Grant Agreement Number: 257528

KHRESMOI

www.khresmoi.eu

D5.1 Report on data source integration

Deliverable number	*D5.1*
Dissemination level	*Public*
Delivery date	*26 August 2011*
Status	*Final*
Author(s)	*Konstantin Pentchev, Vassil Momtchev*

This project is supported by the European Commission under the Information and Communication Technologies (ICT) Theme of the 7th Framework Programme for Research and Technological Development.

Table of Contents

Index of Tables

1 Executive Summary

This deliverable reports on the progress of the tasks T5.1 "Biomedical knowledge server" and T5.2 "Knowledge base creation". The document specifies the developments in the context of WP5 – Biomedical Knowledge Engine Server or simply Knowledge Engine (KE), which is considered a key component of the overall Khresmoi architecture. The KE is responsible for the ontology management process that includes semantic data integration of various use case datasets, their consistent representation in a formal logical model and, finally, the efficient retrieval of knowledge in the KE.

Our work aims to deliver a generic service, capable of addressing the different structure and semantic heterogeneity levels of biomedical knowledge by combining a classic data integration infrastructure with ontologies and efficient reasoning algorithms. The WP5 infrastructure enables the consolidation or federation of knowledge, depending on data source requirements. It persistently stores various types of information such as ontologies, instance data, unstructured text, semantic annotations (i.e. links between ontology instances and text annotations), image meta-data, multilingual semantic networks, user feedback and other system runtime data. The infrastructure needs to handle very large amounts of data that must be processed by repeatable update routines.

This deliverable is complemented by an early version of D5.2 "Large Scale Biomedical Knowledge Server", which integrates a reduced number of datasets, identified by the Khresmoi use cases. All the work in D5.2 is done with the help of new components, developed and presented later in this deliverable.

2 Introduction

In the "Biomedical Knowledge Engine" work package we are in the process of developing a generic infrastructure, capable of processing extremely large, rapidly growing and potentially inconsistent or incomplete information. The current document presents general information integration concepts, the Knowledge Engine (KE) architecture and software infrastructure that will be used for the successful implementation of D5.2 "Large Scale Biomedical Knowledge Server".

Data integration is the process of ensuring interoperability between different data sources by providing a unified view of the information contained in the data sources. A key objective for this process is to build a consistent and homogenous global data model that unifies all sources. Lenzerini in [1] gives a formal definition of data integration (I) as a triple consisting of $I = (G, S, M)$, where G is the global schema (unified view towards the information), S is the source schema and M the mapping between G and S. The mapping between different data sources has to overcome four types of incompatibilities or heterogeneity levels of the information described by Sheth [2]:

- The *system level* reflects scenarios where data is accessed via an intermediate storage interface (i.e. a different file system or a physical separation between system and data)
- The *syntactic or format level* is concerned with the problems of cross-platform data encoding like ASCII, UTF-8, UTF-16 and etc. These compatibility issues are largely addressed by the XML 1.0 format specification and further refined by XML 1.1, so they are beyond the scope of our work;
- The *structure (schema) level* refers to heterogeneity in the entity modelling, their attributes, type hierarchy, cardinalities and the behaviour with respect to the schema constrains e.g. data integrity rules;

- The *semantic (meaning) level* refers to incompatibility in modelling generalisation/specialisations (drug versus organic chemical), composition/aggregation and value level interpretations like homonyms or synonyms;

Figure 1 shows the nested nature of the heterogeneity levels. Once a specific issue is unresolved, it will be propagated in a cascading way to all upper layers.

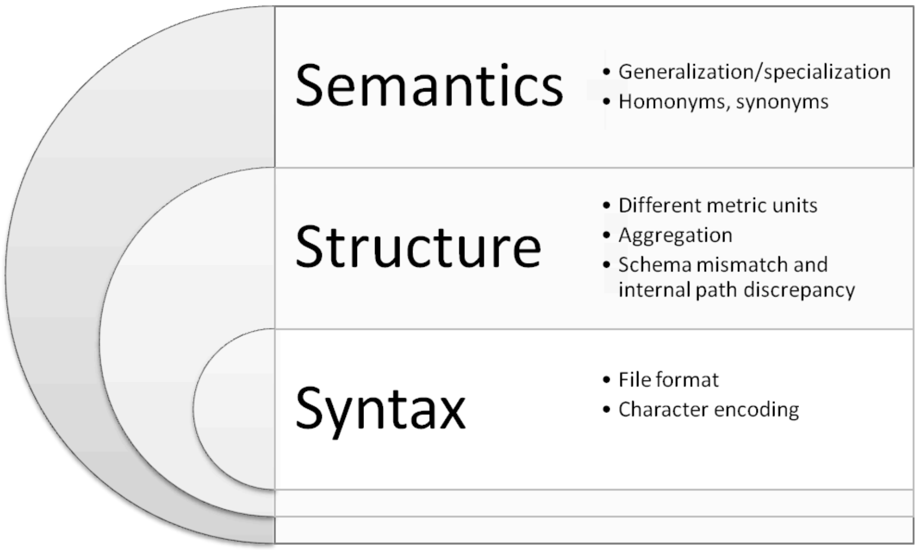


Figure 1: Syntax, structure and semantic heterogeneity levels.

