Deliverable 6.2

Cross-Lingual Semantic Search

Grant Agreement number:	**250467**
Project acronym:	**ATLAS**
Project type:	**Pilot B**

Deliverable D 6.2 **CLIR Tool** **13.07.2012**

Project coordinator name, title and organisation:
Anelia Belogay, CEO; Diman Karagiozov, CTO
Tetracom Interactive Solutions
Tel: +35924950444
Fax: +35924950443
E-mail:anelia@tetracom.com, diman@tetracom.com
Project website address: www.atlasproject.eu

Authors:
Ronald Winnemöller, Cristina Vertan, Diman Karagiozov

Project co-funded by the European Commission within the ICT Policy Support Programme

Dissemination level: **PUBLIC**

Document history

Revision	Date	Author	Description
0.1	28 June 2012	Ronald Winnemöller, University of Hamburg	initial version
0.2	04 July 2012	Cristina Vertan, University of Hamburg	feedback, minor adjustments
0.4	10 July 2012	Diman Karagiozov, Tetracom	document review
1.0	13 July 2012	Anelia Belogay, Tetracom	final version

Statement of originality

This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.

Table of contents

Introduction

This document describes the deliverable 6.2 "Cross-Lingual Semantic Search" of the Work Package 6 of the ATLAS project.

The ATLAS project itself as well as the individual work packages are described well by the ATLAS web homepage:

> “ATLAS (Applied Technology for Language-Aided CMS) is a project funded by the European Commission under the CIP ICT Policy Support Programme. Its main purpose is to facilitate the multilingual Web content development and management, in particular the authoring, versioning and maintenance of multilingual Web sites.”
> (Quotation from <http://www.atlasproject.eu/atlas/project/en>, accessed 15.06.2012)
>
> “The goal of this package 6 is to integrate into ATLAS two important multilingual technologies: machine translation and cross-lingual retrieval. Machine translation will ensure that ATLAS-content could be made available in several languages, or that third-party-content could be translated and integrated into a web-page. Cross-lingual retrieval will give users the opportunity to rapidly access documents inside ATLAS and the online content services independent of the language.”
> (Quotation from <http://www.atlasproject.eu/atlas/project/wp6>, accessed 15.06.2012)

Deliverable 6.2, is created by Ronald Winnemöller, member of the regional computer centre of the University of Hamburg. Ronald Winnemöller is both researcher in the field of semantically oriented natural language processing and IT practitioner at the computer centre. The work of Dr. Winemöller is part of the own contribution of the university of Hamburg.

Deliverable 6.2, a Cross-Lingual Semantic Search is needed within a language-aided CMS framework such as the one ATLAS produces because:

- One of the priorities of the target CMS is multi-linguality. This also involves the desire for cross-lingual functionalities such as placing a query in one language and retrieving documents in a specific (different) language, specifically retrieving automatically translated response documents.
- Faceted search can be greatly enhanced by targeting specific query parameters when these parameters can by analyzed semantically prior to the actual search. For example, when searching for a "person" typed parameter, the system should also infer to retrieve documents addressing "professors", "lawyers", "administrators" etc.
- Especially in combination with the categorization engine (WP3 - Automatic Text Categorization), this semantic process can lead to a robust yet precise two-phased retrieval process that guides users quickly to the desired information.

The main goal of Deliverable 6.2 is the creation of a retrieval engine that

- supports the notion of cross-linguality, i.e. that addresses language-related issues explicitly;
- supports semantic search applications in which users have some degree of freedom to formulate robust queries that are used by the engine to retrieve a set of documents which are further processed by a pre-built precise selection and construction phase in which the data can be further analyzed and enhanced;
- is easily embeddable into the framework built by the ATLAS consortium.

The remainder of this document:

1. clarifies the notions of "cross-linguality" and "semantics" applied to the retrieval engine;
2. gives a short introduction to some utilized concepts of the "semantic web", i.e. RDF, RDF-schema and OWL and to some of the more technical aspects of the underlying search engine employed by Deliverable 6.2;
3. describes the architecture and implementation details of the retrieval engine built for ATLAS;
4. provides information for installing and configuring the retrieval engine;
5. presents some results on a set of input test documents provided by the CMS developers;

Theoretic Fundamentals

In this section, we explain our fundamental assumptions regarding the notion of semantics and of cross-linguality, the building blocks of the ATLAS CLIR engine.

The Notion of "Semantics" in the ATLAS CLIR Environment

(The following section content is a quotation from: Ronald Winnemöller (2009). "Semantic Enterprise Search (but no Web 2.0)", In: Alexander Gelbukh (Hrsg.): 10th International Conference on Intelligent Text Processing and Computational Linguistics. Mexico City, Mexico: Instituto Politecnico Nacional, Reihe Polibits, S. 11-19.)

In order to clarify what we are talking about, we need to discuss what we mean when using the term "knowledge". Unfortunately, even a superficial definition of the nature of "knowledge" is far beyond the scope of a paper like this one but we would like to prevent some common issues:

1. We don't discuss open philosophical issues here (especially we will not argue about truth conditions, belief or justification). Instead, we see "knowledge" in very narrow technical terms as intentionally stored or transmitted information, usually – but not necessarily – contained in electronically represented documents within a specific organization;
2. Despite what we just claimed, "knowledge" does not equal "document content" but rather the information contained therein. This leads to the thought that even though "documents" usually are the unit of retrieval (cf. Mika *et al*[1]), it does not necessarily mean that documents are the basic unit of a storing and indexing process. For instance, there might be transient (e.g. streamed RSS) forms of data that can not be encompassed by the document metaphor. On the other hand, single documents may contain many differently motivated informational items;
3. In the same way as in the previous point, knowledge may be distributed across several source documents (or streams), linguistic or ontological information does not need to be symbolic but can be spread across "meaning aspects" (as discussed

[1] P. Mika, "Microsearch: An interface for semantic search," in Proceedings of the Workshop on Semantic Search (SemSearch 2008), Tenerife, Spain, June 2008, pp. 79–88

2004 by Winnemöller[2]) or across concepts (as in e.g. neuronal networks[3]);
4. Knowledge by our means is seldom static or monotonous, especially in the context of textual data streams.

Another important question is what we try to mend when “searching for knowledge”, i.e. what it means not to know (something):

> Smithson constructed a taxonomy of ignorance[4], based on the notions of error (bias, inaccuracy and confusion) and irrelevance (“mistaking some criterion as support for an argument when it has no bearing on its truth or falsehood”), based on his observations about the role of informational errors. His hypothesis is that not only knowledge is a socially constructed artifact, represented and communicated through symbolic frameworks, but that this is true for ignorance, too. He stated that in order to understand how and why people seek information, and how knowledge is linked with behavior, knowledge about both (perceived) relevance and irrelevance, objective knowledge and ignorance is fundamental.

We conclude from Smithsons findings, that in order to create an ultimately successful service for institutional information retrieval, at first the semantics of knowledge (and its absence) should be made explicit and clear.

So, where is that “semantics”? Where do we state ignorance and where should we create knowledge? When looking at web documents, for example, we see that they usually reflect an intentional act to represent and communicate knowledge. On the other hand, one common act to represent and communicate ignorance (as defined by Smithson) are search queries. Trivially, search engines use the first acts to solve issues of the latter ones - but there is one major problem: the documents are created prior to the queries; in terms of software development: they exist at compile time (when the search engine index is compiled). Queries, on the other hand, come into life at run time. Thinking about this, it does seem awkward that an answer (“Answer” being a socially constructed artifact just as well) should exist prior to its question!

Unfortunately, current search engine technology is mostly based on (syntactic) open web search, which in turn is based on common information retrieval techniques. These provide only basic tools, which are not very effective in a highly socialized and information-wise fine grained environment.

Other tools, like link structure exploitation, also don’t work too well here[5]. To be more specific: while intranet recall seems an issue of providing a highly customized technical solution, the precision of search results can - by definition - only be raised by tuning the search engines relevancy algorithms.

[2] R. Winnemöller, “Constructing text sense representations,” in ACL 2004: Second Workshop on Text Meaning and Interpretation, G. Hirst and S. Nirenburg, Eds. Barcelona, Spain: Association for Computational Linguistics, July 2004, pp. 17–24.

[3] G. Dorffner, Konnektionismus: Von neuronalen Netzwerken zu einer "natürlichen" KI, ser. Leitfäden der angewandten Informatik. Stuttgart: Teubner, 1991.

[4] M. Smithson, “Ignorance and science - dilemmas, perspectives, and prospects,” Science Communications, vol. 15, no. 2, pp. 133–156, 1993.

[5] G.-R. Xue, H.-J. Zeng, Z. Chen, W.-Y. Ma, H.-J. Zhang, and C.-J. Lu, “Implicit link analysis for small web search,” in SIGIR ’03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval. New York, NY, USA: ACM, 2003, pp. 56–63.

Semantic Web methods on the other hand are well suited to shape indexed knowledge according to the real informational situation and needs of institution members. Providing semantically rich machine readable information about resources and the principle of distributed extensibility are key aspects of the Semantic Web theory. Yet, one major drawback is that they still depend on a large amount of manual annotation work (sometimes it is simply assumed that the WWW will eventually contain appropriately annotated resources[6].

This indeed is one well-known problem of knowledge engineering, that annotating text basically is a huge amount of work with no apparent use to the annotator himself. Even in cases where people apparently want to annotate text (e.g. via the so-called "Web 2.0" technologies, i.e. folksonomies and such) they do it rather in a way that they gain reputation in their respective community but not in order to provide semantic annotations for automatic information retrieval[7].

Because of this issue, we think that annotation must come from automatic methods, if they are to be employed on large volumes of data.

Furthermore, we like the view of Chakrabarti that schema-free searches must be enabled, but schema knowledge should be honored by a query language (this enables freetext keyword searches but still rewards complex processing)[8]. While following his advice forbids using strict schematic query wizards or formalized query languages (as proposed by Cunningham *et al*[9] and Codina *et al*[10]), it reveals the necessity for applying NLP methods (and enabling manual editing) in order to discover semantic relationships within the data.

The concept of Cross-Linguality

One of the ATLAS basic concepts is the multilinguality of the content stored within the ATLAS applications. This has several implications:

1. Most documents that are imported into the applications database are only available in a subset of the targeted languages, commonly they will be single-language documents. This leads to the necessity of translating them into the other languages.
2. As stated in the previous section, not only the properties of knowledge have to be identified but also the ones of ignorance. Fairly obvious, this means that not only the documents are translated and stored but also provisions have to be made for queries in several different languages.

