates the document set if new relations need to be established in the AIRSKB. For this purpose, the Core Component checks whether the current document already has a relationship with all other feedback documents represented in the AIRSKB (Lines 14 - 25). For relationships that are not already represented through the AIRSKB, a new one is built and placed in the buffer (Lines 17 - 22). Second, the relationships of the current document are investigated. Idea: inspect each relation of the current feedback document whether another feedback document is involved (Lines 29 - 47). This means that both documents have been used in the business case. If yes, the relation's weight is increased, otherwise it is reduced. The relations are stored in the corresponding buffer (Lines 37 - 45). Third, the relationships that are stored in the buffers are synchronized with those represented in the AIRSKB (Lines 50 - 59). New relations are placed in the AIRSKB (Line 50), used relationships are updated with increased weights (Lines 53 - 54) and not used relationships are updated with reduced weights (Lines 57 - 58). Weights are increased by a certain percentage (up to a maximum of 1) and are also reduced by a certain percentage (up to a minimum of 0.1, see Section 3.3). The following equations are performed

in the function INCREASEWEIGHT() (Line 53) and DECREASEWEIGHT() (Line 57) of the Core Component.

$$weight \leftarrow weight + \frac{weight}{100} * increase\_in\_percent \tag{7.1}$$

$$weight \leftarrow weight - \frac{weight}{100} * decrease\_in\_percent \tag{7.2}$$

The parameters *increase_in_percent* and *decrease_in_percent* can be set by a properties file.

Algorithm 4 presents the function that is used in the AIRS Prototype to update an existing or establish a new Suggest Cluster document in the AIRSKB.

Algorithm 4 Build or update SUGGEST CLUSTER document from system feedback.

```
1: ▷ The function consumes a feedback object (contains meta data like the user's search
2: ▷ query) and a set of feedback documents used in a case. The function uses this
3: ▷ information in order to update or build new Suggest Cluster documents.
4: function processCartFeedback(feedback, documents)
5:     ▷ First step: find Suggest documents from AIRS
6:     ▷ index that match the user's query.
7:     Query ← getUserQueryFromFeedback(feedback)
8:     Query ← addSourceInformationToQuery(Suggest Source)
9:     SearchResult ← irs_component.search(Query)
10:    ▷ Second step: find Suggest Cluster document using found Suggest documents.
11:    Query ← irs_component.buildSuggestClusterQuery(SearchResult)
12:    Query ← addSourceInformationToQuery(Suggest Cluster Source)
13:    ▷ Call IRS Component to find Suggest Cluster document (if exists).
14:    Suggest Cluster document ← irs_component.search(Query)
15:    if document is null then                    ▷ No Suggest Cluster exists, build new one.
16:        new object Suggest Cluster document
17:        ▷ Add all found Suggest documents to Suggest Cluster document.
18:        Suggest Cluster document ← addSuggestDocuments(SearchResult)
19:        ▷ add new Suggest Cluster document to AIRS index and AIRSKB
20:        irs_component.addDocument(Suggest Cluster document)
21:        ontology_component.addDocument(Suggest Cluster document)
22:    end if
23:    ▷ Third step: find context-sensitive relations of Suggest Cluster document
24:    Relations ← ontology_component.getRelations(Suggest Cluster document)
25:    ▷ Forth step: Increase or decrease context-sensitive relations weights
26:    for all Relation of Relations do
27:        used ← false
28:        for all documentTo of documents do
29:            if documentTo part-of Relation then
30:                used ← true
31:                break
32:            end if
33:        end for
```

Algorithm 4 Build or update SUGGEST CLUSTER document from system feedback – Part Two.

```
34:         if used = true then
35:             Relation ← INCREASEWEIGHT(void)
36:             ontology_component.UPDATERELATION(Relation)
37:         else
38:             Relation ← DECREASEWEIGHT(void)
39:             ontology_component.UPDATERELATION(Relation)
40:         end if
41:     end for
42:     ▷ Fifth step: Check for every feedback document if a relation to
43:     ▷ the Suggest Cluster
44:     ▷ document exists, build new one if necessary.
45:     for all documentTo of documents do
46:         used ← false
47:         for all Relation of Relations do
48:             if documentTo part-of Relation then
49:                 used ← true
50:                 break
51:             end if
52:         end for
53:         if used = false then                    ▷ Add new context-sensitive relation
54:             new object NewRelation
55:             ▷ Add Suggest Cluster document to feedback document relationship
56:             NewRelation ← ADDDOCUMENTS(document, documentTo)
57:             ontology_component.ADDRELATION(Relation)
58:         end if
59:     end for
60: end function
```

The function PROCESSCARTFEEDBACK() consumes the same set of feedback documents like the function of Algorithm 3. Additionally, the function consumes a feedback object that comprises metadata like the user's query. In general, the algorithm bases on the approach described in Section 5.5. Like the function of Algorithm 3, the function call can happen immediately after the user has finished the business case or after a defined time span. The first part of the algorithm comprises the search for a Suggest documents that matches the user's query (Lines 7 - 9). For this approach, a query is built through use of the user's original query (Line 7). The search for the Suggest Cluster document that matches the previously-found Suggest documents is performed in the next step of the function (Lines 11 - 14). If the Suggest Cluster document does not exist (Line 15), a new one is built and the Suggest documents are added to it (Line 18). The newly-built Suggest Cluster document is subsequently added to both the AIRSKB (Line 21) and the AIRS index (Line 20). Part three of the algorithm checks whether the Suggest Clusters documents already have relationships with the feedback documents (Lines 26 - 41). The idea behind this is that if the Suggest Cluster document has relationships with DOCUMENTS represented in AIRSKB and the given feedback documents are part of these DOCUMENTS, the weights of

the relationships with the given feedback documents are increased. If the Suggest Cluster document still exists, all context-sensitive relationships of the Suggest Cluster document are investigated in terms of whether the feedback documents are part of the relationships. In the case they are, the weights of the corresponding relationships are increased (Lines 35 - 36), whereas otherwise they are reduced (Lines 38 - 39). Of course, no relationships exist in the case of a new Suggest Cluster document was built. Accordingly, new relationships between the feedback documents and the Suggest Cluster document are established in AIRSKB (Lines 45 - 60).

7.6 Implementation Strategy and Prototype Features

The cornerstone of the AIRS Prototype implementation phase covered a few prototypes that were created during early research. The so-called functional prototypes showed some skills of state-of-the-art technology in combination with a small part of the requirements catalog: a modern retrieval component across one document system or an application to reflect the document landscape as an ontological model.

Later, the AIRS Prototype grew by using these functional prototypes. The research project was divided in four different phases:

1. Describing the theoretical background and clarifying the research focus in detail were the main activities of the first phase. Accordingly, the concept of AIRS and the ontology engineering process was designed. In [93], insights and results of this research phase are given.

2. Phase two started with a case study where part of the domain was investigated to disclose the document linkage potential (see [95]). Technologies for the Ontology Component as well as for the IRS Component have been evaluated. Beside these activities, AIRS Prototype architecture design and first implementation activities were started. Additionally, students face in their theses the opportunity of harvesting document sources and ontology-representation technology evaluation regarding AIRSKB's design patterns. The rest of this phase focuses on the implementation of the AIRS Prototype, especially the AIRS Includes Sources Framework, the AIRS Index & Search Framework and the AIRSKB Framework.

3. In the third phase, the development of AIRS shifted to retrieval optimization based upon an adaptive knowledge network. Therefore, the feedback-processing chain was implemented in the AIRS Prototype system and an evaluation process for information retrieval system development was developed (see [94]). Additionally, advanced retrieval technologies have been developed that combine both a state-of-the-art information search with feedback processing (see [96] and Section 5.2). The AIRS Prototype implementation has also been finished in this phase. The last activities of this step comprised internal evaluation steps.

4. The last phase of system development comprises evaluation steps that were performed in an external evaluation system test as presented in Chapter 8.

The AIRS Prototype is a Java web application that uses information from the IRS index and AIRSKB within the help of the AIRS search index and AIRSKB Framework module.

The Ontology Component and the IRS Component are implemented as wrappers across the AIRSKB Framework and across the AIRS Index Search Framework (in the AIRS Prototype backend component).

The AIRS Prototype architecture is based upon the design presented in Section 4.3. The only differences are that foreign data interfaces and advanced services do not exist[12]. As stated before, AIRS comes as a Java servlet web application. This means that AIRS comprises a backend as well as of a frontend implementation. The backend implementation uses the AIRS framework components.

Figure 7.5: The search result documents are grouped by the sources where they originally appear. The figure shows the search results for the query "Bremsscheibe AND KG:42" grouped by their sources. For example, replacement parts information of the electronic parts catalog is shown on the left (In German "Teile-Information-Suche")

The AIRS Prototype can be installed on a typical server by using the listed sub components: Apache Tomcat servlet container to host the AIRS Prototype application, Apache webserver to host construction paintings as well as advanced static AIRS dependent web services, MySQL database server to host the statistics components, MongoDB database server for the AIRSKB (as long as MongoDB is used as AIRSKB's storage backend) and

[12]The main focus of AIRS Prototype application was to perform end-user tests within a simulation environment. Therefore, interfaces for foreign date services were out of the scope since they were not necessary for the tests.

Apache Solr embedded index to host the IRS index for document retrieval (as long as Apache Solr is used as underlying information retrieval system). Because the AIRS Prototype is implemented in Java and many of the AIRS foreign software components are operating-system independent, the entire AIRS Prototype can be used in any operating-system environment the foreign software components and a Java VM exist for.

Figure 7.6: Faceted search: a user can filter a document result list through use of a drop-down menu that provides source dependent sub categories.

The AIRS Prototype was tested on both the Windows environment and the Linux environment. Like for the server installation, the AIRS Prototype can also be installed on a virtual machine that can be used as a stand-alone application on a single machine for testing purposes. Furthermore, product presentations can be easily undertaken with a portable single-machine AIRS Prototype installation. This is a necessary requirement because AIRS was a research project with the goal of presenting the current status and the possibility of new technologies. In this kind of installation, the same components used for the normal server installation were used. The open source virtualization solution Oracle Virtualbox and an Ubuntu 10.02 LTS operation system were used to host the AIRS application. All other necessary foreign software solutions listed above were installed in Ubuntu. Subsequently, the AIRS Prototype components were deployed on the system (the AIRS index and AIRSKB). Finally, all of the AIRS Prototype dependent software components were configured to start up at system booting time. The result is a ready-to-use AIRS

Prototype environment. Due to the local network capabilities of the Oracle Virtualbox, the AIRS Prototype application can be used after the system boot in two of several ways: in the virtual machine environment by using a standard Internet browser that is installed in the Ubuntu operating system and by using the host system's browser. Accordingly, the AIRS Prototype application can be used in a simple way: the first step is to boot the virtual machine (and keep it in the background) and the next step is the use of a standard Internet browser to run the AIRS Prototype application.

Figure 7.7: A selected search result document.

AIRS' reference implementation offers state-of-the-art retrieval features to the user. Examples of this are the features spelling correction, providing search term suggestions and highlighting of special search characters. These features are not present in the current workshop documents systems. The result documents are grouped by the sources where they originally appear. In the AIRS Prototype, this is implemented through use of result container as shown in Figure 7.5. As long as each source provides document restriction and a category system, the AIRS Prototype provides a faceted search as shown in Figure 7.6. Therefore, the search result can be filtered along special AIRS index fields. A user can select a sub category by selecting an entry of the dynamically built drop-down menu. A detailed search result view opens by clicking on a document from the result set. The detailed document view is shown in a separate window with a sidebar where the user can see related documents or search for related documents information (see Figure 7.7).

Figure 7.8 shows the cart component of the AIRS Prototype that is used to simulate the handing-over point of workshop documents that are used in the workshop processes. AIRS Prototype is a system that was built for the retrieval algorithm evaluation of AIRS in the German after-sales market. Therefore, the AIRS Prototype contains only documents in German language. The outcome is that the AIRS Prototype functions like spelling correction and document search works also only for German languages. In future research, this approach should be extended for more languages (see Section 9.2).

Figure 7.8: A user can add search result documents to a cart, which is used to simulate the handing-over point of document search to the downstream processes.

8 Field Tests and Evaluation

A successful system evaluation requires field tests performed by end users. In best cases, these end users are domain experts. This chapter starts with an explanation of documents and systems that are used in processes of car workshops. Later, this chapter explains how the AIRS Prototype's user interface supports the business cases in the workshops. Furthermore, the experimental setup of the field test is explained and an overview is provided concerning the field test results.

8.1 Automotive Workshop Processes

One of the main tasks of after sales departments of car manufacturers is the support of workshops in maintenance, service and diagnosis of cars (see [96]). Therefore, they provide software products and process solutions for various workshop processes. These are either used in the reception process (customers bring their cars for maintenance or service requests) or during the mechanics' work with the customers' cars. Examples of corresponding software products are (see [96]):

- diagnosis data systems in which error code-specific and symptom-based checks are managed for vehicle diagnosis processes,
- workshop information systems in which workshop literature is managed to support vehicle maintenance, repairs and diagnosis,
- workshop help-systems in which documents are managed that provide information about current remedial measures for technical complaints,
- taxonomies for the standardized recording of symptom locations and symptoms in vehicles used by customers,
- electronic catalogs for replacement parts, and
- systems that contain information about work units and flat rates.

A customer's maintenance, service or repair request is one case that the workshop employee must address. Since modern cars have become more complex, the diagnosis and repair of modern cars is also becoming increasingly complex. This makes the service task more complicated and workshop employees need support in solving it.

Part of the support comprises the information stored in the document systems. From the workshop employees' perspective, documents from various retrieval systems help them to solve the current task. All these documents are structured and attributed in different ways. For example, the segments and attributes of a document that contain replacement parts differ from a document that contains information about a repair instruction. The outcome

is a heterogeneous document landscape where different kinds of documents are accessible through different retrieval systems. Unfortunately, these retrieval systems are restricted to the type of documents that they contain. In general, links between documents of different information sources are not in the scope of these systems. Links between documents of different retrieval systems must be established manually in authoring processes that can be achieved through link-out and link-in services between the retrieval systems[1]. Such links between documents exist, albeit not between all documents that they need to. Because this is a complicated step in the up-stream data processes, it raises the possibility of errors and causes additional costs for the departments.