The work in WP5 "Biomedical Knowledge Management Engine" is focussed on integrating information from heterogeneous sources and overcoming various types of structure and semantic level incompatibilities by developing extract and associate rules. After the semantic heterogeneities are addressed and the information is represented in a consistent knowledge base, further analytics may be applied such as text mining, reasoning, semantic similarity prediction, etc. Thus, the WP5 software infrastructure is called *knowledge management engine* since it will address the challenges in data integration, semantic incompatibility and the knowledge analysis. Later, this document provides an overview of the KE foundation and the choice of the underlying data model, suitable for semantic data integration and advanced knowledge analysis.

The RDF data model is not constrained to a particular application domain and does not define a priori the semantics of any application domain, [3]. Its abstract knowledge representation formalism fits into a wide range of use-cases and application scenarios from meta-data publishing on the web, data integration, implementation of complex formal logical models (i.e. ontologies), etc. Such cross use case compatibility is achieved by a number of W3C standards, specifications and recommendations that cover different aspects of the ontology language layers. By the term ontology language layer, we mean the set of theoretical modelling primitives that every ontology language can be decomposed into:

- The data model determines the mathematical data-structure (e.g. directed acyclic graph) that describes the ontology, e.g. the RDF data model, [3];
- Epistemology defines the language at the conceptual level or specifies data model patterns, used to represent notions like concepts, classes, relations, properties, attributes, roles, etc.;
- The vocabulary determines what sort of symbols are valid for composing expressions in the ontology language by giving naming conventions for various primitives, defined in the data model and the epistemology levels (for instance the vocabularies of Dublin Core (DC) , Simple Knowledge Organization System (SKOS), Web Ontology Language (OWL));

- The syntax determines the structure of the valid expressions within the language and its serialization formats (RDF/XML, Turtle, RDF/JSON, JSON Triples, etc.);
- Semantics is the top layer that determines the meaning of the expressions made in the ontology language; it is often defined in terms of pairs consisting of a mathematical model and a function, which define the correspondence between the expressions of the language and the elements of the model; any sort of inference or induction of implicit triples is performed on the basis of the semantics, e.g. OWL2-RL, SKOS, etc.;

To summarize, the RDF data model is highly abstract and supports the layering of several ontology primitives, which makes it an excellent candidate for the KE internal representation format. Hence, a final contributing reason for its selection over the classical database technology is the fact that the latest SPARQL 1.1 specification [4] fully covers the relation algebra [5].

3 Semantic Data Integration with RDF

This chapter presents an analysis of the different levels of performing data integration. Ziegler and Dittrich [6] define multiple integration levels depending on its specificity. They start from 1) Manual integration – no real integration is done since the interpretation is performed by the end-user; 2) Common user interface – data from relevant sources are displayed in a single view in the application; 3) Integration by applications or middleware –integration is done on the concrete application level where the developers are relieved only from implementing common integration functionality; 4) Uniform Data Access –information integration is realized by *virtual data* or data abstracted from its physical structure in runtime; 5) Common Data Storage – is the physical reorganisation or replication of the existing data to a new place and possibly new global schema. It is a general rule that the integration becomes more efficient when it is moved closer to the physical storage. Thus, when we need to operate with very large amounts of information like in the context of the Khresmoi project, our choice for efficient data integration is practically limited to the data consolidation (or warehousing) and federation approaches. In the next paragraphs we summarize the different trades-offs in the two approaches and their impact on the knowledge engine design.

Data warehousing is the process of centralizing the information into a common physical storage model. It requires the reorganization and consolidation of all data into a global schema, and may either fully replace the old databases or replicate the information on a regular basis. Either way, data consolidation requires the design and execution of extract transform and load scripts that need to resolve the structure and semantic heterogeneities between the source and the global schema during data loading.

Data federation utilizes a different approach, using the Uniform Data Access level described in [6], which nowadays is often referred to as *data virtualisation*. *Data virtualisation* is a technique for abstracting the information from its physical storage and organisation. This enables the cross-data source mediation between the multiple results and query formats during the execution of every request. Thus, all structure and semantic heterogeneities need to be resolved at runtime, which adds efficiency overheads. Furthermore, the pure federation approach is well known for its inability to efficiently deal with many *remote join* operations. A *remote join* is the computational merging of information between two distributed physical and/or logical interfaces.