[6] R. Guha, R. McCool, and E. Miller, "Semantic search," in The Twelfth International World Wide Web Conference (WWW2003), Budapest, Hungary, May 2003.
[7] P. Mika, "Microsearch: An interface for semantic search," in Proceedings of the Workshop on Semantic Search (SemSearch 2008), Tenerife, Spain, June 2008, pp. 79–88.
[8] S. Chakrabarti, "Building blocks for semantic search engines: Ranking and compact indexing in entity-relation graphs," in IIIA-2006: International Workshop on Intelligent Information Access. Helsinki, Finland, July 2006.
[9] H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan, "Gate: A framework and graphical development environment for robust nlp tools and applications," in Proceedings of the 40th Annual Meeting of the ACL, 2002
[10] J. Codina, E. Pianta, S. Vrochidis, and S. Papadopoulos, "Integration of semantic, metadata and image search engines with a text search engine for patent retrieval," in ESWC-SemSearch 2008, Tenerife, Canary Islands, Spain, June 2008, pp. 14–28.

In the context of the CLIR engine, Point 2 is important: obviously, translating queries "on-the-fly" is computationally expensive, so an a-priori solution has to be identified. In this case, the situation is fortunate:

- All documents are translated during or before the indexing stage so that at a robust search-engine level, the retrieval mechanism can be language-agnostic, i.e. search for text occurrence without the need of query translation;
- During phases of fine-grained query handling, textual properties are tagged by their respective language so that only text in the relevant language(s) are eventually returned to the user.
- The CMS system can offer to switch off language recognition which means that documents in any configuration of languages can be returned to the user.

Cross-linguality is therefore achieved in a programmatically transparent fashion, i.e. by ignoring language at a broad level and following tagged data at finer levels. Since the index documents are processed by the ATLAS NLP pipelines beforehand, the text language is already known and tagging is a trivial task.

Basic Technical Principles

Apart from theoretical considerations, technological issues play a vital role in the ATLAS project and also in the CLIR subproject. In this section, therefore, we will explain some basic technical items.

Ontologies

Schemata and ontologies are the semantic building stones of the Nebula5 system. All statements, i.e. "knowledge" (and "ignorance", i.e. queries) are based on RDF, RDF-Schema, OWL and individual ontologies, such as FOAF[11] or the Nepomuk ontologies[12].

The search phase uses these ontologies to generalize class and property information and the RDF query phase can be parameterized in order to use ontological logics. All documents that are indexed should conform to one or more schemata – apart from the technical implementation, this also serves as important developer documentation repository for the individual properties of the indexed documents.

It is therefore important, to know the very fundamentals of these principles in order to understand how to configure the system in order to operate properly. In the Appendix, RDF, RDF-Schema and OWL will be introduced briefly.

Since these topics have been discussed very thoroughly by other parties and this report is not a genuine scientific paper, we will constrain ourselves to quoting the relevant web sources in the appendix.

Using Search Engines for RDF Storage

The search engine used by Nebula5 is Apache Lucene, realized by the Apache SOLR search server that either runs embedded by a Nebula5 component within ServiceMix or

[11] http://www.foaf-project.org/
[12] http://www.semanticdesktop.org/ontologies/

is started externally and just accessed by the internal component, allowing for simple distribution of load volume.
Lucene is, in a sense, typical for so-called "NoSQL" databases as it uses a simple "search document" metaphor: a "document" is just a set of field-value pairs of the datatype string. A typical document may look like this:

id:	2348727733
creation-date:	2012-06-27
text:	The search engine used by Nebula5 is Apache Lucene
author:	Ronald Winnemöller
author:	Winston Churchill

Querying for documents means querying the content of these fields by applying the Lucene search syntax (this syntax is included by quotation in the appendix). Result of a search is a set of documents. For example, the above displayed example document may be returned by queries like:

```
<text:Lucene>
<author:Churchill>
<creation-date:2012* AND text:engine>
etc...
```

Obviously, the semantical aspects are very limited in this approach. For this reason, the core component of Nebula5, the RDFIndex wraps a SOLR/Lucene service but before the actual storage action, incoming RDF messages are translated into search documents by preserving the semantic structure of the RDF models and enriching the search documents by the semantic content at the same time.

The process translates RDF predicates and literal objects into search fields and values and it also integrates some message metadata such as a unique identifier, security information (ACLs), creation-date information and more. Namespaces are stripped and included separately.

Example: The following document represents an incoming RDF model.

```
<document
        xmlns:dc="http://purl.org/dc/elements/1.1/"
        xmlns:foaf="http://xmlns.com/foaf/0.1/"
        xmlns:altas="http://www.atlas.org/ontology/1.0/"
        xml:lang="en">

        <dc:author>
                <foaf:firstname>Jimmy</foaf:firstname>
                <foaf:lastname>Bondi</foaf:lastname>
        </dc:author>
        <dc:title xml:lang="de"> ...german title ...</dc:title>
        <atlas:category>/general/law/bulgarian/civil law</atlas:category>

</document>
```

This RDF model will be translated into the following search document:

id	urn:n5:sample2348727733
creation-date	2012-06-27
namespace	DC http://purl.org/dc/elements/1.1/
namespace	FOAF http://xmlns.com/foaf/0.1/
namespace	ATLAS http://www.atlas.org/ontology/1.0/
_firstname	Jimmy
_lastname	Bondi
_title	...german title ...
_category	general , law , bulgarian , civil law
content	<document xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:foaf="http://xmlns.com/foaf/0.1/" xmlns:altas="http://www.atlas.org/ontology/1.0/" xml:lang="en"><dc:author><foaf:firstname>Jimmy</foaf:firstname><foaf:lastname>Bondi</foaf:lastname></dc:author><dc:title xml:lang="de">german title</dc:title><atlas:category>/general/law/bulgarian/civil law</atlas:category> </document>

At this point, one can search not only by querying the fulltext serialization of the RDF model (field “content”) but also specific properties of the model (“title”, “firstname”, etc...).

The RDFIndex component further includes a simple approach to incorporating RDF schemata: on initialization it reads RDF schema files from a predefined location and searches for subclass and subproperty definitions. These definitions are regarded at search time and queries might be rewritten on the fly before submitted to the underlying search engine. For example, a RDF schema may include the following definitions:

foaf:firstname	subPropertyOf	atlas:name
foaf:lastname	subPropertyOf	atlas:name

A query can now be formulated using these sub-properties:

```
name: Jimmy
```

will be expanded to:

```
name:Jimmy OR firstname:Jimmy OR lastname:Jimmy
```

The result of such a search is not returned as set of documents but rather a merged model which is derived by merging the models of the response documents. This enables a unified view on the search result – the result model constitutes a single microtheory that is consistent with both the query and the input documents, i.e. it supports the a.m. notion of knowledge and ignorance. Because the response model is also valid RDF, subsequent RDF-specific querying can be applied, e.g. running SparQL queries on the model.

Enterprise Service Bus (ESB) Technology

The ESB concept is best described as follows:

> "An Enterprise Service Bus (ESB) is a software architecture model used for designing and implementing the interaction and communication between mutually interacting software applications in Service Oriented Architecture.
>
> As a software architecture model for distributed computing it is a specialty variant of the more general client server software architecture model and promotes strictly asynchronous message oriented design for communication and interaction between applications. Its primary use is in Enterprise Application Integration of heterogeneous and complex landscapes.
>
> The word "bus" reflects the analogy to a computer hardware bus that is common architecture in computer design today. Like in a computer bus it is the basic concept to allow applications to be easily plugged in and out (switched on and off) of the network without impact on other components and without the need to restart the system or even stop running applications.
>
> It is an essential design concept of an ESB that every client directs all its requests through the ESB instead of passing it directly to a potential server. This indirection allows the ESB to monitor and log the traffic. The ESB can then intervene in message exchange and overwrite standard rules for service execution."[13]

The ESB therefore constitutes a container for connecting functional components, component assemblies and other services via an enterprise internal or external messaging infrastructure.

Many implementations include message transformation and reliable routing services as well as a multitude of connectors to external services and clients, known as consumers and producers. Examples of such connectors are Web services messaging standards or

[13] Quoation from <http://en.wikipedia.org/wiki/Enterprise_service_bus>, accessed 21.06.2012

Java Message Service (JMS realizations). The ESB technology is an upwards trending SOA concept : “originally defined by analysts at Gartner, ESB is increasingly seen as a core component in a service-oriented infrastructure.”[14]

The Java Business Integration (JBI) specification is commonly used as a standard for building ESBs: The JBI standard API serves as an interface that allows binding components and service engines to interact with the ESB. JBI internal messaging is based on so-called “normalized messages”, i.e. messages that contain a XML body, a set of metadata headers, optionally binary stream attachments and security subject information.
JBI specifies two types of components:

1. “Service Engines” provide business-logic in an ESB and can also provide data transformation, routing, scripting and application-specific functionality;
2. “Binding Components” are used to send and receive messages from an environment external to the ESB. They may implement particular protocols and transports and usually marshall and unmarshall JBI messages to protocol-specific data formats.

Architecture and Implementation

In this section we present the abstract architecture as well as the actual implementation of the ATLAS CLIR engine.

Architecture

The ATLAS application assembly consists of several binding component endpoints:

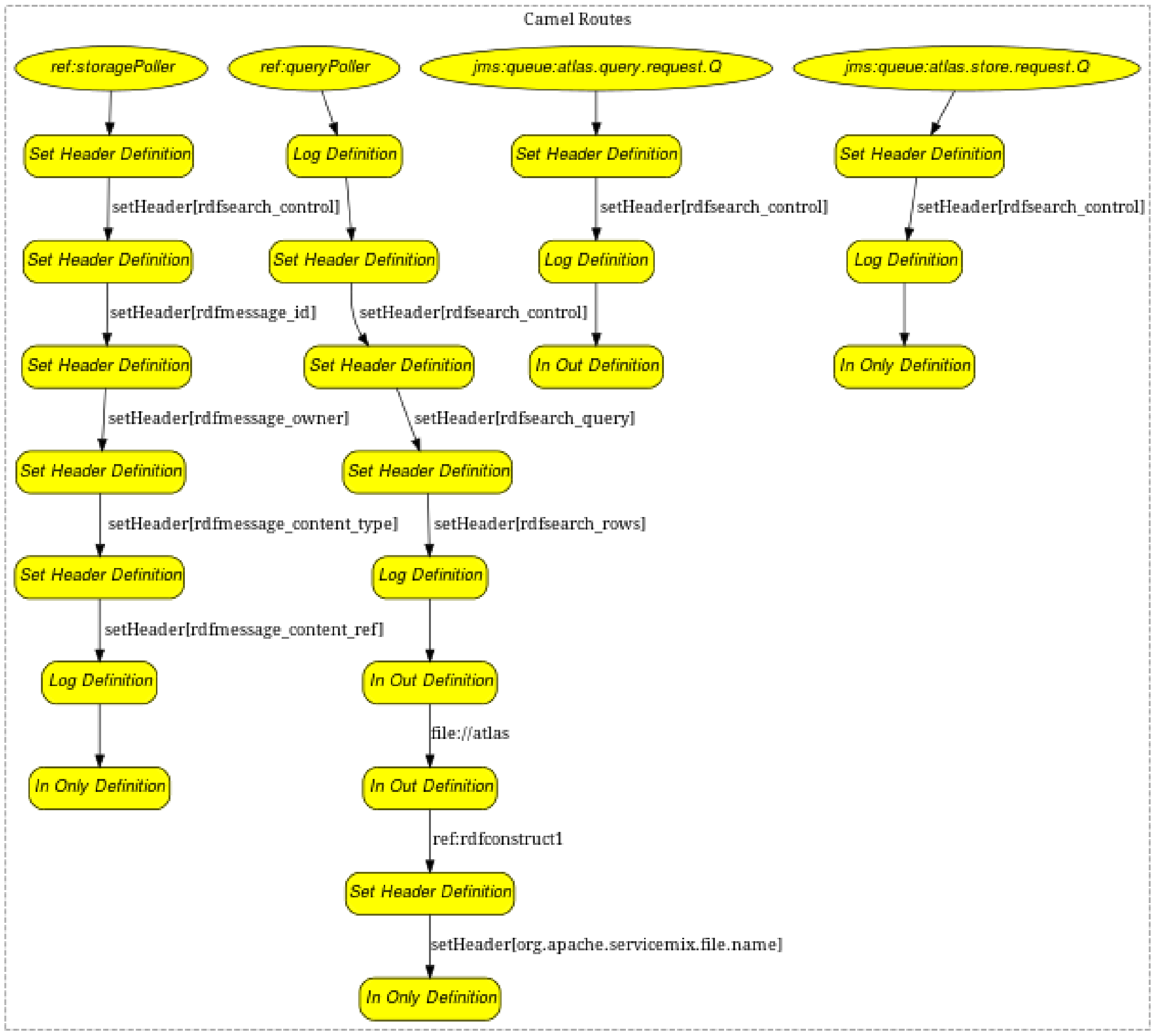
rdfselect1	Applies the default SparQL query to a search result (selects all)
rdfconstruct1	Demo application of introducing new statements, may be used to implement a (limited) kind of propositional logic.
storagePoller	File endpoint; the denoted location can be used to dump RDF files that will be imported into the RDFIndex
queryPoller	File endpoint; XML files that are stored here will be used to query the RDFIndex
resultSender	File endpoint; in case of querying the RDFIndex by using the queryPoller directory, the result of the search will be stored in the directory defined by the resultSender.
atlas.store.request.Q	JMS endpoint: clients connect to this queue in order to import documents
atlas.query.request.Q	JMS endpoint: clients connect to this queue in order to search for documents

These BCs are connected via a handful of Apache Camel routes:

[14] Cf. J. Jeffrey Hanson, JavaWorld.com, 12.12.2005, <http://www.javaworld.com/javaworld/jw-12-2005/jw-1212-esb.html>, accessed 22.06.2012

store-jms	Routing JMS based messages to the RDFIndex for storage
query-jms	Routing JMS based messages to the RDFIndex for search
query-route	Routing file based messages to the RDFIndex for storage
store-route	Routing file based messages to the RDFIndex for search

The individual routes are represented in the following picture.



Implementation

The implementation of the ATLAS CLIR engine is built in a layered fashion. The top/ application layer constitutes the actual Nebula5 part whereas the lower layers are part of the Apache ServiceMix distribution and stem from a multitude of open source projects of whom we will describe the most important ones in this section (mainly by excerpting their description).

Nebula5

Within Nebula5, we aim for Semantic Search (but not for Semantic Web Search) for the following reasons: Search is often limited to searching literal text or URI nodes and is implemented as specific function within a RDF framework (cf. Larq[15], Gnowsis[16] or Sesame[17]). We feel that is an unnecessary limitation because search functionality and RDF framework functionality should be tightly and efficiently integrated. It also should be possible to integrate schema information and use description logics and such when required.

We propose that fusing search engine and semantic web technology at the right level, i.e. enabling semantic annotations and intra-institution-wise distributed extensibility – while maintaining freetext search functionality – will create a certain amount of synergy which can raise the effectiveness of a semantic search approach in an institutional (enterprise) environment.

From our preliminary evaluation of some query logs of our institution we found that queries are strongly biased towards personal information (~28% of all queries) and organizational or structural queries, related to the institution (~36%), such as querying for departments, scripts, elearning courses, etc.. This enterprise-search related aspect of course will have a great impact on the kind of semantics we need to employ — especially named entity processing should be treated with high priority.

Furthermore, we think that the approach should be reasonably open to allow for other kinds of semantics, especially sub-symbolic ones like the TSR-approach proposed 2004 by Winnemöller[18].

Our approach to semantic enterprise search is based on a distributed modular architecture, named Nebula5.

One of the main differences between our architecture and others (typically similar to the one described by Lei *et al*[19]) is this: while the common approach to freetext semantic search (and also to semantic query expansion, such as explained by Umbrich and Blohm[20] or by Tran *et*

[15] Hewlett-Packard Development Company, "Larq - free text indexing for sparql", http://jena.sourceforge.net/ARQ/lucene-arq.html, accessed October 2008.

[16] L. Sauermann, G. A. Grimnes, M. Kiesel, C. Fluit, H. Maus, D. Heim, D. Nadeem, B. Horak, and A. Dengel, "Semantic Desktop 2.0: The Gnowsis Experience," in Proc. of the ISWC Conference, ser. Lecture Notes in Computer Science, vol. 4273/2006. Springer Berlin / Heidelberg, Nov 2006, pp. 887–900

[17] E. Minack, L. Sauermann, G. Grimnes, C. Fluit, and J. Broekstra, "The sesame lucene sail: Rdf queries with full-text search," NEPOMUK Consortium, Technical Report 2008-1, February 2008.

[18] R. Winnemöller, "Constructing text sense representations," in ACL 2004: Second Workshop on Text Meaning and Interpretation, G. Hirst and S. Nirenburg, Eds. Barcelona, Spain: Association for Computational Linguistics, July 2004, pp. 17–24.

[19] Y. Lei, V. S. Uren, and E. Motta, "Semsearch: A search engine for the semantic web," in Knowledge Acquisition, Modeling and Management (EKAW), S. Staab and V. SvÃ¡tek, Eds., Podebrady, Czech Republic, October 2006, pp. 238–245.

[20] J. Umbrich and S. Blohm, "Exploring the knowledge in semi structured data sets with rich queries," in Proceedings of the Workshop on Semantic Search (SemSearch 2008), Tenerife, Spain, June 2008, pp. 89–101.

al[21]) aims to translate natural language queries/ keyword queries into formal expressions – which are subsequently used to search a model repository for matching RDF statements, we instead use conventional freetext queries on our RDF documents.

The most important key heuristic is hidden in the postprocessing step of our architecture – by querying the index we encounter three cases:

1. Using conventional keywords only: documents containing these keywords will be discovered and ranked according to the Lucene tf*idf scheme. Additionally, RDF URI nodes can be discovered, too – exploiting the fact that most RDF URIs contain semantically relevant, human readable parts. For example, a keyword search for "bob homepage" will also reward indexed items containing "<foaf:homepage>" – especially when in conjunction to the literal fragment "bob". This can be quite useful, because many homepages in an institutional environment do not explicitly state that they are homepages!
2. Submitting a mixture of keywords and RDF URIs: queries like "foaf:homepage bob" will find "Bobs homepage" – but not "Jills homepage" with a reference to Bob! Because Lucene query analyzers eliminate non-alphanumeric characters, a domain-less URI is treated like a keyword; i.e. the query ":homepage bob" will not be restricted to the FOAF12 domain but rather work like an ordinary keyword query in the above explained way. In the special case of web documents containing microformats such as RDFa these will be implicitly honoured the same way.
3. Submitting RDF URIs only will exhibit documents with certain semantic properties: the query "foaf:homepage" will return all indexed items that contain homepages in the sense of the homepage element of the FOAF schema, plus the FOAF schema itself (as it also contains the fragment "foaf:homepage").

The query results (possibly filtered by a predefined document relevance threshold or by a first-N-documents-only heuristic) are merged into a single resulting RDF model that can be searched by means of templates, implemented structured RDF querying languages, e.g. SPARQL (cf. Schenk[22]), in order to provide end-user application functionality.

In this way, a query-centric RDF model is constructed dynamically on each search occasion that reflects the "ignorance-artifact" created by the user. Because schemata are discovered as well (and can be further tracked by using the PREFIX RDF document fields), we are not restricted to structured RDF queries only but can also apply description logics in order to further examine query results. For example, we can deduce subclassing etc. On the other hand, when it's just a portion of the textual content, it is being searched for, we can simply output the value of the "nie:content" predicate triple. In this way, we are able to defer complex processing until it is really needed.

(Quotation from: *Ronald Winnemöller (2009)*. "Semantic Enterprise Search (but no Web 2.0)", In: Alexander Gelbukh (Hrsg.): 10th International Conference on Intelligent Text Processing and Computational Linguistics. Mexico City, Mexico: Instituto Politecnico Nacional, Reihe Polibits, S. 11-19.)

Nebula5 is realized as set of RDF-centric components that are integrated into the ServiceMix service container infrastructure. The ATLAS CLIR engine itself is basically an independent

[21] T. Tran, P. Cimiano, S. Rudolph, and R. Studer1, The Semantic Web, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2007, vol. 4825/2008, ch. Ontology-Based Interpretation of Keywords for Semantic Search, pp. 523–536.

[22] S. Schenk, "A sparql semantics based on datalog," in KI 2007: Advances in Artificial Intelligence. Springer, 2007, pp. 160–174.

configuration applied to these components and also deployed to the ServiceMix container.

External Software

The Nebula5 system is based on a set of external and publicly available open source frameworks. Since understanding these components and their role within the framework is important to understand the overall architecture, the most fundamental ones will be described in this section (Excerpts from their web or project documentations are included in the appendix of this document).

Apache ServicMix

Apache ServiceMix is a multi-purpose service container that builds on an OSGi infrastructure and offers many technologies (such as java messaging, web services, web applications, java business integration etc.) bundled into a single, consistent provider application. ServiceMix is an open-source development, but private companies offer consulting and supporting services.

Maven

Maven is a net-based development project management tool that is tightly integrated into ServiceMix. Maven enables the service container instance to retrieve dependencies (jar files that are needed by some installed feature) from local disk locations or from public web repositories.

OSGi Component Building

ServiceMix contains many individual software components from third-party projects in form of so-called “bundles”. These bundles are like ordinary java jar files in many respects except for a handful of specific manifest entries that define external dependencies and how software may access the classes and services provided by the respective bundle. The main benefits of OSGi[23] are:

- for developers: OSGi reduces dependency and versioning complexity by providing a modular architecture for (distributed) component-based systems;
- in business terms: The OSGi modularity concept reduces implementation and operational costs and integrates components in a highly dynamic environment, successfully aiming at application development, maintenance and remote service management.

In practical terms, OSGi solves many issues related to jar library versioning, local service provisioning and library access management. All bundle and feature management functions (installing new modules, uninstalling modules, starting and stopping services, dependency management, etc.) within ServiceMix are handled by OSGi.

Apache Camel

Apache Camel is needed for providing connectivity to and from external clients and services as well as routing document-centric data from the external connectors to the RDF index database. The routing process may also include transformation processes such as templating the RDF search results and returning them as HTML or XML document fragments to the querying party.