The hypothesis is that the workshop processes can be significantly improved if:

- workshop employees need to use only one retrieval system that includes all documents, independently of their source.
- links between documents from different information sources can be established automatically.
- links between documents are dynamic and not only static. They should be relationships with variable weights.
- relationships between the documents can be used to gain or update existing knowledge about the relevance of document relationships.

Therefore, the challenge is to break down the information barrier of the heterogeneous document landscape and use the possibilities of an adaptive document knowledge network. An overall retrieval system should help to reduce costs as well as optimize the information access that is necessary for the workshop processes. Therefore, the domain of workshop processes served as frame for the AIRS Prototype user tests.

8.2 AIRS Prototype User Interface

A selection of data from different workshop systems of a car manufacturer was included in the AIRS data storage via the Indexing Workflow of AIRS (see Section 5.1). Together with the indexed data of the workshop systems, different search functions of the AIRS Prototype can be used to simulate workshop process flow. Such simulations were used for the AIRS Prototype tests. Using the AIRS Prototype application is as simple as the usage of a state-of-the-art free text search engine: a search bar is given to the user where he can input a free text query. Figure 8.1 shows the AIRS Prototype application. One can see the results of a search for “Bremsscheibe” (German for “brake disk”). A box with search results is shown for each document location where the documents came from. Area A of Figure 8.1 marks the search result box for documents from the electronic parts catalog. When a user now selects a search result by clicking on it, the document is presented to the user in a separate window as shown in Figure 8.2. In this example, Area A marks the result document representation and Area B shows the sidebar containing related information.

[1]For this work, a link between documents is a computer-accessible connection between these documents. However, a link also indicates a kind of a semantic relationship between the documents based upon their content (see Section 4.2).

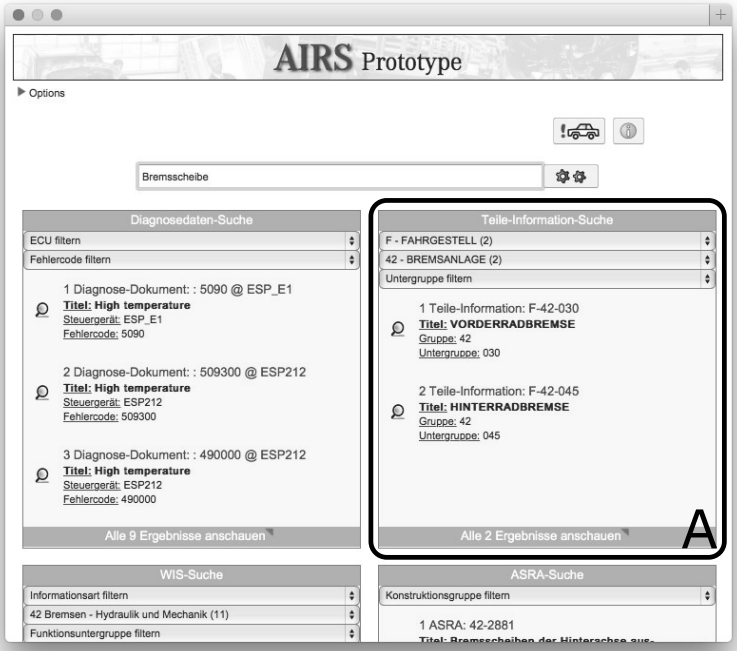


Figure 8.1: AIRS Prototype: search for "Bremsscheibe" (German for "brake disk"). Area A depicts a source-dependent search result box.

The related documents presented in the sidebar are grouped according to their sources and ranked by using the relevance weights. Figure 8.2 Area B, for example, shows two documents of the source wis that are related to the given result document "Hinterachse". As long as the source wis contains part replacement instructions, the relationship between the parts information "Hinterachse" and the related documents means that there is a description in the instruction about how the part has to be replaced. This document relationship weight is dynamically adapted to its current usage during the workshop workflow by the workshop employees.

As stated in Section 3.3, AIRSKB *knows* two different kinds of relationships beside system-generated relationships: static document relationships (is-linked-to) and adaptive document relationships (is-related-to). Those that have a weight are adaptive relationships and those that have a static weight are static relationships. Static document relationships are grouped in the sidebar under the "punctuation mark" category. A user can add a single document result in the cart that is provided by AIRS Prototype (Figure 7.8) by clicking the cart symbol in the right corner below the sidebar (Figure 8.2, Area B).

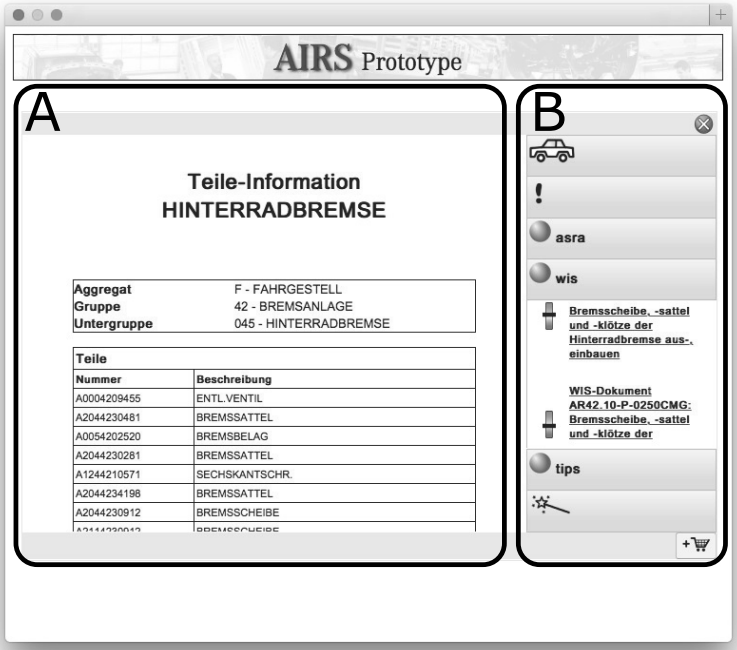


Figure 8.2: AIRS Prototype showing parts information "HINTERRADBREMSE" (German for "rear wheel brakes"). Area A marks the search result document and Area B shows related documents (ranked by the relationship weight) from different sources in a sidebar.

As explained in Section 7.6, AIRS Prototype provides advanced free text search features to the user. Figure 8.3 shows such a feature for searching only in the symptom taxonomy source ("source: symptom") to find information about standardized car fault describing symptoms. This is a helpful feature, especially for the car reception process in which the receptionist opens a new case in the workshop system. Therefore, the user needs to map the customer's complaint to these standardized symptoms. In Figure 8.3, a search for "Vorderachse klappert" (German for "Front axle rattles") is shown. This complaint is subsequently mapped to the standardized symptom taxonomy and all nodes are shown that match this mapping (search string).

If a user ascertains that "knocking" rather than "rattles" is the right complaint description, the user can select the leaf of the taxonomy (which actually represents a result document). This opens the result container as shown in Figure 8.5. The result representation of the symptom node "knocking" shows extended information like the node hierarchy, the node's validity state and the taxonomy version. Important is this result representation especially for advanced retrieval capability of AIRS Prototype because the symptom node

itself can be a starting point for a new retrieval step. This new retrieval based upon the AIRS' knowledge network components is available to the user through use of the sidebar. On the one hand, the user can select documents from the sidebar that are related to the given symptom node and on the other, the user can search actively for related documents. This feature lies behind the magic wand symbol. As shown in Figure 8.4, the user can select a target source to find documents that are somehow related to the given symptom. The only constraint is that the related documents must be from the selected target source.

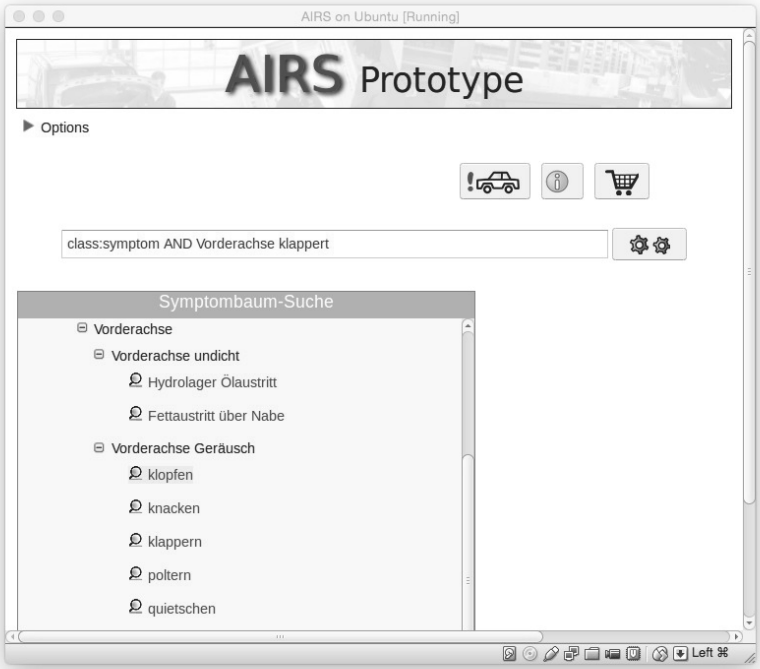


Figure 8.3: A user can search for symptoms that appear in a customer's car. Shown is a search in the symptom taxonomy "class: symptom" for "Vorderachse klappert" (German for "Front axle rattles"). The result container subsequently shows a symptom tree with document results as leaves.

The search for related documents is implemented in the AIRS Prototype as explained in Algorithm 2 of Section 7.5. This feature is bases on AIRSKB methods that are presented in Section 3.2 and Section 3.3: finding the best contest-sensitive pathways between a document to documents of an also given source. Figure 8.5 shows the result of such a source. After selecting the target source, the AIRS Prototype searches in AIRSKB parts

information for documents that are somehow related to a given symptom. The test users can use all features of AIRS Prototype that are presented in this section.

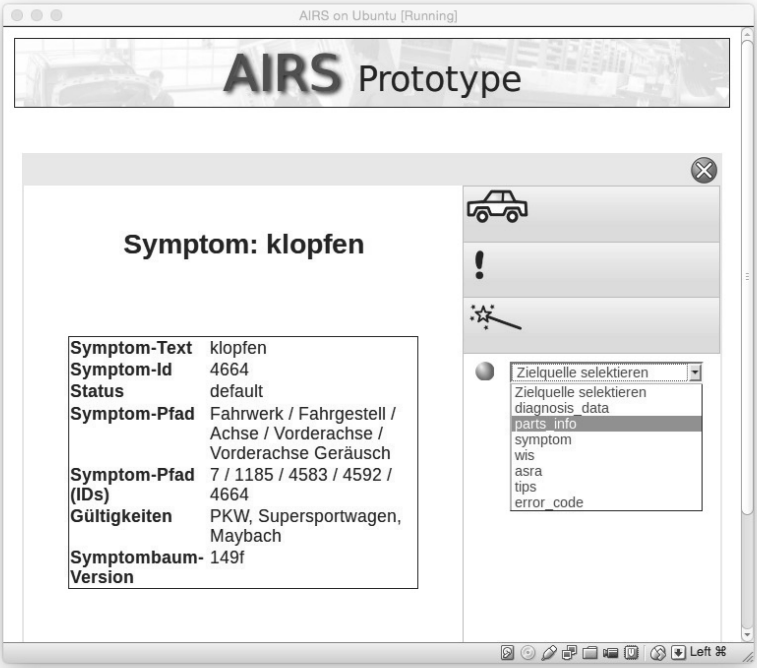


Figure 8.4: After selecting the leaf node "klopfen" (German for "knocking") from the symptom taxonomy tree, the user can start a search for related documents through use of the sidebar.

In Section 5.4, the Suggest Cluster Algorithm was named as an example of extended retrieval that can be established by using the AIRS infrastructure. A user simply needs to match a Suggest Cluster with a query and all related documents are shown to him that other users include in the same kind of business case. Figure 8.6 now shows how the result of the Suggest Cluster Algorithm is presented to the user. In the figure, Area A frames the result box where ranked documents are listed that belong to the Suggest Cluster that matches the user's query "Bremsscheibe wechseln" (German for "replace brake disks").

Figure 8.5: The related documents (if they exist) are presented in the sidebar.

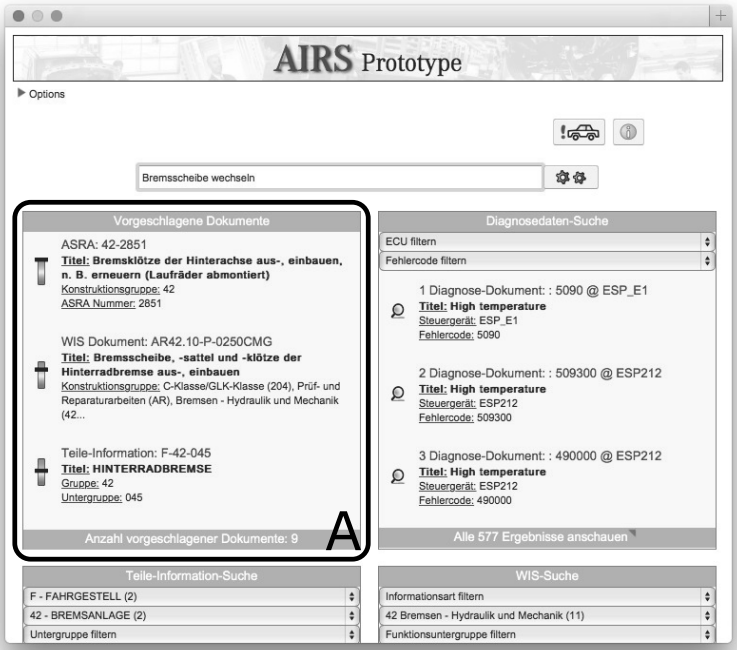


Figure 8.6: AIRS Prototype showing suggested documents for the search string "Bremsscheibe wechseln" (German for "replace brake disks") that match the corresponding Suggest Cluster. Area A shows the suggested documents presented in a special result container.