In the Khresmoi project the integration strategies of data warehousing and federation demonstrate significant trade-offs because of the extreme amounts of semantic enabled data that need to be processed efficiently. Calabria [7] extends the two previously mentioned approaches to four architecture approaches presented in **Table 1**:

Table 1: Different data integration architecture used by the industry.

Integration Architecture	Type of data integration	Advantages	Disadvantages
Data warehouse	Warehousing	• Fast queries • Clean data • Full control over query optimizations	• Stale data • Complex schema • Redundant data
Mediator-based Architecture	Federation	• Current data • Flexible architecture • No duplication of data • Data autonomy	• Slower queries • Complex schema • Little or no data cleansing • Temporary unavailable services cause incomplete data fetching
Service Oriented Architecture	Federation	• Current data • Flexible architecture • Schema tailored to users • No duplicate data	• Requires source availability • Little or no data cleansing
Peer-based Architecture	Federation	• Current data • No global schema • Flexible and independent	• Slower queries • Experimental • Little or no data cleansing • Redundant data

Comparing the different approaches in **Table 1**, we see that each integration architecture offers significant advantages and disadvantages. It shows that the best solution for processing large-scale information by the KE tends towards warehousing because it provides:

- Forward chaining inference and materialization of trivial implicit statements;
- Even and predictable query performance across all information;
- Unlimited capabilities for data cleaning;
- Integration of consistency checking, which is often critical for the biomedical domain;

The federation approach and the Mediator-based Architecture enable the integration of autonomous data sources that cannot be replicated in a warehouse, because their content is updated too often and/or because of security restrictions. The Peer-based Architecture offers excellent flexibility, but the existing disadvantages, such as the lack of a global schema and the slow querie times.

3.1 RDF Warehousing

An RDF warehouse requires the translation of all data sources to RDF triples and loading the statements into different named graphs (contexts). Keeping different dataset in separate named graphs guarantees the minimal provenance information required in order to support incremental information updates.

The Linked Life Data (LLD) service [8] is an example of an RDF warehouse project that demonstrates excellent performance for a wide range of SPARQL queries against billions of RDF statements. The service relies on a highly efficient persistence of RDF, a query optimizer and an integrated forward chaining reasoner that enables the indexed search of implicit statements. Once all the information is consolidated into a single physical structure, resource alignment rules are defined to link related identifiers. Figure 2 depicts six alignments rules, where the dashed lines and the blue text of the captions (used either as part of the URI or literals) designate the criteria for linking the information. Since the specified mapping rules are not universally applicable for arbitrary RDF datasets, they are manually controlled for each specific subset.

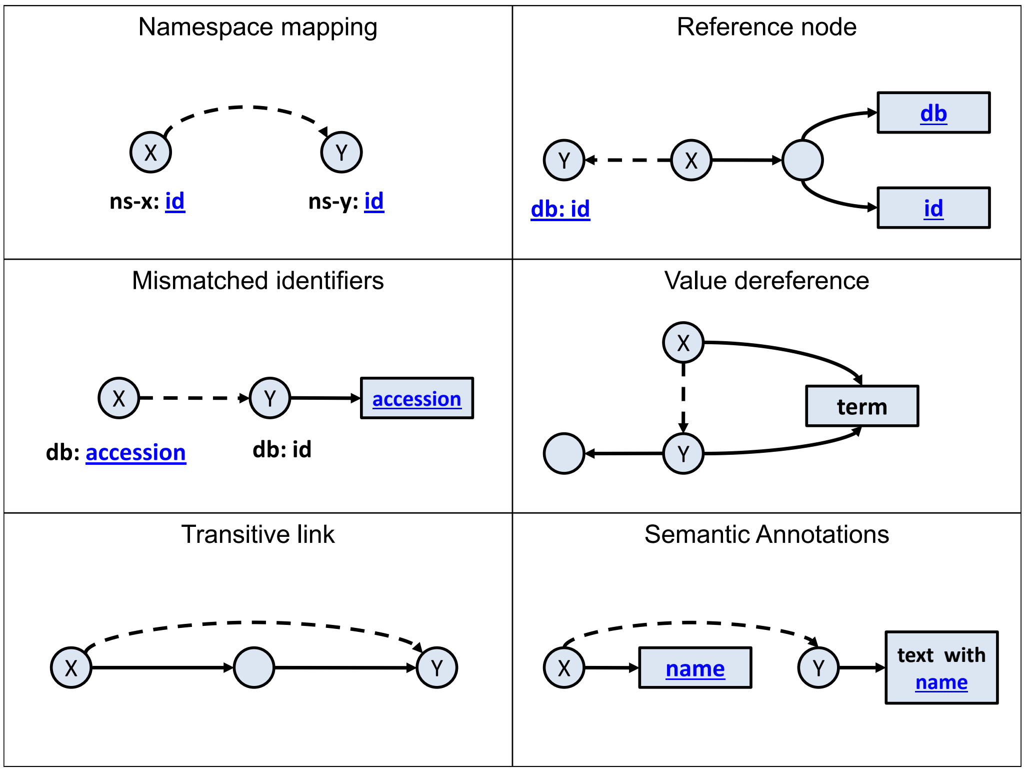


Figure 2 Resource alignment patterns in LLD.