ActiveMQ and JMS

[23] http://www.osgi.org/Technology/WhyOSGi, accessed 21.06.2012

ActiveMQ as an implementation of the Java Messaging Services (JMS) specification is the transport technology that connects ATLAS to the Nebula5/ServiceMix environment.

ATLAS assembly

The ATLAS assembly defines some metadata, by which it can identified within the ServiceMix container:

1. Bundle-Version = 1.0
2. Bundle-Name = Nebula5 :: ATLAS CrossLingual Information Retrieval
3. Bundle-SymbolicName = nebula5-atlas
4. Bundle-Description = CrossLingual Information Retrieval for the ATLAS EU project
5. Bundle-Vendor = ronald.winnemoeller@uni-hamburg.de

The assembly is implemented as Spring-XML file and located in the SERVICEMIX.ROOT/ deploy directory. Any changes to the assembly reflect immediately to the running processes in case the ServiceMix container is running.

The assembly should be regarded as "default" starting point – without intercepting the default behaviour, many endpoints and transformations can be added when desired.
The appendix contains example code that can be used as starting point for developing client-side JMS connectors.

Installation and Configuration

For the ATLAS project we created a graphical installer package that can be started either from a workbench gui or from the command line. The Nebula5 base framework also comes with such an installer. Both of them are very standard-like, easy to use and self-documented.

Requirements

There are a few minimum requirements to meet prior to the installation:

- Java 5 or Java 6 (not earlier, and Java 7 will also not work).
- At least 2 GB disk space (the actual size of the index depends on the volume of the input data. A rough estimate is the size of the input data multipied by 10)
- 2 GB RAM memory
- The JAVA_HOME environment variable must be set correctly.

Installation

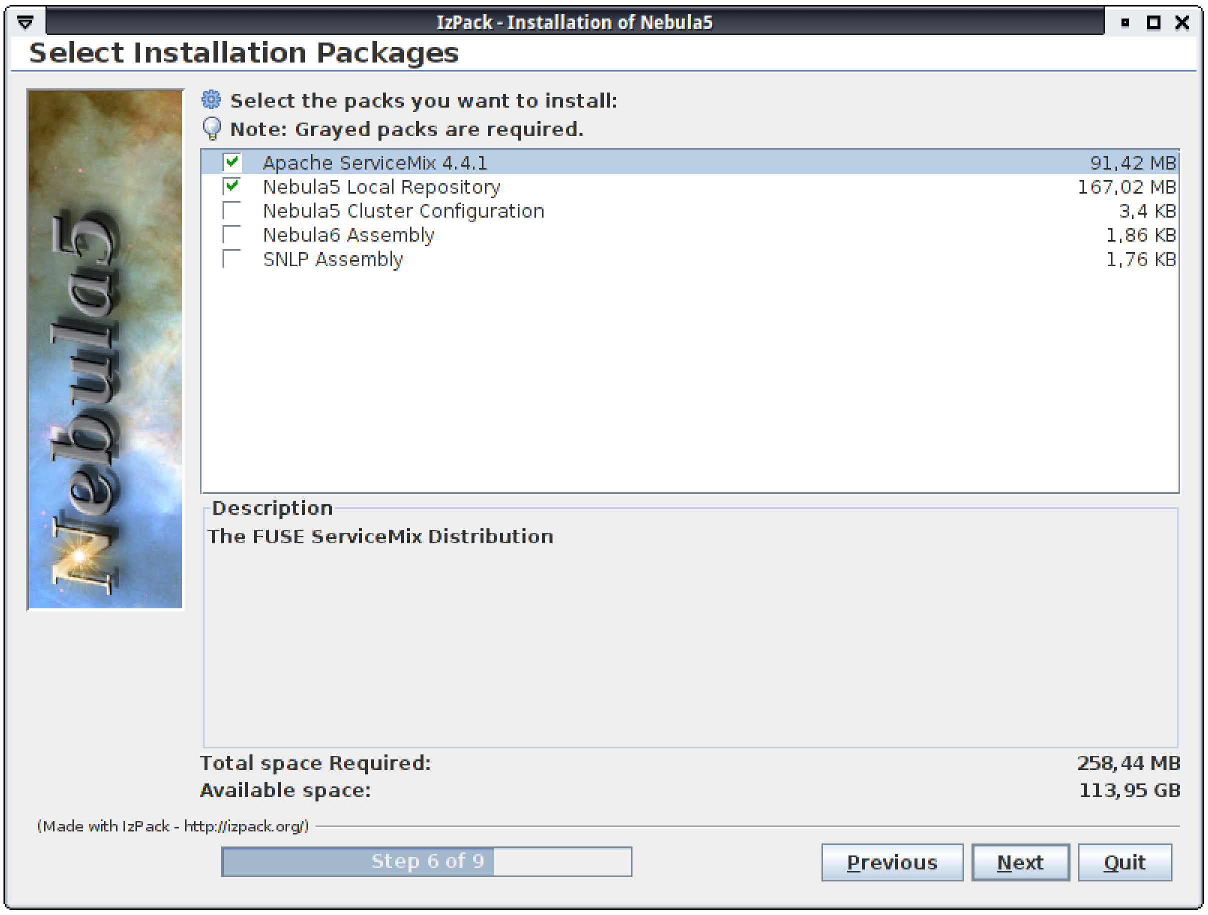
The Nebula5 framework can be installed using a convenient pre-built setup program (based on IzPack[24]). The ServiceMix ESB is included in the installer and can be installed optionally. The installer is run by double-clicking its icon or by command-line:

```
#  java -jar nebula5-0.3-installer.jar
```

The installation procedure will ask the usual questions (display README, accept copyright/ license, accept or edit installation directory - defaults to '/srv/servicemix') and the install about 2GB into the installation location.

[24] http://izpack.org/

For ATLAS, it is sufficient to install only the ServiceMix container and the Nebula5 local repository. All other components are not needed and will not work in the ATLAS environment.


IzPack - Installation of Nebula5
Nebula5
Previous
Next
Quit
Step 6 of 9
(Made with IzPack - http://izpack.org/)

Select Installation Packages

Select the packs you want to install:
Note: Grayed packs are required.

✓	Apache ServiceMix 4.4.1	91,42 MB
✓	Nebula5 Local Repository	167,02 MB
	Nebula5 Cluster Configuration	3,4 KB
	Nebula6 Assembly	1,86 KB
	SNLP Assembly	1,76 KB

Description
The FUSE ServiceMix Distribution

Total space Required: 258,44 MB
Available space: 113,95 GB

The ATLAS CLIR application is also available as installer and can be installed using the same procedure. Here, it is vital to use the *same installation location* as in the previous step.

The ServiceMix container should be started interactively first after running the base installer (i.e. before the ATLAS installer) in order to correct possible failures within the container shell. There might be some warnings and even an error concerning the Joda bundle or some trailing characters in some bundle configuration but this does not affect the ATLAS application performance. The self-configuration log can be viewed continuously by

```
log:tail          (interrupt with CTRL-C)
```

or in a "single-shot" fashion by

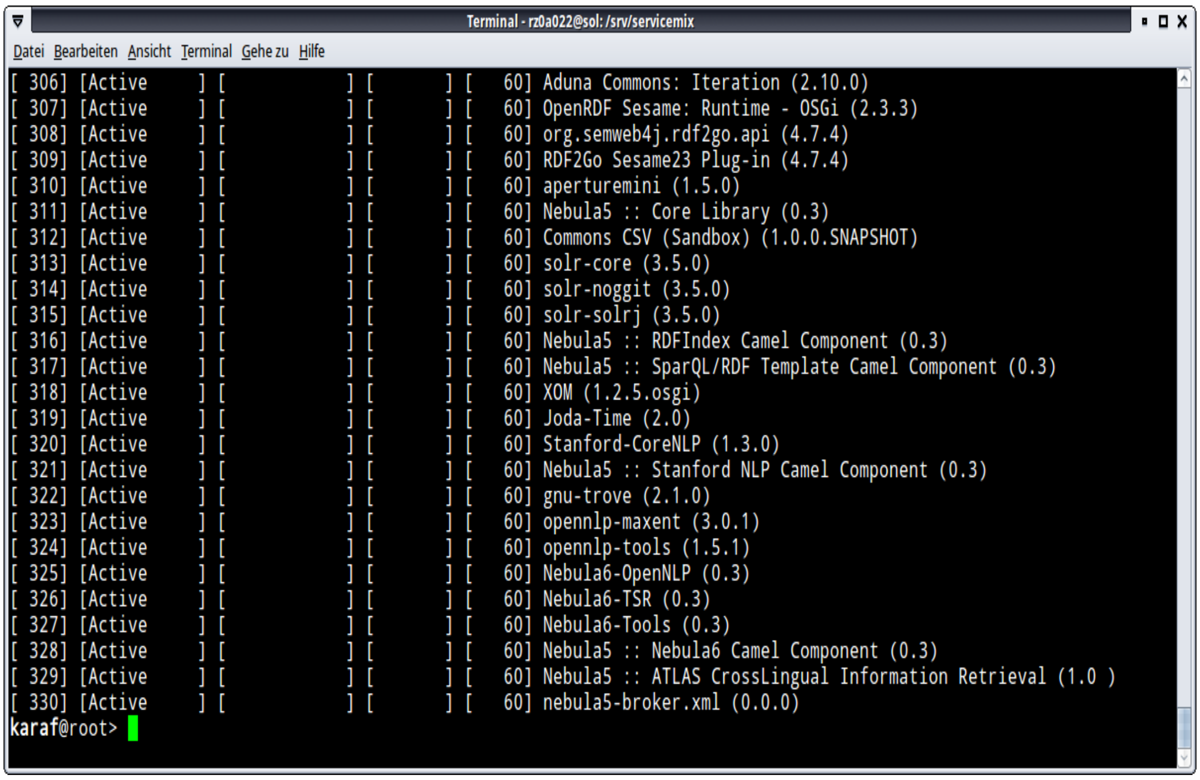
```
log:display
```

The installed bundles are listed by

```
osgi:list
```

The console also provides a help feature – just type TAB twice; The console can be shut down by issuing CTRL-D.