8.3 Experimental Setup of AIRS Prototype Field Tests

As described in Section 6, AIRSKB was designed to include collective intelligence of employees into the retrieval process. With the aid of the automatic inclusion of new document relationships into the AIRS retrieval system, the AIRS Prototype is able to collect the knowledge of workshop employees and combine it with the knowledge of workshop authors. A document relationship network is the resulting data structure. Since AIRSKB is not a static knowledge structure, it changes slowly due to the ongoing evolutionary process. By only including new relationships with AIRSKB and even by doing so carefully (adding new relations if at least a critical number of workshop employees used them), the number of relations in AIRSKB will grow uncontrollably. The resulting AIRSKB is simply a super-set of knowledge from many automotive experts and not a network that includes the collective intelligence of users. Evolution also means survival of the fittest. AIRS Prototype's algorithms meet this challenge by including a concept of aging in AIRSKB and use a ranking mechanism for the document relationships. Relation aging has a direct influence on the relation's weights. AIRS Prototype adapts this principle and updates the document relationships at the end of each business case. Given that the AIRS Prototype application simulates the workshop processes of a document search, the document handing-over point marks the end of the business case. In detail, this means that submitting AIRS Prototype's cart causes an update process of AIRSKB. The update principles are:

- less usage of a relation over time decreases the relation's weight,
- usage of a document without including its related documents to the business case solution decreases the weights of the unused document relations and
- frequent usage of a relation increases its weight.

A still open question about ranking is how to deal with lower ranked related documents. One can cut the related document list according to a too low relation weight. However, even lower weighted relations could be important to less frequent business cases. For the field tests, lower ranked document relations were included in the document representation to examine whether these relations have an impact on the retrieval process. Additionally, upper and lower borders for relation weights were defined for AIRSKB relations. By following this, the lowest possible relation weight is 0.1 and the highest possible relation weight is 1.0. This means if a relationship between two documents is ever established, the relationship between the documents will still exist even if it is very weak (lower border) and a relationship between documents cannot be more "relevant" to the user than 100 percent (upper border).

One of the field tests' goals was to monitor the evolutionary process of AIRSKB's document relationships through the AIRS prototype field test. Accordingly, a decision was made to prepare well-defined test cases to obtain comparable results. In each test case, a task was described that can also happen in real life. One of the test cases was: *The brake disk of a rear wheel brake must be replaced. Find all documents you need for the business case.* These tasks were defined together with domain-specific experts.

Furthermore, two scenarios should be compared in the field tests: the search for related documents using the existing isolated retrieval systems and the same task using the AIRS Prototype. Therefore, surveys were conducted about the search using the isolated retrieval systems and the AIRS Prototype search.

8.4 Performing Field Tests Using the AIRS Prototype

Field tests are difficult to perform because they depend on subjective user opinions about the system that is tested. In [14], Davis postulated the technology acceptance model (TAM) for this approach and extended it later to TAM2 (together with Venkatesh, see [89]). According to the model, the acceptance and usage of a technology depends largely on the ease of use and added value of technology. Both depend strongly on subjective user opinions. AIRS Prototype field tests were inspired by this approach. Accordingly, "before-after" surveys were performed to derive statements about the system's ease of use. System tests based upon use case scenarios were performed and document relations were observed over time to obtain information about usefulness of the system.

AIRS' search capability was verified by means of a sample application and field tests with employees of workshops. The goal of these user tests was:

- The workshop employees perceive quick and easy search for relevant documents using AIRS Prototype as helpful and easy to use.
- A system based upon the AIRS Prototype helps to optimize the entire workshop process by reducing the time a workshop employee needs to search for business case relevant documents.

The workshop employees now need to search for all of the documents they need for a given case that itself is mostly a maintenance or service request for a given customer car. The workshop retrieval systems contain these documents. They were developed to support the maintenance and service processes that appear in the workshops. The workshop employees need to use these workshop retrieval systems every day. AIRS was developed to support the same kind of workshop employee's work routine. Additionally, it could improve the workshop processes significantly. Therefore, AIRS must be proven in the real workshop situations from the same kind of people who perform the maintenance and services processes day by day.

Because the workshop employees are domain experts for their processes, testing the AIRS Prototype means performing an expert test and obtain highly valid statements concerning how the tested system acts in the workshop domain. Finally, the test must contain a comparison between the current workshop systems and the AIRS Prototype regarding document retrieval. This comparison should involve various aspects like the retrieval quality, the retrieval performance and a statement about how good the tested system satisfies the user's information need. The test must depict maintenance and support cases that could also appear in real life workshop situations. Accordingly, the user tests should involve two main aspects:

- A "before-after" comparison of the workshop retrieval systems with the AIRS Prototype and
- the AIRS Prototype system tests regarding domain-specific test scenarios.

The "before-after" comparison was undertaken by using two surveys. The first survey was performed even before the AIRS Prototype system was tested and the survey contains questions about the current workshop retrieval systems. The second survey was performed

after the AIRS Prototype tests. It contains questions about the usage of the AIRS Prototype. AIRS Prototype tests were performed by using two maintenance scenarios. Both scenarios cause a documents search for different kinds of workshop documents.

A user test session at a workshop was subsequently performed as follows: after making an appointment, the workshop was visited for one day. The first agenda point after a "welcome" session and an explanation of the visit's reason comprises a brief introduction to the topic. Later, the workshop employees who joined the test were asked to answer the questions from the first survey to obtain information about the workshop retrieval systems that the employees are currently using to solve their business tasks. A selection of questions from the first survey that the workshop employees were asked to complete is presented in Appendix A.1. As stated above, the idea behind the first questionnaire was to gain an idea of what the workshop employees think about the current retrieval systems without being influenced by the AIRS Prototype. Therefore, questions like "Which of the workshop systems do you know and use for your daily work? Please mark the boxes how often you use the system.", "How quick do you find the information you want?" or "You talk to the customer and you want to search for a workshop help document that matches the problem the customer has with the car. How long does it take to search for the document? Please select." were asked. Another aspect of the first survey was to ascertain something about the workshop employees' careers. These questions have been used to appraise and classify the workshop employee's answers about the workshop retrieval systems. Examples for these kinds of questions were "How long have you been working in the workshop?" or "In which workshop sector are you working? Please choose between *service reception* and *car service*.". Accordingly, it was possible to obtain more information about the target user groups. In this sense, one can divide the users of the workshop systems into four different groups:

1. A *newbie* is someone who has recently started working in a workshop (still in or recently finished training). The current workshop systems are new to the employee. The user might securely deal with new technologies.

2. An *expert* is someone who knows every aspect of the work and who has a lot of experience in dealing with the workshop systems.

3. A *senior expert* is someone who is very experienced in doing the job. Therefore, the employee has the capability to teach other employees in dealing with the current workshop processes.

4. An *old hand* is a person who has worked in workshops for a very long time. The employee knows everything about workshop processes and learned to work with many generations of the workshop system.

Additionally, the workshop employees were asked to characterize themselves to ascertain which group they match. The question was asked indirectly to obtain the best results for a self-characterization: "What do you think, how would your colleagues characterize your work status?". The target group that produces highly valid answers should be *experts* and *senior experts*. They already know the current generation workshop systems (and their limitations) best. Additionally, they are very adept in dealing with new search technologies they are already know from private web search, for instance. After the workshop employees

completed the first survey, a brief introduction to the topic of AIRS was provided, along with an introduction to the usage of AIRS Prototype. The workshop employees came in touch with AIRS Prototype software during a guided "hands-on" session. The workshop employees subsequently had time to become more familiar with the AIRS Prototype software.

In the next step, a document containing the user tests was given to the employees, who were asked to read it carefully. Appendix A.3 shows a part of this document containing two tests scenarios used for the user tests. The document also explains the functions of AIRS to ensure that the workshop employees can answer open questions in using the system by themselves[2]. Following a brief question and answer session about the user tests (to clarify open questions), the workshop employees were asked to perform the tests by themselves. The first test case is about a typical maintenance request regarding a specific car model: the brake discs of the rear wheel need to be replaced. The objective of the test was to find all of the documents that are necessary to solve the task of brake disc replacement. The second case was about a technical complaint regarding a specific car model: the front window of a given car model is broken and needs to be replaced. The workshop employees were asked again to find all of the necessary documents to solve the given business task. After the workshop employees completed the user tests, they were asked to completed the second survey (Appendix A.2). In this second questionnaire, the users were asked about the document retrieval using the AIRS Prototypes. Similar questions to those in the first questionnaire were asked. For example, the following question was asked: "You talk to the customer and you want to search for workshop help documents that matches the problem the customer has with the car. How long does it take to search for the document by using the AIRS Prototype? Please select.". This is the corresponding question to "You talk to the customer and you want to search for a workshop help document that matches the problem the customer has with the car. How long does it take to search for the document? Please select." from the first survey (see Appendix A.1). This approach allowed conducting a "before-after" comparison. The second survey subsequently concludes with an assessment of the workshop employee regarding the research of AIRS and whether it should be continued.

Another part of the two surveys comprises three case descriptions of maintenance situations that can appear in the workshops. For example, the workshop employees were asked to assess how long it would take to find all necessary documents by using the isolated retrieval systems in the first questionnaire. A similar question was asked in the second questionnaire after the users had completed the tests. This similar question was how long it would take to find all necessary documents by using the AIRS Prototype rather than the isolated retrieval systems. All of the three questions were:

1. You read a technical information document and you want to find a replacement instruction document that is not linked in the technical information document but that is necessary for the case. Please mark how long it would take to find it.

2. You are in a reception situation and a customer brings a car for a maintenance request. You want search for a technical information document that matches the customer's car and the complaint. Please mark how long it would take to find it.

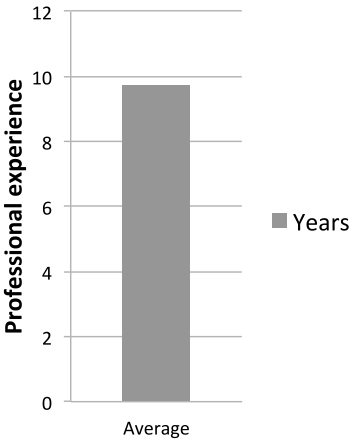
[2]However, it has emerged that this brief description does not contain sufficient information to answer all open questions of the workshop employees during the user tests. Therefore, fundamental questions about the system have been answered during the tests.

3. A vehicle that had an accident is delivered to the workshop. You need to search for all necessary documents that help to repair the car. Please mark how long it would take to find them.

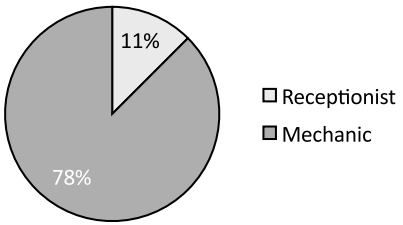
In the first survey the users answered these questions without knowing that they had to answer the questions later in the second survey again regarding the AIRS Prototype system. The user tests concluded when the workshop employees finished the second survey.

8.5 Results of Field Tests

Five workshops were visited during the tests and more than ten workshop employees took part of the user tests (including two after sales experts of headquarter department who have been employed in workshops before with many years of experience in workshop processes). All of them had many years of professional experience in the domain of workshop processes as shown in Figure 8.7 (a).



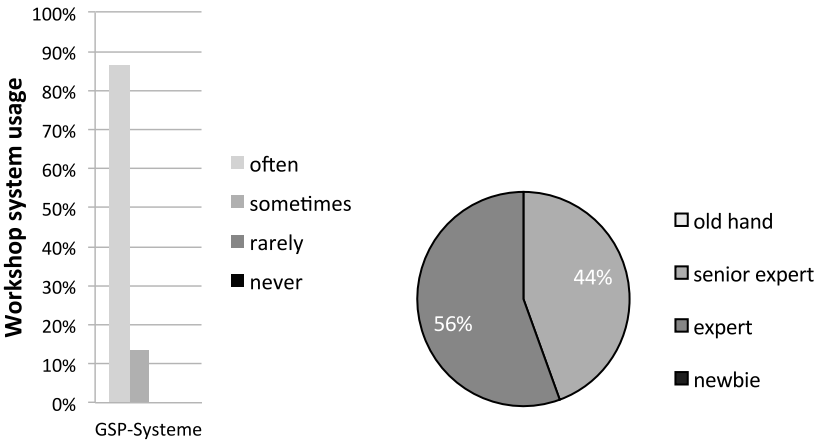
(a) The average of professional experience of the workshop employees



(b) The role of the workshop employees

Figure 8.7: The usage of workshop system during the daily work of the workshop employees who took part of the user tests and the role the workshop employees have in the workshops.

The workshop employees spread across two different groups as depicted in Figure 8.7 (b). Most of them are mechanics who work directly with the customers' cars in the car maintenance, service and repair process. They had only rarely customer contact. By contrast, the smaller group comprises receptionists who are responsible for the car reception process where the customers bring their cars for a service, maintenance or repair requests. The workshop employees were asked how often during their daily work they enter into touch with the workshop systems. Of course, the usage of a workshop system depends on the role of the workshop employee.



(a) How often the workshop employees use the workshop systems during the daily work (the average of named systems).

(b) The mean of the self-characterizations of the workshop employees

Figure 8.8: Professional experience of the workshop employees who took part of the user tests and the mean of self-characterizations of the workshop employees.

In the surveys, the workshop systems were named directly and Figure 8.8 (a) shows the mean of the given answers. The evaluation showed that most of the systems were used very frequently during the daily work. This means that an optimization of the systems should improve the processes in a significant way. Figure 8.8 (b) shows the mean of the self-characterizations the workshop employees were asked for during the first survey.

Table 8.1: Documents used in the *brake disc* replacement task. Documents that describe work units are d_{1asra}, d_{2asra} and d_{3asra}. Document d_{1wis} contains replacement instructions, document d_{1tips} contains repair instructions and document d_{1pi} contains parts information.

