MASTER’S THESIS

On Semantic Searching for Municipal Information Services

by

I. Krooswijk

Supervisors:  Drs. L. Hermans (Everest b.v., ’s Hertogenbosch)
              Dr. L. Aroyo (TU/e, Eindhoven)

Amsterdam, July 2006
Abstract

In this thesis a concept is described for an intelligent search engine for the municipal information services of the city of Amsterdam. These services consist of numerous websites for citizens, but e.g. also of an application for the employees of the multi channel call-center.

The proposed concept for intelligent searching is based on free text search, or keyword based search, not aiming for full text retrieval, but for intelligent concept-based searching. To make this possible, advanced language techniques are used for query filtering and -expansion and innovative open standards are used for searching in a developed knowledge base, based on an ontology. The content used for the concept is based on one of the municipal databases where all the public information is stored.

Personalization is used to show the possibilities for searching within the municipal’s knowledge base, based on user specific information and behavior, by using user profiles from the actual nationwide initiative of DigiD. When presenting these ranked results several navigation and presentation options are used to offer the user better guidance when browsing to the wanted information, if available. Also suggestions are provided based on the search behavior of other users.

All this is combined with the current search techniques the municipality is using, offered by the company Verity b.v., the current search engine provider of the municipality. Since this company uses a lot proprietary standards, a lot of investigation was needed to find a good integration of both aiming for better results.

Based on this concept, a software-architecture has been proposed which evolved in a prototype. This prototype has been successfully implemented and functions as a proof-of-concept, which shows that ontology-based knowledge modeling for the municipality of Amsterdam is feasible, and advanced language techniques are helpful for retrieving qualitative better search results. This has been outlined in a comparing test with one of the current municipal information portals.
Acknowledgements

This is the moment I looked out for, for a long time: writing my thesis. This thesis forms the final piece of my study Computer Science, with as main subject Information Systems, at the University of Technology Eindhoven, The Netherlands. I worked for the company Everest b.v. at a project for the municipality of Amsterdam, for intelligent searching. I had a very instructive time, but carrying out a research like this can't be done alone. Therefore I would like to take this opportunity to thank a few people.

First I especially would like to thank Drs. L. Hermans and MSc. M. Mastop from Everest b.v. for their enthusiastic professional guidance and of course for giving me the opportunity and freedom to work on such a project in such an environment for a great company. Also I would like to thank Dr. L.M. Aroyo and Dr. J. Broekstra from the university, for their help in gathering all the knowledge needed to get to the final result.

Furthermore of the municipality of Amsterdam I would like to thank in alphabetic order A. Hof, M. Marghich and I. Noojens for helping me with specific questions regarding the current projects within the municipality.

Last but not least I would to thank M. Voors from Verity for the sessions for fine-tuning the collaboration of my prototype with the software of Verity, and not to forget are all the colleagues at Everest b.v. and the municipality for the nice times I had with them during my period.

Finally I would like to dedicate this thesis to my parents for their support during the years and especially during this final part.
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TABLE OF CONTENTS</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>LIST OF FIGURES AND TABLES</strong></td>
<td>3</td>
</tr>
<tr>
<td><strong>GLOSSARY OF TERMS AND ABBREVIATIONS</strong></td>
<td>5</td>
</tr>
<tr>
<td><strong>1 INTRODUCTION</strong></td>
<td>7</td>
</tr>
<tr>
<td>1.1 CONTEXT DESCRIPTION</td>
<td>7</td>
</tr>
<tr>
<td>1.2 RESEARCH GOALS</td>
<td>8</td>
</tr>
<tr>
<td>1.3 RELEVANCE</td>
<td>8</td>
</tr>
<tr>
<td>1.3.1 Scientific relevance</td>
<td>8</td>
</tr>
<tr>
<td>1.3.2 Business relevance</td>
<td>9</td>
</tr>
<tr>
<td>1.4 DOCUMENT OVERVIEW</td>
<td>9</td>
</tr>
<tr>
<td><strong>2 PROBLEM DEFINITION</strong></td>
<td>10</td>
</tr>
<tr>
<td>2.1 INTRODUCTION</td>
<td>10</td>
</tr>
<tr>
<td>2.2 COMMONKADS</td>
<td>11</td>
</tr>
<tr>
<td>2.2.2 Organization model</td>
<td>12</td>
</tr>
<tr>
<td>2.2.3 Knowledge model</td>
<td>12</td>
</tr>
<tr>
<td>2.3 CURRENT SEARCH TECHNIQUES</td>
<td>13</td>
</tr>
<tr>
<td>2.4 CURRENT PROJECTS</td>
<td>14</td>
</tr>
<tr>
<td>2.4.1 PIGA</td>
<td>14</td>
</tr>
<tr>
<td>2.4.2 KANA</td>
<td>15</td>
</tr>
<tr>
<td>2.4.3 ECM</td>
<td>16</td>
</tr>
<tr>
<td>2.4.4 Conclusion</td>
<td>17</td>
</tr>
<tr>
<td><strong>3 MUNICIPAL METADATA AND CONTENT</strong></td>
<td>18</td>
</tr>
<tr>
<td>3.1 INTRODUCTION</td>
<td>18</td>
</tr>
<tr>
<td>3.2 DOMAIN MODELING</td>
<td>18</td>
</tr>
<tr>
<td><strong>4 PERSONALIZATION</strong></td>
<td>24</td>
</tr>
<tr>
<td>4.1 INTRODUCTION</td>
<td>24</td>
</tr>
<tr>
<td>4.2 USER MODELING</td>
<td>24</td>
</tr>
<tr>
<td>4.2.1 Introduction</td>
<td>24</td>
</tr>
<tr>
<td>4.2.2 Static data</td>
<td>25</td>
</tr>
<tr>
<td>4.2.3 Dynamic data</td>
<td>25</td>
</tr>
<tr>
<td>4.3 MODELING PROCESS</td>
<td>26</td>
</tr>
<tr>
<td>4.4 USAGE OF USER MODELS</td>
<td>29</td>
</tr>
<tr>
<td>4.4.1 Static data</td>
<td>29</td>
</tr>
<tr>
<td>4.4.2 Dynamic data</td>
<td>29</td>
</tr>
<tr>
<td><strong>5 INTELLIGENT SEARCH AND PRESENTATION</strong></td>
<td>32</td>
</tr>
<tr>
<td>5.1 INTRODUCTION</td>
<td>32</td>
</tr>
<tr>
<td>5.2 SEARCH PROCESS</td>
<td>33</td>
</tr>
<tr>
<td>5.3 INPUT VALIDATION</td>
<td>34</td>
</tr>
<tr>
<td>5.4 QUERY PROCESSING</td>
<td>35</td>
</tr>
<tr>
<td>5.4.1 Introduction</td>
<td>35</td>
</tr>
<tr>
<td>5.4.2 Process steps</td>
<td>36</td>
</tr>
<tr>
<td>5.4.3 Applying scores</td>
<td>38</td>
</tr>
<tr>
<td>5.5 FUZZY SEARCHING</td>
<td>40</td>
</tr>
<tr>
<td>5.5.1 Introduction</td>
<td>40</td>
</tr>
<tr>
<td>5.5.2 Via keyword concepts</td>
<td>43</td>
</tr>
<tr>
<td>5.5.3 Via information-objects</td>
<td>44</td>
</tr>
<tr>
<td>5.5.4 Combined results</td>
<td>45</td>
</tr>
</tbody>
</table>
List of Figures and Tables

Figures

Figure 2.1 CommonKADS models.................................................................................................................11
Figure 2.2 PIGA meta model..........................................................................................................................14
Figure 2.3 PIGA to KANA transformation.......................................................................................................15
Figure 2.4 Verity within KANA..........................................................................................................................15
Figure 2.5 Architecture ECM............................................................................................................................16
Figure 3.1 Domain model.....................................................................................................................................19
Figure 4.1 Static part user model.........................................................................................................................25
Figure 4.2 Dynamic part user model....................................................................................................................25
Figure 4.3 Complete user model..........................................................................................................................26
Figure 4.4 Updating dynamic data.......................................................................................................................29
Figure 4.5 Storage user behavior.........................................................................................................................31
Figure 5.1 Formalization concepts......................................................................................................................32
Figure 5.2 Activity diagram query processing global..........................................................................................33
Figure 5.3 Activity diagram query processing detailed......................................................................................35
Figure 5.4 Activity diagram searching................................................................................................................40
Figure 6.1 Kruchten 4+1........................................................................................................................................52
Figure 6.2 Client/server architecture...................................................................................................................53
Figure 6.3 Use Case diagram...............................................................................................................................55
Figure 6.4 Context view.........................................................................................................................................55
Figure 6.5 Component model...............................................................................................................................56
Figure 6.6 UserService component.......................................................................................................................57
Figure 6.7 SearchService component..................................................................................................................58
Figure 6.8 Activity diagram client application.....................................................................................................60
Figure 6.9 Activity diagram administration process............................................................................................62
Figure 6.10 Activity diagram updating triple store..............................................................................................62
Figure 6.11 System diagram...............................................................................................................................63
Figure 6.12 Subsystem diagram Client.................................................................................................................65
Figure 6.13 Subsystem diagram User service.......................................................................................................65
Figure 6.14 Subsystem diagram Search service..................................................................................................66
Figure 6.15 Deployment diagram.......................................................................................................................67
Figures Appendices

Figure A.1 Spectrum of information systems..........................................................83
Figure A.2 Semantic layer cake.............................................................................84
Figure A.3 Simple RDF statement.......................................................................87
Figure D.1 Sequence Diagram Login................................................................132
Figure D.2 Sequence Diagram Logout ................................................................132
Figure D.3 Sequence Diagram Search .................................................................133
Figure D.4 Sequence Diagram Select .................................................................134
Figure E.1 User Interface Login .........................................................................135
Figure E.2 User Interface Query ...............................................................135
Figure E.3 User Interface Results ......................................................................136
Figure E.4 User Interface Keyword Refinement ................................................136
Figure E.5 User Interface Attribute Refinement ................................................137
Figure E.6 User Interface Logout ......................................................................137
Figure E.7 User Interface Initiate New Query ...................................................138
Figure E.8 User Interface Initiate New Query by related keyword ......................138
Figure E.9 User Interface Content .................................................................139
Figure E.10 User Interface Relevant Information .............................................139
Figure E.11 User Interface Suggestions Other Users .......................................139
Figure E.12 User Interface Suggestions Current User ......................................140
Figure E.13 User Interface Viewing history ....................................................140
Figure E.14 User Interface Query History ......................................................140

Tables

Table 3.1 Classes in domain model ......................................................................20
Table 3.2 Types InformationObjectRelations domain model .................................21
Table 3.3 Types KeyWordRelations domain model .............................................23
Table 4.1 Classes in user model ........................................................................27
Table 5.1 Verity transformation scores ..............................................................38
Table 7.1 Categories comparing research ............................................................73
Table 7.2 Analysis comparing research ..............................................................74
Table 7.3 MECCA Alternatives matrix ..............................................................75

Tables Appendices

Table A.1 OWL building blocks ........................................................................89
Table B.1 Dictionary overview ........................................................................92
Table B.2 Content overview ...........................................................................95
Table B.3 KeyWord overview ...........................................................................97
Table B.4 District overview .............................................................................98
Table B.5 Statistics overview ..........................................................................99
Table B.6 Settings overview ..........................................................................100
Table B.7 User overview .................................................................................103
Table B.8 Session overview ..........................................................................108
Table B.9 Preferences overview .....................................................................110
# Glossary of Terms and Abbreviations

<table>
<thead>
<tr>
<th>API</th>
<th>Application Program Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII</td>
<td>American Standard Code for Information Interchange</td>
</tr>
<tr>
<td>AXIS</td>
<td>Container for SOAP based web services for Tomcat</td>
</tr>
<tr>
<td>CMS</td>
<td>Content Management System</td>
</tr>
<tr>
<td>Cluster</td>
<td>Set of information-objects sharing a commonality</td>
</tr>
<tr>
<td>Concept</td>
<td>Name defining an object in a knowledge domain</td>
</tr>
<tr>
<td>Content Type</td>
<td>Type of the content</td>
</tr>
<tr>
<td>DB</td>
<td>Database</td>
</tr>
<tr>
<td>DBMS</td>
<td>Database Management System</td>
</tr>
<tr>
<td>DC</td>
<td>Dublin Core, metadata specification</td>
</tr>
<tr>
<td>DM</td>
<td>Data Model</td>
</tr>
<tr>
<td>DTD</td>
<td>Document Type Definition</td>
</tr>
<tr>
<td>District</td>
<td>One of the 15 city districts</td>
</tr>
<tr>
<td>Domain</td>
<td>Knowledge domain, collection of concepts</td>
</tr>
<tr>
<td>ECM</td>
<td>Enterprise Content Management</td>
</tr>
<tr>
<td>Free Text Search</td>
<td>Keyword based search without using predefined structures over the data source</td>
</tr>
<tr>
<td>Group</td>
<td>Collection of Information-objects</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HTML</td>
<td>Hyper Text Markup Language</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hyper Text Transfer Protocol</td>
</tr>
<tr>
<td>Header</td>
<td>The title of the information-object</td>
</tr>
<tr>
<td>IA</td>
<td>Intranet Amsterdam</td>
</tr>
<tr>
<td>IEEE</td>
<td>Institute for Electrical and Electronics Engineers</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>IR</td>
<td>Information Retrieval</td>
</tr>
<tr>
<td>Information-object</td>
<td>Object representation within PIGA saved and published as unit of content</td>
</tr>
<tr>
<td>Input screen</td>
<td>The page were the user enters the query</td>
</tr>
<tr>
<td>J2EE</td>
<td>Java 2 Enterprise Edition</td>
</tr>
<tr>
<td>JDK</td>
<td>Java Development Kit</td>
</tr>
<tr>
<td>JSP</td>
<td>Java Server Pages</td>
</tr>
<tr>
<td>JRE</td>
<td>Java Runtime Environment</td>
</tr>
<tr>
<td>Kana IQ</td>
<td>Call Center application</td>
</tr>
<tr>
<td>Keyword</td>
<td>Enkelvoudig deel van ingevoerde query</td>
</tr>
<tr>
<td>Metadata</td>
<td>Data about data, e.g.: information about a resource</td>
</tr>
<tr>
<td>NS</td>
<td>Name Space, XML definition</td>
</tr>
<tr>
<td>OWL</td>
<td>Web Ontology Language</td>
</tr>
<tr>
<td>OWL-DL</td>
<td>OWL Description Logic</td>
</tr>
<tr>
<td>PIGA</td>
<td>Publieks Informatie Gemeente Amsterdam</td>
</tr>
<tr>
<td>Portal</td>
<td>Website that aggregates content and integrates applications based on metadata and user profiles</td>
</tr>
<tr>
<td>Portlet</td>
<td>Part of portal for specific functionalities or dynamic content</td>
</tr>
<tr>
<td>RDF</td>
<td>Resource Description Frame</td>
</tr>
<tr>
<td>RDF</td>
<td>Resource Description Framework, metadata specification</td>
</tr>
<tr>
<td>RDFS</td>
<td>RDF Schema</td>
</tr>
<tr>
<td>RDFS-DL</td>
<td>RDF schema, metadata specification</td>
</tr>
<tr>
<td>RDQL</td>
<td>RDF Data Query Language</td>
</tr>
<tr>
<td>ROI</td>
<td>Return On Investment</td>
</tr>
<tr>
<td>RQL</td>
<td>RDF Query Language</td>
</tr>
<tr>
<td>Ranking</td>
<td>Sorting of the result set based on relevance</td>
</tr>
<tr>
<td>Relation</td>
<td>A semantical link between two domain concepts</td>
</tr>
<tr>
<td>Relevance</td>
<td>The weight an information-object gets based on scores from query processing and relation weights in the ontology</td>
</tr>
<tr>
<td>Resource</td>
<td>Object that can be described with metadata</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------------------------------------------------------</td>
</tr>
<tr>
<td>Result screen</td>
<td>The page where the system shows the results</td>
</tr>
<tr>
<td>SOA</td>
<td>Service Oriented Architecture</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SSO</td>
<td>Single Sign On.</td>
</tr>
<tr>
<td>SW</td>
<td>Semantic Web</td>
</tr>
<tr>
<td>SeRQL</td>
<td>Sesame RQL</td>
</tr>
<tr>
<td>SeRQL-C</td>
<td>Sesame RQL Construct</td>
</tr>
<tr>
<td>SeRQL-S</td>
<td>Sesame RQL Select</td>
</tr>
<tr>
<td>Search Machine</td>
<td>Application that offers the possibility to search for information using search attributes</td>
</tr>
<tr>
<td>Servlet</td>
<td>Application running on Tomcat server</td>
</tr>
<tr>
<td>Spider</td>
<td>Spiders are applications that are able to collect data (crawler) over a network and indexes (indexer) it for search machines</td>
</tr>
<tr>
<td>TU/e</td>
<td>Eindhoven, University of Technology</td>
</tr>
<tr>
<td>Tomcat</td>
<td>Web/application server specific for Java Server Pages and Java Servlets</td>
</tr>
<tr>
<td>UM</td>
<td>User Model</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Identifier</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator, part of URI</td>
</tr>
<tr>
<td>UUP</td>
<td>Unified User Profile</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium</td>
</tr>
<tr>
<td>WIS</td>
<td>Web-based Information System</td>
</tr>
<tr>
<td>WSRP</td>
<td>Web Services for Remote Portlets</td>
</tr>
<tr>
<td>WWW</td>
<td>World Wide Web</td>
</tr>
<tr>
<td>Weight</td>
<td>The weight that a relation has between two concepts</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
<tr>
<td>XML Schema</td>
<td>XML Schema</td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Context description

The main goal of the municipality of Amsterdam (Amsterdam) is to get a better and faster distribution of their municipal information. The way they want to use this information and knowledge, within all its web portals, is to offer its citizens, its local institutions as well as the employees of its multi channel call-center a service to present reliable answers to their questions.

Seen the fact that more and more practice examples of semantic web solutions arise worldwide [1], Amsterdam wants to join this hype and develop a prototype that, on the basis of semantic techniques, offers a smarter solution for keyword based searching within its web-portals. Considering the statistics [2] 80% of all users use the free text search options, instead of the other ones available. Therefore this needs extra attention, especially since Amsterdam aims to use knowledge engineering with respect to their distributed systems and their communicating agents. In paragraph 1.2 the specific research questions are presented in more detail.

Guided by the prototype, described in this thesis, the goal is to present a prove that the intelligent prototype offers much more smart functionalities, so consequently a decision can be made whether the so-called “return on investment” (ROI) is worthwhile taking the risk. To achieve this, the intelligent prototype and the current solution are both taken into account within a comparing research.

In the prototype is not only watched to the contact center application but also to some other information services. The prototype generalizes over these so the solution presented will be more generic and its several features can be implemented in all the current systems. This is also the reason why not the whole process has been taken into account of the call center employees. Steps as answering the phone, clarifying the question and answering the question are omitted and there only has been zoomed in to part of searching with the question and finding the answer.

Paragraph 1.4 outlines how the rest of this thesis is organized to tackle this problem systematically. The other paragraphs of this introduction present more background information needed to understand the current situation and vision of Amsterdam.

The presented concept and prototype will give other researchers or developers for different domains a good base for designing an intelligent search option in their applications. Also the current developments that are used for implementing the search option are interesting for growing flexible environments like the one of Amsterdam. The relevance for Amsterdam is clear with extra attention to the future where the open standards provide enough transparency to be used as a generic format for all the distributed systems so these can easily be integrated.
1.2 Research goals

Knowing about the problems of the municipality, several research topics can be addressed to realize the wanted search concept and its accompanying proof-of-concept. These concern modeling the content. Innovative open standards are used which need to be investigated. For personalization the users need to be modeled. Furthermore linguistic techniques for keyword processing and specific algorithms for intelligent searching need to be investigated. Finally will be shown how the prototype contains these aspects. Therefore this thesis handles one global question:

- "Is it possible to provide more intelligence for searching for- and presenting municipal information, and can this be shown in a prototype search engine?"

In more detail the research questions and goals are the following:

- Investigate how innovative open standards for metadata specifications can be used for content and knowledge modeling.
- Investigate how users can be modeled in the system and how DigiD can be involved in this.
- Investigate personalization techniques based on user profiles, preferences and behavior.
- Investigate how open standards can be combined with the proprietary standards of the current search technology Verity.
- Investigate how linguistic techniques can be combined with the proprietary search techniques of the current search technology Verity.
- Investigate the possibility to exhaustively use an ontology for presenting better navigation options in the search results for faster information retrieval of related resources.
- Design a system with the proposed content modeling techniques, personalization techniques and search techniques.
- Implement a proof-of-concept based on this design.
- Evaluate the proof-of-concept in a comparing research.
- Implement an administration solution.

1.3 Relevance

1.3.1 Scientific relevance

The scientific relevance is about combining conventional search techniques and innovative open standards to a combination specifically usable for an organization like the municipality of Amsterdam. The prototype developed for this domain can be used as a foundation for all kind of organizations, since the proposed methodology is generic.

In addition this research helps to provide an insight into the current developments in the research area of search techniques using leading edge technologies. Currently more and more organizations are planning to use semantic web technologies for organizing their information. Furthermore also more and more organizations are implementing service oriented architecture (SOA) based solutions to become more flexible. Therefore both aspects are very important in this thesis. In combination with a conventional search technique used by Verity, this is a first step to grow to a flexible structure of the information management by the municipality. This will provide a proper foundation for other researchers to further refine the search method, enrich the knowledge base and especially rebuild and improve the part Verity is currently providing.
1.3.2 Business relevance

- Amsterdam

The business relevance for Amsterdam is mainly the faster information retrieval for its call-center agents to reduce costs when handling calls. Furthermore also this advantage holds for the citizens of the municipality by presenting a good look-and-feel when searching and navigating for the wanted information. Another important business goal when using these new techniques is the position of Amsterdam related to other municipalities. Especially within the four most important cities in The Netherlands (G4) Amsterdam will get a leading role when it comes to integrating new techniques in their information services.

- Everest

This research is also for Everest of importance. It allows them to develop new technologies based on the open semantic standards. Not only for the search use case, but also for other purposes these techniques can be used to improve the organization of knowledge and help customers to use it in such a way it reduces costs. The new techniques will increase Everest’s market reach by being able to introducing these new techniques for building their knowledge bases and also for using the specific search option to implement in their current applications for faster information retrieval. Open standards imply lower costs for their customers and, because of the faster information retrieval, lower costs for agents working with the software.

1.4 Document overview

For further reading here the overview of the rest of the document is presented. After analyzing the current situation, projects and identifying the current problems in chapter 2, the modeling of the domain information is presented in chapter 3. In chapter 4 subsequently the modeling of the users is outlined.

Both parts are needed to provide the background information for the presented search concept in chapter 5. Respectively in chapter 6 is described how, based on the current situation, requirements and proposed concept the prototype will be built. Here the complete architecture is described that is used to cover all aspects of the proof-of-concept. Here also implementation issues are handled. In chapter 7 the presented solution will be compared with the solution from the municipality.

The conclusions and recommendations concerning the research are listed in chapter 8. This is where the research questions will be answered. All sub-questions are treated by the chapters throughout the thesis.
2 Problem definition

2.1 Introduction

Amsterdam wants to find out if there exists a possibility to apply a more intelligent search option to their domain information. They experience a knowledge problem within their knowledge domain, when performing a specific task, which is searching for information. From this point of view there has to take place a so-called knowledge management research for deploying a knowledge base to solve this problem. A knowledge base is a system which exists of domain elements and the conventional domain knowledge currently available for the data. It is the knowledge over the data that makes it possible for agents, like persons of software systems, to make correct decisions. This is why knowledge management is a key to success here.

The method for solving this problem this will be described using the CommonKADS [3] method. CommonKADS is an abstract set of models used as a guideline for analyzing the knowledge problems within the municipality. It is some sort of de-facto method defined by a number of models that are used to describe the context, the concepts and the design of the system. For the goal of this project, the paradigm of CommonKADS is partly used, and only as a guideline, since the technical specifications are not used. A short overview is described in paragraph 2.2 where also will be explained what parts of CommonKADS are used here. How the knowledge base is finally set up will be described in chapter 3. The main task, searching in the knowledge base, is explained in detail in chapter 5.

As has been told there are multiple instances and websites, currently running within the municipality, that are experiencing knowledge problems. Currently there are three main projects that need extra attention. Therefore these are analyzed and investigated for what possibilities exist to improve them and build the functionalities needed by all of them. That is why after analysis the main requirements are listed. Based on these requirements it becomes possible to work towards the techniques for the wanted solution and create the architectural design for the all-embracing prototype.

Initially the systems PIGA and KANA are the most interesting for this research. Next to these there is a project oriented for the future were a completely new Enterprise Content Management (ECM) system will be built, which complies with the vision of the municipality, which will finally be the only place for information retrieval. This project is very interesting, especially regarding the functionalities for intelligent searching and their data resources and their structure. This project is most likely to obtain the new search task and the proposed knowledge solutions in the future. For these reasons all three projects are taken into account.

In paragraph 2.3 and 2.4 the current search techniques and all three projects are outlined. The working of the search techniques for as well the back-end as the front-end will be handled in detail for gaining a better view when it later will be used together with the new data structure. Furthermore for each project its data resources are outlined and will be specified how its data is organized and usable for the prototype together with other aspects that were reckoned with.
2.2 CommonKADS

CommonKADS offers the possibilities to execute some knowledge acquisition and modeling that are needed for setting up a knowledge base with respect to intelligent searching. Before elaborating the search method first needs to be analyzed whether the usage of a knowledge base is wanted or not and how is can be used. In Figure 2.1 the complete model-set is presented, where the models used are highlighted.

First a short explanation is given for all models.

- **Organizational model**
  The organizational model provides an analysis of the environment in which the knowledge base has to function. It defines the bottlenecks to be solved, the functions and the solutions for the future agents and their tasks.

- **Task model**
  The task model specifies how the organizational function is achieved by certain system tasks and activities.

- **Agent model**
  The agent model describes relevant features of the agents that are executing the tasks from the task model.

- **Knowledge model**
  The knowledge model outlines the knowledge of an agent relevant to a task, where the knowledge and its structure is described as domain-, task- and inference knowledge

- **Communication model**
  The communication model describes the transactions that are communicated between the agents, wherefore additional tasks need to be taken into account

- **Design model**
  The design model specifies the realization of the problem in computational and representational terms and distinguishes between application, architecture and platform design

As has been explained in the paragraph 2.1 not the complete pyramid of models is used here, but only the most relevant models for clarifying the process of developing the knowledge base. So here only the guidelines for the organization model and the knowledge model are used as a grip for the rest of this thesis. The main goal of the organization- and knowledge analysis is to show how the results of the knowledge analysis can be used to solve the problems that were identified during the organization analysis and to show why the presented solution can be of value for the municipal.
2.2.2 Organization model

The goal of the organization model is to describe the current situation, to identify the problems and to describe the wanted situation. This part is complemented with suggestions for improving the current problems. Specifically for the current problem the knowledge model will be analyzed. That is the reason why both are connected in the pyramid.