After installation of the ATLAS package, listing the installed bundles should include the Nebula5 broker and the ATLAS CLIR component and look like this:


Terminal - rz0a022@sol: /srv/servicemix
Datei Bearbeiten Ansicht Terminal Gehe zu Hilfe

```
[ 306] [Active     ] [            ] [       ] [   60] Aduna Commons: Iteration (2.10.0)
[ 307] [Active     ] [            ] [       ] [   60] OpenRDF Sesame: Runtime - OSGi (2.3.3)
[ 308] [Active     ] [            ] [       ] [   60] org.semweb4j.rdf2go.api (4.7.4)
[ 309] [Active     ] [            ] [       ] [   60] RDF2Go Sesame23 Plug-in (4.7.4)
[ 310] [Active     ] [            ] [       ] [   60] aperturemini (1.5.0)
[ 311] [Active     ] [            ] [       ] [   60] Nebula5 :: Core Library (0.3)
[ 312] [Active     ] [            ] [       ] [   60] Commons CSV (Sandbox) (1.0.0.SNAPSHOT)
[ 313] [Active     ] [            ] [       ] [   60] solr-core (3.5.0)
[ 314] [Active     ] [            ] [       ] [   60] solr-noggit (3.5.0)
[ 315] [Active     ] [            ] [       ] [   60] solr-solrj (3.5.0)
[ 316] [Active     ] [            ] [       ] [   60] Nebula5 :: RDFIndex Camel Component (0.3)
[ 317] [Active     ] [            ] [       ] [   60] Nebula5 :: SparQL/RDF Template Camel Component (0.3)
[ 318] [Active     ] [            ] [       ] [   60] XOM (1.2.5.osgi)
[ 319] [Active     ] [            ] [       ] [   60] Joda-Time (2.0)
[ 320] [Active     ] [            ] [       ] [   60] Stanford-CoreNLP (1.3.0)
[ 321] [Active     ] [            ] [       ] [   60] Nebula5 :: Stanford NLP Camel Component (0.3)
[ 322] [Active     ] [            ] [       ] [   60] gnu-trove (2.1.0)
[ 323] [Active     ] [            ] [       ] [   60] opennlp-maxent (3.0.1)
[ 324] [Active     ] [            ] [       ] [   60] opennlp-tools (1.5.1)
[ 325] [Active     ] [            ] [       ] [   60] Nebula6-OpenNLP (0.3)
[ 326] [Active     ] [            ] [       ] [   60] Nebula6-TSR (0.3)
[ 327] [Active     ] [            ] [       ] [   60] Nebula6-Tools (0.3)
[ 328] [Active     ] [            ] [       ] [   60] Nebula5 :: Nebula6 Camel Component (0.3)
[ 329] [Active     ] [            ] [       ] [   60] Nebula5 :: ATLAS CrossLingual Information Retrieval (1.0 )
[ 330] [Active     ] [            ] [       ] [   60] nebula5-broker.xml (0.0.0)
karaf@root>
```

Configuration

The ATLAS service assembly can be configured at compile time via the ServiceMix xmlbeans facilities but the default values should be sufficient for non-clustered environments.

Starting, Stopping, Inspecting, Access

In order to start the ServiceMix container in server mode, you can use the start command:

```
#        ${N5HOME}/bin/start
```

Similarily, use use the stop command in order to stop the ServiceMix container server:

```
#        ${N5HOME}/bin/stop
```

Logs will be written to ${N5HOME}/data/log/servicemix.log (Log rotation is active by default, therefore make sure you inspect the correct log file).

ServiceMix can also be started in an interactive console mode. This can be used to install further components or employ other maintenance functions. For example, it is possible to configure ssh shell access to the ServiceMix console to access the ServiceMix instance from a remote computer:

```
#        ${N5HOME}/bin/servicemix
```

Test results

Currently, tests have been performed only on monolingual data. The test corpora consist of 9’000 English and 9’000 Bulgarian short documents, found in the SETimes copus[25].

Further tests will be performed as soon as the machine translation engine is ready (due to month 30, end of August 2012). These tests will also include the full integration in ATLAS of CLIR, MT and Summarizer engines. The results will be made available at the following address:

https://docs.google.com/document/d/1oo4Qlok2MfbTuFFjvJvi0Nro50rZKb23JmAX2zXjTCI/edit

25

Francis M. Tyers and Murat Alperen (2010), "South-East European Times: A parallel corpus of the Balkan languages".
Jörg Tiedemann, 2009, News from OPUS - A Collection of Multilingual Parallel Corpora with Tools and Interfaces. In N. Nicolov and K. Bontcheva and G. Angelova and R. Mitkov (eds.) Recent Advances in Natural Language Processing (vol V), pages 237-248, John Benjamins, Amsterdam/ Philadelphia.

Appendix A: ATLAS RDF Schema, Examples

The appendix contains a mention of the RDF schemata that are used in ATLAS as well as an example data record.

Third-party Software Descriptions

In this section we provide detailed descriptions of the most important third-party software components by excerpting the original descriptions from the respective projects website, accompanying documentation, etc.

Apache ServiceMix

"Apache ServiceMix is a flexible, open-source integration container that unifies the features and functionality of Apache ActiveMQ, Camel, CXF, ODE, Karaf into a powerful runtime platform you can use to build your own integrations solutions. It provides a complete, enterprise ready ESB exclusively powered by OSGi.

It is being released under Apache License v2.

The main features are: reliable messaging with Apache ActiveMQ; messaging, routing and Enterprise Integration Patterns with Apache Camel; WS-* and RESTful web services with Apache CXF; loosely coupled integration between all the other components with Apache ServiceMix NMR including rich Event, Messaging and Audit API; complete WS-BPEL engine with Apache ODE; OSGi-based server runtime powered by Apache Karaf"
(Quotation from <http://servicemix.apache.org/>, accessed 21.06.2012)

"Apache ServiceMix is an enterprise-class open-source distributed enterprise service bus (ESB) and service-oriented architecture (SOA) toolkit [disambiguation needed]. It was built from the ground up on the semantics and APIs of the Java Business Integration (JBI) specification JSR 208 and released under the Apache License. ServiceMix 4 also fully supports OSGi. ServiceMix is lightweight and easily embeddable, has integrated Spring support and can be run at the edge of the network (inside a client or server), as a standalone ESB provider or as a service within another ESB. You can use ServiceMix in Java SE or a Java EE application server. ServiceMix uses ActiveMQ to provide remoting, clustering, reliability and distributed failover. The basic frameworks used by ServiceMix are Spring and XBean. ServiceMix is often used with Apache ActiveMQ, Apache Camel and Apache CXF in SOA Infrastructure projects.

Enterprise subscriptions for ServiceMix is available from independent vendors. Characteristics of an ESB include Federation, clustering and container provided failover; Hot deployment and lifecycle management of business objects; True vendor independence by license compliance with the JBI specification"
(Quotation from <http://en.wikipedia.org/wiki/ServiceMix>, accessed 21.06.2012)

Apache Camel

"Apache Camel is a rule-based routing and mediation engine which provides a Java object-based implementation of the Enterprise Integration Patterns using an API (or declarative Java Domain Specific Language) to configure routing and mediation rules. The domain-

specific language means that Apache Camel can support type-safe smart completion of routing rules in an integrated development environment using regular Java code without large amounts of XML configuration files, though XML configuration inside Spring is also supported."
(Quotation from <http://en.wikipedia.org/wiki/Apache_Camel>, accessed 21.06.2012)

"Camel empowers you to define routing and mediation rules in a variety of domain-specific languages, including a Java-based Fluent API, Spring or Blueprint XML Configuration files, and a Scala DSL. This means you get smart completion of routing rules in your IDE, whether in a Java, Scala or XML editor.

Apache Camel uses URIs to work directly with any kind of Transport or messaging model such as HTTP, ActiveMQ, JMS, JBI, SCA, MINA or CXF, as well as pluggable Components and Data Format options. Apache Camel is a small library with minimal dependencies for easy embedding in any Java application. Apache Camel lets you work with the same API regardless which kind of Transport is used - so learn the API once and you can interact with all the Components provided out-of-box.

Apache Camel provides support for Bean Binding and seamless integration with popular frameworks such as Spring, Blueprint and Guice. Camel also has extensive support for unit testing your routes."
(Quotation from <http://camel.apache.org/>, accessed 21.06.2012)

Open Service Gateway initiative framework (OSGi)

"The Open Services Gateway initiative framework is a module system and service platform for the Java programming language that implements a complete and dynamic component model, something that as of 2012 does not exist in standalone Java/VM environments. Applications or components (coming in the form of bundles for deployment) can be remotely installed, started, stopped, updated, and uninstalled without requiring a reboot; management of Java packages/classes is specified in great detail. Application life cycle management (start, stop, install, etc.) is done via APIs that allow for remote downloading of management policies. The service registry allows bundles to detect the addition of new services, or the removal of services, and adapt accordingly."
(Quotation from <http://en.wikipedia.org/wiki/Osgi>, accessed 21.06.2012)

Apache ActiveMQ and JMS Technology

"Apache ActiveMQ is an open source (Apache 2.0 licensed) message broker which fully implements the Java Message Service 1.1 (JMS). It provides "Enterprise Features" like clustering, multiple message stores, and ability to use any database as a JMS persistence provider besides VM, cache, and journal persistency.

Apart from Java, ActiveMQ can be also used from .NET, C/C++ or Delphi or from scripting languages like Perl, Python, PHP and Ruby via various "Cross Language Clients" together with connecting to many protocols and platforms. These include several standard wire-level protocols, plus their own protocol called OpenWire.

ActiveMQ is used in enterprise service bus implementations such as Apache ServiceMix, Apache Camel, and Mule."
(Quotation from <http://en.wikipedia.org/wiki/Apache_ActiveMQ>, accessed 21.06.2012)

Solr and Lucene

"Apache Lucene(TM) is a high-performance, full-featured text search engine library written entirely in Java. It is a technology suitable for nearly any application that requires full-text search, especially cross-platform. Apache Lucene is an open source project available for free download.**"**
(Quotation from <http://lucene.apache.org/core/>, accessed 21.06.2012)

"The Solr search server powers a wide range of applications such as Netflix, AOL, CNET, Zappos and many more.

Its major features include powerful full-text search, hit highlighting, faceted search, dynamic clustering, database integration, rich document (e.g., Word, PDF) handling, and geospatial search. Solr is highly scalable, providing distributed search and index replication, and it powers the search and navigation features of many of the world's largest internet sites.

Solr is written in Java and runs as a standalone full-text search server within a servlet container such as Tomcat. Solr uses the Lucene Java search library at its core for full-text indexing and search, and has REST-like HTTP/XML and JSON APIs that make it easy to use from virtually any programming language. Solr's powerful external configuration allows it to be tailored to almost any type of application without Java coding, and it has an extensive plugin architecture when more advanced customization is required.**"**
(Quotation from <http://lucene.apache.org/solr/>, accessed 21.06.2012)

Search document and Query syntax

"Search Documents are the unit of indexing and search. A Document is a set of fields. Each field has a name and a textual value. A field may be stored with the document, in which case it is returned with search hits on the document. Thus each document should typically contain one or more stored fields which uniquely identify it.**"**
(Quotation from
<http://lucene.apache.org/core/old_versioned_docs/versions/2_9_0/api/all/org/apache/lucene/document/Document.html>, accessed 21.06.2012)

The following text is taken verbatim from the lucene query syntax web page[26]:

Although Lucene provides the ability to create your own queries through its API, it also provides a rich query language through the Query Parser, a lexer which interprets a string into a Lucene Query.

A query is broken up into terms and operators. There are two types of terms: Single Terms and Phrases:

- A Single Term is a single word such as "test" or "hello".
- A Phrase is a group of words surrounded by double quotes such as "hello dolly".