	Field Test n	Field Test $n+1$	Field Test $n+2$	Field Test $n+3$
d_{1asra}	+	−	+	−
d_{2asra}	+	+	+	−
d_{3asra}	+	−	+	+
d_{1wis}	−	−	+	+
d_{1tips}	−	−	+	−
d_{1pi}	−	−	+	+

The result is that most of the workshop employees see themselves as an “expert” or a “senior expert”. This correlates with the workshop employees’ years of professional experiences and means that they are domain experts in their work areas and that they will produce highly relevant answers during the field tests. A “newbie”, for example, is currently in the studying and learning phase and does not have many experiences with the current workshop systems. This kind of user group possibly cannot see or identify process

optimization effects due to the missing experience. By contrast, an "old hand" is perhaps not open-minded to innovations because he sees the current systems running and possibly implicates this as the most important system feature.

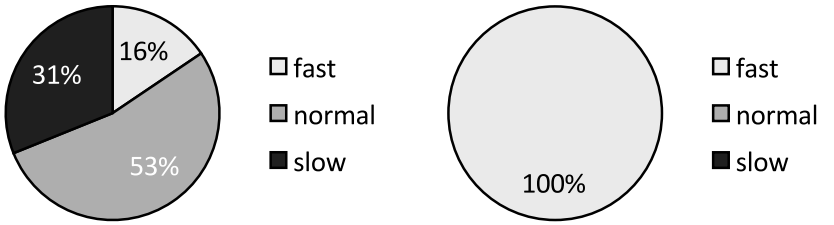
Table 8.2: Related documents of parts information *rear wheel brakes* d_{1pi} and their relation weights over time. The character '+' marks a new relation that has been established after the use case, '↗' marks a relation whose weight has been increased since the last usage, '↘' marks a relation whose weight has been reduced since the last usage and '→' marks a relation whose weight has not changed since the last usage.

	hasWeight($\text{rel}(d_p, d_q)$)$_{t_n}$	hasWeight($\text{rel}(d_p, d_q)$)$_{t_{n+1}}$	hasWeight($\text{rel}(d_p, d_q)$)$_{t_{n+2}}$	hasWeight($\text{rel}(d_p, d_q)$)$_{t_{n+3}}$
$rel(d_{1pi}, d_{1asra})$	0.324 ↘	0.324 →	0.438 ↗	0.285 ↘
$rel(d_{1pi}, d_{2asra})$	0.324 ↘	0.211 ↘	0.285 ↗	0.185 ↘
$rel(d_{1pi}, d_{3asra})$	0.324 ↘	0.324 →	0.438 ↗	0.592 ↗
$rel(d_{1pi}, d_{1wis})$	0.5 →	0.5 →	0.675 ↗	0.911 ↗
$rel(d_{1pi}, d_{1tips})$	–	–	0.5 +	0.324 ↘

In Section 8.4, the workshop employees were asked to work with the system and perform two different test scenarios: a brake disc replacement and a front window replacement. Before performing the tests, the users needed to enter into contact with the AIRS Prototype system. All test persons had no problems in using the AIRS Prototype software. Most of them only need rare assistance in using the system. This can be explained due to the simple search concept of AIRS that is at the deepest level (without AIRSKB-based features) similar to many free text search applications the workshop employees are familiar with: web search, free text search in a mail client program and many more. In other words, the employees understood the concept of search quickly and they were very interested in using the system. The test persons had clear expectations of the AIRS Prototype search. For example, they stated the possibility to search with synonyms and abbreviations is an important feature. Due to this expectation, the workshop employees gave very good and valuable feedback about the system. As stated before, one of the test tasks was to find all documents that are necessary for a brake disk replacement. An example of such a document is the *brake disk* replacement part document itself, which can be originally found in the isolated retrieval system of the electronic parts catalog. In contrast to the AIRS Prototype's free text search, the electronic parts catalog is based upon taxonomic-guided retrieval, presenting lists of parts information (and pictures) as search results at the taxonomic leave-node-level. The component *brake disk*, for example, can be found in the electronic parts catalog under the taxonomic node *rear wheel brakes*. AIRS again turns the principle around and the user can search for the part information *brake disk* directly. Later on, the taxonomic feature can be used as data filter criteria. Figure 8.2 of Section 8.2 shows the parts information list that contains the *brake disk* replacement part. In the figure, Area A frames the parts information, which is also shown as the search result in the electronic parts catalog[3]. In Area B of Figure 8.2 (Section 8.2), documents that are related to the *rear wheel brakes* can be seen. They are grouped by source and sorted by relation

[3]Of course, the retrieval of the original electronic parts catalog contains much more data filter criteria to ensure retrieval hits that match exactly the customer's car. Following the AIRS principle, all these filter criteria can be modeled as contexts and context-attributes.

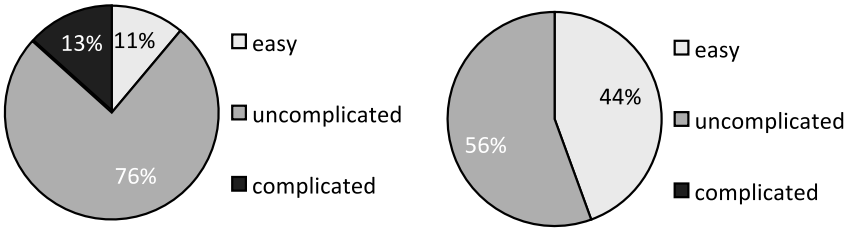
weights. For instance, two documents from the source "wis" (which actually contains replacement instructions for parts) are shown that are related to the parts information *rear wheel brakes*. During the retrieval for documents, the users were able to select result documents from both: the search result list and the related documents box.



(a) Using the isolated retrieval systems **(b)** Using AIRS Prototype

Figure 8.9: Result of user survey about how fast it is to find business case relevant documents.

In Table 8.1, the usage of documents in the field test for the *brake disk* replacement task is given. For example, in field test $n+1$ the user chose only document d_{2asra}. Given that AIRSKB contains a relation-aging concept, which distinguishes between *usage* and *non usage* of a document-relation, the documents that were used (or not used) in a field test have a direct influence on the document relation weights.



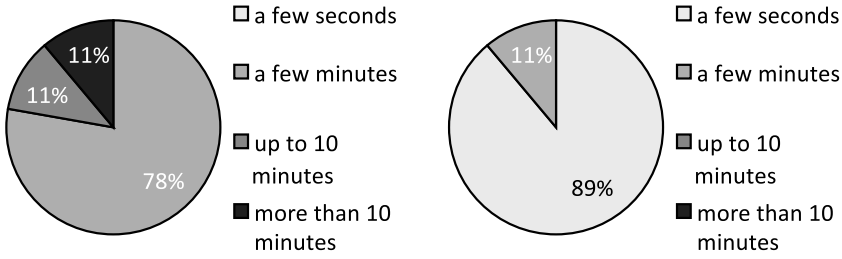
(a) Using the isolated retrieval systems **(b)** Using AIRS Prototype

Figure 8.10: Result of user survey about how easy it is to find business case relevant documents.

Table 8.2 shows the document relations of the part information *rear wheel brakes* (d_{1pi}) and how they changed from field test t_n to field test t_{n+3}. In Table 8.2, $\text{hasWeight}\big(\text{rel}\,(d_{1pi}, d)\,\big)_{t_x}$ is the weight of the relation $rel(d_{1pi}, d)$. Time stamp t_x means the time directly after field test number x.

After the field test $n+3$, document d_{1wis} shows a strong relationship with the parts information *rear wheel brakes*. Because document d_{1wis} contains the replacement instruction for

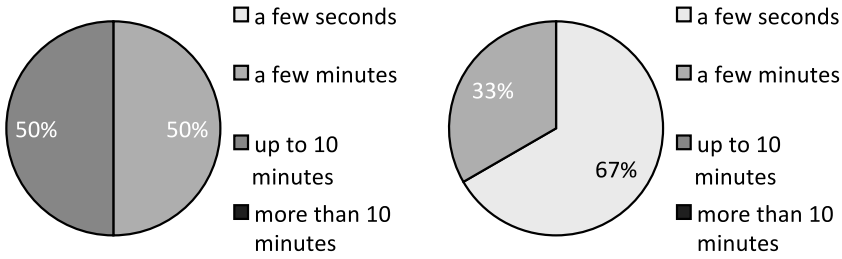
the part *brake disk*, the AIRS Prototype has "learned" a valid document relationship. In future searches, AIRS presents the replacement instruction d_{1wis} as the top-related document to the part information *rear wheel brakes*. It must be noted that AIRS has "learned" this relationship from the workshop employees' collective intelligence. In addition, the workshop employees were asked about their search experience with the AIRS Prototype compared to the isolated retrieval systems.



(a) Using the isolated retrieval systems **(b)** Using the AIRS Prototype

Figure 8.11: Result of user survey about how easy it is to find business case relevant documents.

Figure 8.9 (a) shows the mean of the workshop employee's opinions about the velocity of finding documents using the isolated retrieval systems. As shown 8.9 (b), the users say that the AIRS retrieval for documents is much faster than current retrieval for workshop documents: each of the workshop employees assesses the AIRS search algorithm as fast. Additionally, most of the users perceive the search as "uncomplicated" and a lot of them as "easy" (see Figure 8.10 (b)).

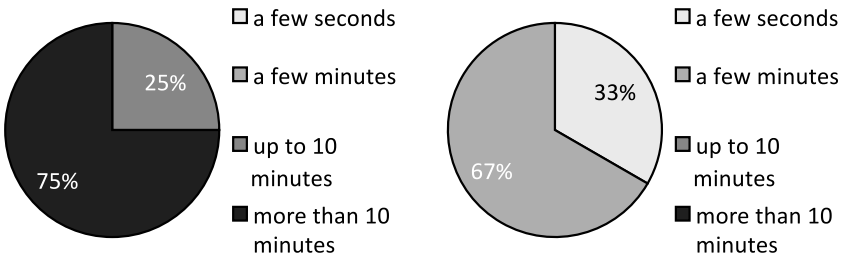


(a) Using the isolated retrieval systems **(b)** Using AIRS Prototype

Figure 8.12: Result of user survey about how easy it is to find business case relevant documents.

By contrast, the group of workshop employees who assessed the current retrieval as "easy" is much smaller: users find AIRS is an easy way to search for business case relevant documents, compared to the current retrieval system landscape. Moreover, the group

who categorizes the search as "complicated" has the same size (see Figure 8.10 (b)). One can interpret this in the following way: the users perceive the technology behind the AIRS Prototype as a necessary development, considering the currently isolated retrieval systems. Figure 8.11 shows the comparison of the question how long it would take to find a replacement instruction document that is not linked in the technical information document but that is necessary for the case. Most of the workshop employees marked "a few seconds" for the AIRS Prototype. This seems a realistic estimation because AIRS searches in all sources concurrently. The users stated that it is very easy to perform many searches by simply typing in a new query. Figure 8.12 shows the results for the search for technical information documents during the reception process. The result implies an interesting point: in contrast to a search using the isolated retrieval systems, a retrieval system based upon AIRS search technology can be used in real time. This means that a receptionist can perform the search while talking to the customer. Using the current retrieval systems, this seems impossible because the workshop employees assess that it would take up to 10 minutes and potentially even longer.



(a) Using the isolated retrieval systems **(b)** Using the AIRS Prototype

Figure 8.13: Result of user survey about how easy it is to find business case relevant documents.

Figure 8.13 presents the results for the last of the three case descriptions: the time to find all documents that are necessary to repair a car that had an accident. This seems the most complex scenario for the AIRS Prototype because many things of a crashed car must be repaired. This includes many documents of different workshop systems. The workshop employees made an interesting statement in the user tests by answering this question. For the current retrieval systems, they mostly choose "more than 10 minutes", which seems to be a realistic estimation. Surprisingly, most of the workshop employees found that with the AIRS Prototype this task should take "a few minutes" or even "a few seconds". The question is how realistic this estimation is. A closer look into AIRS can confirm this estimation because "a few minutes" can mean that "the AIRS Prototype is so fast that I can search for many documents at once and I can find all necessary documents in just a few minutes". Even "a few seconds" makes sense because a user can match a Suggest Cluster with a query. This means that a set of all necessary documents is still suggested and the user can find all of the documents with just one query.

The Suggest Cluster Algorithm was also evaluated during the field tests. The two domain-specific tests scenarios were used to analyze lists of suggested documents produced

by the Suggest Cluster Algorithm over time. In a specific test scenario, a user performs a search for case relevant documents. For each query that results in a case solution, the Suggest Cluster Algorithm identifies (or establishes) a Suggest Cluster document. Subsequently, all documents used in the case are automatically linked to this Suggest Cluster document. Still existing relationships between the Suggest Cluster document and documents that have been used in the case are reweighted instead. The idea is that users perform similar searches when they have to solve the similar tasks. This means that the probability is high that the users "find" the same Suggest Cluster documents with their queries. The list of suggested documents that is related to the Suggest Cluster document can change over time. It should be optimized by using the collective intelligence of the system users. The evaluation comprised the following investigations:

- The list of suggested documents for a Suggest Cluster document changes over time through use of the Suggest Cluster Algorithm (users match Suggest Cluster documents with their queries, new documents are added to the set of suggested documents and relationships between these documents and the Suggest Cluster document are updated). Therefore, the list of suggested documents belonging to a Suggest Cluster document was examined during the evaluation.

- The list of suggested documents is updated automatically by the Suggest Cluster Algorithm, based upon the professional expertise of the users. A document of the list of suggested documents containing information that is not relevant for the given case should be ranked down. The weight of the relationship from the "irrelevant" DOCUMENT to the Suggest Cluster DOCUMENT is reduced. A document that contains information that is not necessary for the case was examined in the field tests, whether it is automatically moved down in the list of suggested documents.