The first step is analyzing the current processes within the domain and the knowledge that is needed for it. Together these form the current situation and are described in paragraph 2.3 and 2.4.

The knowledge problem, the municipality currently has, is that the intelligence within searching for content-items is not optimal. Therefore Amsterdam has as goal to use an ontology (see appendix A). The task to be modeled within the knowledge model therefore is the classification of the domain knowledge. The reason for solving this problem is to allow citizens to retrieve the information they are looking for faster. The same holds for the call center agents where also the money aspect is of importance. The municipality itself acts as the domain expert who provides all needed information about their current processes (projects).

So the wanted situation will be described in the form of how the search process should perform based on the new knowledge base. All these steps are described in chapter 3 and 5.

2.2.3 Knowledge model

As has been mentioned in the previous paragraph the goal is to develop a knowledge base where all domain knowledge is well structured, in the form of an ontology, with the purpose to improve the search option. Therefore a knowledge model will be modeled. This serves as a base for the search engine. What happens after this step with the knowledge model depends on the chosen knowledge based solution. For the search option a design model needs to be created, so the proof-of-concept can be implemented.

So the knowledge model is the tool for the specification of such a knowledge-intensive task. It is oriented on real-world domain aspects were reuse plays a central role. Furthermore its goal is to give an implementation-independent description of knowledge involved in the task. Knowledge in this case is just semantically rich information (“information about information”). In chapter 3 is outlined how the complete knowledge base is set up.

There are three components of which the domain model exists, which are the domain component, the task component and the inference component. First the domain component describes the relevant domain knowledge and information. The task component is goal-oriented and specifies a functional decomposition of the task to be executed. Finally the inference component provides the basic reasoning steps that can be made in the domain knowledge and are applied by tasks.

Task component

The task component presents a domain independent specification of the task to be executed. Therefore the search task will be specified using an activity diagram. The search task with the accompanying activity diagram can be found in chapter 5. The input for the search task consists of the information concepts and the output is a subset of these information concepts, which fulfill the search query.

Domain component

The domain component describes the concepts that exist within the domain. Also relations between these concepts are described. It consists of the domain scheme, in which a schematic description of knowledge and information types are given and the knowledge base which consists of the knowledge instances. The domain component and the inference component are closely related. Therefore also the mappings between domain knowledge and inference knowledge are given. As mentioned earlier, in chapter 3 the process of modeling the domain knowledge is presented.
Inference component

The inference component describes the lowest level of functional decomposition of the knowledge and is closely related to the domain knowledge. Using inference knowledge the knowledge system is able to reason about the information and to infer new information or relations. Here a simple form of inference is used to infer inverse relations. Since concepts from the domain component are linked to other concepts by a specific relation, there also exist relations the other way around by inverse relations. Since the knowledge base is not overdone, this is the only way in which inference knowledge is used here. Since its closeness to the domain knowledge, it is also explained in chapter 3 in the passage about relations.

2.3 Current Search Techniques

Verity [4] is the search technology the municipality currently uses. With its K2 software [5] it offers the current techniques for searching within the municipal's data sources.

Verity uses an index service with a spider to index these data sources to create collections that afterwards are searchable for matching concepts. These collections provide full-text indexes of the information resources, including several types of data formats. Both the service and the collections can be administered by the web based administration tool Verity offers. Furthermore Verity uses a powerful query language to search through these collections. This is primarily based on the so-called query parser, which has to be told how a query should match the content and how scores must be applied to the match. This will be outlined in more detail in paragraph 5.4, where the search process is outlined.

When using Verity's search engine in an application one can use a C++ or Java API to contact the search service on a specific URI and retrieve data. This API makes it possible to specify how the query parser should perform in order to get the correct results with the appropriate scores.

A notification needs to be made here that in the prototype these searching techniques are not used for finding the real content, but for finding related keywords to enrich the query. Since as back-end a knowledge base is developed here, Verity will not be used for searching information-objects, but this will be done using new standards using advanced query languages for data retrieval. So only the keywords are searched for by Verity. This was the reason that one of the research goals was to find out how both would be able to work together. The distinction has been outlined above.

Also numerous parameters can be set on how to group/sort the result set from Verity. Since only related keywords are retrieved the order is not important here, but only the relevance score of the found keyword to the entered keyword. Other techniques that Verity offers, like full text retrieval [6] with e.g. text highlighting for matching objects, is also completely left out.

Now the working of the Verity search engine is clear the current processes of the municipality are outlined in the following paragraph, where also the role of Verity will be pointed out.
2.4 Current projects

2.4.1 PIGA

PIGA [7] is the system where mainly all product and address data of the municipality is stored. It consists of a set of information-objects with mutual typed relations. Furthermore PIGA also contains a thesaurus where all keywords are stored in. These also have mutual typed relations and also contain linguistic variants that are related to base keywords.

Between those two parts relations exist where each keyword is linked to an information-object and vice versa. In the scheme in Figure 2.2 the complete classification of all data in PIGA is presented:

The keywords from the thesaurus make sure the user will end up with the correct related information-object. Input keywords matched to keywords from PIGA and related information-objects are presented to the users result screen.

When considering using PIGA as a data source for proof-of-concept a bid advantage is that it already consists of a rich structure with typed relations, which can be used very well for navigation options in the result screen of the prototype. It not only uses see-also relations, but also relations like “is-member-of” or “is-director-of”.

These relations can help the user gain knowledge about the domain and move faster through the results to get to the wanted information. Also for a contact centre employee it might come in handy to know why certain information-objects are related. So this will actually be a requirement with respect to the intelligent search prototype.
2.4.2 KANA

KANA [8] is the application that especially has been designed for the contact center employees of the municipality. It consists of several parts where KANA IQ is the one importance here. The goal of KANA IQ is to support the employees when answering the questions of the citizens. Therefore it is expected from the system that it offers several methods for searching and finding the right information. The data in KANA consists of list of frequently asked questions with their related answers. This data is enriched with the answer information-objects from PIGA, and therefore KANA exists of much more usable information than PIGA. Although this seems to be an advantage, it actually is a disadvantage. There is no real good overview of what exactly from PIGA is transferred to KANA and also the fact that KANA uses a proprietary format for exporting its content is not very helpful. The integration process of PIGA to KANA works as depicted in Figure 2.3:

![Figure 2.3 PIGA to KANA transformation](image)

The export data format PIGA will be transformed to the import data format of KANA IQ. Also the information-objects in KANA IQ are changed so that they are a one-to-one copy of the PIGA object. But in KANA they will fit into a different structure. In this structure all relations that are described in PIGA are not available anymore. All typed relations are flattened to become a “see-also” relation, and lose their semantics. This is also a disadvantage when considering using KANA for the prototype. Also the thesaurus part and its semantics are lost which are required for the intelligence within the search process of the system.

Furthermore KANA IQ uses an adapted Verity component for searching in its knowledge base with its proprietary data format AOB. When no results are found it is possible to search in other data sources as well. At that point the query is send to the common Verity engine that gathers the information from the external data sources. Verity itself makes sure the indexing and searchability of the other data sources is done correctly.

![Figure 2.4 Verity within KANA](image)
2.4.3 ECM

ECM [9] is the project name of the portal system that the municipality is planning to use within the next year. It is a portal system in which random websites or portals can be build using the BEA portal manager. Each of these sites exists of portlets, in where specific functionality can be added. The big advantage is here that by using this portlet architecture different publications can be made based on the same content. This content is saved in the BEA CSM in again a different uniform structure for all data sources from where the ECM is gathering its data. This structure or metadata is flexible and extendible.

There has taken place a push of data from PIGA to the ECM and an analysis of the PIGA data in ECM form. Here also the PIGA data is converted perfectly and also the links between the keywords and the information-objects and vice versa are saved, but the mutual relations between the information-objects are not saved and these are necessary on behalf of the presentation and navigation of the search results.

In the ECM a few aspects are very important related to the research. First the search option should fit in one portlet, or it should be divided in such a way that it can be distributed over a number of portlets. Also in the future more and more personalized actions must be executable via the websites.

The easiest way to search for information in the view of ECM is searching by metadata. Although the combination to search for metadata in combination with free text search. For both options a portlet can be built which can be used freely. Searching will play a role on all portals. It must be possible to perform keyword based search, but also to search more advanced on the basis of specific characteristics, like the metadata. In the scheme below the architecture of the ECM is presented. From a CMS like PIGA the content is pushed to the BEA. Verity indexes this content and enriches it with keywords and saves it to the BEA repository. The portlets described above on their turn have access to the repository.

One disadvantage of ECM is that it currently communicates with the portal system via HTTP requests. A JSP application is called with as parameter the search query, from where Verity returns the result set in XML format. A requirement by Amsterdam is that these should also communicate via a web service at the Verity side, where the query and the results are sent via SOAP.
2.4.4 Conclusion

After the analysis of the different projects it is possible to determine how the search concept and the prototype are going to be set up globally. Advantages and disadvantages of all three projects and Verity are taken into account in such a way that the concept for intelligence searching and its accompanying prototype will contain all aspects needed by all three projects.

A requirement by Amsterdam is that Verity should do especially what it is intended for: searching. All other functionalities can be done by other software. That is why the intelligent search option adheres as much as possible to the Verity technology. Verity therefore is used for linguistic techniques provided by the API for finding keyword concepts. Using these concepts the knowledge base will be searched for related information-objects.

Considering this knowledge base for the proof of concept the metadata from PIGA is used as a base (see chapter 3). As mentioned before, the data export from KANA still containing all rich relations would be more appropriate, but is not available. Also ECM is going to loose all these relations. Therefore simply PIGA is used as data back-end. PIGA offers for the prototype enough data to work properly and show the necessary results. Currently the structure of PIGA is not modeled as an ontology, so therefore later the administration process makes sure the back-end is transformed in an ontology for better usage of its semantics. This is shown in paragraph 6.6.3 were the administration process is presented. Although there is still the option to add the frequently used questions list from KANA it does not add extra possibilities and therefore is left out here.

An extra advantage for using PIGA is that its metadata and data are easy accessible and readable. This benefits for the administration process of the prototype. This way the content can stay on each individual server within Amsterdam and only the keywords are read into the Verity collections for searching. The generated ontology is only the layer around the content where all relations and other metadata are stored. This principle above totally fits in the view of ECM where each system controls its own content and one central system gathers this information for central information retrieval.

It would be the task of ECM to create a more sophisticated generic list of keyword concepts, which can be used by all systems. Even better is to create one dictionary which is used everywhere. Since Amsterdam has the ambition to be a municipality above the average, agreements need to made with other cities in The Netherlands like the G4 (Amsterdam, Rotterdam, The Hague, Utrecht) so they can get a very sophisticated dictionary with lots of rich relations, which also benefits the controllability by a separate external instance. The Gov United [10] initiative can adopt these ideas to create a central information service.

Technically the proof of concept actually represents in some form the Verity component in Figure 2.5. Although in the prototype it is enclosed in a web service, since this is a future goal of Amsterdam.

At this point, after analyzing the current domain and the most important applications running in it, the exact requirements for the concept and the prototype can be defined. Regarding these requirements [11] there exist different solutions with a mix-up of aspects that need to be chosen for using in the prototype. The most important requirements resulting from the analysis and also from the domain experts are listed below. Their id’s are identical to the ones in [11] and are based on the specification in [12]. In the rest of the document will be shown how these requirements are met in the search concept and the prototype.

- [REQ-VISC-FC-001] Improving the intelligence of the free text search option
- [REQ-VISC-FC-005] Using open standards by W3C for semantic web and appropriate programming/scripting techniques
- [REQ-VISC-FC-006] Reusing the data and relations described in PIGA
- [REQ-VISC-FC-007] and [REQ-VISC-FC-008] Collaboration with current search engine Verity
- [REQ-VISC-NFC-019] Practical applicability in technical and infrastructural architecture of the municipality
- [REQ-VISC-FC-013] and [REQ-VISC-FC-014] Administration solution for automatically updating the ontology
3 Municipal metadata and content

3.1 Introduction

The domain model describes the main concepts within the domain of the municipality. It describes the main entities and their relationships. Furthermore it especially uses the vocabulary of the domain to be easy understandable for its stakeholders. The domain model here is created and described using ontologies (see appendix A). Together with the instances of the concepts it forms a knowledge base. The knowledge engineering method presented in [13] defines a strategy for building up such a knowledge base from scratch. After analyzing the current situation in chapter 2 and describing the metadata languages to be used in appendix A, the described strategy is taken as an inspiration to model the domain for the prototype as an ontology together with the instances of its classes using a meta data language. The main reasons for using an ontology here is to share the structure of the information between software agents and the re-use of the domain knowledge for the multiple systems of the municipality. [13] also points out that developing an ontology is actually the same as defining a set of data and their structure for other programs to use. When creating the ontology for Amsterdam it is therefore important to reckon with the systems that are going to use the ontology and adjust the ontology for optimal usage by these applications. The knowledge base must be well semantically organized and normalized [70] and may not contain redundant information for that reason. The other important aspect is that the knowledge base should be build up in such a way that parts of the domain knowledge can easily be reused elsewhere in the municipality and also can be replaced or renewed with similar parts.

The ontology is built from scratch with the help of the existing meta model of PIGA, as pointed out in paragraph 2.4.4. For each information-object and keyword concept the metadata and relations are extracted from where the ontology was generated. The actual content is not extracted but is callable via a unique id that exists in the metadata. Later in the architecture will be shown where the ontology fits in the whole of the information retrieval process. How the administration process works together with the ontology can be found in paragraph 6.6.3. The point where the search process uses the ontology is also described in paragraph 6.6.2. The next paragraph shows how the strategy is used in this case to create the knowledge base.

Examples of concepts here are presented in the chosen metadata language, namely OWL (see conclusion appendix A), to directly present a short overview of how such description languages look like. The biggest advantage, when using metadata languages, is to share common understanding of the structure of municipal information among citizens, call center employees but especially the software agents. So a well defined metadata specification is required here.

3.2 Domain modeling

An ontology together with its set of individual instances of classes constitutes a knowledge base. The steps used for developing the knowledge base, derived from [13], adjusted and expanded to be applicable here, are the following:

- Define the ontologies

Especially regarding maintainability and reusability, and possible future usage of other ontologies, the knowledge base is divided in several ontologies that are combined via mappings. When analyzing PIGA (see paragraph 2.4.1) it becomes clear, there exists a clear separation between a thesaurus of keywords and a taxonomy of information-objects within the domain. Here these are stored separately in different ontologies, which enables ontology based reasoning. The relations between both are expressed via a mapping. How each separate ontology is build up, will be outlined in the following steps.
For searching domain concepts in the search engine, all content items of PIGA are classified into a hierarchy. Also the thesaurus with keywords is imported in the ontology where each instance of a keyword concept will point to an information-object. To enable spatial reasoning of the information-objects, each information-object is linked to a specific district of the municipal. This way the system can detect more relevant information for the user based on that person’s own district. See Figure 3.1.

![Domain model](image)

**Figure 3.1 Domain model**

Explanation building blocks:

- In Dictionary each keyword concept is mapped to an information-object concept and vice versa.
- In Content all information-object concepts and -instances are stored
- In KeyWords all keyword concepts and -instances are stored
- In Districts all district concepts and -instances are stored

As mentioned above the reason for splitting the whole up like this is that it becomes easier to reuse existing ontologies for some parts (like the keyword thesaurus) or to replace specific parts.

- **Define classes (concepts) per ontology and arrange classes in a (subclass–super class) hierarchy**

When defining classes for each (sub-)ontology, it is important to realize what the exact scope of the ontology is. To be able to determinate whether the ontology is able to handle all requests some competence questions [15] are formulated that must be covered by the ontology. Some example questions are the following:

- What are the related keywords to the keyword with a specific id?
- What are the information-objects within a specific content type?
- What are the information-objects, published after a specific date?
- What are the information-objects from a specific district?
- What are the information-objects from a specific district and its neighboring districts?
- What are the related information-objects to the keyword object with a specific id?
- What are the related information-objects to the information-object with a specific id?

Taking these questions into account in Table 3.1 the classes are defined per ontology. In the following step for each concept the properties and restrictions are described in OWL format, since it is used this way as described in appendix A. In appendix B1 the complete domain model is explained in detail.
Define properties and restrictions for the classes per ontology

Per ontology and per class the properties are presented that represent a relation to another resource in the ontology. For each relation its domain and range are presented.

Content

Each InformationObject is of a specific type of class (Table 3.1) and has a unique id, a label and several properties restricted by a datatype, which can be inspected in the value the specific property has. Below an example is presented of an InformationObject of the type “Instelling” which e.g. also has a so-called object property “Omvat”. The value of the triple (see appendix A) is not a simple datatype, but an object of a specified class. Furthermore the HasWeight property is explained for dictionary and keyword and the same holds here as in keyword. In appendix B2 the complete overview of all classes and properties are presented.

```
<owl:Class rdf:ID="InformationObject">
    <rdfs:label xml:lang="nl">informatie object</rdfs:label>
</owl:Class>

<owl:Class rdf:ID="Instelling">
    <rdfs:label xml:lang="nl">instelling</rdfs:label>
    <rdfs:subClassOf rdf:resource="#InformationObject" />
</owl:Class>

<owl:ObjectProperty rdf:ID="InformationObjectRelation">
    <rdfs:label xml:lang="nl">informatieobject relatie</rdfs:label>
    <rdfs:domain rdf:resource="#InformationObject" />
    <rdfs:range rdf:resource="#InformationObject" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="Omvat">
    <rdfs:label xml:lang="nl">omvat</rdfs:label>
    <rdfs:subPropertyOf rdf:resource="#InformationObjectRelation" />
    <owl:inverseOf rdf:resource="#MaaktDeelUitVan" />
    <HasWeight rdf:datatype="&xsd;decimal">0.66</HasWeight>
</owl:ObjectProperty>

<Instelling rdf:ID="IOB_35283">
    <rdfs:label xml:lang="nl">Zwem- en Sportvereniging Cormeta</rdfs:label>
    <ContentId rdf:datatype="&xsd;positiveInteger">35283</ContentId>
    <DistrictName rdf:resource="&dist;DIS_7" />
    <PubDate rdf:datatype="&xsd;positiveInteger">20050107</PubDate>
    <NrViews rdf:datatype="&xsd;positiveInteger">51</NrViews>
    <Omvat rdf:resource="#IOB_40664" />
    <MaaktGebruikVan rdf:resource="#IOB_38208" />
    <MaaktDeelUitVan rdf:resource="#IOB_38585" />
</Instelling>
```
In Table 3.2 the object properties that are defined for InformationObjectRelation are presented.

<table>
<thead>
<tr>
<th>Property</th>
<th>Inverse property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Omvat</td>
<td>MaaktDeelUitVan</td>
</tr>
<tr>
<td>Behandelt</td>
<td>WordtBehandeldDoor</td>
</tr>
<tr>
<td>HeeftAlsLid</td>
<td>IsLidVan</td>
</tr>
<tr>
<td>HeeftAlsVoorzitter</td>
<td>IsVoorzitterVan</td>
</tr>
<tr>
<td>IsSecretarisVan</td>
<td>HeeftAlsSecretaris</td>
</tr>
<tr>
<td>WerktBij</td>
<td>HeeftInDienst</td>
</tr>
<tr>
<td>MaaktGebruikVan</td>
<td>WordtGebruiktDoor</td>
</tr>
<tr>
<td>AuteurVan</td>
<td>GeschrevenDoor</td>
</tr>
<tr>
<td>Organiseert</td>
<td>GeorganiseerdDoor</td>
</tr>
<tr>
<td>HoudtVergadering</td>
<td>GehoudenDoor</td>
</tr>
<tr>
<td>LeverancierVan</td>
<td>GeleverdDoor</td>
</tr>
<tr>
<td>IsHoofdVan</td>
<td>HeeftAlsHoofd</td>
</tr>
<tr>
<td>IsDirecteurVan</td>
<td>HeeftAlsDirecteur</td>
</tr>
<tr>
<td>Vereist</td>
<td>NodigVoor</td>
</tr>
<tr>
<td>Beschrijft</td>
<td>WordtBeschrevenIn</td>
</tr>
<tr>
<td>StaatBoven</td>
<td>ValtOnder</td>
</tr>
<tr>
<td>VindtPlaatsIn</td>
<td>LocatieVoor</td>
</tr>
<tr>
<td>InformatieOver</td>
<td>InformatieBij</td>
</tr>
<tr>
<td>ChefVan</td>
<td>HeeftAlsChef</td>
</tr>
<tr>
<td>IsCommissarisVan</td>
<td>HeeftAlsCommissaris</td>
</tr>
<tr>
<td>IsBestuurderVan</td>
<td>HeeftAlsBestuurder</td>
</tr>
<tr>
<td>HeeftAlsPlaatsvervangendLid</td>
<td>HeeftAlsPlaatsvervangendLidVan</td>
</tr>
<tr>
<td>ZieOok</td>
<td>ZieOok</td>
</tr>
<tr>
<td>StadsdeelinformatieOver</td>
<td>StadsdeelinformatieBij</td>
</tr>
<tr>
<td>IsGassocieerdMet</td>
<td>IsGassocieerdMet</td>
</tr>
<tr>
<td>WettelijkeBasisVoor</td>
<td>GebaseerdOp</td>
</tr>
<tr>
<td>Beheert</td>
<td>WordtBeheerdDoor</td>
</tr>
</tbody>
</table>

Table 3.2 Types InformationObjectRelations domain model
Dictionary

The mapping dictionary maps keyword concepts to information-object concepts and vice versa. Below the two relations are presented as they were modeled in the ontology.

```xml
<owl:ObjectProperty rdf:ID="KeyWordInformationObjectRelation">
  <rdfs:label xml:lang="nl">relatie trefwoord en informatie object</rdfs:label>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="IsKeyWordOf">
  <rdfs:label xml:lang="nl">is trefwoord van</rdfs:label>
  <rdfs:domain rdf:resource="&keyw;KeyWord" />
  <rdfs:range rdf:resource="&cont;InformationObject" />
  <rdfs:subPropertyOf rdf:resource="#KeyWordInformationObjectRelation" />
  <owl:inverseOf rdf:resource="#HasKeyWord" />
  <HasWeight rdf:datatype="&xsd;decimal">1.00</HasWeight>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="HasKeyWord">
  <rdfs:label xml:lang="nl">heeft trefwoord</rdfs:label>
  <rdfs:domain rdf:resource="&cont;InformationObject" />
  <rdfs:range rdf:resource="&keyw;KeyWord" />
  <rdfs:subPropertyOf rdf:resource="#KeyWordInformationObjectRelation" />
  <owl:inverseOf rdf:resource="#IsKeyWordOf" />
  <HasWeight rdf:datatype="&xsd;decimal">1.00</HasWeight>
</owl:ObjectProperty>

As can be seen there exists also a sub property HasWeight. For both relations there exist sub relations that are more specific and present a more detailed weight. This way the modeling problem of giving a specific relation a certain weight is solved. Other solutions are presented in [16].

```xml
<owl:DatatypeProperty rdf:ID="HasWeight">
  <rdfs:domain rdf:resource="#KeyWordRelation" />
  <rdfs:range>
    <owl:DataRange>
      <owl:oneOf>
        <rdf:List>
          <rdf:first rdf:datatype="&xsd;decimal">0.33</rdf:first>
          <rdf:rest>
            <rdf:List>
              <rdf:first rdf:datatype="&xsd;decimal">0.66</rdf:first>
              <rdf:rest>
                <rdf:List>
                  <rdf:first rdf:datatype="&xsd;decimal">1.00</rdf:first>
                  <rdf:rest rdf:resource="&rdf;nil" />  
                </rdf:List>
                <rdf:rest rdf:resource="&rdf;nil" />  
              </rdf:List>
            </rdf:rest>
          </rdf:rest>
        </rdf:List>
      </owl:oneOf>
    </owl:DataRange>
  </rdfs:range>
</owl:DatatypeProperty>

So also the following property is defined which can be used for a relation between a keyword concept and an information-object concept:

```xml
<owl:ObjectProperty rdf:ID="HasKeyWordWeak">
  <rdfs:label xml:lang="nl">heeft zwak trefwoord</rdfs:label>
  <rdfs:subPropertyOf rdf:resource="#HasKeyWord" />
  <owl:inverseOf rdf:resource="#IsKeyWordOfWeak" />
  <HasWeight rdf:datatype="&xsd;decimal">0.33</HasWeight>
</owl:ObjectProperty>
```
A recommendation is that all weights are already available in the metadata files within the different sources. This way when generating the ontology this value only has to be copies and the administration then not has take place in the ECM system.

**KeyWords**

Each KeyWord has a unique id, a label and several properties restricted by a datatype, which can be found back in the value the specific property has. Below an example is presented of a KeyWord which has specific properties “ZieOok” and “OuderVan”. The HasWeight property has been explained on the previous page for dictionary and the same holds here, with the difference that there is no subdivision in relations but only that the property HasWeight gets one of the specific values described in the HasWeight property.

```xml
<owl:ObjectProperty rdf:ID="KeyWordRelation">
  <rdfs:label xml:lang="nl">trefwoord relatie</rdfs:label>
  <rdfs:domain rdf:resource="#KeyWord" />
  <rdfs:range rdf:resource="#KeyWord" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="ZieOok">
  <rdfs:label xml:lang="nl">Zie ook</rdfs:label>
  <rdfs:subPropertyOf rdf:resource="#KeyWordRelation" />
  <owl:inverseOf rdf:resource="#ZieOok" />
  <HasWeight rdf:datatype="&xsd;decimal">0.33</HasWeight>
</owl:ObjectProperty>

<KeyWord rdf:ID="KWD_9492">
  <rdfs:label xml:lang="nl">kegelen</rdfs:label>
  <KeyWordId rdf:datatype="&xsd;positiveInteger">9492</KeyWordId>
  <ZieOok rdf:resource="#KWD_22582" />
  <OuderVan rdf:resource="#KWD_23322" />
</KeyWord>
```

In Table 3.3 the object properties that are defined for InformationObjectRelation are presented:

<table>
<thead>
<tr>
<th>Property</th>
<th>Inverse property</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZieOok</td>
<td>ZieOok</td>
</tr>
<tr>
<td>OuderVan</td>
<td>KindVan</td>
</tr>
<tr>
<td>Behandelt</td>
<td>WordtBehandeldDoor</td>
</tr>
<tr>
<td>Behandelen</td>
<td>WordenBehandeldDoor</td>
</tr>
</tbody>
</table>

Table 3.3 Types KeyWordRelations domain model

- **Define individuals (instances) per class per ontology**

The final step is defining the instances for all defined concepts. These are generated from the existing XML output from PIGA and this process is described in paragraph 6.6.3. It already needs to be mentioned that the properties Publication date and District Name are randomly generated since they currently are not available in PIGA. In the previous steps and in appendix B2 already samples of instances are presented.
4 Personalization

4.1 Introduction

Systems used by the municipality are very big and eager to get new information in very short periods. So there is a lot of information available in the system. Therefore searching for the desired information can be very time-consuming and users might be overwhelmed with unwanted information. Especially when a user is not able to quickly find what he or she needs can be very useful to let the system learn from users’ behavior and anticipate by presenting hints based on other users’ history but also personal history. Personalization techniques are used to solve this problem.