Multiple terms can be combined together with Boolean operators to form a more complex query (see below).

Lucene supports fielded data. When performing a search you can either specify a field, or

[26] http://lucene.apache.org/core/old_versioned_docs/versions/3_0_0/queryparsersyntax.html, accessed 21.06.2012

use the default field. The field names and default field is implementation specific. You can search any field by typing the field name followed by a colon ":" and then the term you are looking for. As an example, let's assume a Lucene index contains two fields, title and text and text is the default field. If you want to find the document entitled "The Right Way" which contains the text "don't go this way", you can enter:

```
title:"The Right Way" AND text:go
```

or

```
title:"Do it right" AND right
```

Lucene supports modifying query terms to provide a wide range of searching options. Lucene supports single and multiple character wildcard searches within single terms (not within phrase queries):

- To perform a single character wildcard search use the "?" symbol.
- To perform a multiple character wildcard search use the "*" symbol.

The single character wildcard search looks for terms that match that with the single character replaced. For example, to search for "text" or "test" you can use the search:

```
te?t
```

Multiple character wildcard searches looks for 0 or more characters. For example, to search for test, tests or tester, you can use the search:

```
test*
```

You can also use the wildcard searches in the middle of a term.

```
te*t
```

Note: You cannot use a * or ? symbol as the first character of a search.

Lucene supports fuzzy searches based on the Levenshtein Distance, or Edit Distance algorithm. To do a fuzzy search use the tilde, "~", symbol at the end of a Single word Term. For example to search for a term similar in spelling to "roam" use the fuzzy search:

```
roam~
```

This search will find terms like foam and roams.

Starting with Lucene 1.9 an additional (optional) parameter can specify the required similarity. The value is between 0 and 1, with a value closer to 1 only terms with a higher similarity will be matched. For example:

```
roam~0.8
```

The default that is used if the parameter is not given is 0.5.

Lucene supports finding words are a within a specific distance away. To do a proximity search use the tilde, "~", symbol at the end of a Phrase. For example to search for a "apache" and "jakarta" within 10 words of each other in a document use the search:

```
"jakarta apache"~10
```

Range Queries allow one to match documents whose field(s) values are between the lower and upper bound specified by the Range Query. Range Queries can be inclusive or exclusive of the upper and lower bounds. Sorting is done lexicographically.

```
mod_date:[20020101 TO 20030101]
```

This will find documents whose mod_date fields have values between 20020101 and 20030101, inclusive. Note that Range Queries are not reserved for date fields. You could also use range queries with non-date fields:

```
title:{Aida TO Carmen}
```

This will find all documents whose titles are between Aida and Carmen, but not including Aida and Carmen. Inclusive range queries are denoted by square brackets. Exclusive range queries are denoted by curly brackets.

Lucene provides the relevance level of matching documents based on the terms found. To boost a term use the caret, "^", symbol with a boost factor (a number) at the end of the term you are searching. The higher the boost factor, the more relevant the term will be. Boosting allows you to control the relevance of a document by boosting its term. For example, if you are searching for

```
jakarta apache
```

and you want the term "jakarta" to be more relevant boost it using the ^ symbol along with the boost factor next to the term. You would type:

```
jakarta^4 apache
```

This will make documents with the term jakarta appear more relevant. You can also boost Phrase Terms as in the example:

```
"jakarta apache"^4 "Apache Lucene"
```

By default, the boost factor is 1. Although the boost factor must be positive, it can be less than 1 (e.g. 0.2)

Boolean operators allow terms to be combined through logic operators. Lucene supports AND, "+", OR, NOT and "-" as Boolean operators(Note: Boolean operators must be ALL CAPS). The OR operator is the default conjunction operator. This means that if there is no Boolean operator between two terms, the OR operator is used. The OR operator links two terms and finds a matching document if either of the terms exist in a document. This is equivalent to a union using sets. The symbol || can be used in place of the word OR. To search for documents that contain either "jakarta apache" or just "jakarta" use the query:

```
"jakarta apache" jakarta
```

or

```
"jakarta apache" OR jakarta
```

The AND operator matches documents where both terms exist anywhere in the text of a single document. This is equivalent to an intersection using sets. The symbol && can be used in place of the word AND. To search for documents that contain "jakarta apache" and "Apache Lucene" use the query:

```
"jakarta apache" AND "Apache Lucene"
```

The "+" or required operator requires that the term after the "+" symbol exist somewhere in a the field of a single document. To search for documents that must contain "jakarta" and may contain "lucene" use the query:

```
+jakarta lucene
```

The NOT operator excludes documents that contain the term after NOT. This is equivalent to a difference using sets. The symbol ! can be used in place of the word NOT. To search for documents that contain "jakarta apache" but not "Apache Lucene" use the query:

```
"jakarta apache" NOT "Apache Lucene"
```

Note: The NOT operator cannot be used with just one term. For example, the following search will return no results:

```
NOT "jakarta apache"
```

The "-" or prohibit operator excludes documents that contain the term after the "-" symbol. To search for documents that contain "jakarta apache" but not "Apache Lucene" use the query:

```
"jakarta apache" -"Apache Lucene"
```

Lucene supports using parentheses to group clauses to form sub queries. This can be very useful if you want to control the boolean logic for a query. To search for either "jakarta" or "apache" and "website" use the query:

```
(jakarta OR apache) AND website
```

This eliminates any confusion and makes sure you that website must exist and either term jakarta or apache may exist.

Lucene supports using parentheses to group multiple clauses to a single field. To search for a title that contains both the word "return" and the phrase "pink panther" use the query:

```
title:(+return +"pink panther")
```

Lucene supports escaping special characters that are part of the query syntax. The current list special characters are: + - && || ! ( ) { } [ ] ^ " ~ * ? : \

To escape these character use the \ before the character. For example to search for (1+1):2 use the query:

```
\(1\+1\)\:2
```

RDF Schemata

The means of semantic modelling within the Nebula5 framework is taken from the semantic web efforts, i.e. using UTF-8 as text encoding basis, URIs as identifiers, RDF for modelling knowledge and querying and RDF-Schema for conceptualizations.

RDF

"The Resource Description Framework (RDF) is a language for representing information about resources in the World Wide Web. It is particularly intended for representing metadata about Web resources, such as the title, author, and modification date of a Web page, copyright and licensing information about a Web document, or the availability schedule for some shared resource. However, by generalizing the concept of a "Web resource", RDF can also be used to represent information about things that can be *identified* on the Web, even when they cannot be directly *retrieved* on the Web. Examples include information about items available from on-line shopping facilities (e.g., information about specifications, prices, and availability), or the description of a Web user's preferences for information delivery."
(Quotation from <http://www.w3.org/TR/2004/REC-rdf-primer-20040210/#basicconcepts>, accessed 21.06.2012)

“RDF is intended for situations in which this information needs to be processed by applications, rather than being only displayed to people. RDF provides a common framework for expressing this information so it can be exchanged between applications without loss of meaning. Since it is a common framework, application designers can leverage the availability of common RDF parsers and processing tools. The ability to exchange information between different applications means that the information may be made available to applications other than those for which it was originally created.

RDF is based on the idea of identifying things using Web identifiers (called *Uniform Resource Identifiers*, or *URIs*), and describing resources in terms of simple properties and property values. This enables RDF to represent simple statements about resources as a *graph* of nodes and arcs representing the resources, and their properties and values.”
(Quotation from <http://www.w3.org/TR/2004/REC-rdf-primer-20040210/#basicconcepts>, accessed 21.06.2012)

“RDF models statements as nodes and arcs in a RDF's graph model. In this notation, a statement is represented by:

1. a node for the subject
2. a node for the object
3. an arc for the predicate, directed from the subject node to the object node.

In RDF, the English statement:

http://www.example.org/index.html has a creator whose value is John Smith

could be represented by an RDF statement having:

1. a subject http://www.example.org/index.html
2. a predicate http://purl.org/dc/elements/1.1/creator
3. and an object http://www.example.org/staffid/85740

RDF statements are similar to a number of other formats for recording information, such as:

- entries in a simple record or catalog listing describing the resource in a data processing system.
- rows in a simple relational database.
- simple assertions in formal logic

and information in these formats can be treated as RDF statements, allowing RDF to be used to integrate data from many sources."
(Quotation from <http://www.w3.org/TR/2004/REC-rdf-primer-20040210/#rdfmodel>, accessed 21.06.2012,)

RDF Schemata

"RDF itself provides no means for defining such application-specific classes and properties. Instead, such classes and properties are described as an RDF vocabulary, using extensions to RDF provided by the RDF Vocabulary Description Language 1.0, referred to here as RDF Schema.

RDF Schema does not provide a vocabulary of application-specific classes like exterms:Tent, ex2:Book, or ex3:Person, and properties like exterms:weightInKg, ex2:author or ex3:JobTitle.

Instead, it provides the facilities needed to describe such classes and properties, and to indicate which classes and properties are expected to be used together (for example, to say that the property ex3:jobTitle will be used in describing a ex3:Person). In other words, RDF Schema provides a type system for RDF.

The RDF Schema type system is similar in some respects to the type systems of object-oriented programming languages such as Java. For example, RDF Schema allows resources to be defined as instances of one or more classes. In addition, it allows classes to be organized in a hierarchical fashion; for example a class ex:Dog might be defined as a subclass of ex:Mammal which is a subclass of ex:Animal, meaning that any resource which is in class ex:Dog is also implicitly in class ex:Animal as well.

However, RDF classes and properties are in some respects very different from programming language types. RDF class and property descriptions do not create a straightjacket into which information must be forced, but instead provide additional information about the RDF resources they describe."
(Quotation from <http://www.w3.org/TR/2004/REC-rdf-primer-20040210/#rdfschema>, accessed 21.06.2012)

OWL

"The W3C Web Ontology Language (OWL) is a Semantic Web language designed to represent rich and complex knowledge about things, groups of things, and relations between things. OWL is a computational logic-based language such that knowledge expressed in OWL can be reasoned with by computer programs either to verify the consistency of that knowledge or to make implicit knowledge explicit. OWL documents, known as ontologies, can be published in the World Wide Web and may refer to or be referred from other OWL ontologies. OWL is part of the W3C's Semantic Web technology stack, which includes RDF, RDFS, SPARQL, etc."
(Quotation from <http://www.w3.org/2001/sw/wiki/OWL>, accessed 21.06.2012)

External Schemata

The Nebula5 framework depends semantically on the use of several publicly available base schemata in order to create corpora that can be read by many independent applications. In this section we reference the most important ones.

Dublin Core

“Early Dublin Core workshops popularized the idea of "core metadata" for simple and generic resource descriptions. The fifteen-element "Dublin Core" achieved wide dissemination as part of the Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) and has been ratified as IETF RFC 5013, ANSI/NISO Standard Z39.85-2007, and ISO Standard 15836:2009.