The results of the first test case of the field tests have been analyzed. Accordingly, a Suggest Cluster document was identified, which was often "matched" by user queries for the given case. The corresponding list of suggested documents was analyzed over time. Additionally, the list of these documents was "infected" with a document that contains information that is not relevant for the given case. This document should be moved down automatically in the list, when users "find" the corresponding Suggest Cluster document with their queries in future cases. Figure 8.14 shows the ranked list of suggested documents for a corresponding Suggest Cluster document over the time of five field tests. This list was produced by the Suggest Cluster Algorithm. The users "find" the corresponding Suggest Cluster document with their queries in field tests n, $n + 2$ and $n + 3$. Based upon the documents they added to the case solution, the list of suggested documents was updated through using the Suggest Cluster Algorithm. Document d_3_asra was injected manually into the list of suggested documents It contains information that is not necessary for the case solution. According to the hypothesis, the document d_3_asra should be moved down in the list of suggested documents. As shown in Figure 8.14, it was actually moved down to the last position of the list until the end of field test n. It was moved down to the last place of the list because no user included the document in the case solution. This means that the Suggest Cluster Algorithm works as expected. The following was observed during the tests:

- The users do not "find" the Suggest Cluster document in each field test.

- The list of suggested document has been re-ranked when the users "find" the Suggest Cluster document with their queries.
- The "infected" document moved down to the last position of the list over time.

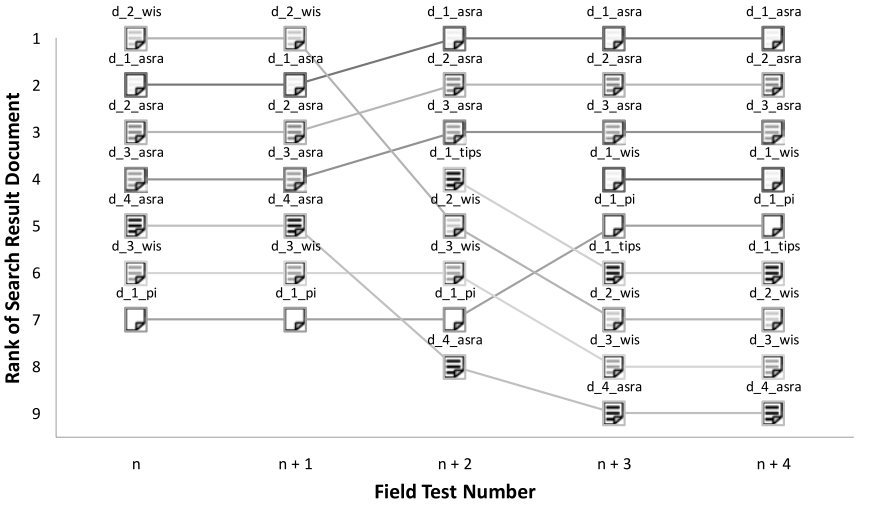


Figure 8.14: Development of suggested documents list produced by the Suggest Cluster Algorithm over time.

The test results can be interpreted as follows: The Suggest Cluster Algorithm works as expected and enables the inclusion of system users' retrieval activity back in the retrieval process since suggested documents are updated automatically. The Suggest Cluster Algorithm helps to use the collective intelligence of systems users to optimize the retrieval for case relevant documents.

Users actually "find" Suggest Cluster documents with their queries for similar tasks. Regarding the underlying business cases, Suggest documents should be optimized to produce Suggest Cluster documents that exactly match the underlying domain. Additionally, Suggest Cluster documents should also be build by domain experts first to optimize the automated feedback processing. For an extended evaluation approach, a quantitative evaluation of the Suggest Cluster Algorithm should be performed to retrieve advanced knowledge about the behavior of Suggest Cluster documents over time.

9 Conclusion and Future Research

This chapter concludes the AIRS research results and gives an outlook on possible further research activities and how they could be performed.

9.1 Summary

AIRS is a system that offers an intelligent and adaptive retrieval across a heterogeneous document landscape. AIRS provides a workflow of modern knowledge network-based information retrieval. This was achieved by applying the following steps in the development of AIRS:

1. Design an overarching generic model that represents all documents, sources and relationships in a given business or organizational context. AIRSKB serves as a model for the heterogeneous document landscape.

2. Build ontological individuals that represent the original documents and their sources in a uniform way to abstract irrelevant diversities. Document Abstracts were built during the Indexing Workflow of AIRS. They have been represented by ontological individuals in AIRSKB.

3. Analyze the source data and construct index documents for a state-of-the-art retrieval system. The built Document Abstracts include all of the information that is necessary for the retrieval. They served as a reference for corresponding index documents.

4. Link each index document to the corresponding ontological individual. This was achieved in the Indexing Workflow of AIRS.

5. Map hard-coded document links to document relationships represented in the ontology. During the Indexing Workflow, existing links between documents have been represented by Document-Relations between the Document representations of the original documents.

6. Develop an evolutionary process for analyzing user behavior and use the corresponding feedback to adapt document relationships over time. The Retrieval & Feedback Workflow of AIRS includes different algorithms that include user feedback to improve the retrieval quality (for example, the Suggest Cluster Algorithm).

AIRS combines the collective intelligence of the retrieval system users with the document authors' knowledge. It used both a state-of-the-art search technology and an ontology that contains a document representation of the heterogeneous document landscape, the AIRSKB. The retrieval algorithm of the AIRS uses both for the documents search in the AIRS index and the AIRSKB. The AIRS index is used to implement a state-of-the-art

vector space retrieval across the document landscape and AIRSKB is used to find related documents regarding a search result document or the define retrieval strategies. The infrastructure of AIRS again can be used to implement extended retrieval algorithms like the Suggest Cluster Algorithm.

AIRS showed what kinds of systems, algorithms and technologies should be used to optimize the access to documents. More specifically, the principle of document abstraction and combining search technologies with a heterogeneous document landscape model represented through an ontology was developed to ensure this document access. The users can easily access documents of the heterogeneous document landscape simply by using an AIRS-based system. AIRS is a suggested system that can be used in different domains where expert knowledge is necessary to solve complex business tasks. Therefore, firms can implement a retrieval system based upon AIRS to optimize critical business processes. This was shown through field tests using experts as users and the AIRS Prototype as the tested system in the domain of workshop processes and car maintenance. The field tests were performed in workshops and workshop employees serve as test users. The AIRS Prototype application is the reference implementation of the AIRS architecture suggestion. It provides a Rich Internet Application that contains all of the functions of AIRS. This prototype was used to perform user tests with real expert user groups and real life test situations. The user tests of the AIRS Prototype showed that the AIRS system is highly accepted by expert users.

A scientific finding of this work is the answer to the question how a knowledge network can be included in document retrieval in heterogeneous document landscapes representing a special domain. The AIRS principle again showed how such a deep integration must be performed and which kinds of retrieval algorithms can be performed on top of the knowledge network. The main questions can be answered:

1. Can a single systems view be provided for all of the case-related documents kept in different retrieval systems? – With AIRS it is possible to establish a homogeneous access structure across the heterogeneous system landscape.

2. Can a seamless access across this disparate and disconnected retrieval systems be designed? – Combining state-of-the-art search technology with knowledge representation approaches serves as the base for advanced document retrieval. The ontological model of the document landscape enables overall document relationships that can be used for both document retrieval and document linkage information.

3. Can the quality of retrieval results and the effectiveness of the retrieval process be improved by exploiting user feedback? – The combination of search technologies with document landscape knowledge provides by the AIRSKB enables the development of new kinds of search algorithms. An example is the Suggest Cluster Algorithm that includes the collective intelligence of system users in a new retrieval approach.

4. Can a technical solution be developed that is accepted by users in the field? – The AIRS Prototype serves as example implementation for the AIRS principle. The prototype application was successfully evaluated in real world workshop situation.

AIRS combines the functionality of state-of-the art search technology with an ontology in that way that index documents of an index are also modeled in an ontology. This

ensures that complex relationships between documents can be established. These relationships can change over time (index documents of information retrieval systems are not designed for this approach). Furthermore, the relationships allow implementing new kinds of retrieval algorithms based upon both vector space retrieval of modern search engines and knowledge representation through ontologies. The Suggest Cluster Algorithm that is introduced in this work provides such an algorithm. It matches the user's query against a cluster of concepts with the help of the IRS Component of AIRS. Subsequently, the cluster (Suggest Cluster document) found is located in the AIRSKB ontology and the Ontology Component searches in the ontology for all related documents of the cluster. The related documents found subsequently serve as the search result. Additionally, the relationships between the Suggest Cluster document and the related documents are updated over time. User feedback is collected automatically by AIRS and the relationships are updated with the help of this feedback. Accordingly, the knowledge of the AIRS users flows back in the system and the retrieval of future searches is optimized. This work introduces this approach of optimizing document retrieval based upon collective intelligence of system users.

AIRSKB is the ontology that is used to represent the heterogeneous document landscape of the isolated retrieval systems. This work introduces the principles of ontology design that comprise defining the application context, performing conceptualization and defining inference rules. Additionally, this work suggests the following steps of ontology engineering for an ontology that is used to represent document landscapes:

1. *Initial activities.* Focusing on the questions the application context defines. The application context provides a set of questions that help to understand the domain and the need for an ontology. Additionally, elements of the ontology are defined in this step. This is necessary for a common understanding of the domain. After these initial activities, the picture of the ontology that needs to be built was very clear. This enormously helps to perform the conceptualization because concepts and attributes and requirements of the ontology were well known before.

2. *Performing conceptualization.* The step comprises defining classes and relationships. The ontology that is used for retrieval in heterogeneous document landscapes (AIRSKB) includes concepts for documents, sources, relationships between the document and attributes that can be used to describe validity conditions of documents. The class model of the AIRSKB was built and attributes of the individuals were described whereby the individuals match the previously-defined classes.

3. *Defining inference rules.* The theoretical frame for the knowledge acquisition means defining the mathematical background. Axioms and equations ensure ontological reasoning.

4. *Ontology representation.* Defining an ontology representation framework or language that can be used to share the ontology among applications and people. This also involves the technology how the ontology can be used in applications.

9.2 Research Opportunities

AIRS provides a new kind of retrieval based upon a deep integration of an ontological model in the search workflow. This model enables advanced retrieval algorithms such as the Suggest Cluster Algorithm (see Section 5.4). This has opened a few new research interests. Examples are questions that address the possibility of establishing new kinds of retrieval strategies on top of the AIRS infrastructure beside the Suggest Cluster Algorithm. Other future research should focus on methods for calculating the cheapest document pathways in the adaptive relationship environment of AIRSKB. Especially the speed and complexity of pathway calculation is an interesting field of research. For example, path finding algorithm of AIRS Prototype bases on breadth-first search in combination of Document candidate reduction using valid Source pathways (see explanation for Algorithm 1 of Section 7.5). Additionally, four measures have been suggested in this work that can be used for pathway calculation (see Section 3.3). Therefore, further research should focus on the evaluation of existing or development new path finding algorithms in the context of AIRS. Existing graph search algorithm as named in [51] can serve as basis. This basis must subsequently be extended to deal with conditional relationships. Furthermore, extended functions for calculation the pathways' weights should be evaluated or developed in further research activities. These functions must subsequently calculate the weights of the pathways to find the most relevant documents of a target source for a given document.

The AIRS Prototype is the reference implementation of AIRS. It has been evaluated in the domain of workshop processes for vehicle maintenance. The field tests have been carried out with workshop employees exhibiting many years of professional experience in workshop services. They showed that the new ontology-based retrieval is superior to the existing retrieval technology and that the collective feedback of workshop experts enables the automatic and valid reconstruction of hidden document relationships. The AIRS research provides new methods and technologies of document retrieval across a heterogeneous and very large document landscape of a special domain. The possibility of document linkage even across system borders, an easy and at the same time powerful retrieval, offered new ways of development of new products for the workshop processes. A product based upon the AIRS Prototype should be developed including the following steps:

1. An extended phase in a partner workshop to identify requirements and parameters for an application implementation.
2. Development of scenarios for an application implementation (including use case and risk analysis).
3. A further qualitative and quantitative evaluation helps to calculate the business case relevance.

The application development frame should be a project that must include the following four points:

1. a use case analysis,
2. an effort and risk analysis,
3. for a production system implementation and integration setup, and

4. application implementation itself.

These points can be performed as project milestones. Figure 9.1 illustrates these points as a time line and explains the main condition and the result of every mile stone.

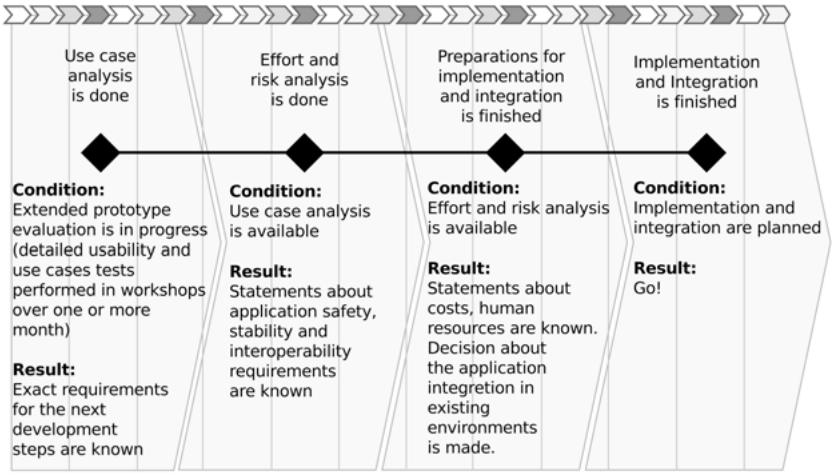


Figure 9.1: Milestones for possible future development of a software solution based upon AIRS Prototype.

The question is how the AIRS principle can be used in other business domains for information access. For example, the medical domain produces case depended scenarios that need exact information for different processes. Furthermore, the literature existing in this domain provides a heterogeneous document landscape too. Different approaches focus on the topic of retrieval for literature in medicine and biology (for example, see [1], [40], [21], [78] and [48]). Therefore, further research approaches should focus on the evaluation of AIRS based software products in different domains. For example, users of booking systems[1] as used in tourism industry must search for offers in many sources.