Since in the municipal there are a lot of users [2], statistics and history of usersn that become a reliable factor when comparing certain behavior with other users’ behavior. Using personalization, the system is able to reason about the information-objects other users have visited in the past in order to provide a user with recommendations for content that the user might be interested in, in combination with the users’ own profile.

Results of a specific search query can be personalized to adjust to the user’s interests. Results here are found concepts but also suggestions etc related to the user. In this chapter will be outlined how the personalization process works and how the system can reason about the users’ behavior. First will be explained how specific user information is modeled in the prototype. The user model can be divided in three components, which are the user profile, user preferences and the viewing history per user. The first two are considered as static data and the last component is considered as dynamic data, described in paragraph 4.2. In paragraph 4.3 the process of modeling the static and dynamic data is explained in detail. In paragraph 5.6 the way the system uses these user models in order to get more personalized search results will be described.

4.2 User modeling

4.2.1 Introduction

According to [17] the aim of user modeling in computer-based dialog systems is primarily to help make systems more cooperative and thus more easily accessible, particularly to inexperienced users. To demand as few as possible knowledge by especially the citizens, user models are used in this prototype. The term user modeling [18] means technically the construction of (computer-based) models based on users’ mental activities and behaviors, often used to make predictions about a system’s usability or as the basis for interactive help systems. The personalization techniques that have been researched are user profile-based and user behavior-based techniques.

Information contained in the user models consist of the exact information that is needed by the prototype to reason about the users' needs. Since the proof-of-concept is designed for a multi-user web environment each user is modeled using a user profile complemented with user preferences. These are considered as static data. Static data is data that does not change often over time. Dynamic data on the other hand is the data of a user that may be changed or expanded over time more often. This type of data must be gathered by the system automatically. Relevant static data is loaded when a user logs in to the prototype. Dynamic data is loaded whenever a user views a specific information-object to compare its behavior to other users’ behavior. Both will be outlined in paragraph 4.2.2 and 4.2.3.

The behavior models are based on the viewed information during a specific search query in a specific session. Mentionable is that all information should be saved directly to the model, and not when the session ends. When working in a web environment this is necessary, since otherwise specific valuable information might get lost.
4.2.2 Static data

Since static data does not change very often over time, these are specifically the characteristics of a person like name, address, etc. These user profiles are set up the same as the profiles used by DigiD [19]. Since the municipality currently is thinking about linking to this system and about how they can use personalized functionalities in their applications this might be useful. In the prototype only one attribute of the profile is taken and is used to prove it can be useful in the intelligent search option. This attribute is the district where a specific user lives in. This attribute is used by the system to be able to only search in the part of the ontology for information-objects that are from the same or a neighboring district the user lives in. Furthermore attributes as gender or age might be usable for ranking specific information-objects higher, but are not taken into account for the prototype.

Next to user profiles also user preferences are added to the user model. User preferences are mostly graphical settings that a user prefers in order to view the information. These are used in the prototype very primitively since there is no extra usefulness to the purpose of intelligent searching. Globally seen the static part of the user models can be presented as follows. See Figure 4.1. The complete part of static data will be outlined in paragraph 4.3.

![Figure 4.1 Static part user model](image)

4.2.3 Dynamic data

Dynamic data changes over time. After a user has logged in and performs actions on the system, these actions are all stored with the specific user in such a way that information can be derived from these actions to determine certain user’s interests for a specific type of information. All information logged is used to update the behavior model of a specific user. So when the system will be used more often and by more users, more information will be stored and the system will become smarter and more precise so the personalized search results and suggestions will become more sophisticated. An important issue therefore is that the information stored here is adequate.

The information per user is saved as follows. For each user a unique session is saved. Within this session the user can perform a number a queries and within a query a user can uniquely view a number of information-objects. Since for each information-object its meta data is available, the system can distract from here the extra information needed for presenting suggestions to the users. At this point the user model consists of the user profile and viewing history per user. Globally seen the dynamic part of the user models can be presented as in Figure 4.2. The complete part of dynamic data will be outlined in paragraph 4.3.

![Figure 4.2 Dynamic part user model](image)
4.3 Modeling process

Since the users and their behavior and technically modeled the same as within the domain model, also users are seen here as concepts, so here the same guideline is used as in paragraph 3.2. No extra explanations are given but only the results of each step, which will finally result in the knowledge base for the user models.

- Define the ontologies

Next to the knowledge base for all information for the search engine also the user information and behavior on the system is stored. This information is stored within a separate knowledge base. As has been mentioned in the previous paragraph the user models exist of user profiles, user preferences and viewing history. These are separated in different ontologies, combined by mappings. See Figure 4.3.

![Diagram of User Model](image)

**Figure 4.3 Complete user model**

Explanation building blocks:

- In Sessions all session viewing information is stored, e.g. queries per session and viewed items per session query.
- In Statistics each Session concept is mapped to a User concept and vice versa.
- In Users all user profiles are stored.
- In Settings each Preferences concept is mapped to a User concept and vice versa.
- In Preferences all preferences for each user are stored.

- Define classes (concepts) per ontology and arrange classes in a (subclass–super class) hierarchy.

Competence questions according to [15], for being able to infer to define the needed classes:

- What are the information-objects a specific user viewed in the past?
- What are the information-objects viewed by other users who also viewed the information-objects that are viewed in the current user session?
- What type of information-objects the user viewed most often in the past?
- Does a user want email notification upon new information-objects?
Taking these questions into account the following concepts are defined per ontology:

<table>
<thead>
<tr>
<th>Users</th>
<th>User</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preferences</td>
<td>Preferences</td>
</tr>
<tr>
<td>Sessions</td>
<td>Session</td>
</tr>
<tr>
<td></td>
<td>SessionQuery</td>
</tr>
</tbody>
</table>

Table 4.1 Classes in user model

Below for each concept the properties and restrictions are described in OWL format, since it used this was as described in appendix A. In appendix B3 the complete user models are explained in detail.

- Define properties and restrictions for the classes

Users

Each User concept has a unique id and several properties restricted by a datatype, which can be seen in the value the specific property has.

```xml
<User rdf:ID="USR_1">
    <UserId rdf:datatype="&xsd;positiveInteger">1</UserId>
    <UserName rdf:datatype="&xsd;string">ikrooswijk</UserName>
    <PassWord rdf:datatype="&xsd;string">760000</PassWord>
    <FirstName rdf:datatype="&xsd;string">Ivo</FirstName>
    <SurName rdf:datatype="&xsd;string">Krooswijk</SurName>
    <EmailAddress rdf:datatype="&xsd;string">ivo@krooswijk.com</EmailAddress>
    <StreetName rdf:datatype="&xsd;string">Albert Cuypstraat</StreetName>
    <StreetNumber rdf:datatype="&xsd;positiveInteger">270</StreetNumber>
    <ZipCode rdf:datatype="&xsd;string">1073BR</ZipCode>
    <DistrictName rdf:datatype="&xsd;string">oud zuid</DistrictName>
    <CityName rdf:datatype="&xsd;string">Amsterdam</CityName>
    <SofiNumber rdf:datatype="&xsd;string">102938422</SofiNumber>
    <DateRequest rdf:datatype="&xsd;string">17-05-2006 08:55</DateRequest>
    <DateExpire rdf:datatype="&xsd;string">17-06-2006 08:55</DateExpire>
    <DateActivation rdf:datatype="&xsd;string">26-05-2006 21:08</DateActivation>
</User>
```

Statistics

The Statistics mapping maps a User concept to a Session concept, both described by their URI’s.

```xml
<owl:Class rdf:about="&sess;SESS_1">
    <isSessionOf rdf:resource="&user;USR_2" />
</owl:Class>

<owl:Class rdf:about="&user;USR_2">
    <HasSession rdf:resource="&sess;SESS_1" />
</owl:Class>
```
Sessions

The Session concept has a unique id and several properties. Each session consists of one or more Queries in which one or more information-objects are viewed.

```xml
<Session rdf:ID="SESS_1">
  <SessionNr rdf:datatype="&xsd;positiveInteger">1</SessionNr>
  <SessionId rdf:datatype="&xsd;string">E69FD43E901275FDDF8F4837F08B39B0</SessionId>
  <HasSessionQuery rdf:resource="#SESQ_1_1" />
  <HasSessionQuery rdf:resource="#SESQ_1_2" />
</Session>

<SessionQuery rdf:ID="SESQ_1_1">
  <QueryNr rdf:datatype="&xsd;positiveInteger">1</QueryNr>
  <HasViewed rdf:datatype="&xsd;positiveInteger">38743</HasViewed>
  <HasViewed rdf:datatype="&xsd;positiveInteger">41424</HasViewed>
  <HasViewed rdf:datatype="&xsd;positiveInteger">87233</HasViewed>
  <HasViewed rdf:datatype="&xsd;positiveInteger">94414</HasViewed>
</SessionQuery>
```

Settings

The Settings mapping maps a User concept to a Preferences concept, both described by their URI’s.

```xml
<owl:Class rdf:about="&user;USR_1">
  <HasPreferences rdf:resource="&pref;PRF_1"/>
</owl:Class>
```

Preferences

Each Preferences concept has a unique id and several properties restricted by a data type, which can be seen in the value the specific property has.

```xml
<Preferences rdf:ID="PRF_1">
  <HitsPerPage rdf:datatype="&xsd;positiveInteger">10</HitsPerPage>
  <EmailNotification rdf:datatype="&xsd;boolean">true</EmailNotification>
</Preferences>
```

- Define individuals per class per ontology

The final step is to define the instances of the defined concepts. Since for storing the users’ behavior dynamic data is used here, these individuals are generated by the system. On the other hand, considering the static data, Users are created when they register to the system or, actually the external DigiD system, and they also are allowed to set preferences. Since this is a task of an external system these individuals are generated manually as described above. See also paragraph 4.4. In appendix B.3 some instances are described.
4.4 Usage of user models

4.4.1 Static data

As has been outlined earlier the static part of the user model is already available in the knowledge base. Because of preventing cold start problems [68], already some user profiles and preferences have been manually inserted into the system.

The user profiles are used for a few options in the prototype. After logging in the district a user lives in is used to search for more specific information closer to the user’s district. Furthermore the user’s email address is used to sending specific information to the user. As has been mentioned earlier the information is copied from the DigiD initiative and does not fulfill all possibilities. Although the goal here was to show that specific information, like the district, could become handy when searching, also other information could be interesting. Attributes as age or gender can be useful when ranking specific information higher. Furthermore the user preferences are very limited and only available to show that they can easily be integrated into the user model. In the future these can be used to look to groups with the same preferences, to detect common interests and behavior and reason about that.

4.4.2 Dynamic data

The dynamic data is used for two purposes, which are based on content-based filtering and collaborative filtering. Although here they are not used for filtering information-objects out of all resources when presenting results, these techniques are used here to present relations to other useful information to the end-user. First the activity diagram in Figure 4.4 will show how the dynamic data will be saved to the user model at the moment a user views a specific information-object.

![Figure 4.4 Updating dynamic data](image)

Figure 4.4 Updating dynamic data
**Own behavior**

Content-based filtering [21] is a technique based on analysis of previously viewed information-objects in relation with the user's personal preferences. Every time a user views a specific information-object, the dynamic data of the user's user model is updated with certain characteristics of this action. Regarding the viewed information-objects the system then can filter specific characteristics of these information-objects to detect the interests of the user by analyzing the objects' characteristics and e.g. present suggestions for the user when new information-objects are created in the system based on these characteristics.

In the proof of concept the most viewed type of information, and the district the most viewed information-objects are related to, are distracted and based on these attributes questions are asked for further information retrieval. How this is handled can be found back in paragraph 5.6. So this is the part for offering recommendations based on the user's own behavior. The meta-data descriptions for all content items are solid and adequate and provide good enough information for providing suggestions. There are no gaps in the information that can make the system present no recommendations.

**Other users’ behavior**

Collaborative filtering [22] goes one step further than content-based filtering and is not based on the behavior of a specific user but also on the behavior of other users. When other users are mentioned always all users are meant. There is no notion of groups of similar users, since in the context of Amsterdam this kind of techniques are useless since the content-items looked for are very diverse. There is a distinction between user-based collaborative filtering and item-based collaborative filtering.

User-based collaborative filtering [23] uses similarity, which usually is computed using satisfactory indications by its users. Most systems are based on ratings a user can give to items after viewing its contents. This is omitted because of the fact that users don't always rate specific content items or only rate negative [2] and therefore no reliable information can be used by the system. Also the first-rating [68] problem is worked around by this.

Item-based collaborative filtering [24] makes it possible to suggest interesting related information-objects to a user when viewing a specific information-object. Information-objects not viewed before are not taken into account here, which means that if the user views an information-object that has not been viewed before by any other user, the system will not be able to present recommendations. This is a slight disadvantage.

In the prototype this last technique is used to present suggestions for interesting information for the user. To present the techniques for finding related information based on other users in Figure 4.5 an example is elaborated.
When the user logs in to the system a new session is started. Only after entering a query and looking for a specific information-object the session query and its related viewed items are saved. When not viewing any information-object within a query, nothing will be saved to the system.

What the system does is assuming that when a number of information-objects is viewed within one query and another user also views some (settable number) of these information-objects, then also the other information-object viewed by the first user might be of interest. These can subsequently be suggested to the user for viewing. Since users can ‘accidentally’ view information-objects that have nothing to do with their query, there is no relevancy measure performed on the presented suggestions.

E.g. when user A visited during one query for information-object a, b, c and d and user B visits a, b and c, then a suggestion to user B is presented that information-object d might be useful. All visited nodes are stored in for each single query. And all queries are stored in a single session, which on its turn is related to the user.

**Example**

Gathered from Figure 4.5:

- User 1 has viewing path $P_1 = \{I_1, I_2, I_3, I_4\}$
- User 2 has viewing path $P_2 = \{I_2, I_3, I_4\}$
- User 3 has viewing path $P_3 = \{I_2, I_3, I_4, I_5\}$

The suggestions for user 2 would be $\left( P_1 - P_2 \right) \cup \left( P_3 - P_2 \right) = \{I_1, I_5\}$

Here the number of matching objects is set to 3 which is settable when implementing the prototype. How these techniques are used in the search process can be found back in paragraph 5.6.
5 Intelligent search and presentation

5.1 Introduction

The main task to be specified, as described in chapter 2.2, is the search option. On most websites there exist several specific search options. The most used option is the “free text” search or “keyword based” search as used by Google [25], where the user needs to enter natural language words to search for specific content. It is a fast and easy way which often delivers results quickly, but where not all content can be found easily, or too much non-wanted results are returned.

The “free text search” option is mostly used for non-structured content, like text resources in more than one domain in where specific occurrences of text need to be found. For the concept here only structured content is used in a single domain. For all content-items, concepts are defined by metadata that describe the content-item, together with a set of keywords that are related to that content-item. So in the system only the metadata of a content-item is saved. Therefore “full-text retrieval” is not used. Full text retrieval has the disadvantage that when a user searches with a specific keyword, and the keyword occurs somewhere in the document, it is possible that the returned document is completely not related to the keyword and therefore irrelevant. Furthermore the query consists of several other attributes that are used to broaden or narrow down the set related content items.

Therefore the keyword concepts that are related to the content-items together with the content headers of the content-items will be matched against the input keywords. The content headers always contain relevant terms. Below the concepts for keyword, information-objects and relations are formalized to describe the two-way search process for information-objects in the following paragraphs.

So always keep in mind that a user initially enters some natural language and finally wants to find one or more information-objects. The keyword concepts the system contains are actually some intermediaries pointing to the right information-objects. As you can see one match is retrieved via keyword concepts and the other match is directly retrieved via the content headers. If matches occur via both strategies, only one match is retrieved by the system with a combined score. Furthermore when matching by keyword concepts linguistic techniques are applied, which are described in paragraph 5.4. When matching directly on the title of the information-objects only containment is checked. Other attributes that are contained in an information-object are not used for ranking purposes but for filtering the result set for objects that meet a specific attribute value. This will be explained further in this chapter.

Before explaining the steps of the search process in detail, first in paragraph 5.2 an overview is given of how the system gets from a query to a ranked results set of information-objects.
5.2 Search Process

As has been mentioned in the introduction of this paragraph there actually is a two-way search process for information-objects. One via the keyword concepts and one directly through the headers of the information-objects themselves. To get a good view of the search process via the keyword concepts, Figure 5.2 is showing the main parts of the process which uses linguistic techniques. How this process is implemented into the prototype can be found in more detail in chapter 6.5.5, where all sequence diagrams are presented that describe the exact flow. Before a user is able to search first authentication is needed from the user models, but this is not taken into account here. The search process directly through the headers of the information-objects in discussed in paragraph 5.5. Here also first input validation takes place, but then directly the query is matched to the headers. No refinement is applied.

![Activity diagram query processing global](image)

Figure 5.2 Activity diagram query processing global

The steps above are explained in detail in paragraph 5.4. The input validation part is first discussed in paragraph 5.3. How the final result set is presented, consisting of information-objects from both search parts, will be outlined in paragraph 5.6.
5.3 Input validation

As the introduction of this chapter tells, the only input possibility for the search question is an input field for a query for natural language keywords. Explicitly this means that other input options like searching for frequently asked questions, starting letter, etc. are not available, simply because it is out of scope of this research.

Next to the keywords it is also possible to add specific search attributes to the query. This can be seen as an initial refinement of the query, which is also possible when the results or content are presented. Doing this only a specific part of the ontology will be searched. The system interprets this as an advanced search. When a simple search is invoked, the extra attributes are not taken into account and the whole ontology will be searched. The reason why a user can also choose search attributes in the input screen is because it might be helpful for call center employees, who possess knowledge of the system, when clarifying the user’s question, to get faster to specific results.

For the input keywords there doesn’t have to be any attention for Boolean operators (and, or, difference), exact matches (“exact”) or regular expression matched (regular*). Furthermore punctuation marks are filtered out during validation of the input. Also the input query will be trimmed and lowercased to offer it to the search option. Therefore before searching with the keywords, there first takes an input validation place. This is as minimal as possible since otherwise there might be a conflict with exact matches. Therefore only the very obviously mistakes are omitted. When the resulting query is not valid (empty after applying filters) there will be invoked an error message and the query will not even be presented to the query processing engine to filter and enrich it further.

The entered keywords form a space separated input sentence where each next keyword is preceded by a space. This sentence should be converted to a list where each keyword is single entry. This way each keyword can be analyzed separately, and afterwards each keyword can get an initial weight when the priority queue is offered to the spread activation algorithm for searching in the ontology. This process is outlined in paragraph 5.4. Formally the input query is defined as follows:

\[ Q = \{ KW, CT, CA, DN, ND, PP, ST \} \]

where

- **KW** = keyword string entered by user, after input validation has taken place
- **CT** = content type, the type of content the user searches for, default all
- **CA** = content attribute, the type of attribute the user searches for, default all
- **DN** = name of the district, the district in which the user searches, default users own district
- **ND** = flag for searching in neighboring districts
- **PP** = publication period, the period in which the information objects should be published
- **ST** = search type, flag whether all attributes should be taken into account (simple, advanced)

Within the validation step of the input only the user entered keywords are validated as described above. Formally this can be described as follows:

**Example**  

\[ K_n \] is uppercased and \[ K_1 \] is “/” and preceded by white spaces

\[ KW = " K_1 K_2 ... K_n " \]  
\[ \Rightarrow \{ apply \ transformation \ algorithm : remove \ K_1 \ and \ lowercase \ K_n \} \]

\[ KW" = " K_2 K_n " \]
5.4 Query processing

5.4.1 Introduction

When the input keywords are validated correctly, the process of finding related concepts can start. Here for the strategy [26] and [67] use have been consulted. This process takes the validated keyword string K from query Q (paragraph 5.3) and uses several linguistic techniques to transform the query into a set of related keyword concepts that are related to the validated input keywords. Globally seen this is done by two abstractions which are filtering and expansion, both done via several linguistic techniques (see Figure 5.2). For all keywords in the initial keyword list a check will be performed to see whether there are matching- and related keywords/synonyms or there is no matching keyword in the systems’ keyword collection. So first the keyword set is reduced and optimized for unwanted keywords and then the keyword set is expanded with related keywords. These two steps will result in a priority queue of found concepts and their relation score to the input keyword. The expansion of the input query uses the linguistic techniques below. A remark needs to made that these techniques, as required, are going to be implemented using the Verity search API, which has some restrictions regarding the proposed process. This fine-tuned (Verity) process is visualized in Figure 5.3.

The linguistic techniques needed for finding keywords based on the input query, mentioned in Figure 5.3, are now going to be explained in more detail.
5.4.2 Process steps

As Figure 5.3 points out first the input query is filtered for noise, then depending on the language related Dutch concepts are retrieved. This is done by a combination of typographical, morphological and semantical variants. Finally these are grouped by the system and returned.

1) ‘Noise filtering’

Although during the input retrieval already some noise from the query is filtered, there takes a second filtering place in the second step. The term ‘noise’ here can be defined as unwanted string elements in the keyword set. Noise filtering consists of two steps to filter the keyword set. These are:

- Words with less than three characters
- Words from the superfluous word list

Example $K_1$ has only two characters and $K_n$ is the word “the”

$$KW = \{ K_1, K_2, \ldots, K_n \}$$

$$\Rightarrow \{ \text{apply filtering algorithm : remove } K_1 \text{ and } K_n \}$$

$$KW' = \{ K_2, \ldots, K_{n-1} \}$$

It would be a good addition to filter out diacryts, and replace them with their base variants (e.g. é becomes e), without losing score for being seen as a typographical variants.

2) Multilingual deduction

Since the keyword collection only consists of Dutch and English keywords that are linked to the ontology, there should be performed a language detection for each keyword in the set. This way all words in the keyword set can be deducted to the Dutch language. There has been made an English dictionary with about 400 translated words, that are relating back to the Dutch keyword concepts:

Example $K_1$ is the English variant of the Dutch concept $K_1'$

$$KW = \{ K_1, K_2, \ldots, K_n \}$$

$$\Rightarrow \{ \text{apply language algorithm : replace } K_1 \text{ by } K_1' \}$$

$$KW'' = \{ K_1', K_2, \ldots, K_n \}$$

3) Typographical variants / Ontographical variants:

When a word does not occur in the keyword/synonym collection the system cannot detect whether it is a typographical mistake or an ontographical variant. In an ideal situation the most closely related word can be found by calculating a minimal edit distance between both words, where two words only may differ a settable percentage of the length of the input keyword to be relevant. In Verity it is only possible set a fixed maximal number of allowed mistyped characters. This comparison apply to as well Dutch as English keyword concepts.

Example $K_1$ differs on n character places from the English concept $K_1'$ with the Dutch variant $K_1''$ where n is set to 1 (e.g. “eeldery” is brought back to “elderly” which is related to the Dutch concept “ouderen”)

$$KW = \{ K_1, K_2, \ldots, K_n \}$$

$$\Rightarrow \{ \text{apply language algorithm : restore } K_1 \text{ to } K_1', \text{ replace } K_1' \text{ by } K_1'' \}$$

$$KW'' = \{ K_1'', K_2, \ldots, K_n \}$$
4) **Morphological variants**

This concerns the normalization of each keyword to its stem, which is called stemming. E.g. “alarming” will be brought back to “alarm”. Mostly this happens to plurals, verbs, etc. Also compound words are brought back to their bases. Like “alarmbell” will become “alarm” and “bell”. Verity uses a list for compound words, which is kept in tact. A disadvantage of Verity is that when looking for compound words, typographical variants are not taken into account anymore.

*Example*  
K_1′ is “alarming” and is a morphologic variant of the basic word K_1 “alarm”

\[
KW = \{K_1, K_2, ..., K_n\} \\
\Rightarrow \{apply\ language\ a\ lg\ orithm:\ restore\ K_1\ by\ K_1′\} \\
KW' = \{K_1', K_2, ..., K_n\}
\]

5) **Semantical variants**

Here the expansion takes place of each individual keyword with synonyms and related keyword concepts via the Verity collections.

*Example*  
L_1 is a synonym for K_1 and L_2 is related to K_2

\[
KW = \{K_1, K_2, ..., K_n\} \\
\Rightarrow \{apply\ language\ a\ lg\ orithm:\ replace\ K_1\ by\ L_1\ and\ add\ L_2\} \\
KW' = \{L_1, K_2, ..., K_n\} \cup \{L_2\}
\]

Techniques that are not used by the system in the filtering/expansion process:
- Syntactical variants: No different word combinations, like reversed syllables.
- Key phrase determination: minister president could be seen as minister-president
- Proper name recognition: Bush can be seen as an important person rather than a collection trees

Applying these techniques will finally result in a set, in the form of a priority queue containing matching keyword concepts with an initial relevance score. This set of found keyword concepts is only based on linguistic techniques. Since each keyword concept also exists in the ontology and also has links to other keywords, the related keyword concepts are offered to the user in the presentation part to make a query refinement/expansion. Verity uses a proprietary method to find the keyword concepts. For its so-called query parser a query can be defined that retrieves all concepts by using the techniques described above. This is described in the following paragraph.
5.4.3 Applying scores

Based on the linguistic techniques described in paragraph 5.4.2 the Verity engine is able to determine a score for the relevance between the input query and the found keyword concept. Since from [2] is derived that about 70% of the users only enters a single keyword, there has been made a distinction in the number of single keywords n, for applying different transformation scores. In table 5.1 below the query parser rules are explained on how they calculate the score per case of n.