Figure 3 illustrates the semantic aspect of the instance mappings.

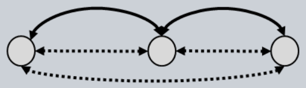
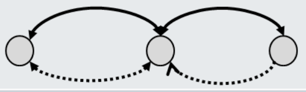
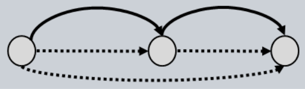
Relationship	Semantics	Example
Exact match	Transitive equivalence	
Close match	Equivalent only for search purposes	
Broader match	Generalization of a concept	
Narrower match	Specialization of a concept	Inverse of broader match

Figure 3: Instance mappings in LLD using the SKOS vocabulary.

Local RDF storage engines can provide full control over the query optimisations and statistics on each value's associations, allowing the calculation of optimal execution plans for complex information joins. Queries with unbound predicates are especially difficult to optimize. The SPARQL query presented in **Table 2** lists all unique predicates for resources of type Protein. The first pattern executed against LLD 0.8 results in 16,505,340 possible bindings. The second pattern to be merged with the first one results in 5,120,886,447 possible bindings. This yields a total of 8.45E+16 tuples if naive optimisation is used. However, the total execution time for the presented query is less than 60 seconds despite its extreme complexity. Similar types of queries are not practical for any sort of pure federated environments because of the previously mentioned *remote join* limitation.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX uniprot: <http://purl.uniprot.org/core/>