AIRS is designed to work language independent: the ontology approach using AIRSKB as knowledge network for document relationships can include multiple-language documents. The retrieval algorithms work fine for many languages as long as the underlying search technology supports the languages. The AIRS Prototype was developed to evaluate the AIRS algorithms in a German language speaking workshop environment. The combination of German language search and collective knowledge retrieval worked fine (Section 8.5). Further research should focus on evaluating AIRS for multiple languages. One interesting question is how the Suggest Cluster Algorithm can be extended with multiple language concepts and how Suggest Cluster documents differ for various languages.

Other possible research questions focus on the usage of AIRSKB in different domains and application scenarios:

[1]See [28] and [47] for more information about booking systems and challenges in tourism industry.

- How do relationships between documents change over time when AIRSKB is used in different domains? AIRSKB provides an adaptive knowledge network of documents, which represents the knowledge about document relationships of domain experts. Different domains mean different knowledge networks of document relationships. A possible solution is that each domain uses its own AIRSKB "instance".

- How can relationships between documents of AIRSKB be transferred to new documents, which are similar to documents that already exist in AIRS Index and AIRSKB? For example, replacement parts of a car are related to installation instruction documents for a special car model. The question is whether these relationships can be transferred to similar replacement parts and similar installation instruction documents when a new car model is released?

- How to implement an update process for DOCUMENTS of AIRSKB? The question is how AIRS Index documents and AIRSKB representations change if the original documents change; for example, if an attribute is omitted in a new document version.

A Appendix

The appendix shows two examples of the questionnaires the workshop employees were asked to fill out. The last part of the appendix presents a document containing the two test cases the workshop employees had to solve during the user tests.

A.1 Questionnaire 1

Appendix A.1 shows an excerpt of a completed questionnaire. This questionnaire starts with a short introduction into the topic and it continues with closed questions (multiple choice) and open questions (free text fields) about the current document search.

FRAGE 5: Was denken Sie, wie schätzen Ihre Kollegen Sie ein?

- Bitte kreuzen Sie an.
- Bitte *nur* ein Kreuz.

☐ Ein „alter Hase", der alles weiß, sich mit allem auskennt und alles schon gesehen hat.

☒ Jemand, der schon lang dabei ist. Ihn fragt man, wenn man ein Problem hat.

☐ Ein Fachmann, der sich in seinem Gebiet gut auskennt.

☐ Ein Neuling, der noch nicht lang dabei ist.

FRAGE 6: Wie viele Schritte und wie viele verschiedene Werkstatt-Systeme benötigen Sie, um die geschilderten Aufgaben zu erledigen?

- Bitte wählen Sie die Anzahl der Schritte und die Anzahl der werkstatt-systeme aus.

Beispiel 1: Ich öffne Programm X und suche nach Dokument Y.

→ 2 Schritte
→ 1 Werkstatt-System

Beispiel 2: Ich frage Kollege X. Öffne danach Programm Y und Suche nach Dokument Z.

→ 3 Schritte
→ 1 Werkstatt-System

Beispiel 3: Ich öffne Programm A und suche nach Information X. Danach öffne ich Programm B und benutze Information X für die Suche nach Dokument Z.

→ 4 Schritte
→ 2 Werkstatt-Systeme

1. Sie wollen herausfinden, ob es zu einer in einem TIPS-Dokument beschriebenen Abhilfe eine Reparaturanleitung gibt, wenn keine WIS-Referenz vorhanden ist.

   4 Schritt(e)

   1 Werkstatt-System(e)

2. Sie nehmen das Fahrzeug eines Kunden an. Während Sie mit dem Kunden sprechen, wollen Sie gleich nachschauen, ob für das geschilderte Problem ein TIPS-Dokument existiert.

   5 Schritt(e)

   1 Werkstatt-System(e)

3. Ein Unfallfahrzeug wird in Ihre Werkstatt angeliefert. Sowohl elektronische als auch mechanische Probleme sind zu beheben. Sie wollen alle Dokumente zusammenstellen, die Sie für den Vorgang benötigen.

   > 5 Schritt(e)

   2 Werkstatt-System(e)

FRAGE 8: Schätzen Sie die Zeit, die Sie benötigen, um die Aufgaben aus Frage 7 zu erledigen.

- Bitte kreuzen Sie an.
- Bitte *nur* ein Kreuz je Frage.

1. Sie wollen herausfinden, ob es zu einer in einem TIPS-Dokument beschriebenen Abhilfe eine Reparaturanleitung gibt, wenn keine WIS-Referenz vorhanden ist.

   ☐ einige Sekunden

   ☒ einige Minuten

   ☐ bis zu zehn Minuten

☐ mehr als zehn Minuten

2. Sie nehmen das Fahrzeug eines Kunden an. Während Sie mit dem Kunden sprechen, wollen Sie gleich nachschauen, ob für das geschilderte Problem ein TIPS-Dokument existiert.

☐ einige Sekunden

☒ einige Minuten

☐ bis zu zehn Minuten

☐ mehr als zehn Minuten

3. Ein Unfallfahrzeug wird in Ihre Werkstatt angeliefert. Sowohl elektronische als auch mechanische Probleme sind zu beheben. Sie wollen alle Dokumente zusammenstellen, die Sie für den Vorgang benötigen.

☐ einige Sekunden

☐ einige Minuten

☐ bis zu zehn Minuten

☒ mehr als zehn Minuten

FRAGE 9: Treffen folgende Aussagen Ihrer Meinung nach zu?

- Bitte kreuzen Sie an.
- Bitte *nur* ein Kreuz je Frage.

1. Es würde bei der täglichen Arbeit in der Werkstatt helfen, wenn man wüsste, was Kollegen aus anderen Werkstätten bei bestimmten Service-Problemen gemacht haben.

A.2 Questionnaire 2

Appendix A.2 shows an excerpt of a completed questionnaire. This questionnaire was presented after the field tests. It contains similar questions as the first questionnaire, asking specifically about the AIRS Prototype.

☐ nicht so gut — *Wenn man etwas sucht, findet man es schon irgendwann. Aber alle wichtigen Informationen sind nicht immer dabei.*

☐ schlecht — *Ich bin nicht zufrieden mit der Suche. Wichtige Dokumente fehlen in den Suchergebnissen.*

☐ unentschlossen — *Ich weiß nicht...*

FRAGE 2: Was werden Sie ihren Kollegen über AIRS erzählen?

- Bitte kreuzen Sie eine der Möglichkeiten an.

☐ Ich finde den AIRS-Prototype super.

☒ Der AIRS-Prototype ist hilfreich.

☐ Der AIRS-Prototype ist schon okay.

☐ Der AIRS-Prototpye ist der falsche Ansatz.

FRAGE 3: Schätzen Sie die Zeit, die Sie benötigen, um die Aufgaben mit Hilfe des AIRS-Prototype zu erledigen.

- Bitte kreuzen Sie an.
- Bitte *nur* ein Kreuz je Frage.

1. Sie wollen herausfinden, ob es zu einer in einem TIPS-Dokument beschriebenen Abhilfe eine Reparaturanleitung gibt, wenn keine WIS-Referenz vorhanden ist.

   ☒ einige Sekunden

   ☐ einige Minuten

   ☐ bis zu zehn Minuten

☐ mehr als zehn Minuten

2. Sie nehmen das Fahrzeug eines Kunden an. Während Sie mit dem Kunden sprechen, wollen Sie gleich nachschauen, ob für das geschilderte Problem ein TIPS-Dokument existiert.

☒ einige Sekunden

☐ einige Minuten

☐ bis zu zehn Minuten

☐ mehr als zehn Minuten

3. Ein Unfallfahrzeug wird in Ihre Werkstatt angeliefert. Sowohl elektronische als auch mechanische Probleme sind zu beheben. Sie wollen alle Dokumente zusammenstellen, die Sie für den Vorgang benötigen.

☐ einige Sekunden

☒ einige Minuten

☐ bis zu zehn Minuten

☐ mehr als zehn Minuten

FRAGE 4: Treffen folgende Aussagen Ihrer Meinung nach zu?

- Bitte kreuzen Sie an.
- Bitte *nur* ein Kreuz je Frage.

1. Es würde bei der täglichen Arbeit in der Werkstatt helfen, wenn man wüsste, was Kollegen aus anderen Werkstätten bei bestimmten Service-Problemen gemacht haben.

A.3 User Tasks

Appendix A.3 shows an excerpt of a document containing the two test cases the users needed to solve.

FALL 1: Bei einem Kundenfahrzeug vom Typ „C 320 CDI“ sollen an der Hinterachse sowohl die Bremsscheiben als auch die Bremsklötze gewechselt werden.

Aufgabe:

1. Stellen Sie bitte mit Hilfe des AIRS Prototypen alle Dokumente für einen Service-Vorgang zusammen, die Sie finden (WIS-Dokumente, TIPS-Dokumente, Teile-Information, ASRA-Dokumente, Symptome, Fehler-Codes und Diagnose-Daten).
2. Fügen Sie dazu bitte die Dokumente dem Vorgang hinzu (Warenkorb).
3. Schließen Sie bitte den Vorgang ab, indem Sie den Warenkorb als PDF downloaden und die PDF auf dem Windows Desktop ablegen.

Hinweise zu Fall 1:

- Die Konstruktionsgruppe 42 „Bremsanlage“ ist für die Suche relevant
- Die Konstruktionsgruppe 40 „Räder“ ist ebenfalls relevant
- Bremsscheiben tauschen
- Laufräder müssen abmontiert werden

Notizen:

FALL 2: Bei einem Kundenfahrzeug vom Typ „C 320 CDI“ ist die Frontscheibe gerissen, sie muss getauscht werden.

Aufgabe:

1. Stellen Sie bitte mit Hilfe des AIRS Prototypen alle Dokumente für einen Service-Vorgang zusammen, die Sie finden (WIS-Dokumente, TIPS-Dokumente, Teile-Information, ASRA-Dokumente, Symptome, Fehler-Codes und Diagnose-Daten).
2. Fügen Sie dazu bitte die Dokumente dem Vorgang hinzu (Warenkorb).
3. Schließen Sie bitte den Vorgang ab, indem Sie den Warenkorb als PDF downloaden und die PDF auf dem Windows Desktop ablegen.

Hinweise zu Fall 2:

Notizen:

Glossary

Adaptive Relationships AIRSKB relationships defined by Inference Rules that include System-Relations and Document-Relations with changing weights between 0.1 and 1.

AIRS Advanced Ontology-based Information Retrieval System is a system that combines the functionality of state-of-the-art search technology with ontological knowledge to enable retrieval in Heterogeneous Document Landscapes. The architecture of AIRS includes three main components (IRS Component, Ontology Component, Statistics Component), which are managed by the Core Component.

AIRS Backend The server component of AIRS.

AIRS Client The user interface of AIRS.

AIRS Include Source Framework A framework suggested by the AIRS Prototype implementation and used in the Indexing Component to encapsulate the Indexing Workflow of AIRS. It uses the AIRSKB Framework and the AIRS Index & Search Framework for this approach.

AIRS Index & Search Framework A framework suggested by the AIRS Prototype implementation and used in the IRS Component to encapsulate the functions of the underlying retrieval system solution.

AIRS Prototype A Rich Internet Application that serves as a reference implementation for AIRS based upon the AIRS' architecture suggestion. The AIRS Prototype was used during the field tests for the evaluation of methods and algorithm of AIRS.

AIRSKB AIRS Knowledge Base is an ontology that represents the concepts of a Heterogeneous Document Landscape. It defines an adaptive knowledge network used by AIRS to enable adaptive document relationships in retrieval processes.

AIRSKB Framework A framework suggested by the AIRS Prototype implementation and used in the Ontology Component to encapsulate the functions of the underlying knowledge representation technology.

Application Context First step of ontology development based upon Knowledge Representation in Context. Describes the usage of a list of questions to clarify fundamental points, including why is an ontology necessary, what it should look like, and for what it should be made for.

Attribute-Relation Element of AIRSKB (written as: Attribute-Relation). Represents relations between attributes or segments of document concepts of the Heterogeneous Document Landscape.

Best Context-Sensitive-Pathways Class of AIRSKB Pathways defined by Inference Rules that represent the best-r ranked Pathways of Conditional Relationships of the Heterogeneous Document Landscape.

Conceptualization Second step of ontology development based upon Knowledge Representation in Context. Involves the implementation of a complex ontological model of classes, individuals and semantic relationships between the individuals.

Conditional Relationship Concept of the Heterogeneous Document Landscape. Represents a relationship between documents that is restricted via a set of Validity Conditions.

Context Element of AIRSKB (written as: CONTEXT). Represents a class of Validity Conditions of a Heterogeneous Document Landscape. Belongs to a SOURCE via context-of-source relationship and to a CONTEXT-ATTRIBUTE via context-attribute-of-context relationship.

Context-Attribute Element of AIRSKB (written as: CONTEXT-ATTRIBUTE). Represents a Validity Condition of a Heterogeneous Document Landscape. Belongs to a CONTEXT via a context-attribute-of-context relationship and a DOCUMENT via a context-attribute-of-document relationship.

context-attribute-of-context Element of AIRSKB. A SYSTEM-RELATION between a CONTEXT-ATTRIBUTE and a CONTEXT.

context-attribute-of-document Element of AIRSKB. A SYSTEM-RELATION between a CONTEXT-ATTRIBUTE and a DOCUMENT.

context-of-source Element of AIRSKB). A SYSTEM-RELATION between a CONTEXT and a SOURCE.

Context-Sensitive Relations Class of AIRSKB relationships defined by Inference Rules that represent Conditional Relationships of the Heterogeneous Document Landscape.

Context-Sensitive-Pathways Class of AIRSKB Pathways defined by Inference Rules that represent Pathways of Conditional Relationships of the Heterogeneous Document Landscape.

Core Component The main component of the AIRS architecture that controls the Retrieval & Feedback Workflow and manages the interaction between the Core Component, the Ontology Component and the Statistics Component.