<table>
<thead>
<tr>
<th>Case</th>
<th>Queryline</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>n=1</td>
<td>[0.8] concept &lt;matches&gt; standard[1]</td>
<td>When a single keyword matches the concept name, the concept name is returned with relevance 80%</td>
</tr>
<tr>
<td>n=1</td>
<td>[0.7] concept &lt;contains&gt; standard[1]</td>
<td>When a single keyword is contained somewhere in a longer concept name as a single word, the concept name is returned with relevance 70%</td>
</tr>
<tr>
<td>n=1</td>
<td>[0.6] &lt;YESNO&gt; ( &lt;WORD&gt; standard[1] )</td>
<td>When a single keyword is contained in a longer concept name, but as part of a single word, the concept name is returned with relevance 60%</td>
</tr>
<tr>
<td>n=1</td>
<td>[0.5] &lt;YESNO&gt; ( &lt;LANG/NL&gt; standard[1] )</td>
<td>When a single keyword is contained in the stem-or compound words list, the related concept name, is returned with relevance 50%</td>
</tr>
<tr>
<td>n=1</td>
<td>[0.5] &lt;YESNO&gt; ( &lt;TYPO/1&gt; standard[1] )</td>
<td>When a single keyword differs one character with a concept name, the concept name is returned with relevance 50%</td>
</tr>
<tr>
<td>n=1</td>
<td>[0.2] &lt;YESNO&gt; ( &lt;TYPO/2&gt; standard[1] )</td>
<td>When a single keyword differs two characters with a concept name, the concept name is returned with relevance 20%</td>
</tr>
<tr>
<td>n&gt;1</td>
<td>[0.8] concept &lt;matches&gt; FOREACH wrd IN standard CAT wrd OPT &quot; &quot; END</td>
<td>When the complete keyword string matches the concept name, the concept name is returned with relevance 80%</td>
</tr>
<tr>
<td>n&gt;1</td>
<td>[0.6] concept &lt;contains&gt; FOREACH wrd IN standard CAT wrd OPT &quot; &quot; END</td>
<td>When the complete keyword string is contained somewhere in a longer concept name, the concept name is returned with relevance 60%</td>
</tr>
<tr>
<td>n&gt;1</td>
<td>&lt;LANG/NL&gt; [0.5]&lt;YESNO&gt;( ( FOREACH wrd IN standard CAT &quot;&lt;WORD&gt; &quot; + wrd OPT &quot;&lt;NEAR&gt;&quot; END ) )</td>
<td>When one of the words contained somewhere in a longer concept name as a single word, the concept name is returned with relevance 60%</td>
</tr>
<tr>
<td>n&gt;1</td>
<td>&lt;LANG/NL&gt; [0.4]&lt;YESNO&gt;( ( FOREACH wrd IN standard CAT &quot;&lt;WORD&gt; wrd OPT &quot;&lt;OR&gt;&quot; END ) )</td>
<td>When one of the words is exactly contained in the stem- or compound words list, the concept name is returned with relevance 40%</td>
</tr>
<tr>
<td>n&gt;1</td>
<td>&lt;LANG/NL&gt; [0.4]&lt;YESNO&gt;( ( FOREACH wrd IN standard CAT &quot;&lt;WORD&gt; wrd OPT &quot;&lt;OR&gt;&quot; END ) )</td>
<td>When one of the words is contained in the stem- or compound words list, the concept name is returned with relevance 40%</td>
</tr>
<tr>
<td>n&gt;1</td>
<td>[0.3]&lt;YESNO&gt;( ( FOREACH wrd IN standard CAT &lt;TYPO/1&gt; wrd OPT &quot;&lt;NEAR&gt;&quot; END ) )</td>
<td>When one of the words is contained in the stem- or compound words list, the concept name is returned with relevance 30%</td>
</tr>
<tr>
<td>n&gt;1</td>
<td>[0.2]&lt;YESNO&gt;( ( FOREACH wrd IN standard CAT &lt;TYPO/1&gt; wrd OPT &quot;&lt;OR&gt;&quot; END ) )</td>
<td>When one of the words differs two characters with a part somewhere in a concept name, the concept name is returned with relevance 20%</td>
</tr>
</tbody>
</table>

Table 5.1 Verity transformation scores

In Table 5.1 the terms within Verity’s query parser mean the following. standard is the input query string by the user and concept is the keyword concept to be matched. The other tags obvious in the context.
Since a 100% score only can be derived when a complete query matches the complete header of the information-object, the scores start here from 80%. Together with a domain expert has been analyzed all rules and well-considered scores have been determined in relation to the other rules of both cases in such a way they correspond best.

There stick some disadvantaged to the manner Verity uses, which are listed below:

- A disadvantage is that Verity is not able to return the keyword that caused the match. Only the matching concepts can be returned. But mostly this can be easily derived by the user’s common knowledge when browsing through the result list.

- The maximum number of typing errors is restricted to two. It is recommended that this is not a fixed value, but should depend on the length of the input query, which can be done easily without Verity using Lehvenstein algorithm for edit-distance.

- Some combinations of rules are not allowed by the Verity query parser. E.g. stemmed words cannot be combined with the detection of typing errors. Or searching further on one of the parts of a compound word is not supported. Because of this actually some extra intelligence should be build in.

- Another disadvantage of Verity is that when multiple lines may result in the same concept with a different score they both are returned with that different score. Therefore when the final list of concepts is returned there takes place a detection of duplicate concepts where one concept remains with a combined matching score.

- Below an example, for calculating the final relevance score $f$, which is adapted from the rank normalization approach in [72] to be applicable here.

\[
\begin{align*}
\text{Example} & \quad \text{Two keyword concepts match input string KW, one with relevance 0.9 and one with relevance 0.7} \\
& \quad \quad \quad \quad \quad \quad f = (h + (l \cdot (1 - h))) \cdot 100 \\
& \quad \quad \quad \quad \quad \quad \text{where} \\
& \quad \quad \quad \quad \quad \quad h = \text{highest relevance score} \\
& \quad \quad \quad \quad \quad \quad l = \text{lowest relevance score} \\
& \quad \quad \quad \quad \quad \quad \text{with } (n-1) \text{ iterations for } n \text{ matching concepts} \\
& \quad \quad \quad \quad \quad \quad \text{So the combined score for the example above:} \\
& \quad \quad \quad \quad \quad \quad f = (0.9 + (0.7 \cdot (1 - 0.9))) \cdot 100 = 97%
\end{align*}
\]

Hereby it is accomplished that keyword concepts that are found more than once, will retrieve a higher score than keyword concepts that are only found once. So the output of this step is a set of found keyword concepts, based on the input query, with for each concept a score indicating the linguistic relevance to the input query. How this is used as a base for searching for information-objects will be showed in the next paragraph.
5.5 Fuzzy searching

5.5.1 Introduction

To be able to find the correct information-objects and their relations, matching is used. Matching can be defined as comparing a query to a certain information-object concept found by following a path through their semantic weighted associations. This is explained in [29] where a hybrid approach is presented for searching in semantic content. It combines classical search techniques with spread activation techniques. Matching thus is an essential part, since it is exactly the part where the knowledge is used to find the correct concepts in the knowledge base. The structure of the knowledge base is already described in chapter 3.

Often this is referred to as fuzzy matching, since it might be possible that the keywords that are searched are not even present in the content. The fact that the system tries to think for the user, by detecting possible mistakes or knowledge gaps, helps in presenting the information-objects the user is most likely interested in. Because of the possibility that also irrelevant matches might be returned, the user is able to converge and diverge the result set or refine the query, so irrelevant matches can be easily filtered out.

As already described in the introduction of this chapter there are two strategies for matching the user query to the information-objects. In Figure 5.1 this is explained. One way to find related information-objects is to find matching keyword concepts that on their turn are related to the information-objects. The other way is to directly match to the content headers of the information-objects. When searching for matching concepts also a related relevance score is calculated. For both strategies different, but coherent, score calculation techniques are applied. When different matches are related to the same information-object, a merge takes place, where a combined score will be calculated. In Figure 5.4 the process of this relevance score calculation is presented schematically.

![Figure 5.4 Activity diagram searching](image)

Both ways for searching, together with their different approaches for score calculation, will be described in the following paragraphs. First the user query, the concepts of the knowledge base and the result set are formalized.
• A formal description of a query object is already presented in paragraph 5.3.

It is the keyword string that is matched against the keyword- and information-object concepts in the domain model. All other attributes that optionally can be entered are used to narrow down or broaden the search area in the ontology and therefore also the result set related to a specific query. For example when a specific content-type has been set, only concepts of that type are matched against, or when specifying a certain district only information are returned that are related to that district.

• The domain model $DM$ can be represented formally as follows:

$$DM = \langle \{KC\}, L, \{IC\} \rangle$$

where

$KC = \text{keyword concept}$

$L = \text{relation between keyword concept and information object concept}$

$IC = \text{information object concept}$

So the domain model consists of keyword and information-objects that are described below.

 o Each keyword concept $KC$ can be formalized as follows:

$$KC = \langle KI, KL, \{REL\} \rangle$$

where

$KI = \text{keyword concept id}$

$KL = \text{keyword concept label}$

$REL = \{RL, WG, RI\}$ is the set relations to other keyword concepts

where

$RL = \text{label of relation}$

$WG = \text{weight of the relation}$

$RI = \text{id of the range concept}$

 o An information-object concept $IC$ can be characterized as follows:

$$IC = \langle II, IL, IT, ID, IP, IV, \{REL\} \rangle$$

where

$II = \text{information object concept id}$

$IL = \text{information concept label}$

$IT = \text{type of the information object}$

$ID = \text{district name for the information object}$

$IP = \text{publication date of the information object}$

$IV = \text{number of times the information object has been viewed}$

$REL = \{RL, WG, RI\}$ is the set relations to other keyword concepts

where

$RL = \text{label of relation}$

$WG = \text{weight of the relation}$

$RI = \text{id of the range concept}$
The result set $RS$ has the following formal representation

$$RS = \{MK\}$$

where

$MK = IC \cup \{SC, MR\}$ is a matching keyword concept

where

$SC =$ the relevance score of the query to the matching information object

$MR =$ the match reason, indicated by the keyword that made the match possible

A result set contains match objects that exist of all fields of the matching information-object complemented with the score and the reason for the match. In the following paragraphs will be described how the search processes for both strategies exactly work and also how the result sets both produce will be combined into one final result set containing unique matches.
5.5.2 Via keyword concepts

Matching via keywords depends on two steps, which bring along two sub scores. These are the transformation score, which is determined by the Verity engine as described in paragraph 5.4.3. This transformation score tells about the ‘linguistic distance’ between an input query and a found keyword. The second sub score is the mapped weight on the relation between keyword concept and information-object in the knowledge base. There are three types of relations between both, with three different weights, which describe the level of relatedness to the information-object for each keyword.

In the first part during the process of filtering and expanding the query, keywords get specific weights, depending on the transformations that have been made regarding the initial query. These are put in a priority queue. The other attributes of the query, described in paragraph 5.3, are not used. This is the point were the enriched search query and the entered parameters are used for searching in the ontology for related information-object concepts. Searching will be performed by using a ‘spread activation search’ [29], which takes the priority queue as parameter. This way by taking the relation weights between matching keywords and information-object concepts, a result set can be built with precise relevancy factors and returned to the user.

The calculation of the final score is depends on the two steps. The initial value is the score retrieved from the Verity engine that indicates the relevance from the input keyword to the found keyword concept. That is multiplied by the relation scores on the path to an information-object. The total matching score $s$ of the input query to an information-object can easily be measured with the following calculation:

$$s = t \times r \times 100$$

where

$t =$ transformation score

$r =$ relation weight

Since for each information-object also its directly related information-objects are returned, also for these objects their relation score needs to be presented. This comes down to a breadth first search, where only one layer is presented per object. Also the relation between a main information-object and its related information-objects has a weight. Therefore it is quite simple to calculate the relation score $ss$ for each related object. Simply the relation weight is multiplied by the score already retrieved for the main information-object.

$$ss = s \times w \times 100$$

where

$w =$ relation weight

When browsing further to a deeper layer the scores are decreased relatively by the relation weight to that layer. So in layer $n$ (path-length = $n$) the score is multiplied and decreased $n$ times.
5.5.3 Via information-objects

For the direct matching on the titles of the information-objects another strategy is used. Here no linguistic techniques are used. Note that first also the input validation of paragraph 5.3 is applied. First there will be checked if the complete input query literally matches the header of the information-object. In the same step is checked whether the complete string only matches a part of the header.

**Example**  
Input query “Stadsdeel Zuidoost” completely matches “Stadsdeel Zuidoost”, but it only partly matches “Dienstverlening Stadsdeel Zuidoost”.

The first case is a 100% match, since the title always is about the content of the information-object. The second case has been set to a match with a base score of 90%. Only in the case there is no (partly) complete match of the input query, and if the query exists of more than one word, then for each individual word the same two checks will be performed. Here the base scores chosen subsequently are 80% and 70%. All determined by the domain experts, keeping in mind that a header-match has a slightly higher priority than a keyword concept match.

**Example**  
When input query “Stadsdeel Zuidoost” has no matches, the word “Stadsdeel” of the input query actually does match the header “Stadsdeel centrum” and therefore retrieves a lower matching score.

For the partly matches also is taken account what part of the concept title they match. When a keyword only matches a small part of the header this means less relevance than when the keyword matches a big part of the content header. Therefore there takes place an extra calculation which decreases the base score slightly, to get a better spread result set regarding the final scores.

**Example**  
Input query “media” matches a big part of “mediatheek” and only a small part of “Raadscommissie Kunst en Cultuur, Lokale Media, Sport en Recreatie, Bedrijven, Deelnemingen en Inkoop”.

In the first case the overlap score is 0.5 (5/10) and in the second case (5/101). Therefore the first case is likely to be slightly more relevant than the second case. The final relevance score $f$ of a partly match can therefore be calculated as follows:

$$f = (bs - (\frac{1}{os}/100) \times s) \times 100$$

where

- $bs = \text{base score chosen by domain experts}$
- $os = \text{overlap score calculated by: length(keyword) / length(content header)}$
- $s = \text{spreading factor making sure very small matches will not score heigh}$

In the example above, with $s = 0.5$, the following final scores are calculated:

**Input:** "media"

**Match 1:** "mediatheek!" with base score $bs = 0.9$ and overlap score $os = 0.5$

**Match 2:** “Raadscommissie Kunst en Cultuur, Lokale Media, Sport en Recreatie, Bedrijven, Deelnemingen en Inkoop” with base score $bs = 0.9$ and overlap score $os = 0.05$

$$\text{Final score } f_1 = (0.9 - (\frac{1}{0.5}/100) \times 0.5) \times 100 = 89\%$$

$$\text{Final score } f_2 = (0.9 - (\frac{1}{0.05}/100) \times 0.5) \times 100 = 80\%$$

In the first case, since the partly match, the score is only a little bit decreases, since it is likely to be more relevant. Because the second header is much longer and likely to be less relevant, it has a lower score, but still in the same region as the first case.
5.5.4 Combined results

Both strategies will produce a result set with their found information-objects. It is possible that the same matching information-object occurs in both lists.

Example “Stadsdeel Zuidoost” is a direct match and also has a related keyword concept “stadsdeel” and therefore is also found via the keyword concept way.

This means that the same object in both lists will retrieve a different relevance score. Since both lists will be added to a final list, each object only may occur once, with only one score. Therefore both result lists need to be merged and the doublures need to be filtered out. A combined score must be calculated and the match reasons must be summed up.

The final scores for a unique information-object after merging will be calculated as follows, again based on rank normalization presented in [72]. The priority for a match to a content header has the priority since the header always guaranteed tells about the content and since no linguistic techniques are applied here, if there is a match, it is has more value. Therefore this score is taken as the base.

When then are two matches it says the matching information-object is of more importance then when an information-object only has one of the two matches. Therefore its resulting score must be higher than the base. The calculation takes the other percentage and multiplies that with the rest (1.0 – base) and adds that to the base. More formally for the final score $f$:

$$f = (bs - (ks \ast r)) \ast 100$$

where
- $bs =$ base score retrieved from direct matching
- $rs =$ rest score calculated by $1 - b$
- $ks =$ score retrieved from keyword matching

Note that a combination of factors can never pass the 100% border, so exact title matches always get the preference. For the related information-objects the same principle holds as described earlier. The final score $rf$ for these objects again takes a multiplication by its relation weight $rw$.

$$rf = f \ast rw \ast 100$$

where
- $rw =$ the relation weight between the main object and the related object

Furthermore for future work, weights can be defined over specific parts of the ontology [69]. These context-weights can for example be specified in the user profiles. When call-center employees only possess knowledge over a particular type of information the information-objects of this type can rated higher. This is recommendation for further work, since it was not of enough relevance to work it out here. Also trust-weights can be defined for parts of the ontology. Since the complete ontology is built up here from data within the municipality, trust is not really a relevant aspect here.
5.6 Presentation

5.6.1 Introduction

The presentation part consists of the found results but more important of guidance for navigation to other related information. Although the search process is finished in principle, it is of course possible the user did not find the wanted information. In case there are no results at all the user should get a well defined message that there are no results found, were the query is repeated, and additional options in dialogue form for refining the query, were all attributes of the query can be re-chosen.

Also when there are results the user cannot find the useful information. Here there are two possibilities. The wanted information is in the result set, or it is not in the result set. When it is available in the result set, but not directly visible the user should get options to sorting- or grouping the information to be able to get more quickly to the content item. Important here is that it is not the search part that fulfills these options but the presentation part. Initially the result set is presented descending based on the relevance and no grouping is applied.

When the wanted information is not in the result set, or the user is still not able to find the information fast enough, there again is the option to perform a refinement/expansion of the query attributes by resetting one of the search attributes to converge / diverge the result set. This also meets the call from users not wanting too many results in the result set. This way they can get a better overview of the results.

These options are presented to the user in a dialogue based way split up in several parts each performing other functions of the dialogue. Using the dialogue in the system the system can possible solve the knowledge gap of the user. This is only used very basically in the concept, since the knowledge base is too diverse to be able to detect specific needs by a user. The system offers a dialogue using specific user interface elements to diverge and converge the result set or to initiate new queries. It also remembers earlier queries by the user to be able to go to an earlier point in the dialogue.

One of the goals of the prototype was to give the user the feeling of working with all the advantages of an ontology using the rich relations by using a semantic presentation. So in the result set and in the query refine options, everywhere are presented options to navigate through the ontology, based on the attributes and relations. This corresponds to the trend that finding by association [71] is more and more an important form of information retrieval.

The query refine options are accessible from as well the results page as when a specific content page is selected. On a content page also other possibilities are offered to the user based on the personal user model and history and also the history of other users with resembling search behavior.

5.6.2 Presenting results

After there has taken place an ontology match between the keywords and the information-objects the unique results are stored in an array and returned to the user. How these results can eventually be grouped and sorted will be described in paragraph 5.6.3. Also the initial choices for these options are described here. So the initial ranking is based on the initial group- and sort settings and is not calculated by the system.

Per information-object in the result list a number of items are displayed, which are all standard fields of an information-object (see paragraph 5.5) and the following fields, which are also already mentioned in paragraph 5.5.

- Relevance
- Reason of the match (keyword match or header match)

Each related information-object itself is also presented with its header and its relevance. These related headers and the main header are all links for the user to retrieve the real content of that information-object which are identified by the content id. Furthermore the reason of the match, next to the relevance score, can help the user to faster determine if an information-object is interesting or not. This holds especially for the call-center employees who already possess a lot of knowledge when using the system and can ask callers more specific questions about their needs.
Concluded can be when the wanted information is likely to already be in the result set, the user can restructure the result set in such a way the wanted information-object comes to the front. Also the result set can be refined, since a smaller list is clearer to users. When it is likely the information-object is not already in the result set, the user can refine the search query to get a qualitative better result set, or the user can broaden the result set to get more related objects. These options are outlined in the following two paragraphs.

5.6.3 Grouping and sorting

Conventional systems often offer long lists of results. In combination with not using the linguistic techniques and poor ranking systems, for users it is hard to finally find the wanted match in a result list. Where the attributes of the information-object first were used to find and filter specific objects in the ontology here they are used to sort or group (cluster) the results or even sort per group. Below they are explained.

Sorting

By sorting all the information-objects in the result the user might find the wanted match faster. Although in many systems searching and sorting can be performed independently, here first the searching takes place, and afterwards the sorting can be performed. The attributes that are used for sorting are:

- None – Default sorting by descending relevance
- Publication date – Might indicate if the information is about an actual event.
- Number of views – Might indicate if an information-object is a hot item, if it is viewed a lot of times
- Relevance – This is a calculated attribute that indicates the relevance between the query and the result.

The results can be sorted descending and ascending on all three attributes.

Grouping

When a user searches for information-objects the result is often a large list of matches without further structure. Grouping (or clustering) the resulting set of resources from a search query into clusters creates:

- Overview of common attributes between resources
- Viewing of data per subject / per part of the query
- Better overview of the information

According to [30] the groups must be efficiently browsable and fast enough. This way users can decide fast enough whether a specific cluster is of interest for investigating more, which is exactly the goal of clustering.

The attributes that are used for grouping are:

- None – Only the sorting option is used
- District – Per district all information-objects are listed
- Type – Per content type all information-objects are listed

Especially when a user narrowed down the result set beforehand, by searching on a specific attribute, might give the user the insight to broaden the result set again, since easier can be concluded that the wanted result is not in the list. When grouping is selected the group title (e.g., when clustering on district, “centrum”) is not presented within the information-object anymore.

[30] also describes the aspect of coherent clustering which means that results within one cluster all have the same kind of topic. Since after searching the total set of results mostly already consist of objects with the same topic, here, as already has been mentioned, the clustering is based on the attributes. This contributes to the purpose to create structure in the results for faster detecting interesting information.
Combination

The combination is both is also possible to even get faster to the wanted result in the result set. When clustering is applied for one of the two attributes, the sort option can be applied to each cluster. E.g. when the result set is clustered on district and sorted descending on relevance, the results are first grouped per district and consequently each group is sorted descending on relevance.

Initially the result set will be presented to user without clusters and with a descending relevance. This way the most relevant information-object is always presented first. Furthermore when no clustering is applied the result set is presented by pages, which can be browsed by the user. In the user profile is captured the preference of the number of hits per page. E.g. when a result set consists of 42 hits, en the users preference is to show 10 hits per pages, then the result set is shown within 5 pages. This concept benefits the overview of the user. When there is a cluster attribute selected paging is not applied. Since all groups can be collapsed and decollapsed, there is no need for such an aspect for gaining overview.

A recommendation is that when a user is logged in to the system, the sort and group options can be applied according to the settings in the user profile. For example when a call-center agent is specialized in a certain area within the municipality, these information-object scan get a higher relevance score. Also citizen’s aspects like age or gender might be used to rank specific information higher. To fulfill this, the user profiles must be expanded with this information.

5.6.4 Converge and diverge results

As is mentioned in the introduction of this chapter a user must be able to broaden or refine the result set. This can be done by refining / expanding the keywords or by changing the scope of the attributes to search in a bigger area of the ontology.

For the attributes three different types of refinement / expansion have are used.

- **Type** – The types are the subclasses of the information-object class in the ontology. In paragraph 3.2 is shown that there exist several subclasses. The user is able to choose an exact subset of these types to search in. For example the user can choose to only search for “producten” en “instellingen”. Default the user searches in all classes, if initially not a class is selected. Afterwards the user can add or remove classes to specify the search area.

- **District** – For the district has been chosen not to add or remove individual districts but only for zones of districts. When a user is logged in and chooses for an advanced search, default the district is gathered from the user profile and only within that district is searched. Afterwards the user can choose to expand the search area to also all neighboring districts or to all districts. When a simple search is invoked default all districts is searched in.

- **Publication date** – To search in time there has been chosen to use 5 pre-defined threshold dates (always, within last 5 years, within last year, within last month, within last week). When a user selects a period, only those information-objects are selected that are published within the chosen period. Default the search will be performed on all published information-objects.

Searching for information-objects is performed in two ways. One way is directly matching to the headers, and the other way was via keyword concepts. An initial query matches to several keywords, which are called the “found keywords” the complete set of neighboring keyword concepts of all found concepts are called the “related keywords”. For keyword refinement there are two options are presented:

- **Browsing by related keywords** – A new query can be initiated by selecting a related keyword concept the user beforehand did not have any notice of. All other attributes remain the same. These are not used in the search process/

- **Refining by found keywords** – Found keyword concepts are the ones that directly are related to the result set. The user can select one of the concepts to search only for that keyword concept to converge the result set.

- **Expanding with a new query** – Since there exists no management for entered query the user can broaden a result set by entering new query. On the other hand the system keeps track of previously entered queries. This way the user is able to return to a previous point in the search process and can perform new refinements / expansions.
5.6.5 Suggestions

In the content part of the system there will be provided suggestions to the user, since this was a requirement by the municipality. Globally these are an option to detect user’s preferences on behalf of their search history, and the option to detect behavior that other users also seem to have. How the data for both is stored is already described in the chapter about user modeling. How both are used in this search concept is listed below:

- **Detect most viewed type and district** – The system is able to provide an option to let the user retrieve an email when new information of a specific type or within a specific district is published. Since the system knows all information-objects a user has viewed it can easily determine these values and present these options.

- **Detect the viewed objects within a query** – Within a user session a user can perform one or more queries. Per query the unique viewed information-objects are stored. This way the system can detect overlap between the current user’ behavior and previous behavior by others, and provide other interesting information-objects. This is described in paragraph 4.4.2.
6 Architectural design

6.1 Introduction

The architecture outlines the design elements of the intelligent search prototype for Amsterdam. Since Amsterdam wants a well described prototype for being able to integrate an intelligent search option in their information services, it is required to describe how the system works and communicates with its environment. The way the prototype has been set up fits for all current projects within Amsterdam. The domain analysis and especially the requirements form the basis for the architecture. Furthermore the other chapters form a proper basis for understanding the way the prototype has been set up. Where the former chapters were focused on the functional aspects, this chapter mainly focuses on the technical aspects. According to [31] a software-architecture comprises the organization of a system embodied by its components, their relationships to each other and to the environment and the principles guiding its design and evolution.

The architecture presents a detailed view of the parts, the proof-of-concept consists of, and how they communicate. During the creation of the software-architecture of the prototype, the context in which the system operates is defined. In addition, future workers on the project can easily understand the purpose of the system and its constraints. Therefore reducing the complexity of the system is one of the main developing goals here. It also gives a detailed feedback to the various stakeholders related to the municipal. Business goals in this case are an enhancement of the system’s quality and a possible reduction of time and money when implementing functionalities of the prototype in the end product.

On the other hand, the architecture makes it possible to create better system requirements and objectives, due to the overview of the system. This is an iterative process since the requirements specification is denoted prior to this architecture. In addition, documentation of distribution/partitioning of the system can be done more precisely. Due to good documentation another advantage, considering the future usage of the prototype’s functionalities, is that it can reduce maintenance costs and support new developments. This will be outlined further in chapter 7.

The prototype has a client-server architecture, where the domain model and the search intelligence is captured by the server and the input, presentation and navigation logic are kept in the client. The client is a web-based application which provides a user interface for performing the correct tasks. Both client and server are developed fully object-oriented (OO). One of the key factors of OO is enhanced by the architecture. Due to it’s overview it is possible to detect reusable parts and eliminating unnecessary development overhead.