SELECT DISTINCT ?predicate
WHERE {
        ?subject rdf:type uniprot:Protein .
        ?subject ?predicate ?object.
     }
```

Table 2: SPARQL query with unbound predicate.

In conclusion, warehouse architectures, like the one implemented by the LLD project, guarantee fast queries because they offer a single data model, storage engine-provided query optimisation strategies and the possibility of data quality control and cleaning. However, the central control over the information comes at a cost. Every warehouse has redundant and stale data, which requires regular updates. In some scenarios, the warehousing approach is not compatible with specific data source licences or imposed security restrictions. The next chapter will investigate how the KE architecture may overcome these limitations.

3.2 SPARQL Endpoint Federation

The service-oriented and mediator-based architectures are very similar, with the exception of the API interface that is used to communicate with the federated databases. In the service-oriented architecture the integration middleware will support any type of data objects returned by the service and will automate their mappings to the global model. In the context of RDF, the mediator-based architecture is very close to the SPARQL 1.1 federation working draft [9]. The working draft is produced as a response to the increasing number of public SPARQL endpoints, the need to integrate information distributed across the web and to overcome the licence or security restrictions imposed by specific data sources. The SPARQL language syntax and semantics are extended with the `SERVICE` and `BINDINGS` keywords that enable the query rewrite to compose a query that delegates specified triple patterns to a series of services.

The SPARQL federation is a very promising approach to the simple integration of very large non-semantic databases. The R2RML working draft [10] presents a language for expressing customized mappings from relational databases to RDF dataset that could be further exposed as a virtual SPARQL endpoint over the mapped relational data. Still, a major challenge for efficient query execution is the limited interface that does not include the sharing of any statistics to be used in the optimisation. Nonetheless, the access to virtual SPARQL endpoints is a practical way to overcome security and licence limitations.

4 Biomedical Knowledge Engine

The KE is a core Khresmoi component. [11] defines the persistence layer responsible for the storage, retrieval and integration of information. Furthermore, it automates the execution of batch core services to process huge amounts of information. Figure 4 defines the KE reference architecture and the different supported external interfaces:

- Datasets: the data sources to be processed by the KE. The complete list of the possible datasets, their licence and purpose is presented in [12] and how they are used in KE is described D5.2; may change with the evolution of the project;
- Batch processing and Extract Transform Load (ETL): the layer responsible for overcoming the structure heterogeneity, cleaning data and aligning the semantic incompatibilities between the different data sources; the output of each ETL job is a stream of RDF statements that can be directly fed to the semantic database;
- Semantic Database: the component responsible for controlling the persistent information, required indexes, forward chaining reasoner and query optimisation statistics; the semantic database is realized with an RDF database instance;
- Knowledge Engine: implements all specialized functionality required on top of the semantic database, exposes specialized data access interfaces and offers a standard SPARQL endpoint; the KE also includes a web interface for knowledge navigation, exploring and linked data publishing;
- Web services: summarize all interfaces required to be implemented by the Knowledge Query Service (KQS) defined in [11];
- SPARQL/Sesame: a standard API, exposed by the KE to access the persisted information with other semantic clients; Sesame is de facto considered a standard interface for RDF repository access;

- Exports: offer specialized information dumps and interface for bulk operations like knowledge mining or analysis;

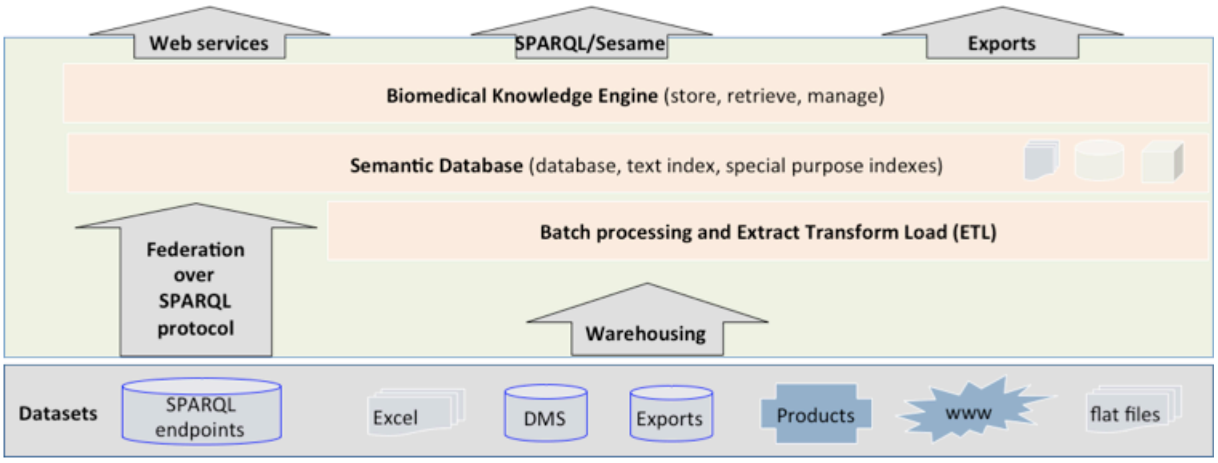

Web services
SPARQL/Sesame
Exports
Biomedical Knowledge Engine (store, retrieve, manage)
Semantic Database (database, text index, special purpose indexes)
Batch processing and Extract Transform Load (ETL)
Federation over SPARQL protocol
Warehousing
Datasets
SPARQL endpoints
Excel
DMS
Exports
Products
www
flat files

Figure 4 Biomedical Knowledge Engine reference architecture.

4.1 Batch processing and Extract Transform Load

The Khresmoi project presents the problem of dealing with dynamic heterogeneous data, originating from many different sources. The relevant information will have to be extracted from the data sources, transformed into a format, supported by the system, and semantically disambiguated before loading it into the knowledge base. This process is commonly referred to as Extract-Transform-Load (ETL).

ETL is a data insensitive process, which requires high efficiency not only in terms of good performance and scalability, but also in easy maintenance and traceability of each individual step of the complete integration process. Talend Open Studio (TOS) is an open-source solution for performing data integration, based on the Eclipse platform that offers off-the-shelf a powerful toolkit and infrastructure for designing data processing tasks known as jobs. TOS uses java code, generated by Java Emitter Templates (JET), a part of the Eclipse EMF Framework. The templates are based on the ETL process model and the data model, allowing for high-level abstraction programming or meta-programming. JET templates are similar to the Java Server Pages (JSP) syntax and primarily contain static code, which is output "as is". The fixed content is enriched by a number of JSP-like tags that are evaluated and interpreted by the generation engine in various ways [13]. A summary of the available tags is given in **Table 3** JET Template Tags:

Type	**Syntax**	**Description**
JET Directive	*<%@ jet attributes %>*	Declares the beginning of the template
Include Directive	*<%@ include file="URI" %>*	Includes another template
Expression	*<%= expression %>*	Inserts the expression result
Scriptlet	*<% code %>*	Executes the code fragment

Table 3 JET Template Tags

In order to get a better understanding of the code generation, used by TOS, consider the following excerpt from tGateDocumentToRDF – a component, responsible for the serialization of GATE semantic annotations to RDF. As a first step, the component has to create/open a file on the system for writing, based on user input:

```
<%@ jet
imports="
      org.talend.core.model.process.INode
      org.talend.core.model.process.IConnection
      org.talend.designer.codegen.config.CodeGeneratorArgument
      org.talend.core.model.process.ElementParameterParser
      "
%>
<%
      CodeGeneratorArgument codeGenArgument = (CodeGeneratorArgument)
argument;
      INode node = (INode)codeGenArgument.getArgument();
      String cid = node.getUniqueName();
      boolean append = (Boolean)
ElementParameterParser.getObjectValue(node, "__APPEND__");
      String location = (String)
ElementParameterParser.getObjectValue(node, "__LOCATION__");
%>

java.io.Writer writer_<%=cid%> = null;
      try {
                  java.io.File f = new java.io.File(<%=location%>);
writer_<%=cid%> = new java.io.OutputStreamWriter(new
java.io.FileOutputStream(f, <%=append%>));