Document Concept of the Heterogeneous Document Landscape. Represents a single piece of information used to solve a particular task. It could have attributes or segments and is connected to only one retrieval system or source location. The concept Document is also represented in the AIRSKB through the ontological element DOCUMENT (written in small capitals).

Document Abstract Concept of the Heterogeneous Document Landscape. A part of a Document defined by a subset of attributes or segments of the Document containing information that is necessary for the retrieval.

Document Location Concept of the Heterogeneous Document Landscape. , *see* Source

Document Search Algorithm An algorithm of the Retrieval & Feedback Workflow of AIRS that enriches ordinary natural language vector space retrieval with search functionality for related documents regarding a single search result by using the AIRSKB.

Document-Attribute Element of AIRSKB (written as: DOCUMENT-ATTRIBUTE). Represents an attribute or segment of a Document of the Heterogeneous Document Landscape. Belongs to a DOCUMENT via a document-attribute-of-document relationship and a SOURCE via a document-attribute-of-source relationship. Means both an ontological class or ontological individual of class DOCUMENT-ATTRIBUTE.

document-attribute-of-document Element of AIRSKB. A SYSTEM-RELATION between a DOCUMENT-ATTRIBUTE and a DOCUMENT. Represents attributes or segments that a particular Document of the Heterogeneous Document Landscape already has.

document-attribute-of-source Element of AIRSKB. A SYSTEM-RELATION between a DOCUMENT-ATTRIBUTE and a SOURCE. Represents attributes or segments that Documents of the Source of the Heterogeneous Document Landscape can have.

document-of Element of AIRSKB. A SYSTEM-RELATION between a DOCUMENT and a SOURCE. Represents the connection between Document concepts and Source concepts of the Heterogeneous Document Landscape.

Document-Relation Element of AIRSKB (written as: DOCUMENT-RELATION). Represents relationships between Document concepts of the Heterogeneous Document Landscape. The Inference Rules differentiate between Adaptive Relationships and Static Relationships.

Heterogeneous Document Landscape A domain where documents that can appear in different source locations (databases, isolated retrieval systems, file systems and others) are used to solve particular problems.

Indexing Workflow The workflow of AIRS to include Documents of the Heterogeneous Document Landscape in the AIRS Index as well as representing the Documents as ontological individuals in the AIRSKB.

Inference Rules Last step of ontology development based upon Knowledge Representation in Context. A mathematical model that serves as a frame for navigation through the ontology.

IRS Component The component of AIRS that encapsulates the functionality of an underlying retrieval system. For example, enterprise search solutions can serve as an underlying retrieval system.

is-attribute-related-to Element of AIRSKB. An ATTRIBUTE-RELATION between two DOCUMENT-ATTRIBUTES. Represents a relationship between two attributes or segments of Documents of the Heterogeneous Document Landscape.

is-linked-to Element of AIRSKB. A Document-Relation between two Documents of type Static Relationships. Represents an immutable relationship between two documents of the Heterogeneous Document Landscape.

is-related-to Element of AIRSKB. A Document-Relation between two Documents of type Adaptive Relationships. Represents an adaptive relationship between two documents of the Heterogeneous Document Landscape.

is-source-linked-to Element of AIRSKB. A Source-Relation between two Sources of type Static Relationships. Represents an immutable relationship between two documents of the Heterogeneous Document Landscape. Indicates the existence of at least one is-linked-to relationship between two Documents of both Sources.

is-source-related-to Element of AIRSKB. A Source-Relation between two Sources of type Adaptive Relationships. Represents an immutable relationship between two documents of the Heterogeneous Document Landscape. Indicates the existence of at least one is-related-to relationship between two Documents of both Sources.

Knowledge Representation in Context Methodology of ontology development based upon the engineering steps of Application Context, Conceptualization and Inference Rules.

Masking Relations Type of AIRSKB relationships defined by Inference Rules that group Adaptive Relationships and Static Relationships is-related-to and is-linked-to as well as is-source-related-to and is-source-linked-to.

Ontology A knowledge base described by an Application Context and defined through a Conceptualization in combination with Inference Rules.

Ontology Component A component of the AIRS architecture used in the Index Workflow and the Retrieval & Feedback Workflow. It serves as access structure to the AIRSKB and provides functions for path finding and path calculation.

Pathways Concept of the Heterogeneous Document Landscape. Represents a path through the Heterogeneous Document landscape defined through a concatenation of Relationships.

Relationship Concept of the Heterogeneous Document Landscape. Represents a relationship between two documents.

Retrieval & Feedback Workflow The document search workflow of AIRS that includes the Document Search Algorithm and the Suggest Cluster Algorithm. The workflow is controlled by the Core Component.

Search Instance Object The transport object for user queries as well as search results during the Retrieval & Feedback Workflow of AIRS. The object is transmitted between the AIRS Client and AIRS Backend.

Source Concept of the Heterogeneous Document Landscape. Sources are document locations. They represent a containers of documents with similar attributes or segments, including databases, isolated retrieval systems, file systems or other locations. The concept Source is also represented in the AIRSKB through the ontological element SOURCE (written in small capitals).

Static Relationships AIRSKB relationships defined by Inference Rules that include SYSTEM-RELATIONS and DOCUMENT-RELATIONS with constant weights.

Statistics Component A component of the AIRS architecture used in the Retrieval & Feedback Workflow. It encapsulates the feedback processing functions of AIRS.

Suggest Cluster Algorithm Retrieval algorithm of the Retrieval & Feedback Workflow that uses the AIRS infrastructure to enable the retrieval for related documents based upon concept cluster.

Suggest Cluster Document A conjunction of Suggest documents used in the Suggest Cluster Algorithm. Represents an activity in the real world that can be described through different combinations of synonym terms.

Suggest Document Representation of a concept through a set of semantically-related terms used in the Suggest Cluster Algorithm.

System-generated Relationships AIRSKB relationships defined by Inference Rules that include all SYSTEM-RELATIONS.

System-Relation Element of AIRSKB (written as: SYSTEM-RELATION). Represent a relations between DOCUMENTS, SOURCES, ATTRIBUTES, CONTEXTS, CONTEXT-ATTRIBUTES or DOCUMENT-ATTRIBUTES. These kinds of relationships represent the structure of the Heterogeneous Document Landscape and are categorized as System-generated Relationships through the Inference Rules.

Validity Condition Concept of the Heterogeneous Document Landscape. Represents a restriction of the use of Documents for a particular business task.

Index

Bibliography

[1] J. M. Abasolo and M. Gomez. MELISA: An Ontology-based Agent for Information Retrieval in Medicine. In *Proceedings of the first international workshop on the semantic web (SemWeb2000)*, pages 73–82, 2000.

[2] S. Alag. *Collective Intelligence in Action*. Manning Publications, 2008.

[3] H. Benbya, G. Passiante, and N. A. Belbaly. Corporate Portal: a Tool for Knowledge Management Synchronization. *International Journal of Information Management*, 24(3):201–220, 2004.

[4] V. R. Benjamins, D. Fensel, and A. G. Perez. Knowledge Management Through Ontologies. In *Proc. Practical Aspects of Knowledge Management*, pages 5.1–5.12, 1998.

[5] D. Borthakur. *The Hadoop Distributed File System: Architecture and Design*. The Apache Software Foundation, 2007.

[6] P. Bouquet, F. Giunchiglia, F. van Harmelen, L. Serafini, and H. Stuckenschmidt. C-OWL: Contextualizing Ontologies. In D. Fensel, K. P. Sycara, and J. Mylopoulos, editors, *International Semantic Web Conference*, volume 2870 of *Lecture Notes in Computer Science*, pages 164–179. Springer, 2003.

[7] S. Braun, A. Schmidt, and A. Walter. Ontology Maturing: a Collaborative Web 2.0 Approach to Ontology Engineering. In *Proceedings of the WWW Workshop on Social and Collaborative Construction of Structured Knowledge*, volume 273, Banff, Canada, 2007.

[8] R. Cattell. Scalable SQL and NoSQL Data Stores. *SIGMOD Record*, 39(4):12–27, 2010.

[9] B. Chandrasekaran, J. Josephson, and R. Benjamins. What Are Ontologies and Why Do We Need Them? *IEEE Intelligent Systems*, 14(1):Page 20–26, 1999.

[10] H. Chen, R. H. L. Chiang, and V. C. Storey. Business Intelligence and Analytics: From Big Data to Big Impact. *MIS Quarterly*, 36(4):1165–1188, 2012.

[11] C. W. Choo, B. Detlor, and D. Turnbull. The Intranet as Infrastructure for Knowledge Work. In *Web Work*, pages 71–100. Springer, 2000.

[12] D. Connolly and S. Hawke. OWL Web Ontology Language: Errata. `http://www.w3.org/2001/sw/WebOnt/errata`, 2004. [Online; accessed 18-09-2016].

[13] T. H. Davenport and D. J. Patil. Data Scientist: The Sexiest Job of the 21st Century. *Harvard Business Review*, 90:70–76, 2012.

[14] F. D. Davis. *A Technology Acceptance Model for Empirically Testing new End-User Information Systems: Theory and Results*. PhD thesis, Massachusetts Institute of Technology, 1985.

[15] R. de Zoeten and J. Rohmann. Call Center. In *Handbuch Electronic Business*, pages 383–414. Springer, 2002.

[16] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Commun ACM*, 51:107–113, 2008.

[17] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by Latent Semantic Analysis. *Journal of the American Society for information science*, 41(6):391, 1990.

[18] B. Detlor. The Corporate Portal as Information Infrastructure: Towards a Framework for Portal Design. *International Journal of Information Management*, 20(2):91–101, 2000.

[19] R. Diestel. *Graphentheorie (in German)*, volume 2. Springer Berlin, Heildelberg, New York, 1996.

[20] L. Ding, P. Kolari, Z. Ding, and S. Avancha. Using Ontologies in the Semantic Web: A survey. *Ontologies*, pages 79–113, 2007.

[21] G. Stuart Doig and F. Simpson. Efficient Literature Searching: a Core Skill for the Practice of Evidence-based Medicine. *Intensive Care Med*, 29:2119–2127, 2003.

[22] M. D. Ekstrand, J. T. Riedl, and J. A. Konstan. Collaborative Filtering Recommender Systems. *Foundations and Trends in Human-Computer Interaction*, 4(2):81–173, 2011.

[23] M. E. Falagas, E. I. Pitsouni, G. A. Malietzis, and G. Pappas. Comparison of PubMed, Scopus, Web of Science, and Google Scholar: Strengths and Weaknesses. *The FASEB journal*, 22(2):338–342, 2008.

[24] M. Farhoodi, M. Mahmoudi, A.M.Z. Bidoki, A. Yari, and M. Azadnia. Query Expansion Using Persian Ontology Derived from Wikipedia. *World Applied Sciences Journal*, 7(4):410–417, 2009.

[25] M. Fernández-López and A. Gómez-Pérez. Overview and Analysis of Methodologies for Building Ontologies. *The Knowledge Engineering Review*, 17(2):129–156, 2002.

[26] D. H. Fischer. Ein Lehrbeispiel fur eine Ontologie: OpenCyc (in German). *Information Wissenschaft und Praxis*, 55(3):139–142, 2004.

[27] S. Ghemawat, H. Gobioff, and S. T. Leung. The Google File System. *SIGOPS Oper. Syst. Rev.*, 37(5):29–43, 2003.

[28] R. Goecke, T. Eberhard, and J. Roth. Neue Wege zur Navigation durch die Datenflut der Reiseangebote–auf der Suche nach neuer Beratungsqualität im digitalen Zeitalter (in German). *Arbeitsbericht der Fakultät für Tourismus, Fachhochschule München*, 2010.

[29] I. Gordelik. Vom Aschenputtel in die Unternehmensspitze - Der Aufstieg des professionellen Call Center-Managements (in German). In *Handbuch Kundenmanagement*, pages 773–787. Springer, 2008.

[30] T. R. Gruber. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[31] M. Gruninger and J. Lee. ONTOLOGY. *Communications of the ACM*, 45(2):39, 2002.

[32] N. Guarino. Understanding, Building and Using Ontologies. *Int. Journal Human-Computer Studies*, 45(2/3), 1997.

[33] N. Guarino and P. Giaretta. Ontologies and Knowledge Bases: Towards a Terminological Clarification. In N. Mars, editor, *Towards Very Large Knowledge Bases*, pages 25–32. Amsterdam: IOS Press, 1995.

[34] N. Guarino, D. Oberle, and S. Staab. What is an Ontology? In S. Staab and R. Studer, editors, *Handbook on Ontologies*, International Handbooks on Information Systems, pages 1–17. Springer Berlin Heidelberg, 2009.

[35] Y. Guo, H. Harkema, and R. J. Gaizauskas. Sheffield University and the TREC 2004 Genomics Track: Query Expansion Using Synonymous Terms. In E. M. Voorhees and L. P. Buckland, editors, *Proceedings of the Thirteenth Text REtrieval Conference, TREC 2004, Gaithersburg, Maryland, USA, November 16-19, 2004*, volume Special Publication 500-261. National Institute of Standards and Technology (NIST), 2004. `http://trec.nist.gov/pubs/trec13/papers/usheffield.geo.pdf` [Online; accessed 18-September-2016].

[36] I. Gurevych. Anwendungen des semantischen Wissens über Konzepte im Information Retrieval (in German). Technical report, Natural Language Processing Gruppe, EML Research gGmbH, 2004. `http://atlas.tk.informatik.tu-darmstadt.de/Publications/2005/knowledge05.pdf` [Online; accessed 18-September-2016].

[37] J. M. Haake, M. Roos, and T. Schümmer. POSUKO: POrtable SUchKOntexte zur Vernetzung von Bibliotheken, Verlagen und Wissenschaftlern [Poster] (in German), 2014. `https://opus4.kobv.de/opus4-bib-info/frontdoor/index/index/docId/1563` [Online; accessed 18-September-2016].

[38] D. Hawking. Challenges in Enterprise Search. In *Proceedings of the 15th Australasian Database Conference*, volume 27, pages 15–24. Australian Computer Society, Inc., 2004.