This chapter describes the architecture in different perspectives. In paragraph 6.2 the goals and constraints of the architecture are explained. After this, in paragraph 6.3 and 6.4 respectively the method and style of the architecture are explained. Finally paragraphs 6.5 until 6.9 describe the architecture in a more formal way, using UML [32] and the view paradigm Kruchten 4+1 architecture model [33] as a guide (see also 6.3), which contains four different views with different focuses on the architectural concepts of the system, and some additional scenario’s that can be seen as the fifth view on the system. For each view is indicated what notation and what style has been chosen for denoting the view.
6.2 Architecture goals and constraints

6.2.1 Goals

In this paragraph, the main goals of the architecture are described which were meant to dictate the design and development process of the software during its lifetime.

• **Modularity** – The architecture has to be modular such that it is possible to adapt the software for future enhancement and alternate components. The goal is to create a single coherent model that does not need changes when implemented somewhere else.

• **Content** – The server on which the knowledge base is stored can be replaced or extended with a different data or parts of the data, hence the controlling of the data stores must be replaceable. Also, the storage type of the data must be extendible for future data and possible expansions, hence the data processing must be adaptable and replaceable, which is related to the modularity of the software.

• **Service oriented** – Since it is a goal of the municipality for maintainability services need to be used to implement the wanted functionalities. The service for providing the user information must be replaceable, such that a new kind of user information service can be integrated with the software (Example, a European civilian service, instead of a national civilian service). The service for searching in the content must be replaceable, such that other services that e.g. better operate on the data can be integrated within the whole.

• **User interface input/output** – The type of input devices for the user interface must be replaceable, e.g. touch screen in a kiosk or speech recognition systems. Therefore the software must be extensible for new types of input devices to control the software, which implies a different component for the user interface.

• **Performance load** – The application logic must be spread over the client and the service in such a way that they can operate separately and both perform well.

6.2.2 Constraints

In this paragraph, the constraints on the architecture are described. They are meant to constrain the architecture to meet the requirements of some demands.

• **Reuse** – The system must use as much as possible techniques the municipality currently uses. Furthermore as much as possible open source products must be used, since this is a vision of Amsterdam.

• **Performance** – The administration process may not influence on the search process, and therefore it must be possible to invoke the administration process on specific times.

• **Flexible for future technologies** – The architecture must be flexible, so new technologies and devices can be integrated in the system. It must be made in such a way that parts can easily be left out when for example another output device is used, as in the goals is described, but the core search part still works the same.
6.3 Architecture method

For the system a method for designing the architecture has been chosen. A large number of architectural description languages (ADL) have been developed to represent different aspects of architectures. Most architectural description languages invoke a certain method for developing an architecture and vice versa. Therefore a method and the description language used with it often overlap. For example the so-called “Unified Modeling Language” (UML) often implies the use of the Kruchten 4+1 method. UML is applicable in many different areas and it is widely accepted. This is the main reason to use UML instead of other ADL’s. Additionally UML is focused on Object Orientation, which is extremely helpful in a modular system.

So UML will be the language used to describe the architecture in the rest of the document. The Kruchten 4+1 architectural method contains several views of the architecture, each showing a different aspect and can individually be modeled using UML. The different views are depicted below in Figure 6.1.

![Diagram of Kruchten 4+1 views](Logical View) → (Development View) ← (Scenarios) → (Process View) ← (Physical View)

Figure 6.1 Kruchten 4+1

The ‘logical view’ shows the logical (static) structure of the system. It shows how the system is decomposed into several parts with specific tasks and the associations between the different parts of the system. The ‘process view’ shows the dynamic behavior of the system. The ‘physical view’ shows the actual mapping of components on physical hardware. The ‘development view’ describes the organization of the software in the development environment. The description of these views will be complemented with illustrative use cases, or scenarios, which will become the fifth view. All views will be further described in this chapter to provide a good total view of the prototype.

In the center the ‘scenarios’ the system should handle are modeled. These are for example the actions a user of the system can perform. The scenarios are based on the requirements. Around these scenarios the other views will be built.
6.4 Architectural style

6.4.1 Introduction

Each individual view can be described using a specific style. A style can be defined as a set of rules that specify the vocabulary, structure and behavior of the system. Furthermore it is a set of guidelines to support the application of that style. In [31] a number of architectural styles are explained. Because of the top-down structures of many software applications also mixes of different architectural styles are possible. Here the styles of as well the search system as the administration system are described. Both processes in the system use one common component which is RDF database.

6.4.2 Search process

For the search part of the proof-of-concept has been chosen to use a client-server architecture. Since Amsterdam in the future wants to provide their search engine via a web service this architecture perfectly fits on the situation. In this type of architecture the client applications perform requests to services and handle the input and output of the system, and the services on their turn handle the client requests and send responses with the requested data. The most used area of this type of applications is the area of distributed multi-user systems, like the search service of Amsterdam.

From [31] the following definitions are derived within this context:

- **Client** – Application that makes server requests and handles input/output with the system environment
- **Server** – Application that services requests from clients
- **Client/Server System** – Application that is built from client and server applications

This is exactly how the prototype is built up. The goal is to let the service do all the processing, which is denoted as a 2-tier application. Therefore has been chosen to use a thick-client variant, since clients here are able to remember states because they use session management. In this variant the application logic is separated as follows over the two parties, where the client only performs significant processing, and most of the processing happens on the side of the service. Below in Figure 6.2 the separation of the logic is presented.

![Client/server architecture](Figure 6.2 Client/server architecture)

In the following paragraphs all design choices are mentioned and the applied architectural style is explained in more detail. Everywhere will be indicated on which part of the system the choice is applicable to.

6.4.3 Administration process

For the architecture of the administration process the pipe and filter style is used. The process consists of a series of filters and transformations that fulfill certain tasks. These are typically used for batch transformations, where each individual component is consumer and producer. The first step of the process is reading the input, which is the output from PIGA in XML format, and the last step is adding the data in OWL format to the data repository. Each sub process is actually a filter and the data transfers between the sub processes are the pipes. Each sub process gets input via its incoming pipe, performs an action on the input and sends the output over its outgoing pipe to its descendant. In paragraph 6.6.3 the process is described in more detail where each sub process is explained.
6.5 Logical view

6.5.1 Introduction

The first view in this architectural model, which is going to be examined, is the logical view, which tells about the structure of the system. Since an object-oriented design method is used, this consists of the class models and sequence diagrams, based on specific use cases. A context model is presented to show which external software is used for the proof-of-concept. The decomposed class models consist of a number of object classes, placed in packages, which is showed in the development model. This model is not only used for functional analysis, but also identifies common mechanisms and design elements across the various parts of the system. Generally collaboration diagrams (communication diagrams) show how the different parts of the system are interacting. More specifically it shows system objects and the messages and calls that are passed between them. Sequence diagrams on their turn show a more detailed version of the application’s logic. The diagrams in this view are only presented for the service since that is the place where the most important application logic is processed. The process view shows the working of the client in more detail described in an activity diagram. The notation used for these models is UML as was mentioned in paragraph 6.1. The style that is used here is the object-oriented style. The goal here was to create a single coherent model that does not need changes when transferred to other systems. Before all models are presented, first the use-cases will be specified.

6.5.2 Use cases

Within the search system four use cases are defined. A use-case description is a generalized description of how the system will be used, to show the intended functionality of the system. A use case is always initiated by a user with a particular goal in mind and is satisfied when that goal is reached. Thus a use case captures who (actor) is doing what (interaction) with the search system and for what purpose (goal), without dealing with the real system internals (black-box principle). The only actor here is the ‘searcher’ (citizen or call-center employee). In the use-case diagram in Figure 6.3 the use cases are specified for the user actor.

- **Use Case Login**
  Since multiple users are registered to the service, a user needs to log on, entering the username/password combination or by invoking an anonymous login.

- **Use Case Logout**
  When it is possible other users want the use the system after a specific user session, so the user needs to be able to log out. After logging out, the login screen will be displayed.

- **Use Case Search**
  When a user is logged in to the system it is possible to initiate search queries. The results will be returned to the user. When navigating through these results or refining specific search attributes the query is refined and automatically a new query is initiated.

- **Use Case Select**
  When viewing the result list the user might want to view the content of a specific information-object in the result list. From here again the user can directly initiate refined search queries or look for other information-objects based on suggestions from the system.
6.5.3 Context model

The context model shows the system as a black-box and shows with which external systems (terminators) it communicates with. In Figure 6.4 the context of the client-server application is presented with two external systems.

- **Metadata service**

The metadata service is used for requesting and/or updating two types of information. First the user models are stored here, and secondly the metadata of the searchable content is stored here. The advantage of storing the user models on an external system is that these user models now also can be used by systems other than the search system, e.g. by other services like systems for obtaining specific permits or systems that also make use of the DigiD initiative. Also the same end-user environment can be implemented on different devices that all make use of this service. By using the service in more environments the user models can even be updated with information from which eventually the personalization techniques can benefit. Another advantage is that multiple systems can use the metadata service to search within a specific part of the metadata. E.g. a portal for health care or medical care can only use a specific part of the municipal-wide metadata to perform their search queries on, even when its administration is kept on a central place.
• **Dictionary service**

The dictionary service is the service to which the search service connects to. It contains the word collections described in paragraph 5.4 needed for expanding and filtering a user query to improve it and querying the metadata service. To improve the user query all words and word combinations are checked and brought back to basic keyword where after the query is expanded with semantically related keywords to broaden the result set (see paragraph 5.4). Because of the municipal’s requirements the proprietary techniques and data formats of Verity are used in the dictionary service. This is because Amsterdam already has embedded these techniques in all their systems and the intelligent search method therefore was required to be built on top of this.

• **End-user environment**

The user communicates with the client application via a device. The end-user environment therefore consists of the users and the devices that communicate with the prototype. The prototype is developed for web-based deployment and not for other devices. So the device allows the user to communicate via the client application with the search service. Before searching is possible the same interaction takes place first to identify the user for realizing personalization. The client application is not further divided in components, since it is a web application based on a set of script files. To be able to describe the flow of the client application, the process view shows the complete flow and the working of all individual files.

### 6.5.4 Component model

The component model presents the components the system exists of. As described by the context part of the end-user environment only the components of the server are explained here. There are two service components available at the server, which are the so-called user service and the search service. See Figure 6.5. After describing the services, both will be described in more detail on class level.

![Component model](image)

The user service in the prototype communicates with the metadata service to request information about specific users and their behavior (see paragraph 4.4). It also sends information to the service about the user’s behavior on the client to update the user model for being able to provide the users with suggestions with interesting information. The advantage of storing the user models on an external system is that these user models now also can be used by systems other than the search system, e.g. by other services like systems for obtaining specific permits or systems that also make use of the DigiID initiative. Also the same search system can be implemented on different devices that all make use of this service. By using the service in more environments the user models can even be updated with information from which eventually the personalization techniques can benefit.
The search service in the prototype also communicates with the metadata service but for the information the users are searching for. The metadata is not updated by the search service, but by an external administration process, which is described in paragraph 6.6.3. The search service sends a query to the metadata service and gets back a result set with links to where the actual data can be retrieved. The advantage is that multiple systems can use the metadata service to search within a specific part of the metadata. E.g. a portal for health care or medical care can only use a specific part of the municipal-wide metadata to perform their search queries on. Another advantage is that the administration of the metadata can be kept on a central place.

Below both components are outlined in more detail. In Figure 6.6 and Figure 6.7 the class models of respectively the user service and the search service are presented. In appendix C the complete class-descriptions are presented for both services.
6.5.5 Sequence diagrams

To describe the sequence of messages between objects collaboration- or sequence diagrams can be used. On one hand collaboration diagrams offer the possibility to specify the sequence of messages between objects in a collaboration. The style of the diagram emphasizes the relationships between the objects as opposed to the sequence of the messages. On the other hand sequence diagrams contain the same information as collaboration diagrams, but emphasize the sequence of the messages instead of the relationships between the objects. Therefore to provide a good overview of the working of the system, sequence diagrams are chosen to show how the classes work together and how the dataflow runs through the application.

A sequence diagram, the invocation of methods in each object, and the chronologic order in which the invocation occurs, is captured. This makes the sequence diagram a very useful mechanism to easily represent the dynamic behavior of a system. A sequence diagram is two-dimensional in nature. On the horizontal axis, it shows the life of the object that it represents, while on the vertical axis, it shows the sequence of the creation or invocation of these objects. Because a sequence diagram uses class names (see paragraph 6.5.4) and object name references, it is also very useful in elaborating and detailing the dynamic design and the sequence and origin of invocation of objects. So since the sequence diagram is one of the most widely used dynamic diagrams in UML, it is used for the description of the dynamic flow for as well the client as the service application. For all use-cases described in paragraph 6.5.2 the sequence diagram is presented in appendix D.
6.6 Process view

6.6.1 Introduction

In the process view is described what the system’s main processes are, how they behave, and how they are related and possibly communicate with each other. There are two main processes that are important for the municipality. These are the search process and the administration process. The process description of the concept for intelligent searching was already presented in chapter 5. In the sequence diagrams in paragraph 6.5.5 already the sequence of actions for the search process that occur in the system are denoted and give a good overview. In paragraph 6.6.2 and 6.6.3 also the processes of the client application and the administration process are described in more detail. Both processes will be described using activity diagrams which denote the workflow of each process. Activity diagrams again are part of the UML specification. The style used for the process-view is the client/server style where multiple clients are involved communicating with a web-service (see paragraph 6.8). So although activity diagrams can be used for several purposes here they are used to describe the workflow of the search process and the administration process.

Both processes are independent from each other by means of that they do not communicate with each other but do have common resources, which will be described in the next paragraphs.

6.6.2 Search process

The process of the client application has a workflow where the user can interact with. The client application itself does invoke calls to the service. This is depicted in the sequence diagrams where a login request or a search query is received by the service and handled before responding to the client application. In Figure 6.8 the activity diagram of the client application is presented. First each processing unit will be explained below shortly.

- **LoginRequest** – Graphical user interface for entering user information for logging in to the system.
- **LoginProcess** – Checks user input using LoginValidation bean, if the information is correct then proceed to LoginSuccess otherwise to LoginRetry.
- **LoginSuccess** – Checks user validity by calling the user service, if user exists than proceed to SearchRequest otherwise to LoginRetry.
- **LoginRetry** – Clears all login information and proceeds to LoginRequest.
- **SearchRequest** – Graphical user interface for entering search query for searching for information-objects.
- **SearchProcess** – Graphical user interface for entering search query for searching for information-objects.
- **SearchSuccess** – Checks for corresponding information-objects and retrieves result set, and proceeds to PresentResults.
- **SearchRetry** – Clears all query information and proceeds to SearchRequest.
- **PresentResults** – Graphical user interface for presenting search results, options for converging/diverging the result set and navigation options.
- **PresentContent** – Graphical user interface for presenting content, options for converging/diverging the result set, navigation options and suggestions.
- **SearchRefine** – Gathers information from presentation part and invokes search process again with new query.
- **LogoutRequest** – Logs the user out of the system and presents the login screen again

Combining this process with the service calls to the services, where the search concept from chapter 5 is implemented, should provide a good overview of how the complete process from entering keywords to viewing a match from the result set, is set up.
Figure 6.8 Activity diagram client application
6.6.3 Administration process

For the search process to work correctly with all needed linguistic word lists for in the search concepts, and correct up-to-date information-objects to search for, an extra process is needed, which makes sure all this is done right. Below the data collections that are needed by the processes are listed.

- **Synonym list** – List where all Dutch synonyms are linked to a Dutch base word. This is a generated list by a process administered by Verity.
- **English word list** – List where all English words are linked back to a Dutch base word list. This is fixed word list, created manually.
- **Compound / stemmed word list** – List were all stemmed / compound variants are linked to a base word. This is a fixed list, administered by Verity.
- **Denied words list** – List with all words that are filtered out of the query. Fixed word list administered by Verity.
- **Ontology** – Ontology containing all base Dutch keyword concepts with related information-object. This is administered by a separate process.

Each of the lists above contains word variants that are linked back to a basic keyword. This basic keyword has a corresponding keyword concept in the ontology on which can be searched later, as has been described in paragraph 5.5. As is explained in paragraph 6.4.3 the administration process uses a pipe-and-filter style for converting the PIGA output to the data structures and adding it to the right repositories. By an automatic retrieval process is made sure the data and its relations are correctly gathered from the PIGA sources for adding it to the repositories. For creating the Dutch synonym list Verity is responsible. Since the techniques used here are trademarks of the company they cannot be explained in to much detail. The ontology is the only part not stored by Verity and therefore is explained below. The process for gathering Dutch synonyms is almost likewise, so still can be obtained a good view of Verity does there.

Creating ontology

The data that is going to be used for the ontology is the content from PIGA. Content from other systems used within the municipality are not suitable (see paragraph 2.4.4). The current PIGA output in the form of XML documents is located on the following internet address: http://www.piga.acceptatieloket.asp4all.nl/_spidercontent/ and is IP restricted to only allow access to the municipal servers.

Next to a big set of XML representing the actual or information-objects, also some extra XML documents are available that describe the relations between all information-objects and the keywords that are related to them (meta model). These extra XML documents are used for building the ontologies where all intelligence will be presented in for building the result set. The information-objects are only accessed when a user wants to view its contents. Important to notice is that ECM also uses a PIGA output, but in the ECM version all relations are left out, and therefore this version is not suitable.

Since it is the intention to let each system being updated by its own administrators and let ECM only be a central place for data gathering and no publishing, the administrators themselves must make sure the data is correct.

The automatic management process works as outlined in the activity diagram in Figure 6.9 and Figure 6.10.
The steps of the process are explained below:

- **Retrieve PIGA XML output** – All output files are retrieved via HTTP and stored temporarily.

- **Normalize PIGA XML output** – When all files containing the metadata are retrieved script parses these files to normalize them, so each tag will be placed on a single line. This way the transformation script can be executed correctly, since it parses the files line per line.

- **Transform XML to OWL** – The transformation script is based in Perl. Also XQuery can be used for this. But this is very slow. Also since perl is supported by the servers the applications of Amsterdam are hosted on, perl was used. The script uses pattern matching for detecting the occurrences of important metadata to be transformed to correct OWL. Herefore has been reckoned with RDF/OWL language guidelines provided by [35] and [13].

- **Update Triple store** – A small Java application connects to the RDF/OWL database service, which clears the repository, adds all previous generated OWL files as triples (see appendix A). To be able to use the new metadata, the application server needs to be restarted.

The process described above is an automated process that ideally runs once per every fixed period. The main requirement for a successful conversion is that the initial input is available on the server. This is the server where the XML output of PIGA exists on. The other main requirement of course is that the web server is running and set up correctly on which the triple store resides for data retrieval.
6.7 Development view

6.7.1 Introduction

The development view describes the static organization of the software in its development environment. The way the development view is used here, as mentioned by [36] is as an overview of the placement of actual files and packages in the system and development environment. Because of modularity aspects and for reusability and maintainability of the system it is divided into subsystems. Although the final system does not recognize these systems, for modeling purposes it is a good tool to show the packages and it contents. In paragraph 6.5 already an overview of the classes was given. By means of system diagrams and subsystem diagrams a complete overview of the files and packages of the system is presented.

According to [37] a subsystem can be divided in three parts, which are described for each subsystem, which are:

- **Interface part** – Describes the operations the subsystems delivers to its users
- **Specification part** – Consists of the use cases, in case the subsystem contains a user interface
- **Realization part** – Describes internal elements of a subsystem or references to other diagrams in which these are described

[37] also gives a guideline for defining subsystems: "Maximal cohesion, minimal coupling", which means that each subsystem should have maximal cohesion and the subsystems mutually should have minimal coupling. This is what is taken into account in the prototype. Therefore in Figure 6.2 already was mentioned that the application logic was spread over both the client and the service. The part of logic the client handles is purely the validation of the user input and the receiving of the output of the service for grouping and sorting. This way the search services only has to perform what it is intended for.

6.7.2 Subsystem diagrams

A subsystem diagram in UML is an instantiable package and may contain other components. It is recognizable as a unit within the system. Mostly this means it can be executed on its own and used by other subsystems or the end-users. This is exactly what is needed for the proof-of-concept for possible transferability within the municipality. As will be described in paragraph 6.8, Apache Tomcat is used as container for web applications, and Apache Axis is used for container for the web services, which implies the usage of self-deployable web archives. The two services and the client are therefore enclosed in web archives (packages). So the three main subsystems are the web archives. Both services also consist of other used packages which are described per service. First in Figure 6.11 the system diagram is presented including the supported interface functions for the main components. Afterwards also the subsystem diagrams are presented for each subsystem.

![Figure 6.11 System diagram](image-url)
Client

- Interfaces

For the client application there are no interfaces defined, since it is a web application. The interfaces it uses of both services can be found short in Figure 6.11 and in detail in the sequence diagrams in paragraph 6.5.5.

- Specification

Since the client application is the only component with user interfaces, for the client application below the use cases are worked out and are mapped to the processing units from the process view in Figure 6.8. It becomes clear in these descriptions at what point there takes place a communication between the client package and the service package. Also a mapping is given between the use cases and the user interface elements, which are presented in appendix E.

  o Login

  | The user enters user information (LoginQuery), see Figure E.1 |
  | The validation bean validates the input values (LoginProcess i.c.w. ValidationBean) |
  | If login information is not entered correct a login error message appears (LoginRetry) |
  | If login information is entered correct the user information is gathered and sent to the user service by invoking a call to the user service (LoginSuccess) |
  | The user profile is received back from the user service and the client proceeds to the search part (LoginSuccess). |

  o Search

  | The user enters keywords and optionally sets search attributes (SearchQuery), see Figure E.2 |
  | The validation bean validates the input values (SearchProcess i.c.w. ValidationBean) |
  | If query information is not entered correct a query error message appears (SearchRetry) |
  | If login information is entered correct the query information is gathered and sent to the search service by invoking a call to search service (SearchSuccess) |
  | The results and meta results are received back from the search service and the client proceeds to the results part (PresentResults). |

  o Refined search

  | The user selects one of the options to refine the query (PresentResults, PresentContent), see Figure E.4, E.5, and E.8 |
  | The new query information is gathered and sent to the search service by invoking a call to search service (SearchRefine, SearchSuccess) |
  | The results and meta results are received back from the search service and the client proceeds remain in the results part (PresentResults). |

  o Select

  | The user selects a possible match from the results list, or related objects sublist (PresentResults, PresentContent), see Figure E.3, E.9, E.10, E.11, and E.13 |
  | The content of the possible match is received after a HTTP call to the content server |
  | The user is provided with the content and suggestions (PresentContent) |

  o Logout

  | The user selects and confirms the logout option (SearchRequest, PresentResults, PresentContent), see Figure E.6 |
  | The client proceeds to the login part (LoginRequest). |
• Realization

In the process view already has been explained how the workflow behaves and all particular files work together. The class packages that are needed for proper functioning of this system are presented in Figure 6.12.

![Subsystem diagram Client](image)

**Figure 6.12 Subsystem diagram Client**

- The user/bean package consists of the following class:
  - `LoginValidation` – Checks whether both fields for username and password are filled in.

- The search/bean package consists of the following class:
  - `QueryValidation` – Checks whether the keyword is filled in after performing the first filtering (see paragraph 5.3) and checks if all attributes have a correct value.

- The search/engine package consists of the following class:
  - `ResultsOrder` – Handles the grouping and sorting of the received result set from the search service.

Also other resources are used for the client application like images, scripts, styles and applets. The classes above represent handlers of the application logic performed by the client.

**UserService**

• Interfaces

The user service only contains one interface for the outside, which is the UserService class. This class offers four functions to be invoked, as is showed in Figure 6.11 and communicates inside with the classes in the user/engine packages, as is showed in Figure 6.13.

• Realization

![Subsystem diagram User service](image)

**Figure 6.13 Subsystem diagram User service**

The elements within the package of the user service are already explained in paragraph 6.5.4 and in appendix C.
SearchService

- Interfaces

The search service only contains one interface for the outside world, which is the SearchService class. This class offers three functions to be invoked, as is showed in Figure 6.11 and communicates inside with the classes in the search/util and search/engine packages, as is showed in Figure 6.14.

- Realization

![Subsystem diagram Search service](image)

Figure 6.14 Subsystem diagram Search service

Here also each class within the packages is already explained in paragraph 6.5.4 and in appendix C.
6.8 Physical view

6.8.1 Introduction

In the physical view, also called the deployment view, the software is mapped to the hardware. Here the distributed aspect is outlined and will be described how the several software parts will be installed and perform on the municipal’s physical devices and how communication takes place with clients and other specific software. This view is mostly about the non-functional requirements [11]. There is only one configuration taken into account, since tests are directly executed in the development environment, because of the single-user license of Verity. In the next paragraph the deployment diagram is presented together with an explanation about the processing nodes, the software mapped to it and the communication between the nodes.

6.8.2 Deployment diagram

In Figure 6.15 the deployment diagram is showed. The two main hardware components are the user environment and the municipal environment, where several subcomponents are running on. The software mapped to these components will be explained in paragraph 6.8.3.
6.8.3 Implementation aspects

The complete prototype is built using Java 1.5 [38]. The choice for the Java programming language is based on the requirement that the prototype must be able to run on the systems Amsterdam currently uses. Since Java's platform independency, it always can be transferred to other hardware or operating systems. Another guideline Amsterdam maneuvers is the one of open-source software. Furthermore also the external systems the prototype uses, offer programming API's in Java. In the proof-of-concept the techniques are based on Java 2 enterprise edition (J2EE). J2EE is a platform for application development using the 1.5 standard and offers the wanted techniques and reusable components needed for this prototype. In includes the Java Development Kit (JDK) for application programming needed for the administration tool, and implementing the web services. Furthermore it offers facilities as Java Server Pages (JSP) and Java Beans needed for implementing the client. For the end-user environment it offers Java applet for navigation. For short, to let the prototype run properly, several existing technologies and libraries are used. Below an overview will be presented with all used technologies and software. For every piece of software or technology will be explained in what part of the deployment diagram it will fit in. Only Verity is omitted here since it is already explained in paragraph 2.3

- **Web browser**

For using the client application a web browser is needed that supports JavaScript, Cascading Style Sheets (CSS) and uses the Java Runtime Environment 1.5 (JRE) for being able to display Java applets. The application is only tested on Internet Explorer 6.0 for its graphical user interface. It is tested for as well Internet Explorer 6.0 as Mozilla Firefox 1.5.0.2 for functionalities.

- **Java Applets**

Java applets [38] are little applications that can be sent along with a Web page to a user. So they are always running client-side in the end-user environment. Here a Java applet fulfills the task as a navigation widget for browsing through the ontology. The web browser in which the client is invoked, must therefore use the JRE version 1.5.