      } catch (java.io.IOException e) {
                  throw new RuntimeException("Error while serializing
data!", e);
      }
```

Table 4 Meta-programming with JET.

JET templates enable the easy extension of the environment with custom components that introduce new features to be used within the graphical designer. A key advantage of the TOS environment, compared with the other data integration or mediation technologies, is that it uses the formal pipeline descriptions to generate executable java classes, which perform the actual work. A pair of formal description and generated java code is referred to as a *job*. Compared to the same tasks implemented in pure java code, these jobs have a performance decrease lower than 10^{-3}. Jobs can be exported as stand-alone executable java archives (JARs) to be deployed to a production server. At the same time the intuitive TOS graphical user interface provides an easy way to debug the data flows and run them in a batch mode or scheduled sequences. Parallelization is another issue addressed by TOS on job-design level, allowing the simultaneous execution of sub-processes on multiple threads.

In the context of the Khresmoi project we have developed text analysis components to integrate the existing WP1 tools and infrastructure.

Table 5 lists all newly developed Talend components and explains their purpose:

Name	**Family**	**Description**
tGateAnnotator	GATE	Annotates text using a specified GATE application (*.xgapp); suitable for parallelization
tGateInstantiator	GATE	Loads a specified number of GATE application instances
tGateDocumentToRDF	GATE	Transforms a GateDocument object into RDF statements (triples)
tGateDocumentToXML	GATE	Serializes a Gate document in XML format
tMimirInitializer	Mimir	Creates and opens a new Mimir index
tMimir	Mimir	Extracts data from GATE documents and adds it to a Mimir index
tMimirClose	Mimir	Closes and finalizes a Mimir index

Table 5 Talend components that integrate the WP1 infrastructure.

The extensions of TOS provide a powerful infrastructure for populating and maintaining the Khresmoi knowledge base. Several processes are already implemented as jobs and used in practice to create and support the knowledge base prototype.

So far, three key scenarios/tasks have been identified: converting heterogeneous data into RDF; importing data into the semantic database (OWLIM); and integrating data with WP1 annotation tools. For an example TOS workflow Figure 5.

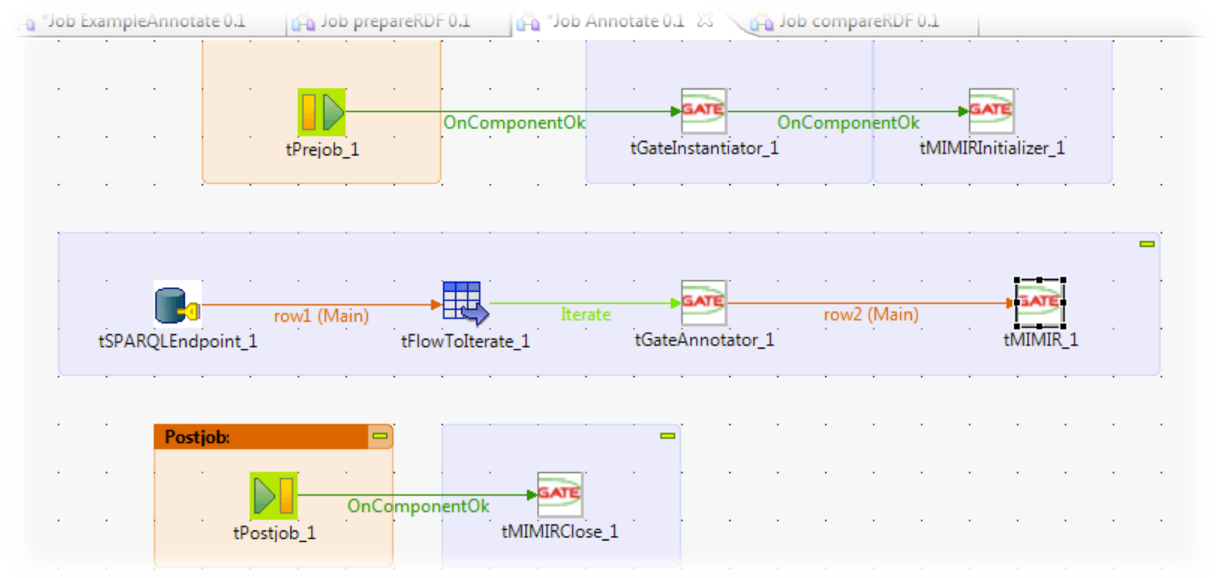


Figure 5: Data is extracted from a remote SPARQL endpoint, annotated using a predefined GATE application and indexed by infrastructure, created in work package 1.

4.2 Semantic Database

OWLIM is a product family of semantic databases, fully implemented in Java and compliant with the most popular RDF connectivity APIs – Sesame and Jena. It comes in three editions: OWLIM-Lite, OWLIM-SE (Standard Edition) and OWLIM-Enterprise that all share the same inference mechanisms, rule language and rule compiler. Thus, the product family ensures smooth interoperability and capacity expansion from small research prototypes to big enterprise clusters, capable of processing millions of queries [14] with industrial strength resilience and automatic fail-over. OWLIM-Lite is the fastest repository where all operations are executed in memory. Its scalability is limited only to the available hardware RAM. In the typical usage scenarios it is designed for datasets of 100 million statements and it features the following key characteristics:

- Reasoning and query evaluation are performed entirely in main memory;
- It employs a persistence strategy that ensures data preservation and consistency;
- The loading of data, including the time for forward chaining, is extremely fast;
- Easy configuration;

OWLIM-SE (previously BigOWLIM) is suitable for handling massive volumes of data and very intensive querying activities. It is designed as a commercial-grade database management system. This has been made possible through:

- File-based indices, which enable it to scale to billions of statements even on desktop machines;
- Special-purpose index and query optimization techniques, ensuring fast query evaluation against very large volumes of data;
- Optimized handling of owl:sameAs (identifier equality) to boost efficiency for data integration tasks;
- Efficient invalidation of inferred statements, which allows efficient delete operations;

OWLIM-Enterprise (previously BigOWLIM Replication Cluster) is designed for resilience and parallel query-answering performance through:

- Parallel execution of queries on multiple worker nodes;
- Dynamic configuration of cluster;
- Automatic fail-over and synchronization - no single-point of failure;

Table 6 summarizes all OWLIM features and the common application scenarios, depending on the data scale:

Product name	Features	Application
OWLIM-Lite	• Fast • In memory • Non-trivial inference • SPARQL queries	*Small* repositories of up to 10 million statements
OWLIM-SE	• Highly scalable • Non-trivial inference • Multi-user support • Optimized owl:sameAs handling • Hybrid queries (SPARQL + fulltext, geospacial etc) • RDF rank • Plugin extendable	Repositories of up to 20 billion statements
OWLIM-Enterprise	• All OWLIM-SE features • industrial strength resilience • linearly scalable parallel query performance • load-balancing • automatic fail-over	Replication cluster infrastructure based on OWLIM-SE

Table 6 OWLIM edition comparison.

All editions come with several standard prebuilt rule-sets, namely RDFS, OWL-Horst (similar to pD*), OWL-Max (RDFS with most of OWL 2) and OWL 2 profiles RL and QL [15]. Due to the fact that OWL 2 QL is designed for query rewriting over relational databases, the OWLIM implementation using forward chaining reasoning is suboptimal and no claim for the complete support of this profile is made. The users are able to build their own custom rule-sets using datalog like rules with inequality constraints. **Table 7** gives an example of the rule language syntax by showing the implementation of owl:FunctionalProperty.