The consolidation of RDF motivated an effort to translate the mixed-vocabulary metadata style of the Dublin Core community into an RDF-compatible DCMI Abstract Model (2005). The DCMI Abstract Model was designed to bridge the modern paradigm of unbounded, linked data graphs with the more familiar paradigm of validatable metadata records like those used in OAI-PMH.”
(Quotation from <http://dublincore.org/metadata-basics/> accessed 22.06.2012)

Friend of a Friend (FOAF)

“FOAF is about your place in the Web, and the Web's place in our world. FOAF is a simple technology that makes it easier to share and use information about people and their activities (eg. photos, calendars, weblogs), to transfer information between Web sites, and to automatically extend, merge and re-use it online.

The Friend of a Friend (FOAF) project is creating a Web of machine-readable pages describing people, the links between them and the things they create and do.**”**
(Quotation from <http://www.foaf-project.org/about>, accessed 22.06.2012)

OSCAF/NEPOMUK Ontologies

The vision of the Social Semantic Desktop defines a user’s personal information environment as a source and end-point of the Semantic Web: Knowledge workers comprehensively express their information and data with respect to their own conceptualizations. Semantic Web languages and protocols are used to formalize these conceptualizations and for coordinating local and global information access. The Resource Description Framework RDF serves as a common data representation format. We identified several additional requirements for high-level knowledge representation on the social semantic desktop. With a particular focus on addressing certain limitations of RDF, we engineered a novel representational language akin to RDF and the Web Ontology Language OWL, plus a number of other high-level ontologies.

The specifications for the following published NEPOMUK Ontologies are given below.

NRL is NEPOMUK's Representational Language. Designed on top of RDF, it addresses certain limitations on the part of RDF/S. In particular it includes support for Named Graphs, which although being a widely-popular notion, have not been supported by any representational language so far. It is also based on a view concept for the tailoring of ontologies. This view concept turned out to be of additional value, as it also provides a mechanism to impose different semantics on the same syntactical structure.

NAO - The NEPOMUK Annotation Ontology is an ontology for annotation, providing vocabulary which is commonly required to annotate resources on the semantic desktop. NAO includes graph metadata vocabulary for describing, or annotating, existing named graphs.

NIE - The NEPOMUK Information Element set of ontologies provide vocabulary for describing information elements which are commonly present on the semantic desktop. The following documents collectively make up the complete specifications for the Nepomuk Information Element Ontology Framework:

- **NIE** (core) - NEPOMUK Information Element Core Ontology : The NEPOMUK Information Element Framework is an attempt to provide unified vocabulary for describing native resources available on the desktop.
- **NFO** - NEPOMUK File Ontology: NEPOMUK File Ontology (NFO) intends to provide vocabulary to express information extracted from various sources. They include files, pieces of software and remote hosts.
- **NCO** - NEPOMUK Contact Ontology: NEPOMUK Contact Ontology describes contact information, common in many places on the desktop. It evolved from the VCARD specification (RFC 2426) and has been inspired by the Vcard Ontology by Renato Ianella. The scope of NCO is much broader though.
- **NMO** - NEPOMUK Message Ontology: NEPOMUK Message Ontology extends the NEPOMUK Information Element framework into the domain of messages. Kinds of messages covered by NMO include Emails and instant messages.
- **NCAL** - NEPOMUK Calendar Ontology: The NEPOMUK Calendaring Ontology intends to provide vocabulary for describing calendaring data (events, tasks, journal entries) which is an important part of the body of information usually stored on a desktop. It is an adaptation of the ICALTZD ontology created by the W3C RDF Calendar Task Force of the Semantic Web Interest Group in the Semantic Web Activity.
- **NEXIF** - NEPOMUK EXIF Ontology: EXIF is a common standard of basic image metadata used in digital cameras and in image management software. Masahide Kanzaki created an ontology that allows the EXIF metadata to be expressed in RDF. NEXIF is a simple adaptation of the Kanzaki's ontology to fit it into the NEPOMUK Information Element Framework.
- **NID3** - NEPOMUK ID3 Ontology: ID3 is a common standard for audio metadata. It has become widespread with the profusion of MP3 files. The NEPOMUK ID3 ontology (NID3) makes it possible to express ID3 information in RDF, thus bringing it within reach of RDF-enabled application.
- **PIMO** - Personal Information Model ontology can be used to express Personal Information Models of individuals.
- **TMO** - Task Model Ontology can be used to describe personal tasks of individuals, as well known as to-do lists.

(Quotation from <http://www.semanticdesktop.org/ontologies/>, accessed 22.06.2012)

ATLAS Schemata

Apart from pre-defined public schemata, the ATLAS CLIR application also uses ATLAS specific definitions such as "summary", "keyword", "category" etc.:

```
<?xml version="1.0" encoding="UTF-8" ?>
```

```
<rdf:RDF xmlns:n5="http://n5.uni-hamburg.de/ontologies/2009/base#"
       xmlns:owl="http://www.w3.org/2002/07/owl#"
       xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"   xmlns:rdf="http://
www.w3.org/1999/02/22-rdf-syntax-ns#"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema#"  xmlns:dc="http://purl.org/
dc/elements/1.1/"
       xmlns:foaf="http://xmlns.com/foaf/0.1/"        xmlns:nfo="http://
www.semanticdesktop.org/ontologies/2007/03/22/nfo#"
       xmlns:nie="http://www.semanticdesktop.org/ontologies/2007/01/19/nie#"
       xmlns:tika="http://tika.apache.org/0.9/metadata/"
       xmlns:atlasdata="http://www.atlas-project.eu/data/"
       xmlns:atlas="http://www.atlas-project.eu/2011.11/"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
              http://www.w3.org/1999/02/22-rdf-syntax-ns#
              http://www.w3.org/2000/07/rdf.xsd
              http://purl.org/dc/elements/1.1/
              http://dublincore.org/schemas/xmls/qdc/2008/02/11/dc.xsd
              http://xmlns.com/foaf/0.1/
              http://friend2friend.net/docs/xml-datatypes/foaf/foaf.xsd
       ">

<!--
       ONTOLOGY :
              Dublin Core + FOAF + Nepomuk Ontologies
              +  additional  ATLAS  properties  as  defined  below  (extendable  on
demand)
-->

<rdf:Description rdf:about="http://www.atlas-project.eu/2011.11/Configuration">
       <rdfs:comment>
              Base objects for configuring the Nebula5 ATLAS JMS
              Marshalling
       </rdfs:comment>
       <rdfs:label>Configuration</rdfs:label>
</rdf:Description>


<rdf:Description rdf:about="http://www.atlas-project.eu/2011.11/Document">
       <rdfs:comment>
              ATLAS RDFXML Document
       </rdfs:comment>
       <rdfs:label>Document</rdfs:label>
</rdf:Description>


<rdf:Property rdf:about="http://www.atlas-project.eu/2011.11/summary">
       <rdfs:comment>
              Automatically derived summary of ATLAS source document
              fulltext
       </rdfs:comment>
       <rdfs:label>summary</rdfs:label>
       <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
       <rdfs:domain rdf:resource="http://xmlns.com/foaf/0.1/Document"/>
</rdf:Property>

<rdf:Property rdf:about="http://www.atlas-project.eu/2011.11/keywords">
       <rdfs:comment>
              Automatically derived keywords from ATLAS source document
              fulltext
       </rdfs:comment>
       <rdfs:label>summary</rdfs:label>
```

```
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
        <rdfs:domain rdf:resource="http://xmlns.com/foaf/0.1/Document"/>
</rdf:Property>


<rdf:Property rdf:about="http://www.atlas-project.eu/2011.11/categories">
        <rdfs:comment>
                Automatically derived categories from ATLAS source document
                fulltext
        </rdfs:comment>
        <rdfs:label>summary</rdfs:label>
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
        <rdfs:domain rdf:resource="http://xmlns.com/foaf/0.1/Document"/>
</rdf:Property>

</rdf:RDF>
```

Sample Data

For a quick overview of actual data, we provide a simple example:

```
<!-- original doc in english language, specific entries in other languages -->

<document
        xmlns:dc="http://purl.org/dc/elements/1.1/"
        xmlns:foaf="http://xmlns.com/foaf/0.1/"
        xmlns:altas="http://www.atlas.org/ontology/1.0/"
        id="12345"
        xml:lang="en">

        <dc:author>
                <foaf:firstname>James</foaf:firstname>
                <foaf:lastname>Bond</foaf:lastname>
        </dc:author>

        <dc:author>
                <foaf:firstname>Jimmy</foaf:firstname>
                <foaf:lastname>Bondi</foaf:lastname>
        </dc:author>

        <dc:title xml:lang="de"> ...german title ...</dc:title>
        ...
        <atlas:summary xml:lang="fr"> ... summary .... </atlas:summary>
        <atlas:summary xml:lang="cz"> ... summary .... </atlas:summary>
        <atlas:summary xml:lang="en"> ... summary .... </atlas:summary>

        <foaf:page> ... source location .... </foaf:page>

        <atlas:category>/general/law/bulgarian/civil law</atlas:category>

        <!-- the following elements are "future tech 1" -->

        <atlas:ne xml:lang="cz"> ... personname/locname/etc. .. </atlas:ne>
        <atlas:ne xml:lang="cz"> ... personname/locname/etc. .. </atlas:ne>
        <atlas:ne xml:lang="fr"> ... personname/locname/etc. .. </atlas:ne>
        ...

</document>
```

The default ATLAS Assembly XML

The assembly file defines the endpoints, routes and transformations that really create the ATLAS application:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:atlasdata="http://www.atlas-project.eu/data"
xmlns:atlas="http://www.atlas-project.eu/2011.11"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:util="http://www.springframework.org/schema/util"
xmlns:osgi="http://www.springframework.org/schema/osgi"
xmlns:camel="http://camel.apache.org/schema/spring"
xmlns:broker="http://activemq.apache.org/schema/core"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="
http://activemq.org/config/1.0
http://activemq.apache.org/schema/core/activemq-core-5.6.0.xsd
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util.xsd
http://www.springframework.org/schema/osgi
http://www.springframework.org/schema/osgi/spring-osgi.xsd
http://www.springframework.org/schema/osgi-compendium
http://www.springframework.org/schema/osgi-compendium/spring-osgi-
             compendium.xsd
http://camel.apache.org/schema/spring
http://camel.apache.org/schema/spring/camel-spring.xsd
http://www.springframework.org/schema/osgi
http://www.springframework.org/schema/osgi/spring-osgi.xsd
">
<!--
ATLAS CrossLingual Information Retrieval
(c) 2012 University of Hamburg ; Dr. Ronald Winnemoeller
-->
<manifest>
Bundle-Version = 1.0
Bundle-Name = Nebula5 :: ATLAS CrossLingual Information Retrieval
Bundle-SymbolicName = nebula5-atlas
Bundle-Description = CrossLingual Information Retrieval for the ATLAS EU project
Bundle-Vendor = ronald.winnemoeller@uni-hamburg.de
</manifest>
<camelContext id="atlas" xmlns="http://camel.apache.org/schema/spring">
<endpoint id="rdfselect1"
       uri="rdf://select?resultHeader=sparql&amp;scriptURI=
       file:${karaf.home}/data/atlas/select-all.sparql"/>
<endpoint id="rdfconstruct1"
       uri="rdf://construct?resultHeader=sparql&amp;scriptURI=
       file:${karaf.home}/data/atlas/construct.sparql"/>
<endpoint id="storagePoller"
       uri="file://${karaf.home}/ATLAS_STORAGE?include=.*.rdf|.*.xml"/>
<endpoint id="queryPoller"
       uri="file://${karaf.home}/ATLAS_QUERIES?include=.*.rdf|.*.xml"/>
<endpoint id="resultSender"
             uri="file://${karaf.home}/ATLAS_RESULTS??
                    fileName=result.rdf&amp;fileExist=Override"/>
<route id="store-jms">
<from uri="jms:queue:atlas.store.request.Q" />
       <setHeader headerName="rdfsearch_control">
```