- **Java Server Pages**

Java Server Page (JSP) [38] is the technology used for controlling the content and appearance of web pages in the client application through the use of servlets, small programs that are specified in the web page and run on the Web server to modify the Web page before it is sent to the user who requested it. It is often referred to as the Servlet API. So each time after a request from the end-user environment the JSP calls a Java program that is executed by the Web server before the page is sent to the user.

- **Java Beans**

JavaBeans [38] make it possible to build re-useable components that can be deployed on any operating system platform. Like Java applets, beans give web applications extra interactive capabilities. Here the client application uses Java Beans. In this case these consist of precompiled java classes that are able to handle HTML forms to check for unwanted input. Another bean is able to handle the result set that is returned by the search service. So the application logic is nicely separated from the presentation logic.

- **Apache Tomcat**

Apache Tomcat 5.5.15 [39] is used as the container for web applications that are based on JSP technologies. In this system it used for the client application to process the JSP files it consists of. This quietly implies that Apache Axis is used as the web service container. This way the prototype software will fit seamlessly in the production environment of Amsterdam.
• **Apache Axis**

Apache Axis 1.4. [40] can be used as a standalone server for providing web-services technology like offering SOAP functionality. It can simply be added to the Tomcat application container and can be used as a container for the user service and the search service. Also when an advanced installation is used, you can add the soap functionality to one of your own web applications in Tomcat. By doing this, you can create a complete self-deploying web-service that can automatically run when installed, and respond to their own service calls. This is what is done for the proof-of-concept. To keep the ease of use high and for modularity purposes, the client and both services are deployed as single web applications.

• **Sesame/SERQL**

For the data retrieval from the knowledge base, Sesame 1.2.5 0 is used. It offers a web based interface for administering the data repositories and offers an Java API for updating and querying the repository over HTTP. Although there are other solutions like Jena [41] and also several other query languages, Sesame in combination with SERQL is chosen because of its good performance [42] and its good support. Although Sesame 2.0 is also developed, it is still is an Alpha release, so here is chosen for the stable release. SERQL is from the same company as Sesame and offers two parts, which are SERQL-S for selecting or querying the data, and SERQL-C for constructing data to add to the repository. Both are used here.

• **Perl**

For the parsing functionality for converting the PIGA output to legal OWL files in the administration process, Perl [43] is used. Especially the pattern matching of Perl offers a powerful tool for fast conversion. Here also XQuery [44] was tested, but seemed about 10 times slower than the Perl scripts.

• **SOAP**

SOAP is a lightweight protocol for exchange of information in a decentralized, distributed environment [45]. The client here sends SOAP requests to the web services, and the web service returns the response in a so-called SOAP response. It can potentially be used in combination with a variety of other protocols, but in practice, it is used with HTTP. Axis offers a good support for SOAP with several standard packages. Although client and service here exist on the same hardware device, in practice they are most likely to resist on different systems and therefore must communicate via a protocol like SOAP.
6.9 Use Case scenario’s

6.9.1 Introduction

A use case scenario is an overview of the specific tasks a user can perform on the system. All possible functionalities of the system will be described using these use cases. They are used to further consolidate the previous views and explain their functionalities and structures, while e.g. the sequence diagrams on their turns also utilize use cases. Where a use case is described by a specification in a use case diagram, scenario’s can be described as a specific version of a use case and is in some sense an abstraction of the most important requirements [11]. Although the diagrams are actually redundant with the other views there are two important purposes for describing them, which are the discovery of architectural elements during the design as well as the validation of the prototype. Therefore for the two most important use cases Search and Select are used for describing real-life examples of using the prototype, which show the real intelligence of it.

6.9.2 Scenario’s

The use cases for the prototype are described in paragraph 6.5.2. Below several example scenarios are described for using the systems that comprise the use cases mentioned above. Since the current search techniques are involved in an ongoing process for improving the search facilities for the municipal’s websites, there are no real golden standards that can be used for describing scenarios for proving the quality of the intelligent search prototype. Therefore some scenarios are described with real-life examples, covering all intelligence aspects of the proof-of-concepts, which show the power of the prototype: getting faster to the wanted results.

• Scenario 1

<table>
<thead>
<tr>
<th>Goal</th>
<th>The users wants information about renewing a passport, a very often requested information object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>The user wants to enter the keyword “paspoort” but makes a typing mistake by entering “[pas[poort”, since the “p” and the “[“ are next to each other on the keyboard.</td>
</tr>
<tr>
<td>Result</td>
<td>Amsterdam</td>
</tr>
<tr>
<td>Responsible step</td>
<td>Prototype</td>
</tr>
</tbody>
</table>

• Scenario 2

<table>
<thead>
<tr>
<th>Goal</th>
<th>A foreign user searches for information about a specific billiards-club and knows that club is situated in “Biljartcentrum Osdorp”.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>The user enters the English keyword “billiardss” (with accidentally an extra s).</td>
</tr>
<tr>
<td>Result</td>
<td>Amsterdam</td>
</tr>
<tr>
<td>Responsible step</td>
<td>Prototype</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Responsible step</th>
<th>Multilingual detection</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Typographical error detection</td>
</tr>
<tr>
<td></td>
<td>Query refinement</td>
</tr>
<tr>
<td></td>
<td>Semantic information object associations</td>
</tr>
</tbody>
</table>
### Scenario 3

<table>
<thead>
<tr>
<th>Goal</th>
<th>A user wants to find information about dietitian practices in the district “oud-zuid”.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>Since this is considered as a healing “medium”. The user enters the keyword “media”.</td>
</tr>
<tr>
<td>Result</td>
<td><strong>Amsterdam</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Prototype</strong></td>
</tr>
</tbody>
</table>

**Responsible step**
- Semantic keyword associations
- Grouping by district

### Scenario 4

<table>
<thead>
<tr>
<th>Goal</th>
<th>A user wants information about a good kindergarten in the neighborhood “Oud-zuid” for his child.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>The user enters the keyword “peuterspeelzaal” and directly selects to search in his own district.</td>
</tr>
<tr>
<td>Result</td>
<td><strong>Amsterdam</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Prototype</strong></td>
</tr>
</tbody>
</table>

**Responsible step**
- Direct district refinement
- Sorting results

### Scenario 5

<table>
<thead>
<tr>
<th>Goal</th>
<th>A user wants to buy a house and therefore wants up-to-date information about the costs of a “huisvestingsvergunning” in “Oostwatergraafsmeer”.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>The user enters the keyword “vergunning”.</td>
</tr>
<tr>
<td>Result</td>
<td><strong>Amsterdam</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Prototype</strong></td>
</tr>
</tbody>
</table>

**Responsible step**
- Time refinement
- Type refinement
- Semantical information object associations
7 Prototype comparison

7.1 Introduction

During the final phase of the research a comparing research has been out carried to prove that the proposed prototype overall offered much more intelligent functionality than the current websites. To conduct a good comparing research of both search engines a few types of analyses are possible.

In principle this is a product comparison between the prototype and one of the current websites offering search possibilities, in this case the site http://www.amsterdam.nl. The essence is to make to prove the prototype is a much better practice and its features can offer advantages for a number of aspects in the current websites.

To make the correct choice it is necessary to find a structured way of finding the best testing method. Here for exist two well known methods, which are the so-called MECCA (Multi Element Component Comparison and Analysis method) [46] and the DELPHI-method [47]. These are examples of make/buy analyses which are usually used to make a decision of, as the name tells us, to make or buy a product. This can actually be mapped to this problem exactly. Buying and using existing proprietary standards and technologies or making and using open standards and data formats. Below you will find an explanation of both methods, starting with DELPHI and later on MECCA, and the continuation of the research.

- The DELPHI method mainly works along the educated opinion of the user by for example holding inquiries, in which all different aspects of the packages to be researched are questioned. Simply because of the fact that there was no objective practical knowledge available about the prototype an inquiry could not be held.

- The MECCA method for comparing products by setting a score for predefined areas related to both products. The product with the best score is the best product in this make/buy analysis. The method asks for selection criteria with several attributes separated in a few main categories in which they are hierarchically grouped. These categories with their attributes were designated along the several, but also according to the quality requirements or demands of the packages.

Because the MECCA method does not have the disadvantages the Delphi Method knows, the Mecca method is chosen.
### 7.2 MECCA

As has been outlined in the introduction of this chapter, each main category gets a certain weight allocated, which indicates the importance of that aspect within the whole of the evaluation of both search engines. These weights are assigned in correspondence with the domain experts of Amsterdam and hold of course for both alternatives. Furthermore these weights are granted in such a way that the sum of the weights together is one. A higher weighing factor (also known as the multiple factor), expresses a more important aspect. After this each category gets a score based on differences between the working of the systems from the viewpoint of the current category. This score can have a minimum of one and a maximum of five. The ultimate score will be the granted score multiplied by the weighing factor which shows the importance. This takes place per category. This total product score will be the sum of the category scores. Consequently both packages are evaluated honestly, of which some aspects are of more importance than others. Below the formula is presented used for determining the final score $s$:

$$s = \sum_{i=1}^{n} ws(i)$$

where

$n = number\ of\ categories$

$i = category\ number$

$ws(i) = wr(i) \cdot us(i),\ weighted\ score\ for\ category\ i$

$wr(i) = weighted\ ratio,\ (0 - 1)\ for\ category\ i$

$us(i) = unweighted\ score,\ (0 - 5)\ for\ category\ i$

The categories that are determined for the analysis together with the domain experts are described in Table 7.1:

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance</td>
<td>What is the response time of the system? How does it handle large result sets? Does the system crash?</td>
</tr>
<tr>
<td>Costs</td>
<td>What are the costs using the system? Are there any costs for administering the system? Are there extra costs for training purposes? Costs for new hardware or software?</td>
</tr>
<tr>
<td>Maintainability</td>
<td>Can the intelligent search procedure easily be integrated in the current systems / applications and afterwards easy maintainable for upcoming updates in comparison to the current search techniques?</td>
</tr>
<tr>
<td>Accessibility</td>
<td>Is the system easy understandable? Can inexperienced users use the system directly? Does the system not slow down the experienced users? Does the user interface not slow down the train of thoughts of the users?</td>
</tr>
<tr>
<td>Quality</td>
<td>Do the search results correspond to the expectations based on the implemented knowledge model and search procedure? Are there any linguistics used? Does the presentation of the results also offer intelligent options for further browsing through the result set? How do the use case scenarios of paragraph 6.9.2 perform on the system?</td>
</tr>
<tr>
<td>Reusability</td>
<td>Is the system built in such a way that it is reusable and easy transferable to other environments? Is it modular, so also only some components can be reused? Is it adjustable so it can also be used for other output devices?</td>
</tr>
</tbody>
</table>

Table 7.1 Categories comparing research
7.3 Analysis

The categories can be found back below in Table 7.2, where for each alternative a short description is given on how the alternative performs considering the current category. This will give a clear view of what exactly has been judged during this analysis.

<table>
<thead>
<tr>
<th>Category</th>
<th>Amsterdam.nl</th>
<th>Prototype</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quality</td>
<td>The quality is good for all regular search queries. The system returns usually the most important results on top. When this is not the case it is often hard to get to the wanted result, the user has several options to proceed, but it is not clear where to go to. Also some linguistic variants are not taken into account by the system.</td>
<td>The quality here for regular search terms is the same. But when using linguistic variants like the English language or some typing errors, the system still returns correct results. Furthermore when a wanted result cannot be found directly, the system offers enough navigation options to get to the wanted results quickly. Also see paragraph 6.9.2.</td>
</tr>
<tr>
<td>Performance</td>
<td>The performance of the system is good. It quickly shows results. How is performs with many requests at the same time has not been tested. Also larger result sets are handled well.</td>
<td>For smaller result sets the system performs well. When the complete ontology needs to be searched and the result set is bigger, the system becomes slow. This probably because of some imperfections in the software.</td>
</tr>
<tr>
<td>Costs</td>
<td>Since Verity is currently already used, the costs only consist of the hardware that is needed, the software licenses, and the consultancy from Verity. When no proprietary standards are used, someone from the municipality can administer the search process (partly), which decreases costs.</td>
<td>There still exist costs for the part Verity offers to the prototype. Furthermore the open standards that are used do not imply extra costs. Because of the accessibility there is no need for extra training. The system is built in such a way is can be understood very fast. Finally no new hardware/software is needed since prototype perfectly fits in the current environment. When Verity could be replaced the costs will decrease.</td>
</tr>
<tr>
<td>Maintainability</td>
<td>The maintainability is a little hard here, since e.g. several systems use a specific Verity component. Also since less open standards are used and the proprietary formats are Verity are maneuvered, the maintainability is restricted that only domain experts can solve problems and maintain the software. Although the system uses several parts these are a little to coherent.</td>
<td>Since the prototype in comparison with Amsterdam.nl uses a service-oriented architecture, the several parts can be easily maintained apart from each other. Also because of the open standards less tool-specific maintainability problems might occur. The automatic administration process for the ontology does not bring any extra maintainability effort along.</td>
</tr>
<tr>
<td>Accessibility</td>
<td>The accessibility is good, a query can be entered quickly, even by inexperienced users. The problem is the accessibility of the results. These are not presented clear and navigation to the wanted information is often hard.</td>
<td>After logging in with DigiD the user should get a familiar feeling with the system. Query input and result output are easy accessible. Because of the dialogue based presentation the user is guided better to the wanted information</td>
</tr>
<tr>
<td>Reusability</td>
<td>The Verity web application for searching and its proprietary standards for its data managements can not easily be integrated in other systems. Only when Verity is used on the target systems, the reusability of the software is good. Furthermore the software is not easily dividable so only some components can be reused.</td>
<td>Because of the use of open standards the reusability, also when transferring to possible other domains, is good. The service-oriented architecture with its self-deploying java-based services can easily be integrated in other environments. A slight problem is Verity with its linguistic techniques, which should be installed on the target systems. Since the software is build modular, specific parts can be left out, or easily replaced so e.g. also other output devices can be used.</td>
</tr>
</tbody>
</table>

Table 7.2 Analysis comparing research
7.4 Alternatives matrix

By comparing all attributes mentioned above, for both packages, it is now possible to draw a conclusion, using the scoring mechanism mentioned earlier. In the package-alternatives matrix in Table 7.3 below, you can see the results of the analysis. First for each category the weighted score is determined by multiplying the weighted ratio with the not weighted score. Summing up all values will determine the final score for the current alternative.

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Alternative</th>
<th>Amsterdam.nl</th>
<th>Intelligent search prototype</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Wr</td>
<td>Ns</td>
<td>Ws</td>
</tr>
<tr>
<td>1. Performance</td>
<td>15 %</td>
<td>4</td>
<td>0,60</td>
</tr>
<tr>
<td>2. Costs</td>
<td>15 %</td>
<td>3</td>
<td>0,45</td>
</tr>
<tr>
<td>3. Quality</td>
<td>25 %</td>
<td>2</td>
<td>0,50</td>
</tr>
<tr>
<td>4. Maintainability</td>
<td>15 %</td>
<td>4</td>
<td>0,60</td>
</tr>
<tr>
<td>5. Reusability</td>
<td>20 %</td>
<td>3</td>
<td>0,60</td>
</tr>
<tr>
<td>6. Accessibility</td>
<td>10 %</td>
<td>3</td>
<td>0,30</td>
</tr>
<tr>
<td>Total</td>
<td>100 %</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.05</td>
<td>3.55</td>
</tr>
</tbody>
</table>

Table 7.3 MECCA Alternatives matrix

Legend:

Wr = Weighing ratio  
Ns = Not weighted score  
Ws = Weighted score

7.5 Conclusion

Out of this research the conclusion can be drawn that especially the difference in quality and reusability were decisive to assert the prototype is overall offering more functionality despite its worse performance. Especially since the quality aspect is slightly better there can be assumed that the time to search for specific content is less and therefore, especially when used by the contact center employees, more calls be answered per time unit. This influences the costs aspect indirectly.

Concerning the scores both products were quite competitive considering the remaining attributes for each category. Of course knowing that the prototype was made because of the fact that there was a lack of functionality in the current search options, it is easy to detect it gaps and built further on that. Because still Verity had to be used, some restrictions were taken into account. Otherwise the proof-of-concept would probably have scored all points for the quality aspect.

The performance of the prototype is especially poor when no attribute refinement has been applied, so the complete ontology is searched, and when big result sets are returned. This is probably reversed proportional to the quality and something to work on for future workers.
8 Conclusions and recommendations

8.1 Conclusions

After the research, design and implementation of the proof-of-concept, conclusions can be drawn about the search concept and the prototype. These conclusions are drawn based on the research questions and goals that were formulated before starting with this research. Below, related to these questions, will be described how these questions are handled in the processes, which goals are met, and which are not met. For the goals that are not met, or other possible improvements, recommendations are offered for further work in paragraph 8.2.

The main question to see if it is possible to provide more intelligence for the searching for- and presenting of municipal information, can be answered positively. As can be seen in the comparing research the quality aspect of the proof-of-concept offers improvement. Several types of realistic questions based on important cases are answered better than by the current system. The main reasons for this are the extended usage of linguistic techniques and the constitution of the knowledge base, which uses rich relations between the concepts for faster information retrieval.

Although the searching process in improved, still both could be used better. The meta-model of PIGA is quite limited, and the generated ontology could be made much more intelligent. The goal for integrating Kana is not met. A first step has been made. The collaboration with the Verity linguistic techniques works perfect, although it would be better to leave the Verity engine out and also let the query enrichment part work together with the ontology. The same word-relations that are now administered by Verity can also be put in the knowledge base and extended furthermore, and the algorithms used by Verity are too limited to be used more freely. Therefore the search concept is restricted to the possibilities Verity offers.

Personalization is successfully added to the proof-of-concept. Although the principle of finding similar behavior is very basic, it still shows to be useful for future users. Furthermore the integration of the usage of the geographic position of a citizen has been well adopted by Amsterdam. Although there could have been used more elements from the user profile, like age or gender, this shows these kinds of attributes are useful within intelligent searching. Taking DigiD as a base for this shows for Amsterdam that in can be integrated in future techniques.

The open standards for building the knowledge base offer enough flexibility for implementing the knowledge base. Although Sesame cannot handle the complete OWL syntax, here the OWL syntax is used in such an extent that Sesame was able to add the data to its repositories. The architecture the prototype is based on also fulfils the needs of Amsterdam in the future based on the SOA architecture based on the open standards, and can easily be integrated in the future architecture of Amsterdam.

For the current situation the administration process fulfils the needs perfectly. It fits the wanted situation and it is fast. Although it would be better to build an application that is not completely automatic, it fulfils perfectly for the current data. When eventually will be chosen for a richer knowledge base with more types of data, and which is eager to be extended more often, an administration tool comes in handy, which ideally runs centrally in the ECM system of the municipality.
8.2 Recommendations

As is concluded in paragraph 8.1 the concept offers improvement but is still offers the maximum of intelligent techniques. A number of aspects are mentioned that are somehow out of scope of this research, and would have taken too much time. Therefore suggestions for further work are offered here, that can be added for future implementation.

- **Domain modeling**

  The knowledge base can be enriched by a better keyword thesaurus with more types of relations. Currently only synonyms, see-also relations and English variants are present. Also the fact the keyword concepts often contain more than one word should be changed. Only singular keyword concepts should be used, which are related to the information-objects. E.g. where now the keyword concept “kunst en cultuur” exists, only the words “kunst” and “cultuur” should be used as keyword concepts. This offers a better base for the linguistic techniques. In the best case there should be used a dictionary that is generally used which can be integrated in the system. Here Amsterdam should work together with other local governments to profile themselves as one common governmental information service.

  Furthermore in the future it is advisable to let experienced call-center agents to be able to add their specific knowledge to the system. This can be often used keywords by users that are currently not in the system, or to specify relations from a keyword to an information-object that is currently not present. Also they should be able to have influence on the ranking. When they know specific matches often occur, and the system does not rank these matches high enough, the call-center employee should be able to intervene. Another related recommendation would be that the scores in the knowledge base and in the search concept are administered real-time. Then when certain matches often occur, the system itself can give higher scores.

- **User modeling**

  Although in the concept the user models are used with success, they still can offer more to the user. The preferences’ part within the user model can contain more information so presentation generation can be used. Currently a default presentation is used for all users. Especially for the call-center agents that use the application much more often than citizens, it might be useful to detect what options an agent often uses, and e.g. leave out certain other parts of the user interface, so for that agent it is even possible to get quicker to the wanted results, since superfluous distractive parts are left out. Although the prototype is build up modular, for other devices the system can easily be rebuild and leave those parts out. But this is not generic. In the ideal situation there is one flexible system and user interface that can be built up generically based on the user’s preferences and output device.

  Since the personalization techniques are quite limited currently the system should be tested with experienced call-center agents, but also with all kinds of different citizens, who do not possess the expert knowledge. The experiences and specific needs of all users can be used to further improve the personalization part of the system. Furthermore the performance of handling the collaborative filtering technique is a little bottleneck in the prototype and is recommended to be worked out better.

  For experienced call-center employees in a specific area of the knowledge base it is recommendable to add weights to specific sub-contexts of the knowledge base. When an agent is specialized in a specific type of information, the information in the specialization area can be ranked higher.

- **Search concept**

  Since Verity is somehow restricted in the search process there might be considered to use other search techniques. Especially the word-lists Verity uses should be administered by some one of the municipality. A recommendation is, taking this proof-of-concept into account, to think of a search process that is ideally for the municipality and than look what Verity can offer. If that is not sufficient there might be considered not to use off-the-shelf software but letting developed a specific application fulfilling the exact job, extensively using ontologies, instead of proprietary formats. When using this proof-of-concept as a base, first the software need to be improved since is contains some imperfections that have a negative influence on the performance.
Bibliography

[1] Citizen Services: Government Call Center Solutions for Open Government from Accenture

[2] Verity statistics website,
http://web-zoek-prod.ver55.asp4all.nl/reporting/index.html


[4] Autonomy website,
http://www.autonomy.com

[5] Autonomy website for K2 software,
http://www.autonomy.com/content/Products/IDOL_K2/index.en.html

http://www.n3labs.com/pdf/nlp-supported-full-text.pdf

[7] P. van der Waal, 08-08-2000, PIGA specifications v1.1

[8] Luc Steegmans, 10-11-2005, Kana Ontwerppdocument versie 0.90

[9] Peter Bekkers, 03-08-2005, ECM Architectuur v1.0

[10] Gov United website,
http://www.govunited.nl/live/


[12] Scott Bradner, Key words for use in RFC’s to Indicate Requirement Levels, RFC 2119,
http://www.ietf.org/rfc.html

Ontology Development 101: A Guide to Creating Your First Ontology


[16] Jeen Broekstra, 14-03-2006, Lecture slides, Modeling in RDF


[18] Alfred Kobsa, 1993, User Modeling: Recent Work, Prospects and Hazards,

[19] Digital Identity website
http://www.digid.nl

[21] Arnd Kohrs, Content based filtering,  
http://www.kom.e-technik.tu-darmstadt.de/acmmm99/ep/kohrs/  

[22] F. Heylighen, 2001, Collaborative Filtering  
http://pespmc1.vub.ac.be/COLLFILT.html  

[23] Michael J. Pazzani, Irvine Irvine, A Framework for Collaborative, Content-Based and Demographic Filtering,  
http://www.ics.uci.edu/~pazzani/Publications/AIREVIEW.pdf  


[25] Google website,  
http://www.google.com/  


[28] Amit Sheth, Clemens Bertram, David Avant, Brian Hammond, Krysztof Kochut, Yashodhan Warke, 2002 Managing Semantic Content for the Web  


[32] Unified Modelling Language website,  
http://www.uml.org  


[34] Sesame website,  
http://www.openrdf.com  


[38] Sun Java website,  
http://java.sun.com  

[39] Apache Tomcat website,  
http://tomcat.apache.org  

[40] Apache Axis website,  
http://ws.apache.org/axis
[41] Jena website,  
http://jena.sourceforge.net

http://simile.mit.edu/reports/stores/

[43] Perl website,  
http://www.perl.org

[44] XQuery specification,  
http://www.w3.org/TR/xquery/

[45] SOAP specification,  
http://www.w3.org/TR/SOAP


http://www.w3.org/2001/sw/

http://www.w3.org/Submission/WSMO/#L2326

http://www.loa-cnr.it/Papers/FOIS98.pdf


[52] Resource Description Framework, RDF Triples,  
http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/#section-Graph-syntax

[53] American National Standards Institute,  
http://wwwansi.org/

[54] Unicode website,  
http://www.unicode.org/

[55] Web naming and addressing URI and URL,  
http://www.w3.org/Addressing/

[56] RFC3986: Uniform Resource Identifier (URI): Generic Syntax,  
http://rfc.net/rfc3986.html

[57] XML - eXtensible Markup Language,  
http://www.w3.org/XML/

[58] XML Schema  
http://www.w3.org/XML/Schema


[64] D. Brickly, R. Guha, 23-01-2003, RDF vocabulary description language 1.0: RDF Schema, W3C working draft


[68] Derry OSullivan, Barry Smyth, David C. Wilson, Kieran McDonald and Alan Smeaton, Improving the Quality of the Personalized Electronic Program Guide


[70] Amit Sheth, Clemens Bertram, David Avant, Brian Hammond, Krysztof Kochut and Yashodhan Warke, 2002, Managing Semantic Content for the Web


Appendix A: Semantic Web technologies

1. Introduction

According to [48] the semantic web is about two things. It is about common formats for interchanging data, where on the original Web only the interchange of documents existed. Furthermore it is about languages for recording how the data relates to real world objects. That allows a person, or a machine, to start in a specific database and then through an unending set of databases that not are connected by wires but by being about the same thing. In this thesis semantic web technologies are used to create an intelligent ‘web’ of data for searching within municipal data resources as described in chapter 3.

In every intelligent system where a user can retrieve and use data, some sort of metadata is needed which is able to describe specific content elements. When a system needs to guide the user to specific information, it must be able to parse the data based upon this metadata before effectively sending it to the user. In this perspective it is very important that the format and style of the metadata is clearly specified for as well the clarity for the user as the performance of system. For short it must be computer readable, interpretable, understandable and of course available. For this goal several open standards are specified by [48] that can fulfill the needs. This matches the requirement of Amsterdam who, as a local government, strive for open standards and open source products.

In the prototype the domain model has a central role and describes the domain concepts that are used by the search systems using the open standard languages. The language used to represent the domain model influences what the search system can do with the information provided by the domain model. In the prototype the domain model is described by a repository of information-object concepts, keyword concepts and their relationships based on the meta model from the existing data resource PIGA (see paragraph 2.4.1). This ontology suffices for the prototype, allowing the system to relate the information-objects to the concepts and mutual. The prototype needs to be able to query this information, using its relations to create a result set based on the specified input query and provide smart presentation and navigation options to guide the user to the wanted information, if available.