```
Id: prp_fp
p <rdf:type> <owl:FunctionalProperty>
x p y1 [Constraint y1 != y2]
x p y2
-------------------------------
y1 <owl:sameAs> y2
```

Table 7 OWLIM language example – the definition of owl:FunctionalProperty

The OWLIM semantic database will guarantee excellent data loading and query execution speed of the KE [16]. The upcoming deliverable D5.3: "Scalability and performance evaluation report" will provide a detailed report of the Large Biomedical Knowledge Server performance, delivered in D5.2.

4.3 Semantic Database Extensions

Nowadays it is expected that every semantic database provides SPARQL support. Still, there are often data intensive computations, which go well beyond the standard query expressivity either because 1) they tend to incorporate complex procedure logic or 2) the language algebra is not sufficient to cover it, like in the case of a vector space model. Regardless of the specific problem, the motivation is to push the computation as close to the data as possible, in order to guarantee decent query efficiency. Such an optimisation guarantees that no significant amounts of data will be moved outside the database process address space and the query execution planner can benefit from using a low level interface. This type of optimization is nowadays standard for every database system. In the context of OWLIM, this data efficiency problem is resolved with the introduction of the plug-in API that allows the mapping between *special predicates* and a piece of software logic, added to the class path of the database. The *special predicates* are special purpose Internationalized Resource Identifiers (IRIs), used in SPARQL triple patterns on the predicate position to denote special query evaluation strategies. For instance, OWLIM 4.x supports geospatial resource indexing, which is not practical to be stored as precomputed information by the forward-chaining reasoner.