[39] S. Helmke and W. Dangelmaier. *Marktspiegel Customer Relationship Management: Anbieter von CRM-Software im Vergleich*. Springer-Verlag, 2013.

[40] W. R. Hersh and D. Hickam. Information Retrieval in Medicine: the SAPHIRE Experience. *Journal of the American Society for Information Science (1986-1998)*, 46(10):743, 1995.

[41] G. Heyer, U. Quasthoff, and T. Wittig. *Text Mining: Wissensrohstoff Text (in German)*. W3L GmbH, ISBN 978-3-937137-30-8, 2006.

[42] O. Hoeber, X. D. Yang, and Y. Yao. Conceptual Query Expansion. In P. S. Szczepaniak, J. Kacprzyk, and A. Niewiadomski, editors, *Advances in Web Intelligence*, volume 3528 of *Lecture Notes in Computer Science*, pages 190–196. Springer Berlin Heidelberg, 2005.

[43] T. Hofmann. Probabilistic Latent Semantic Indexing. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 50–57. ACM, 1999.

[44] C. W. Holsapple and K. D. Joshi. A Collaborative Approach to ONTOLOGY DESIGN. *Commun. ACM*, 45(2):42–47, 2002.

[45] I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From SHIQ and RDF to OWL: The Making of a Web Ontology Language. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(1):7–26, 2003.

[46] H. Huber. Selbstlernende Suche (in German). *Informatik-Spektrum*, 28(3):189–192, 2005.

[47] B. Humm and O. Juwig. Semantische Beratung im Tourismus-Sektor (in German). In *Corporate Semantic Web*, pages 101–110. Springer, 2015.

[48] L. J. Jensen, J. Saric, and P. Bork. Literature Mining for the Biologist: From Information Retrieval to Biological Discovery. *Nature Reviews Genetics*, (9):119–129, 2006.

[49] M. Klebl and B. J. Krämer. Distributed Repositories for Educational Content. *eleed*, 7(1), 2010. `https://eleed.campussource.de/archive/7/2771/index_html` [Online; accessed 18-September-2016].

[50] M. Klettke, M. Bietz, I. Bruder, A. Heuer, D. Priebe, G. Neumann, M. Becker, J. Bedersdorfer, H. Uszkoreit, A. Maedche, S. Staab, and R. Studer. GETESS - Ontologien, objektrelationale Datenbanken und Textanalyse als Bausteine einer Semantischen Suchmaschine. (in German). *Datenbank-Spektrum*, 1:14–24, 2001.

[51] S. O. Krumke and H. Noltemeier. *Graphentheoretische Konzepte und Algorithmen (in German)*. Springer, 2009.

[52] M. Lapp. Intranet-Internes Internet (in German). In *Neue Märkte, neue Medien, neue Methoden-Roadmap zur agilen Organisation*, pages 145–168. Springer, 1998.

[53] N. Leavitt. Will NoSQL Databases Live Up to Their Promise?. *IEEE Computer*, 43(2):12–14, 2010.

[54] V. Levenshtein. Binary Codes Capable of Correcting Deletions and Insertions and Reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

[55] Z. Liu and W. W. Chu. Knowledge-based Query Expansion to Support Scenario-specific Retrieval of Medical Free Text. *Information Retrieval*, 10(2):173–202, 2007.

[56] M. Mariano Fernández López. Overview of Methodologies for Building Ontologies. In *Proceedings of the IJCAI-99 Workshop on Ontologies and Problem Solving Methods (KRR5) Stockholm, Sweden, August 2*, pages 4–1–4–13, 1999.

[57] S. Madden. From Databases to Big Data. *IEEE Internet Computing*, 16(3):4–6, 2012.

[58] A. Maedche, B. Motik, L. Stojanovic, R. Studer, and R. Volz. Ontologies for Enterprise Knowledge Management. *IEEE Intelligent Systems*, 18(2):26–33, 2003.

[59] A. Maedche and S. Staab. Learning Ontologies for the Semantic Web. In *Semantic Web 2001 (at WWW10), May 1, 2001, Hongkong, China*, 2001. `http://www.aifb.kit.edu/web/Inproceedings488` [Online; accessed 18-September-2016].

[60] A. Maedche and S. Staab. Measuring Similarity Between Ontologies. In A. Gómez-Pérez and V. R. Benjamins, editors, *EKAW*, Lecture Notes in Computer Science, pages 251–263. Springer, 2002.

[61] R. Mandala, T. Tokunaga, and H. Tanaka. Combining Multiple Evidence from Different Types of Thesaurus for Query Expansion. In *SIGIR*, pages 191–197. ACM, 1999.

[62] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, UK, 2008.

[63] M. McCandless, E. Hatcher, and O. Gospodnetić. *Lucene in Action*. Manning Publications, 2005.

[64] S. Meier, D. Lütolf, and S. Schillerwein. Anwendungsbereiche eines Intranets (in German). In *Herausforderung Intranet*, pages 55–118. Springer, 2015.

[65] N. Noy, A. Chugh, W. Liu, and M. Musen. A Framework for Ontology Evolution in Environments. *The Semantic Web-ISWC 2006*, pages 544–558, 2006.

[66] N. F. Noy and M. Klein. Ontology Evolution: Not the Same as Schema Evolution. *Knowledge and Information Systems*, 6(4):428–440, July 2004.

[67] N. F. Noy and D. L. McGuinness. Ontology Development 101: A Guide to Creating Your First Ontology. Technical report, Stanford Knowledge Systems Laboratory and Stanford Medical Informatics, 2001.

[68] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web. 1999.

[69] M. P. Papazoglou, P. Traverso, S. Dustdar, F. Leymann, and B. J. Krämer. 05462 Service-Oriented Computing: A Research Roadmap. In F. Cubera, B. J. Krämer, and M. P. Papazoglou, editors, *Service Oriented Computing (SOC)*, number 05462 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2006. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany. `http://drops.dagstuhl.de/volltexte/2006/524/` [Online; accessed 18-September-2016].

[70] S. Pepper. The TAO of Topic Maps: Finding the Way in the Age of Infoglut, 2000.

[71] H. S. Pinto and J. P. Martins. Ontologies: How can They be Built? *Knowledge and Information Systems*, 6:441–464, 2004.

[72] D. Roman, U. Keller, H. Lausen, J. De Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel. Web Service Modeling Ontology. *Applied Ontology*, 1(1):77–106, 2005.

[73] T. Saracevic. Evaluation of Evaluation in Information Retrieval. In E. A. Fox, P. Ingwersen, and R. Fidel, editors, *SIGIR*, pages 138–146. ACM Press, 1995.

[74] A. Singhal. Modern Information Retrieval: A Brief Overview. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 24(4):35–43, 2001.

[75] D. Smiley and E. Pugh. *Solr 1.4 Enterprise Search Server*. Packt Publishing, 2009.

[76] S. Staab. Wissensmanagement mit Ontologien und Metadaten (in German). *Informatik Spektrum*, 25(3):194–209, 2002.

[77] S. Staab, R. Studer, H.-P. Schnurr, and Y. Sure. Knowledge Processes and Ontologies. *IEEE Intelligent Systems*, 16(1):26–34, 2001.

[78] R. Steinbrook. Searching for the Right Search - Reaching the Medical Literature. *New England Journal of Medicine*, 354(1):4–7, 2006. PMID: 16394296.

[79] W. G. Stock. *Information Retrieval: Informationen suchen und finden. (in German)*. Einführung in die Informationswissenschaft. Oldenbourg, München, 2007.

[80] W. G. Stock. Begriffe und semantische Relationen in der Wissensrepräsentation (in German). *Information - Wissenschaft und Praxis*, 60(8):403–420, 2009.

[81] X. Su and T. M. Khoshgoftaar. A Survey of Collaborative Filtering Techniques. *Advances in artificial intelligence*, 2009:4, 2009.

[82] Y. Sure, M. Ehrig, and R. Studer. Automatische Wissensintegration mit Ontologien (in German). In K. Hinkelmann U. Reimer, editor, *Workshop Modellierung fuer Wissensmanagement auf der Modellierung 2006*, Innsbruck, AT, März 2006.

[83] J. Thom and F. Scholer. A Comparison of Evaluation Measures Given How Users Perform on Search Tasks. In *ADCS2007 Twelfth Australasian Document Computing Symposium*. RMIT University, School of Computer Science and Information Technology, 2007.

[84] P. D. Turney and P. Pantel. From Frequency to Meaning: Vector Space Models of Semantics. *Journal of Artificial Intelligence Research*, 37(1):141–188, 2010.

[85] M. Uschold and M. King. Towards a Methodology for Building Ontologies. In *Workshop on Basic Ontological Issues in Knowledge Sharing, held in conjunction with IJCAI-95*, Montreal, Canada, 1995.

[86] A. van der Lans. Enterprise Search and Retrieval (ESR): The Binding Factor. In P. Baan, editor, *Enterprise Information Management*, Management for Professionals, pages 175–211. Springer, 2005.

[87] C. J. van Rijsbergen. *Information Retrieval*. Butterworths, London, 1979. `http://www.dcs.gla.ac.uk/Keith/Preface.html` [Online; accessed 18-September-2016].

[88] P. Vassiliadis. A Survey of Extract-Transform-Load Technology. *IJDWM*, 5(3):1–27, 2009.

[89] V. Venkatesh and F. D. Davis. A Theoretical Extension of the Technology Acceptance Model: Four Longitudinal Field Studies. *Management Science*, 46(2):186–204, 2000.

[90] E. M. Voorhees. Query Expansion Using Lexical-Semantic Relations. In W. Bruce Croft and C. J. van Rijsbergen, editors, *SIGIR*, pages 61–69. ACM/Springer, 1994.

[91] H. Wache, T. Vögele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Hübner. Ontology-Based Integration of Information - A Survey of Existing Approaches. In *IJCAI-01 workshop: ontologies and information sharing*, pages 108–117, 2001.

[92] X. H. Wang, D. Q. Zhang, T. Gu, and H. K. Pung. Ontology Based Context Modeling and Reasoning using OWL. In *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops, 2004.*, pages 18–22. IEEE, 2004.

[93] J. Werrmann. Modellierung im Kontext: Ontologie-basiertes Information Retrieval (in German). Technical report, Department of Mathematics and Computer Science, FernUniversität in Hagen, 2011. `http://deposit.fernuni-hagen.de/2769/` [Online; accessed 18-September-2016].

[94] J. Werrmann. An Advanced Approach to User-based and System-centered Evaluation for the Improvement of Business-oriented Document Retrieval Systems. In *SDPS 2012 Berlin Conference*. Society for Design and Process Science, 2012.

[95] J. Werrmann. Harvesting Domain-Specific Data Resources for Enhanced After-Sales Intelligence in Car Industry. In J. Altmann, U. Baumöl, and B. J. Krämer, editors, *Advances in Collective Intelligence 2011*, volume 113 of *Advances in Intelligent and Soft Computing*, pages 145–167. Springer Berlin / Heidelberg, 2012.

[96] J. Werrmann. Workshop Process Optimization Based on the Collective Intelligence of Workshop Employees Involved in After-Sales Intelligence of Mercedes-Benz Cars. *International Journal of Cooperative Information Systems*, 22(03):1340005, 2013. `http://www.worldscientific.com/doi/pdf/10.1142/S0218843013400054` [Online; accessed 18-September-2016].

[97] T. White. *Hadoop: The Definitive Guide*. O'Reilly, third edition edition, May 2012.

Online-Buchshop für Ingenieure

VDI nachrichten

BUCHSHOP

Online-Shops

Fachliteratur und mehr - jetzt bequem online recherchieren & bestellen unter: www.vdi-nachrichten.com/Der-Shop-im-Ueberblick

Täglich aktualisiert: Neuerscheinungen VDI-Schriftenreihen

Im Buchshop von vdi-nachrichten.com finden Ingenieure und Techniker ein speziell auf sie zugeschnittenes, umfassendes Literaturangebot.

Mit der komfortablen Schnellsuche werden Sie in den VDI-Schriftenreihen und im Verzeichnis lieferbarer Bücher unter 1.000.000 Titeln garantiert fündig.

Im Buchshop stehen für Sie bereit:

VDI-Berichte und die Reihe **Kunststofftechnik:**

Berichte nationaler und internationaler technischer Fachtagungen der VDI-Fachgliederungen

Fortschritt-Berichte VDI:

Dissertationen, Habilitationen und Forschungsberichte aus sämtlichen ingenieurwissenschaftlichen Fachrichtungen

Newsletter „Neuerscheinungen":

Kostenfreie Infos zu aktuellen Titeln der VDI-Schriftenreihen bequem per E-Mail

Autoren-Service:

Umfassende Betreuung bei der Veröffentlichung Ihrer Arbeit in der Reihe Fortschritt-Berichte VDI

Buch- und Medien-Service:

Beschaffung aller am Markt verfügbaren Zeitschriften, Zeitungen, Fortsetzungsreihen, Handbücher, Technische Regelwerke, elektronische Medien und vieles mehr – einzeln oder im Abo und mit weltweitem Lieferservice

Die Reihen der Fortschritt-Berichte VDI:

1 Konstruktionstechnik/Maschinenelemente
2 Fertigungstechnik
3 Verfahrenstechnik
4 Bauingenieurwesen
5 Grund- und Werkstoffe/Kunststoffe
6 Energietechnik
7 Strömungstechnik
8 Mess-, Steuerungs- und Regelungstechnik
9 Elektronik/Mikro- und Nanotechnik
10 Informatik/Kommunikation
11 Schwingungstechnik
12 Verkehrstechnik/Fahrzeugtechnik
13 Fördertechnik/Logistik
14 Landtechnik/Lebensmitteltechnik
15 Umwelttechnik
16 Technik und Wirtschaft
17 Biotechnik/Medizintechnik
18 Mechanik/Bruchmechanik
19 Wärmetechnik/Kältetechnik
20 Rechnerunterstützte Verfahren (CAD, CAM, CAE CAQ, CIM ...)
21 Elektrotechnik
22 Mensch-Maschine-Systeme
23 Technische Gebäudeausrüstung

ISBN 978-3-18-384910-9