So since the goal is to build a prototype for semantic searching first an overview is presented of the available semantic web languages. Since the semantic web is still under development only the currently available techniques are discussed (see chapter 3 of this appendix). First in chapter 2 of this appendix a definition is given of what an ontology exactly is. Finally in chapter 4 of this appendix, after the different techniques, the specific building blocks of the chosen language will be outlined that are needed for creating the ontology.

Later in the architectural design is shown how the ontology is embedded in the whole and especially how it communicates with the other parts, like the current search engine of Verity with its proprietary format, and how specific modeling issues were handled.
2. Definition

The term ontology originates from the field of philosophy, where it means a description of the nature of being (a "theory of existence"). In Computer Science (CS) and especially the semantic web world the most frequently quoted definition is "a specification of a conceptualization" [49] and is seen as an indication for different types of information retrieval systems. The main difference is that in Philosophy an ontology is a systematic and exhaustive account of Existence, only driven by the task of making the ontology.

When creating ontologies in CS there always should be kept in mind what exactly the meaning is of making specific choices, and therefore restrictions of concepts included in the ontology. A specific domain, as part of the world, is represented, where it is common and better to leave out the concepts that are not relevant for the task having in mind. State that the definition, "an ontology is a specification of a conceptualization," leaves room for many possible interpretations, and despite an attempt to clarify and formalize the definition further in [50], new meanings of the term ontology continued to proliferate. In [51] this spectrum (see Figure A.1) of information artifacts, which at some time had been classified as ontologies, is discussed. On the hand of the results of that work the different types of information artifacts will be explained, so a better insight in ontologies can be acquired.

![Figure A.1 Spectrum of information systems](image)

As can be seen there are several types of information systems visualized. All are shortly explained below to finally present the definition of ontologies as used in this thesis.

- A **catalogue** (or controlled vocabulary) is nothing more than an enumerated list of all relevant classes/concepts within the domain in a tree structure (classification).
- A slightly more complex information system may provide a **set of simple natural language texts files** and allow string matching.
- A **glossary** is an indexed catalogue with a natural language description and preferred terms for each concept/class.
- A **thesaurus** is a glossary with a hierarchical structure.
- A **taxonomy** is a collection of catalogue terms organized into a hierarchical structure, where each term in a taxonomy is in one or more parent-child relationships to other terms in the taxonomy. The properties of more general classes/concepts are inherited by the more specific classes/concepts. Also the content is included.
- A **frame-based system** is a taxonomy, which provides relations between objects and restrictions on what and how classes/concepts of objects can be related to each other.
- A **formal ontology** is a controlled vocabulary expressed in an ontology representation language (see paragraph 0) that consist of a grammars for using vocabulary terms to express something meaningful within a specified domain of interest. The grammar contains formal constraints (e.g., specifies what it means to be a well-formed statement, assertion, query, etc.) on how terms in the catalogue of the ontology can be used together and form a semantic relation.
3. Metadata Definition Languages

3.1. Introduction

Metadata Definition Languages (MDL) are used to make the ontologies computer interpretable. For this purpose there are several standards and description languages developed, which are visualized in the language stack below in Figure A.2.

![Figure A.2 Semantic layer cake](image)

The three highest levels, which should eventually lead to trustability of the presented information and knowledge, still need to be built. Therefore only the lower levels are explained below, which are also used for the prototype, as has been indicated earlier.

From bottom-up the essential layers are organized as follows. Each successive level is an extension of its prior one. ASCII, Unicode and URI’s form the technical base structures that need to be adhered by layers on top. XML only exists of the syntactic level. XML-tags (defined by DTD or a XML scheme) already can give certain meaning to the parts of its own contents, where its syntactic level is only known locally. RDF/RDFS is used to give meaning in more standardized and better organized way by adding limited semantics. Finally an ontology vocabulary like OWL allows the full expressiveness of ontologies a way it is computer interpretable. In the following paragraphs these description languages are introduced to get a better understanding of their meaning and possibilities. It must be noticed that there are several ways to input the data, or “triples”, as listed in [63], but in this thesis and in the prototype only the XML variant is used for creating the ontology.
3.2 Basic structures

**Unicode**

Although the predecessor of XML, SGML, was based on the American Standard Code for Information Interchange (ASCII) [53], XML itself is based on the character set UNICODE [54], which is just a standard containing a set of well specified character codes able to code multilingual texts. ASCII is the most common format for text files in computers and on the Internet but fairly old. Newer systems all use the Unicode standard. There are several schemes to encode the Unicode characters. Often used, and also in the prototype is UTF-8, where UTF stands for Unicode Transformation Formats, and the 8 only tells it is the 8 bit standard.

**URI**

According to [55] a Uniform Resource Identifier (URI) [56] is a compact sequence of characters that identifies an abstract or physical resource. Such a single abstract, e.g. a XML document, contains elements and attributes that are defined for usage by all kinds of software tools. All these different applications need to be able to recognize the tags and attributes, especially when they are using more than one descriptive document. To prevent "collisions", occurring when mark-up can belong to more then one definition standard, the mechanism of namespaces identifies the set of tags where a predicate is specified. Names from namespaces may appear as qualified names, which contain a single colon, separating the name into a namespace prefix and a local part. The prefix, which is mapped to a URI reference, selects a namespace. The combination of the universally managed URI namespace and the document's own namespace produces identifiers that are universally unique. URI references, which identify namespaces, are considered identical when they are exactly the same character-for-character and may always be functionally equivalent. A namespace is declared using a family of reserved attributes. Such an attribute's name must either be xmlns or have xmlns: as a prefix. So for short it is a local shortcut to a document for preventing naming collisions.

3.3 Extensible Markup Language

The Extensible Markup Language (XML) [57] is defined by the W3C and is the lowest description language for the semantic web. XML itself is just a text format based on Unicode and does not consist of any structuring rules when describing metadata. It does not provide for semantics but only flexible syntax.

In comparison to HTML, XML has no fixed set of tags and semantics for it, and uses it to create all kinds of structures. Although the developer is allowed to choose tag names, there are some rules. Every opening tag must have a corresponding closing tag. Tag pairs may be nested. Opening tags can contain attributes with quoted values. Data may be freely mixed with tags, although most of the time data is enclosed within a tag pair.

Adding structure or rules to an XML file can be achieved by means of a DTD (Document Type Definition) or XML Schema [58]. According to W3C, the purpose of a Document Type Definition is to define the legal building blocks of an XML document. It defines the document structure with a list of legal elements. A DTD can be declared inline in an XML document, or as an external reference. Like DTD's, XML Schema can define the structure, content and semantics of an XML document. It is specified in XML itself. However, an XML schema allows for stricter validation of an XML document; it can do everything a DTD can do and more. For example, a DTD can only define the type of the content within elements of an XML document as 'string', where an XML schema can specify the required type of content within an element. The schema may be irregular or not completely defined and is therefore called semi structured, which is, according to [59] defined as object like, schema-less and self-describing. To prevent the occurrence of equally named elements, namespaces can be used.

XML in combination with a DTD or XML Schema is currently widely used (according to W3C) as a technique for description of metadata. XML is so established and well-known that it can be used in almost all circumstances. In the architecture chapter can be seen that also SOAP is based on XML. The only disadvantage of an XML Document is that it is very inefficient in bandwidth and storage space, because it is very verbose.
3.4. Resource Description Format

RDF

The Resource Description Format (RDF) [60] is built on top of the syntax of XML and has been specified for being able to add meaning/semantics to documents for describing and interchanging purposes, as has been explained in the previous paragraph. XML elements, defined via DTD or XML schema, already can specify meaning to parts of document content itself. RDF does the same on a more standardized manner and in a better organized way. But when e.g. in XML a tag `<city>` is defined, a computer needs to detect what it exactly means, but still has no idea of its content. The big advantage of RDF is that it is able to reuse other semantic systems and can be identified via the namespaces mentioned earlier and findable by its URI. So actually a namespace is a web-link to usable semantics.

RDF is meant to make statements about resources. The foundation of RDF is a model for representing resources together with named properties and property values, which both are so-called first-class citizens. The basic data model of RDF consists of three object types:

- **Resources**: All things being described by RDF expressions are called resources. This may be web pages or other objects that can uniquely be identified. The resource identifier is the earlier mentioned URI.
- **Properties**: A property is some kind of relation, aspect or attribute used as a type/value combination to describe a resource, where the value is represented as a literal.
- **Statements**: A specific resource together with a named property plus the value of that property for that resource is an RDF statement.

It is intended to provide a simple way to make statements about Web resources and is based on the idea of identifying content using Web Identifiers (URI’s) (see paragraph 3.2) and describing this content in terms of simple named properties and property values. Each property's value is either another resource (specified by a URI) or a literal (a string encoded conforming to XML-specified syntax rules). Data in RDF is modeled in the form of a set of statements, or RDF triples, which contains some information about that resource. The three elements of a single statement are the resource being described, the property's name, and the property's value. Or differently named subject, predicate and object:

- **The subject**: The resource about which the statement is made, which is a URI, blank node or another statement, since they can be nested and chained.
- **The predicate**: A certain property of the subject.
- **The object**: The value of the property mentioned by the predicate, which can be another resource or a literal (any primitive data type defined in XML).

The subject-predicate-object triples can be interpreted as a directed labeled graph, where both subjects and objects are nodes and predicates define edges or relations between both. Subject and object are interchangeable. The object of one triple may be the subject of another triple. This does not hold for literals, which can only be subjects.
A simple graph is presented in Figure A.3 below. The triple is: <Subject>Document1<Property>Author<Object>"John Smith", which denotes that John Smith is the author of Document1.

![Simple RDF statement](image)

Figure A.3 Simple RDF statement

For more information consult [61] and [62].

**RDFS**

RDF properties may be thought of as attributes of resources and in this sense correspond to traditional attribute-value pairs. RDF properties also represent relationships between resources. However, even though RDF provides for resource description with help of a graph model it still is not machine interpretable. Because anything may be stated about anything a machine still cannot be interpret the intended semantics of some relation nor does RDF provide any mechanisms for describing the relationships between properties and other resources. Although RDF abstracts its data model from its syntax, there is still a need for a universal agreement of the semantics and interpretation of certain terms. This way the semantics of statements are machine understandable and graph models are reusable for several resources. That is were RDFS [63] comes in. RDF Schema defines a language on top of RDF that supports the definition process. By predefining a small RDF vocabulary for defining other RDF vocabularies, RDF schema can be used to specify the vocabulary for particular application domains, which is not possible with RDF. RDF Schema structures give sufficient information to allow more expressive queries regarding the concepts semantics and their relationships in the application domain.

RDFS provides the missing semantics of RDF. It is extremely useful for specifying the kind of objects to which predicates can be applied. RDFS expressions are also valid RDF expressions. Important properties are subClassOf, domain and range. This extra structure can be used to define queries. For more information about RDFS refer to [64].
3.5. Web Ontology Language

Using RDFS still the number of semantic facilities is quite restricted. Also the ones that are available are not powerful enough and in some cases not sufficient. To extend the semantic web in such a way that it is machine understandable a powerful language is needed to create and formally describe ontologies.

Therefore the Semantic Web effort has produced OWL [65], an ontology language for the web. As described in OWL [66] is intended to be used when the information contained in documents needs to be processed by applications and can be used to explicitly represent the meaning of terms in vocabularies and the relationships between those terms. This is done by defining the domain in terms of individuals, classes and properties. Individuals refer to real-world entities with relations between each other (properties).

Individuals can be grouped in classes of individuals sharing some common properties. OWL further defines basic properties which enforce a hierarchy of classes and properties. Furthermore it also supports in contrast to RDF reasoning over its properties. By supporting this, OWL even can inference new properties and links based upon present information.

OWL supports property restrictions, cardinality restrictions, versioning and annotation features, class axioms like disjunction, union, intersection, and many more features. There are three different sublanguages of OWL to suit different groups of users. These are called OWL Lite, OWL DL and OWL Full, with OWL Lite being the simplest and OWL Full being the most expressive of the three. OWL Full can be viewed as an extension of RDF, while OWL Lite and OWL DL can be viewed as extensions of a restricted view of RDF.

OWL Lite is designed for users, who primarily need a classification hierarchy and simple constraints. OWL DL is designed for maximum expressiveness while retaining computational completeness and decidability. OWL DL includes all OWL language constructs, but they can be used only under certain restrictions. OWL Full is meant for maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. For example, in OWL Full a class can be treated simultaneously as a collection of individuals and as an individual on its own.
3.6. Building blocks

So an OWL model is a set of RDF triples which define a set of classes, individuals and OWL properties. An OWL model provides no guarantee that the information described is sound, but can be verified by an observer following rules of logic and observations of the world, like Sesame (see paragraph 6.8.3) or Jena [41]. The building blocks mentioned are those that are needed for the goal of the prototype. There are basically four building blocks in ontologies:

- Classes (concepts)
- Properties, which describe the features and attributes of the classes
- Restrictions, which describe the value a property is allowed to have
- Individuals, these are instances of a class with certain restrictions

Below the most important constructs used are outlined and an explanation is given why they are useful regarding the modeling of the domain:

<table>
<thead>
<tr>
<th>Tag Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owl:Class</td>
<td>Each class in the ontology is defined by this tag and represents a set of individuals</td>
</tr>
<tr>
<td>rdf:ID</td>
<td>Uniquely identifies a class in the ontology as attribute of the owl:Class tag</td>
</tr>
<tr>
<td>rdf:about</td>
<td>Describes another uniquely identified resource as attribute of the owl:Class tag</td>
</tr>
<tr>
<td>rdfs:subClassOf</td>
<td>Defines a class as a subclass of another class as attribute of the owl:Class tag. Here all subclasses of information-object are defined using this construct.</td>
</tr>
<tr>
<td>owl:ObjectProperty</td>
<td>Represents a link between two classes or individuals</td>
</tr>
<tr>
<td>owl:DataTypeProperty</td>
<td>Represents a link between a class or individual and data of a certain type (string, int, etc.)</td>
</tr>
<tr>
<td>rdfs:domain</td>
<td>Represents for both ObjectProperty and DatatypeProperty the domain of the relation.</td>
</tr>
<tr>
<td>rdfs:range</td>
<td>Represents for both ObjectProperty and DatatypeProperty the range of the functions.</td>
</tr>
<tr>
<td>rdfs:subPropertyOf</td>
<td>Defines a property as a subproperty of another property as attribute of the owl:ObjectProperty tag or owl:DataTypeProperty tag. Here all subproperties of information-object relations and keyword relations are defined using this construct.</td>
</tr>
<tr>
<td>rdfs:label</td>
<td>Defines the title of a domain concept</td>
</tr>
<tr>
<td>xml:lang</td>
<td>Defines the language of concept label as attribute of the rdfs:label tag</td>
</tr>
<tr>
<td>individual</td>
<td>An individual of a specific class with all the contracts that are defined for the class representing its information</td>
</tr>
</tbody>
</table>

| Table A.1 OWL building blocks |
4. Conclusion

So for the prototype has been chosen for the expressivity of OWL, since it is more expressive than XML and RDF and needs to perform reasoning tasks. Expressibility refers to the type and complexity of the information the language can describe. Reasoning means that software agents are able to infer related information based on the ontology, find specific information relevant to a concept and determine whether some statement about a concept or a set of concepts is true or not. All this reasoning can be done independently of the actual contents in the ontology, which is therefore left out of the ontology and is just a set of documents identifiable by the content id stored in the ontology.

Although the interpreter Sesame is not able to understand all OWL syntax constructs when saving information into its triple store, it still can be used for the simple constructs and then OWL offers a better understandable syntax. Sesame (see paragraph 6.8.3) on its turn can convert OWL syntax back to RDF statements. Furthermore with the basic OWL constructs it already presents a solid base for future purposes when Amsterdam decides to enrich its information and wants to get more out of the ontology languages presented in the previous paragraphs. Since it is XML based it remains easy to share the ontology with other systems, and even with systems of other cities, to eventually be able to make it a nationwide initiative.

The way the ontology is built up now offers class based reasoning for retrieving specific parts of the ontology to search in. Furthermore it consists of several communicating smaller ontologies with well defined mappings between them to keep it modular for maintainability purposes and transferability. Also reusability by other systems is supported by this choice. In chapter 6 will become clear how the semantic web technologies will be combined with the existing techniques.
Appendix B: Domain & User models

B1. Mapping domain & user models

Mapping: Ontologies

B2. Domain models

Dictionary

Mapping: HasKeyWord

Dictionary is a mapping between KeyWords and InformationObjects. Each InformationObject concept is related to one or more KeyWord concepts through HasKeyWord and its inverse is IsKeyWordOf. Indicates each keyword concept is related to one or more InformationObject concepts.
Submappings: HasKeyWord

HasKeyWordWeak is a sub-property of HasKeyWord and indicates that an information-object has a weak relation to a keyword with a weighing value of 0.33.

HasKeyWordMiddle is a sub-property of HasKeyWord and indicates that an information-object has a middle relation to a keyword with a weighing value of 0.67.

HasKeyWordStrong is a sub-property of HasKeyWord and indicates that an information-object has a strong relation to a keyword with a weighing value of 1.00.

Table B.1 Dictionary overview
Content

Concept: InformationObject

The InformationObject concept has the following properties:

- **HasKeyWord**: indicates a related KeyWord
- **InformationObjectRelation**: Indicates a related InformationObject
- **DistrictName**: Indicates the district it is situated in
- **ContentId**: Indicates the unique id
- **PubDate**: Indicates the publication date
- **NrViews**: Indicates the number of views

SubConcept: Instelling

The Instelling concept is a subclass of the InformationObject class.

Relations: InformationObject

- Each information-object retrieves a unique content id indicated by a positive integer
- Each information-object retrieves a district name indicated by a District concept
- Each information-object has a number of views indicated by a positive integer
- Each information-object has a publication date specified by a positive integer
The InformationObjectRelation concept indicates that an InformationObject concept is related to zero or more other InformationObject concepts.

Each InformationObjectRelation concept has a weight indicated by a decimal in the range of \{0.33, 0.66, 1.00\}.

The LocatieVoor property is a sub-property of an InformationObjectRelation concept and has a specific weight.
Instance: Instelling (InformationObject)

An instance of InformationObject concept with the following values:

- **Label**: Fietskluis Metrohalte
- **ContentId**: 102576
- **DistrictName**: DIS_9
- **NrViews**: 23
- **PubDate**: 18-05-2000
- **WordtBeheerdDoor**: Parkeergebouwen Amsterdam
- **HasKeyWord**: fietsen
- **Type**: Instelling

| Table B.2 Content overview |
Keywords

Concept: KeyWord

The KeyWord concept has the following properties:

- **IsKeyWord**: indicates a related InformationObject
- **KeyWordRelation**: Indicates a related KeyWord
- **ContentId**: Indicates the unique id

Relations: KeyWord

Each KeyWord retrieves a unique keyword id indicated by a positive integer

Concept and Instance: KeyWordRelation

The KeyWordRelation concept indicates that a KeyWord concept is related to zero or more other KeyWord concepts.

Each KeyWordRelation concept has a weight indicated by a decimal in the range of {0.33, 0.66, 1.00}
Appendix B: Domain & User models

The OuderVan property is a sub-property of a KeyWordRelation concept and has a specific weight.

Table B.3 KeyWord overview

<table>
<thead>
<tr>
<th>Label</th>
<th>KeyWordId</th>
<th>OuderVan</th>
<th>ZieOok</th>
<th>IsKeyWordOf</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heroïneverstrekking</td>
<td>20882</td>
<td>verslaafden</td>
<td>verslavingszorg</td>
<td>Polikliniek Zuidoost</td>
<td>KeyWord</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Medische suppletie Unit</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

An instance of the KeyWord concept with the following values:

- **Label**: Heroïneverstrekking
- **KeyWordId**: 20882
- **OuderVan**: verslaafden
- **ZieOok**: verslavingszorg
- **IsKeyWordOf**: Polikliniek Zuidoost
- **IsKeyWordOf**: Medische suppletie Unit
- **Type**: KeyWord
**Districts**

**Concept: District**

The District concept has the following properties:

- **NeighbourOf**: indicates the neighbouring districts of this district

**Relations: District**

Each District has zero or more neighbouring Districts

**Instance: District**

An instance of KeyWord concept with the following values:

- **Label**: centrum
- **NeighbourOf**: oud zuid
- **Type**: District

Table B.4 District overview
B3. User Models

Statistics

Mapping: HasSession

Statistics is a mapping between Users and Sessions. Each User concept is related to one or more Session concept through HasSession and its inverse is IsSessionOf indicates each Session concept is related to one User concept.

Table B.5 Statistics overview
Settings

Mapping: HasPreferences

Settings is a mapping between Users and Preferences. Each User concept is related to one Preferences concept through HasPreferences and its inverse is IsPreferencesOf indicates each Preferences concept is related to one User concept.

Table B.6 Settings overview
Users

Concept: User

The User concept has the following properties:

- HasPreferences: indicates the related preferences
- HasSession: Indicates a related session
- UserId: Indicates the unique id
- UserName: Indicates the username
- PassWord: Indicates the password
- FirstName: Indicates the first name
- SurName: Indicates the family name
- EmailAddress: Indicates the email adress
- StreetName: Indicates the name of the street
- StreetNumber: Indicates the house number
- ZipCode: Indicates the zipcode
- DistrictName: Indicates the name of the district
- CityName: Indicates the name of the city
- SofiNumber: Indicates the social security number
- DateRequest: Indicates the date the account was requested
- DateExpire: Indicates the date the account expires
- DateActivation: Indicates the date the account was activated
Appendix B: Domain & User models

Relations: User

- Each user retrieves a unique user id indicated by a positive integer.
- Each user retrieves a unique user name for authentication indicated by a string.
- Each user has a first name indicated by a string.
- Each user has a surname indicated by a string.
- Each user retrieves a password indicated by a string.
- Each user has a unique social security number.
- Each user has a unique email address indicated by a string.
- Each user lives in a street indicated by a string.
- Each user lives on a number in a street indicated by a positive integer.
- Each user lives in a district indicated by a string.
Each user lives on a zip code indicated by a string

Each user lives in a city indicated by a string

Each user requested the profile on a specific date indicated by a string

Each user activated the profile on a specific date indicated by a string

Each user profile expires on a specific date, when not activated, indicated by a string

Table B.7 User overview
Instance: User

An instance of the User concept with the following values:

- HasPreferences: Preferences with id 1
- HasSession: Session with id 2
- UserId: 1
- UserName: ikrooswijk
- PassWord: 760000
- FirstName: Ivo
- SurName: Krooswijk
- EmailAddress: ivo@krooswijk.com
- SofiNumber: 102938422
- StreetName: Albert Cuyp straat
- StreetNumber: 270
- ZipCode: 1073
- DistrictName: oud zuid
- CityName: Amsterdam
- DateActivation: 26-05-2006
- DateExpire: 17-06-2006
- DateRequest: 17-05-2006
- Type: User
Sessions

Concept: Session

The Session concept has the following properties:

- **IsSessionOf**: indicates the related user
- **HasSessionQuery**: Indicates a related query
- **SessionId**: Indicates the unique id of this session for all users
- **Session**: Indicates the unique number of the session for this user

Relations: Session

Each session consist of one or more queries indicated by the SessionQuery concept.

Each session has a unique session number indicated by a positive integer.

Each session has a unique session id provided by the session mechanism indicated by a length 32 character string.
Instance: Session

An instance of the Session concept with the following values:

*HasSessionQuery*: SessionQuery with id 1
*HasSessionQuery*: SessionQuery with id 2
*SessionId*: E69FD43E901275FDDF8F4837F8B39B0D
*SessionNr*: 1
*IsSessionOf*: User with id 2
*Type*: Session
Concept: SessionQuery

The SessionQuery concept has the following properties:

- **HasViewed**: indicates the information-object that has been viewed
- **QueryNr**: indicates the number of the query in the session

Relations: SessionQuery

Each session query has a unique query number indicated by a positive integer.