```
PREFIX geo-pos: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX geo-ont: <http://www.geonames.org/ontology#>
PREFIX omgeo: <http://www.ontotext.com/owlim/geo#>

SELECT distinct ?airport_name
WHERE {
  ?a1 geo-ont:name "Bournemouth" .
  ?a1 geo-pos:lat ?lat1 .
  ?a1 geo-pos:long ?long1 .
  ?airport omgeo:nearby(?lat1 ?long1 "80mi" ) .
  ?airport geo-ont:name ?airport_name .
  ?airport geo-ont:featureCode geo-ont:S.AIRP .
  ?airport geo-pos:lat ?lat2 .
  ?airport geo-pos:long ?long2 .
  ?a2 geo-ont:name "Brize Norton" .
  ?a2 geo-pos:lat ?lat3 .
  ?a2 geo-pos:long ?long3 .
  FILTER( omgeo:distance(?lat2, ?long2, ?lat3, ?long3) < 80)
}
```

Table 8: Special predicate query that uses geo-spatial indexes.

Table 8 shows a query that finds all the airports within 80 miles of Bournemouth and filters out those that are more than 80 kilometres from Brize Norton. The omgeo:nearby and omgeo:distance predicates have a special purpose, configured via the OWLIM plug-in API. The two IRIs call for redirecting the triple pattern to an external index .The return results are combined with the rest of the query.

The OWLIM plug-in API will be the primary mechanism to implement the newly emerging Knowledge Query Services (KQS) initially described in [11], which are not effectively expressed using 'pure' SPARQL queries. Hence, by using the plug-in API it is possible to introduce new *special predicates* that extend the SPARQL algebra and prevent the necessity to add a new web service for the required functionality.

5 Conclusion

In order to integrate the highly heterogenous data required for the Khresmoi project, the RDF model with additional support for different ontologies has been chosen as the foundation of the KE. The scale of information to be integrated and the requirements for query evaluation speed, data model consistency and data privacy make the warehousing approach preferable to federation. Consequently, ETL infrastructure was developed in order to manage the highly dynamic data, thus providing a formal and repeatable process for updating the KE. The OWLIM product family will provide the storage engine with native RDF and SPARQL support and a light API for extending the KE with additional functionality.

As the use cases for the Khresmoi project are still in development, the datasets to be integrated are subject to change. Hence, their description and processing is not in the scope of this document. A list of the integrated datasets and their modelling will be provided in D5.2. Still, we can conclude that the infrastructure for data integration, management, update, access and evaluation is present.

6 References

[1] Lenzerini, M. (2002): Data integration: a theoretical perspective. In: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. ACM, New York, 233–246.

[2] Sheth A 1998, Changing Focus on Interoperability in Information Systems: From System, Syntax, Structure to Semantics, in Interoperating Geographic Information Systems, M. F. Goodchild, M. J. Egenhofer, R. Fegeas, and C. A. Kottman (eds) Kluwer Publishers.

[3] Lassila O., and Swick R.: Resource Description Framework (RDF) Model and Syntax Specification, 1999, available at: http://www.w3.org/TR/PR-rdf-syntax/

[4] Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF (January 2008), http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/

[5] Renzo Angles and Claudio Gutierrez, The Expressive Power of SPARQL. Available at: http://www.dcc.uchile.cl/~cgutierr/papers/expPowSPARQL.pdf

[6] Ziegler, Patrick & Dittrich, Klaus R.: Three decades of data integration - All problems solved? Kluwer (2004), S. 3-12.

[7] Calabria A., Data Integration for Clinical Genomics, Doctorate Thesis, 2009, available at: http://boa.unimib.it/bitstream/10281/19219/3/Phd_unimib_716358.pdf

[8] Vassil Momtchev, Deyan Peychev, Todor Primov, Georgi Georgiev. Expanding the Pathway and Interaction Knowledge in Linked Life Data. In Proc. of International Semantic Web Challenge, 2009.

[9] Prud'hommeaux E., Seaborne A., SPARQL 1.1 Federation Extensions, W3C Working Draft 1 June 2010

[10] Souripriya Das, Seema Sundara, Richard Cyganiak. R2RML: RDB to RDF Mapping Language W3C Working Draft 24 March 2011

[11] Emmanuel Jamin, Vassil Montchev, Konstantin Pentchev, revised by Allan Hanbury. D6.3.1 State of the art, concepts and specification for the "Early software architecture".

[12] Valerie Mapelli, Jungyeul Park, Khalid Choukri. D6.1 Initial version and report on the KHRESMOI data collection.

[13] F. Budinsky, et al. Eclipse Modeling Framework. Addison-Wesley, 2004.

[14] Scaling to Millions of Concurrent SPARQL Queries on the Cloud. http://www.slideshare.net/ontotext/owlim-cluster

[15] Barry Bishop, Spas Bojinov., Implementing OWL 2 RL and OWL 2 QL rule-sets for OWLIM

[16] Bizer, Ch., Schultz, A.: Benchmarking the Performance of Storage Systems that expose SPARQL Endpoints. In: Proceedings of the 4th International Workshop on Scalable Semantic Web knowledge Base Systems, SSWS2008, 2008