```
             <constant>STORE</constant>
      </setHeader>
      <log message="STORE MESSAGE ID = ${in.header.rdfmessage_id} ."
             loggingLevel="INFO" logName="cool"/>
      <inOnly uri="rdfindex://atlas" />
</route>
<route id="query-jms">
<from uri="jms:queue:atlas.query.request.Q" />
      <setHeader headerName="rdfsearch_control">
             <constant>SEARCH</constant>
      </setHeader>

<log message="SEARCH QUERY = ${in.header.rdfsearch_query}"
             loggingLevel="INFO" logName="cool"/>

      <inOut uri="rdfindex://atlas" />
</route>

<route id="query-route">
<from uri="ref:queryPoller" />
      <log message="SEARCH QUERY = ${body}"
             loggingLevel="INFO" logName="cool"/>

      <setHeader headerName="rdfsearch_control">
             <constant>SEARCH</constant>
      </setHeader>
      <setHeader headerName="rdfsearch_query">
             <xpath resultType="java.lang.String">//search/@query</xpath>
      </setHeader>
      <setHeader headerName="rdfsearch_rows">
             <xpath resultType="java.lang.String">//search/@rows</xpath>
      </setHeader>

      <log message="SEARCH QUERY = ${in.header.rdfsearch_query}"
             loggingLevel="INFO" logName="cool"/>

      <inOut uri="rdfindex://atlas" />
      <inOut uri="ref:rdfconstruct1" />

      <setHeader headerName="org.apache.servicemix.file.name">
             <constant>result.rdf</constant>
      </setHeader>

      <inOnly uri="ref:resultSender" />
</route>
<route id="store-route">
<from uri="ref:storagePoller" />
      <setHeader headerName="rdfsearch_control">
             <constant>STORE</constant>
      </setHeader>
      <setHeader headerName="rdfmessage_id">
             <xpath resultType="java.lang.String">/*/*/*[1]</xpath>
      </setHeader>
      <setHeader headerName="rdfmessage_owner">
             <constant>{http://www.atlas-project.eu/data}admin</constant>
      </setHeader>
      <setHeader headerName="rdfmessage_content_type">
             <constant>application/rdf+xml</constant>
      </setHeader>
      <setHeader headerName="rdfmessage_content_ref">
             <xpath resultType="java.lang.String">/*/*/*[2]</xpath>
      </setHeader>
```

```
       <log message="STORE MESSAGE ID = ${in.header.rdfmessage_id} ."
              loggingLevel="INFO" logName="cool"/>
       <inOnly uri="rdfindex://atlas" />
</route>


</camelContext>
<!-- Common Utilities -->
<bean
class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer"/>
<bean                                                      id="connectionFactory"
class="org.apache.activemq.pool.PooledConnectionFactory">
       <constructor-arg value="tcp://localhost:61666" />
       <property name="maxConnections" value="8" />
</bean>
<bean class="org.apache.servicemix.common.osgi.EndpointExporter" />

</beans>
```

ATLAS Test client

The following code implements a Junit test that can be used as starting point for developing client modules:

```
package de.uhh.nebula5.atlas;

import java.io.BufferedReader;
import java.io.InputStreamReader;

import javax.jms.DeliveryMode;
import javax.jms.Destination;
import javax.jms.Message;
import javax.jms.MessageConsumer;
import javax.jms.Queue;
import javax.jms.QueueConnection;
import javax.jms.QueueSender;
import javax.jms.QueueSession;
import javax.jms.Session;
import javax.jms.TextMessage;

import junit.framework.TestCase;

import org.apache.activemq.ActiveMQConnectionFactory;

import de.uhh.nebula5.Nebula5;

/**
*
* The Class JMSClientTest.
*/
public class JMSClientTest extends TestCase {

/** The Constant STORE_REQUESTQ. */
public static final String STORE_REQUESTQ = "atlas.store.request.Q";

/** The Constant STORE_RESPONSEQ. */
public static final String STORE_RESPONSEQ = "atlas.store.response.Q";

/** The Constant QUERY_REQUESTQ. */
public static final String QUERY_REQUESTQ = "atlas.query.request.Q";

/** The Constant QUERY_RESPONSEQ. */
public static final String QUERY_RESPONSEQ = "atlas.query.response.Q";

/** The Constant OWNER. */
public static final String OWNER = "@owner";

/** The Constant ID. */
public static final String ID = "@id";

/**
* Test jms test.
*
* @throws Exception
* the exception
*/
public void testStoreMessage() throws Exception {
```

```
/* Set up JMS environment */
ActiveMQConnectionFactory connectionFactory = new ActiveMQConnectionFactory(
"tcp://localhost:61666");

QueueConnection queueConnection = connectionFactory
.createQueueConnection();
QueueSession queueSession = queueConnection.createQueueSession(false,
Session.AUTO_ACKNOWLEDGE);
Queue requestQueue = queueSession
.createQueue(JMSClientTest.STORE_REQUESTQ);
QueueSender queueSender = queueSession.createSender(requestQueue);
Destination tempResponseQueue = queueSession
.createQueue(JMSClientTest.STORE_RESPONSEQ);

/* Set up JMS requestMessage */
TextMessage msg = queueSession.createTextMessage();
msg.setStringProperty(JMSClientTest.ID, "34764376784884");
msg.setStringProperty(JMSClientTest.OWNER,
"{http://atlas.eu/user}admin");

msg.setStringProperty(Nebula5.RDFMESSAGE_ID, "34764376784884");
msg.setStringProperty(Nebula5.RDFMESSAGE_OWNER,
"{http://www.atlas-project.eu/data}admin");
msg.setStringProperty(Nebula5.RDFMESSAGE_CONTENTTYPE,
"application/rdf+xml");
msg.setStringProperty(Nebula5.RDFMESSAGE_CONTENTREF,
"file://sample.rdf");

BufferedReader in = new BufferedReader(new InputStreamReader(this
.getClass().getClassLoader()
.getResourceAsStream("sample-doc.rdf")));
StringBuffer sbuf = new StringBuffer();
String line = null;
while ((line = in.readLine()) != null) {
sbuf.append(line + "\n");
}
in.close();

String rdftext = sbuf.toString();
msg.setText(rdftext);
msg.setJMSReplyTo(tempResponseQueue);
msg.setJMSCorrelationID(String.valueOf(System.currentTimeMillis()));

/* Print some stuff about the request */
System.out.println("########################################");
System.out.println("#");
System.out.println("# About to send msg.getText(): " + msg.getText());
System.out.println("# msg.getJMSReplyTo(): " + msg.getJMSReplyTo());
System.out.println("# msg.getJMSCorrelationID(): "
+ msg.getJMSCorrelationID());
System.out.println("#");
System.out.println("########################################");

/* Start connection, send the message and thats it folks */
queueConnection.start();
queueSender.send(msg, DeliveryMode.NON_PERSISTENT, 1, 2000);

// Clean up after ourselves
queueSender.close();
queueSession.close();
queueConnection.close();
```

```
}

/**
* Test jms test.
*
* @throws Exception
* the exception
*/
public void testQueryMessage() throws Exception {

/* Set up JMS environment */
ActiveMQConnectionFactory connectionFactory = new ActiveMQConnectionFactory(
"tcp://localhost:61666");

QueueConnection queueConnection = connectionFactory
.createQueueConnection();
QueueSession queueSession = queueConnection.createQueueSession(false,
Session.AUTO_ACKNOWLEDGE);
Queue requestQueue = queueSession
.createQueue(JMSClientTest.QUERY_REQUESTQ);
QueueSender queueSender = queueSession.createSender(requestQueue);
Destination tempResponseQueue = queueSession
.createQueue(JMSClientTest.QUERY_RESPONSEQ);
MessageConsumer responseMessageConsumer = queueSession
.createConsumer(tempResponseQueue);

/* Set up JMS requestMessage */
TextMessage msg = queueSession.createTextMessage();
msg.setStringProperty(JMSClientTest.ID, "34764376784884");
msg.setStringProperty(JMSClientTest.OWNER,
"{http://atlas.eu/user}admin");

msg.setStringProperty(Nebula5.RDFSEARCH_CONTROL,
Nebula5.RDFSEARCH_CONTROL_SEARCH);
msg.setStringProperty(Nebula5.RDFSEARCH_QUERY, "@content:Business");
msg.setStringProperty(Nebula5.RDFSEARCH_ROWS, Nebula5.RDFSEARCH_ROWS_DEFAULT);

msg.setText("<search query=\"@content:Business\" rows=\"20\" />");
msg.setJMSReplyTo(tempResponseQueue);
msg.setJMSCorrelationID(String.valueOf(System.currentTimeMillis()));

/* Print some stuff about the request */
System.out.println("########################################");
System.out.println("#");
System.out.println("# About to send msg.getText(): " + msg.getText());
System.out.println("# msg.getJMSReplyTo(): " + msg.getJMSReplyTo());
System.out.println("# msg.getJMSCorrelationID(): "
+ msg.getJMSCorrelationID());
System.out.println("#");
System.out.println("########################################");

/* Start connection, send the message and thats it folks */
queueConnection.start();
queueSender.send(msg, DeliveryMode.NON_PERSISTENT, 1, 2000);

Message responseMessage = responseMessageConsumer.receive();

/* Print some stuff about the response */
System.out.println("########################################");
System.out.println("#");
System.out.println("# responseMessage.getClass(): "
+ responseMessage.getClass());
```

```
System.out.println("# responseMessage.getJMSDestination(): "
+ responseMessage.getJMSDestination());
System.out.println("# responseMessage.getJMSCorrelationID(): "
+ responseMessage.getJMSCorrelationID());

/* Get the TextMessage from the response and print it out */
TextMessage textMessage = (TextMessage) responseMessage;

System.out.println("# textMessage.getText(): " +
textMessage.getText());
// System.out.println("# Message: " + responseMessage.toString());
System.out.println("# JBI Done: "
+ responseMessage.getStringProperty("JBIDone"));
System.out.println("#");
System.out.println("#######################################");

// Clean up after ourselves
queueSender.close();
queueSession.close();
queueConnection.close();
}

}
```