Each session query consists of one or more viewed content items id’s indicated by a positive integer.
Instance: SessionQuery

An instance of the SessionQuery concept with the following values:

- HasViewed: SessionQuery with id 1
- QueryNr: 1
- Type: SessionQuery

Table B.8 Session overview
Preferences

Concept: Preferences

The Preferences concept has the following properties:

- ArePreferencesOf: indicates the related User
- EmailNotification: Indicates the email notification
- HitsPerPage: Indicates the number of hits per page

Relations: Preferences

- Each preference has a number of hits per page indicated by a positive integer.
- Each preference has a email notification indicated by a Boolean.
An instance of the Preferences concept with the following values:

*ArePreferencesOf*: User with id 1  
*HitsPerPage*: 10  
*EmailNotification*: true  
*Type*: Preferences

Table B.9 Preferences overview
# UserService Class

```java
public class UserService
extends java.lang.Object
```

**Title:** User Service  
**Description:** The main class for the user service which provides the functionalities to be invoked by all clients for finding information in the user models, by calling the user engine  
**Copyright:** Copyright (c) 2006  
**Company:** Everest b.v.  
**Version:** 1.0  
**Author:** Ivo Krooswijk

## Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
</table>
| ```java
UserService()
``` | Constructor for UserService class |

## Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| ```java
java.util.HashMap getMostViewed(java.lang.String userId)
``` | Returns the most viewed attributes of the information-objects viewed by a specific user |
| ```java
java.util.HashMap getStatistics(java.util.HashMap sessionStats)
``` | Retrieves the statistics based on a comparison of the current user's behavior and and other users' behavior |
| ```java
java.util.HashMap getUserProfile(java.util.HashMap profileMap)
``` | Returns the user profile and preferences based on a login profile |
| ```java
void updateStatistics(java.util.HashMap userStats)
``` | Updates the behavior of a single user by adding the content ids of the viewed information-objects to the session of the user |
### Package user.engine

#### Class Summary

<table>
<thead>
<tr>
<th>Class</th>
<th>Title</th>
<th>Description</th>
<th>Copyright</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>StatsSearcher</td>
<td>Statistics Searcher</td>
<td>Queries the RDF store for statistics. Here the behavior of a specific user can be retrieved and the behavior comparing to other users, also the most viewed items can be retrieved.</td>
<td>Copyright (c) 2006</td>
<td>Everest b.v.</td>
</tr>
<tr>
<td>StatsUpdater</td>
<td>Statistics Updater</td>
<td>Updates the statistics with a specific user's search behavior by storing viewed information-objects within a query</td>
<td>Copyright (c) 2006</td>
<td>Everest b.v.</td>
</tr>
<tr>
<td>UserEngine</td>
<td>User Engine</td>
<td>Provides functionality for querying and updating the user repository for user information and statistics</td>
<td>Copyright (c) 2006</td>
<td>Everest b.v.</td>
</tr>
<tr>
<td>UserSearcher</td>
<td>User Searcher</td>
<td>Queries the RDF store for the user profile and preferences.</td>
<td>Copyright (c) 2006</td>
<td>Everest b.v.</td>
</tr>
</tbody>
</table>
public class StatsUpdater extends java.lang.Object

Title: Statistics Updater
Description: Updates the statistics with a specific user's search behavior by storing viewed information-objects within a query
Copyright: Copyright (c) 2006
Company: Everest b.v.

Version: 1.0
Author: Ivo Krooswijk

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StatsUpdater()</td>
<td>Constructor for StatsUpdater class</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>closeConnection()</td>
<td>Closes the connection to the main-memory user repository on the Sesame server</td>
</tr>
<tr>
<td>openConnection(java.lang.String serverSpec, java.lang.String loginName, java.lang.String passWord, java.lang.String repositoryId)</td>
<td>Opens the connection to the main-memory user repository on the Sesame server</td>
</tr>
<tr>
<td>updateStats(java.util.HashMap userStats)</td>
<td>Updates the behavior of a single user by adding the content ids of the viewed information objects to the session of the user</td>
</tr>
</tbody>
</table>
user.engine
Class UserEngine

java.lang.Object

public class UserEngine
extends java.lang.Object

Title: User Engine
Description: Provides functionality for querying and updating the user repository for user information and statistics
Copyright: Copyright (c) 2006
Company: Everest b.v.

Version: 1.0
Author: Ivo Krooswijk

Constructor Summary
UserEngine()
Constructor for the UserEngine class

Method Summary
java.util.HashMap getMostViewed(java.lang.String userId)
Retrieves a list of titles and ids of information-objects that are most viewed by a specific user

java.util.HashMap getStatistics(java.util.HashMap statsMap)
Searching in the user repository for statistics based on the own search behavior and other users' search behavior

java.util.HashMap getUserProfile(java.util.HashMap profileMap)
Retrieves the attributes from the profile and preferences of a specific user

void updateStatistics(java.util.HashMap userStats)
Updates the behavior of a single user by adding the content ids of the viewed information-objects to the session of the user
user.engine
Class StatsSearcher

public class StatsSearcher
extends java.lang.Object

Title: Statistics Searcher
Description: Queries the RDF store for statistics. Here the behavior of a specific user can be retrieved and the behavior comparing to other users, also the most viewed items can be retrieved.
Copyright: Copyright (c) 2006
Company: Everest b.v.

Version: 1.0
Author: Ivo Krooswijk

Method Summary

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>closeConnection()</td>
<td>Closes the connection to the main-memory user repository on the Sesame server</td>
</tr>
<tr>
<td>getMostViewed(String userId)</td>
<td>Retrieves the attributes and values that most often occur in the viewed information-objects of a specific user</td>
</tr>
<tr>
<td>getOtherUsersHistory(String viewedPath, String userId)</td>
<td>Retrieve suggestions on behalf of other users' search behavior</td>
</tr>
<tr>
<td>getUserHistory(String viewedPath)</td>
<td>Retrieves a list of titles and ids of information-objects viewed in the current session</td>
</tr>
<tr>
<td>openConnection(String serverSpec, String loginName, String passWord, String userRepositoryId, String contentRepositoryId)</td>
<td>Opens the connection to the main-memory user repository on the Sesame server</td>
</tr>
</tbody>
</table>
### user.engine

**Class StatsUpdater**

```java
java.lang.Object
   user.engine.StatsUpdater

public class StatsUpdater
    extends java.lang.Object
```

**Title:** Statistics Updater  
**Description:** Updates the statistics with a specific user's search behavior by storing viewed information-objects within a query  
**Copyright:** Copyright (c) 2006  
**Company:** Everest b.v.

**Version:** 1.0  
**Author:** Ivo Krooswijk

### Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>StatsUpdater()</strong></td>
<td>Constructor for StatsUpdater class</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void closeConnection()</code></td>
<td>Closes the connection to the main-memory user repository on the Sesame server</td>
</tr>
<tr>
<td><code>void openConnection(java.lang.String serverSpec, java.lang.String loginName, java.lang.String passWord, java.lang.String repositoryId)</code></td>
<td>Opens the connection to the main-memory user repository on the Sesame server</td>
</tr>
<tr>
<td><code>void updateStats(java.util.HashMap userStats)</code></td>
<td>Updates the behavior of a single user by adding the content ids of the viewed information objects to the session of the user</td>
</tr>
</tbody>
</table>
## User engine

### Class UserSearcher

```java
public class UserSearcher extends java.lang.Object
```

**Title:** User Searcher  
**Description:** Queries the RDF store for the user profile and preferences.  
**Copyright:** Copyright (c) 2006  
**Company:** Everest b.v.  

**Version:** 1.0  
**Author:** Ivo Krooswijk

### Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>UserSearcher()</code></td>
<td>Constructor for UserSearcher class</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void closeConnection()</code></td>
<td>Closes the connection to the main-memory user repository on the Sesame server</td>
</tr>
<tr>
<td><code>void openConnection(java.lang.String serverSpec, java.lang.String loginName, java.lang.String passWord, java.lang.String repositoryId)</code></td>
<td>Opens the connection to the main-memory user repository on the Sesame server</td>
</tr>
<tr>
<td><code>java.util.HashMap searchUser(java.util.HashMap profileMap)</code></td>
<td>Retrieves a list of all needed profile and preference attributes of the user that logs in</td>
</tr>
</tbody>
</table>
## Class SearchService

```java
public class SearchService extends java.lang.Object
```

<table>
<thead>
<tr>
<th>Constructor Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SearchService()</code></td>
</tr>
<tr>
<td>Constructor for SearchService class</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>java.util.Vector getRelatedObjects(java.lang.String contentId, float mainRelevance)</code></td>
</tr>
<tr>
<td>Searching in the ontology for related information-objects for a specific information-object.</td>
</tr>
<tr>
<td><code>java.util.HashMap getResults(java.util.HashMap resultsRequest)</code></td>
</tr>
<tr>
<td>Searching in the ontology for information-objects matching a specific query</td>
</tr>
<tr>
<td><code>void updateViews(java.lang.String contentId)</code></td>
</tr>
<tr>
<td>Updates the number of views for a specific information-object</td>
</tr>
</tbody>
</table>
## Package search.engine

### Class Summary

<table>
<thead>
<tr>
<th>Class</th>
<th>Title</th>
<th>Description</th>
<th>Copyright</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>ContentSearcher</td>
<td>Content Searcher</td>
<td>Searches the ontology for information-objects via relations through found keyword concepts or via matching headers.</td>
<td>Copyright (c) 2006</td>
<td>Everest b.v.</td>
</tr>
<tr>
<td>ContentUpdater</td>
<td>Content Updater</td>
<td>Updates the viewing information of an information-object</td>
<td>Copyright (c) 2006</td>
<td>Everest b.v.</td>
</tr>
<tr>
<td>QueryCooker</td>
<td>Query Cooker</td>
<td>Creates a QueryAdvanced object out of a QuerySimple object by converting the keyword string to a priority queue with found keyword concepts</td>
<td>Copyright (c) 2006</td>
<td>Everest b.v.</td>
</tr>
<tr>
<td>SearchEngine</td>
<td>Search Engine</td>
<td>Provides functionality for querying and updating the content repository for keyword- and information-object concepts</td>
<td>Copyright (c) 2006</td>
<td>Everest b.v.</td>
</tr>
</tbody>
</table>

## Package search.util

### Class Summary

<table>
<thead>
<tr>
<th>Class</th>
<th>Title</th>
<th>Description</th>
<th>Copyright</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>KeyWord</td>
<td>KeyWord</td>
<td>A found keyword concept with a basic transformation score</td>
<td>Copyright (c) 2006</td>
<td>Everest b.v.</td>
</tr>
<tr>
<td>Match</td>
<td>Match</td>
<td>Representing a match object in the result list</td>
<td>Copyright (c) 2006</td>
<td>Everest b.v.</td>
</tr>
<tr>
<td>Query</td>
<td>Query</td>
<td>Superclass for QuerySimple and QueryAdvanced</td>
<td>Copyright (c) 2006</td>
<td>Everest b.v.</td>
</tr>
<tr>
<td>QueryAdvanced</td>
<td>Advanced Query</td>
<td>For holding advanced queries, inherits from Query, where keyword concepts are hold in a priority queue with basic transformation scores</td>
<td>Copyright (c) 2006</td>
<td>Everest b.v.</td>
</tr>
<tr>
<td>QuerySimple</td>
<td>Simple Query</td>
<td>For holding simple queries, inherits from Query, where keywords are hold in comma separated string</td>
<td>Copyright (c) 2006</td>
<td>Everest b.v.</td>
</tr>
<tr>
<td>RelatedKeyWord</td>
<td>Related KeyWord</td>
<td>A concept related to a specific Keyword where also its relation is stored</td>
<td>Copyright (c) 2006</td>
<td>Everest b.v.</td>
</tr>
</tbody>
</table>
search.engine
Class SearchEngine

public class SearchEngine
extends java.lang.Object

Title: Search Engine
Description: Provides functionality for querying and updating the content repository for keyword- and information-object concepts
Copyright: Copyright (c) 2006
Company: Everest b.v.

Version: 1.0
Author: Ivo Krooswijk

Constructor Summary

<table>
<thead>
<tr>
<th>searchengine.SearchEngine()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constructor for SearchEngine class, which initializes the settings file</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>searchengine.SearchEngine</th>
<th>expandQuery(QuerySimple simpleQuery)</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.util.Vector</td>
<td>create an advanced query out of a simple query, by searching for keyword concepts</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>java.util.Vector</th>
<th>getRelatedObjects(java.lang.String contentId, float mainRelevance)</th>
</tr>
</thead>
<tbody>
<tr>
<td>search in the ontology for relating content-items for a specific information-object</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>java.util.HashMap</th>
<th>searchContent(QuerySimple simpleQuery, QueryAdvanced advancedQuery)</th>
</tr>
</thead>
<tbody>
<tr>
<td>retrieve all matching information-objects from the ontology by searching via matching keywords and via matching headers</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>updateViews(java.lang.String contentId)</th>
</tr>
</thead>
<tbody>
<tr>
<td>updates the number of views for a specific information-object</td>
<td></td>
</tr>
</tbody>
</table>
Class ContentSearcher

java.lang.Object

search.engine.ContentSearcher

public class ContentSearcher
extends java.lang.Object

Title: Content Searcher
Description: Searches the ontology for information-objects via relations through found keyword concepts or via matching headers.
Copyright: Copyright (c) 2006
Company: Everest b.v.

Version: 1.0
Author: Ivo Krooswijk

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>ContentSearcher()</td>
<td>closeConnection()</td>
</tr>
</tbody>
</table>

Constructor for OntologySearcher class

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void closeConnection()</td>
<td>Closes the connection to the main-memory content repository on the Sesame server</td>
</tr>
<tr>
<td>java.util.Vector getRelatedObjects(java.lang.String contentId, float mainRelevance)</td>
<td>Searching for related objects related to a specific information-object</td>
</tr>
<tr>
<td>void openConnection(java.lang.String serverSpec, java.lang.String loginName, java.lang.String passWord, java.lang.String repositoryId)</td>
<td>Opens the connection to the main-memory content repository on the Sesame server</td>
</tr>
<tr>
<td>java.util.HashMap searchContentByConcept(QueryAdvanced advancedQuery)</td>
<td>With the found keyword concepts in the AdvancedQuery the ontology is searched for related objects</td>
</tr>
<tr>
<td>java.util.HashMap searchContentByHeader(QuerySimple simpleQuery)</td>
<td>With the keyword string for the SimpleQuery the ontology is searched for matching headers</td>
</tr>
</tbody>
</table>
### search.engine
#### Class ContentUpdater

```java
public class ContentUpdater extends java.lang.Object
```

**Title:** Content Updater  
**Description:** Updates the viewing information of an information-object
**Copyright:** Copyright (c) 2006  
**Company:** Everest b.v.

**Version:** 1.0  
**Author:** Ivo Krooswijk

---

### Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ContentUpdater()</td>
<td>closeConnection()</td>
<td>Closes the connection to the main-memory content repository on the Sesame server</td>
</tr>
<tr>
<td></td>
<td>openConnection(java.lang.String serverSpec, java.lang.String loginName, java.lang.String passWord, java.lang.String repositoryId)</td>
<td>Opens the connection to the main-memory content repository on the Sesame server</td>
</tr>
<tr>
<td></td>
<td>updateViews(java.lang.String contentId)</td>
<td>Updates the number of views for specific information-objects</td>
</tr>
</tbody>
</table>
search.engine
Class QueryCooker

public class QueryCooker extends java.lang.Object

Title: Query Cooker
Description: Creates a QueryAdvanced object out of a QuerySimple object by converting the keyword string to a priority queue with found keyword concepts
Copyright: Copyright (c) 2006
Company: Everest b.v.

Version: 1.0
Author: Ivo Krooswijk

Constructor Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constructor</td>
<td>QueryCooker()</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method</td>
<td>getCookedQuery()</td>
</tr>
<tr>
<td>Method</td>
<td>getMaxSearchTime()</td>
</tr>
<tr>
<td>Method</td>
<td>getQueryParser()</td>
</tr>
<tr>
<td>Method</td>
<td>getQueryText()</td>
</tr>
<tr>
<td>Method</td>
<td>getQueryThreshold()</td>
</tr>
<tr>
<td>Method</td>
<td>getScorePrecision()</td>
</tr>
<tr>
<td>Method</td>
<td>getSearchTime()</td>
</tr>
<tr>
<td>Method</td>
<td>getServerSpec()</td>
</tr>
<tr>
<td>Method</td>
<td>getSortSpec()</td>
</tr>
<tr>
<td>Method</td>
<td>setCollections(java.lang.String[] collections)</td>
</tr>
<tr>
<td>Method</td>
<td>setFields(java.lang.String[] fields)</td>
</tr>
<tr>
<td>Method</td>
<td>setMaxResults(int maxResults)</td>
</tr>
<tr>
<td>Method</td>
<td>setMaxSearchTime(int maxSearchTime)</td>
</tr>
<tr>
<td>Method</td>
<td>setQueryParser(java.lang.String queryParser)</td>
</tr>
<tr>
<td>Method</td>
<td>setQueryText(java.lang.String queryText)</td>
</tr>
<tr>
<td>Method</td>
<td>setQueryThreshold(short queryThreshold)</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>setScorePrecision</code></td>
<td>Sets the precision of the score field</td>
</tr>
<tr>
<td><code>setSearchTime</code></td>
<td>Sets the time of the search in milliseconds</td>
</tr>
<tr>
<td><code>setServerSpec</code></td>
<td>Sets the search server specification</td>
</tr>
<tr>
<td><code>setSortSpec</code></td>
<td>Specifies the sort specification (default is &quot;Score desc&quot;)</td>
</tr>
</tbody>
</table>
java.lang.Object
   └─ search.util.KeyWord

Direct Known Subclasses:
   RelatedKeyWord

public class KeyWord
   extends java.lang.Object

- Title: KeyWord
- Description: A found keyword concept with a basic transformation score
- Copyright: Copyright (c) 2006
- Company: Everest b.v.
- Version: 1.0
- Author: Ivo Krooswijk

### Constructor Summary

- KeyWord()
  - Constructor for KeyWord class

### Method Summary

```
<table>
<thead>
<tr>
<th>java.util.HashMap</th>
<th>getHashMap()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Retrieves all fields from this class</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>int</th>
<th>getKeyWordId()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Retrieves the keyword id</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>java.lang.String</th>
<th>getKeyWordTitle()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Retrieves the keyword title</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>float</th>
<th>getWeight()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Retrieves the weight of the relation to another concept</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>setKeyWordId(int keyWordId)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sets the keyword id</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>setKeyWordTitle(java.lang.String keyWordTitle)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sets the keyword title</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void</th>
<th>setWeight(float weight)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sets the weight</td>
</tr>
</tbody>
</table>
```
search.util
Class Match

java.lang.Object
   search.util.Match

public class Match
extends java.lang.Object

Title: Match
Description: Representing a match object in the result list
Copyright: Copyright (c) 2006
Company: Everest b.v.

Version: 1.0
Author: Ivo Krooswijk

Constructor Summary
Match()
   Constructor for Match class

Method Summary
   int getContentId()
   Retrieves the content id
   java.lang.String getContentTitle()
   Retrieves the content header
   java.lang.String getContentType()
   Retrieves the content type
   java.lang.String getDistrictName()
   Retrieves the district name
   java.util.HashMap getHashMap()
   Retrieves all fields from this class
   java.lang.String getMatchWord()
   Retrieves the matching keyword
   int getNrViews()
   Retrieves the number of views
   java.lang.String getPubDate()
   Retrieves the publication date
   java.util.Vector getRelatedObjects()
   Retrieves the related information-objects
   float getRelevance()
   Retrieves the relevance
   void setContentId(int contentId)
   Sets the content id
   void setContentTitle(java.lang.String contentTitle)
   Sets the content header
   void setContentType(java.lang.String contentType)
   Sets the content type
   void setDistrictName(java.lang.String districtName)
   Sets the district name
   void setMatchWord(java.lang.String matchWord)
   Sets the matching keyword
   void setNrViews(int nrViews)
   Sets the number of views
   void setPubDate(java.lang.String pubDate)
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void setRelatedObjects(java.util.Vector relatedObjects)</code></td>
<td>Sets the related information-objects</td>
</tr>
<tr>
<td><code>void setRelevance(float relevance)</code></td>
<td>Sets the relevance</td>
</tr>
</tbody>
</table>
## Class Query

**Title:** Query  
**Description:** Superclass for QuerySimple and QueryAdvanced  
**Copyright:** Copyright (c) 2006  
**Company:** Everest b.v.  
**Version:** 1.0  
**Author:** Ivo Krooswijk

### Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Method Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Query()</code></td>
<td>Constructor for Query class</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getContentAttr()</code></td>
<td><code>java.lang.String</code></td>
<td>Retrieves the attribute</td>
</tr>
<tr>
<td><code>getContentType()</code></td>
<td><code>java.lang.String</code></td>
<td>Retrieves the content type</td>
</tr>
<tr>
<td><code>getDistrictName()</code></td>
<td><code>java.lang.String</code></td>
<td>Retrieves the district name</td>
</tr>
<tr>
<td><code>getNeighbourDistricts()</code></td>
<td><code>java.lang.String</code></td>
<td>Retrieves the flag for searching in neighboring districts</td>
</tr>
<tr>
<td><code>getPubPeriod()</code></td>
<td><code>int</code></td>
<td>Retrieves the publication period</td>
</tr>
<tr>
<td><code>getSearchType()</code></td>
<td><code>java.lang.String</code></td>
<td>Retrieves the type of search</td>
</tr>
<tr>
<td><code>setContentAttr(java.lang.String)</code></td>
<td><code>void</code></td>
<td>Sets the content attribute</td>
</tr>
<tr>
<td><code>setContentType(java.lang.String)</code></td>
<td><code>void</code></td>
<td>Sets the content type</td>
</tr>
<tr>
<td><code>setDistrictName(java.lang.String)</code></td>
<td><code>void</code></td>
<td>Sets the district name</td>
</tr>
<tr>
<td><code>setNeighbourDistricts(java.lang.String)</code></td>
<td><code>void</code></td>
<td>Sets the flag for searching in neighboring districts</td>
</tr>
<tr>
<td><code>setPubPeriod(int)</code></td>
<td><code>void</code></td>
<td>Sets the publication period</td>
</tr>
<tr>
<td><code>setSearchType(java.lang.String)</code></td>
<td><code>void</code></td>
<td>Sets the type of search</td>
</tr>
</tbody>
</table>
search.util
Class QueryAdvanced

java.lang.Object  
   search.util.Query  
      search.util.QueryAdvanced

public class QueryAdvanced
extends Query

Title: Advanced Query
Description: For holding advanced queries, inherits from Query, where keyword concepts are held in a priority queue
with basic transformation scores
Copyright: Copyright (c) 2006
Company: Everest b.v.

Version: 1.0
Author: Ivo Krooswijk

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryAdvanced()</td>
<td>Constructor for QueryAdvanced class</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.util.Vector getKeyWords()</td>
<td>Retrieves the keywords</td>
</tr>
<tr>
<td>int getSearchTime()</td>
<td>Retrieves the search time for finding related keywords</td>
</tr>
<tr>
<td>void setKeyWords(java.util.Vector keyWords)</td>
<td>Sets the keywords</td>
</tr>
<tr>
<td>void setSearchTime(int searchedTime)</td>
<td>Sets the search time needed to find related keywords</td>
</tr>
</tbody>
</table>

Methods inherited from class search.util.Query

getContentType, getDistrictName, getNeighbourDistricts, getPubPeriod, getSearchType, getContentAttr, setContentType, setDistrictName, setNeighbourDistricts, setPubPeriod, setSearchType
search.util
Class QuerySimple

java.lang.Object
   search.util.Query
      search.util.QuerySimple

public class QuerySimple
extends Query

Title: Simple Query
Description: For holding simple queries, inherits from Query, where keywords are hold in comma separated string
Copyright: Copyright (c) 2006
Company: Everest b.v.

Version:
   1.0
Author:
   Ivo Krooswijk

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>QuerySimple()</td>
</tr>
<tr>
<td>Constructor for QuerySimple class</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.lang.String getKeyWords()</td>
</tr>
<tr>
<td>Retrieves the keyword string</td>
</tr>
<tr>
<td>void setKeyWords(java.lang.String keyWords)</td>
</tr>
<tr>
<td>Sets the keyword string</td>
</tr>
</tbody>
</table>

Methods inherited from class search.util.Query

<table>
<thead>
<tr>
<th>Methods inherited from class search.util.Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>getContentAttr, getContentType, getDistrictName, getNeighbourDistricts, getPubPeriod, getSearchType, setContentAttr, setContentType, setDistrictName, setNeighbourDistricts, setPubPeriod, setSearchType</td>
</tr>
</tbody>
</table>
search.util
Class RelatedKeyWord

java.lang.Object
   search.util.KeyWord
      search.util.RelatedKeyWord

public class RelatedKeyWord
extends KeyWord

Title: Related KeyWord
Description: A concept related to a specific Keyword where also its relation is stored
Copyright: Copyright (c) 2006
Company: Everest b.v.

Version: 1.0
Author: Ivo Krooswijk

Constructor Summary
RelatedKeyWord()
Constructor for RelatedKeyWord class

Method Summary
java.util.HashMap getHashMap()
Retrieves all fields from this class
java.lang.String getRelationType()
Retrieves the relation type
void setRelationType(java.lang.String relationType)
Sets the relation type

Methods inherited from class search.util.KeyWord
getKeyWordId, getKeyWordTitle, getWeight, setKeyWordId, setKeyWordTitle, setWeight
Appendix D: Sequence diagrams

Use Case: Login

Figure D.1 Sequence Diagram Login

Use Case: Logout

Figure D.2 Sequence Diagram Logout
Use Case: Search

Figure D.3 Sequence Diagram Search
Use Case: Select content item

Figure D.4 Sequence Diagram Select
Appendix E: User interface elements

Figure E.1 User Interface Login

Figure E.2 User Interface Query
Appendix E: User interface elements

Figure E.3 User Interface Results

<table>
<thead>
<tr>
<th>Number</th>
<th>Title</th>
<th>Date</th>
<th>Views</th>
<th>Relevance</th>
<th>Type</th>
<th>Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Garagevergunning</td>
<td>2 Feb 2004</td>
<td>40</td>
<td>70%</td>
<td>Product</td>
<td>Stadsdeel: Osdorp</td>
<td>Vergunningen en ontheffingen voor ondernemers</td>
</tr>
<tr>
<td>2</td>
<td>Melding klein kansspel</td>
<td>3 Apr 2004</td>
<td>23</td>
<td>70%</td>
<td>Product</td>
<td>Stadsdeel: Osdorp</td>
<td>Vergunningen en ontheffingen voor ondernemers</td>
</tr>
<tr>
<td>3</td>
<td>Geluidsapparatuur in horeca-inrichtingen</td>
<td>25 May 2005</td>
<td>49</td>
<td>70%</td>
<td>Product</td>
<td>Stadsdeel: Osdorp</td>
<td>Vergunningen en ontheffingen voor ondernemers</td>
</tr>
<tr>
<td>4</td>
<td>Promotionele kansspelen</td>
<td>8 Feb 2004</td>
<td>35</td>
<td>70%</td>
<td>Product</td>
<td>Stadsdeel: Osdorp</td>
<td>Vergunningen en ontheffingen voor ondernemers</td>
</tr>
</tbody>
</table>

Figure E.4 User Interface Keyword Refinement

Er zijn 75 resultaten gevonden voor 1 treefwoord concept voor vergunning binnen 2710 ms.
De gevonden treefwoord concepten zijn: Vergunningen en ontheffingen vo

On Semantic Searching for Municipal Information Services
Appendix E: User interface elements

Figure E.5 User Interface Attribute Refinement

Figure E.6 User Interface Logout
Appendix E: User interface elements

Figure E.7 User Interface Initiate New Query

>> Wilt u verderen/vernaaijen op trefwoord?

(Dubbeklik op trefwoord om hiermee verder te zoeken)

Figure E.8 User Interface Initiate New Query by related keyword
Youngsterdam

Telefoon 020 663 3312
Internet http://www.youngsterdam.nl
Postcode 1091 DC
Straatnaam Van Musschenbroekstraat 40
Plaatsnaam Amsterdam
Landnaam Nederland
Plattegrond 🗺️

Bijzonderheden Youngsterdam maakt media voor en door jongeren in Amsterdam. Youngsterdam is voor jongeren die willen acteren, presenteren, filmen, monteren, websites willen maken of met radio-uitzendingen bezig willen zijn. Youngsterdam beschikt over een eigen digitale mediawerkplaats en biedt haar medewerkers media cursussen aan.

E-mail info@youncsterdam.nl

Figure E.9 User Interface Content

= Verwante informatie:

- maakt deel uit van Stichting Welzijn Oost, Ondersteunings- en sturingsbureau (100%)

Figure E.10 User Interface Relevant Information

= Andere bezoekers bekeken ook de volgende informatie:

- Stichting Welzijn Oost, Ondersteunings- en sturingsbureau
- Meidencentrum Silver Scissors Oost
- Buurtcentrum De Werf
- Peuterspeelzaal De Werf
- Jongeren Ontmoetingsplek Transvaal (JOT)

Figure E.11 User Interface Suggestions Other Users
Wilt u e-mail ontvangen over media als er nieuwe informatie komt:

- van het type instelling. Ja bevestig »
- binnen het stadsdeel oudorp. Ja bevestig »

> U heeft de volgende informatie reeds bekeken:

- Youngsterdam

Hierop zocht u...

1. vergunning
2. vergunningen en ontheffingen voor ondernemers