Integrative Cancer Research
INTEGRATE
Through Innovative Biomedical Infrastructures

SEVENTH FRAMEWORK PROGRAMME

ICT-2010-270253

INTEGRATE

Driving excellence in Integrative Cancer Research through Innovative Biomedical Infrastructures

STREP
Contract Nr: 270253

Deliverable: 2.1 State-of-the-Art Report on Standards

Due date of deliverable: (10-31-2011)
Actual submission date: (10-31-2011)

Start date of Project: 01 February 2011 Duration: 36 months

Responsible WP: Philips

Revision: Final

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)		
Dissemination level		
PU	Public	x
PP	Restricted to other programme participants (including the Commission Service	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (excluding the Commission Services)	

0 DOCUMENT INFO

0.1 Author

Author	Company	E-mail
Gaston Tagni	Philips	gaston.tagni@philips.com
Alejandro Garcia Ruiz	UPM	agarcia@infomed.dia.fi.upm.es
Kristof De Schepper	Custodix	kristof.deschepper@custodix.com
Brecht Claerhout	Custodix	brecht.claerhout@custodix.com
Elias Neri	Custodix	elias.neri@custodix.com

0.2 Documents history

Document version #	Date	Change
V0.1	08/29/2011	Starting version, template
V0.2	08/29/2011	Definition of ToC
V0.3	10/14/2011	First complete draft
V0.4	10/14/2011	Integrated version (send to WP members)
V0.5	10/14/2011	Updated version (send PCP)
V0.6	10/14/2011	Updated version (send to project internal reviewers)
Sign off	10/28/2011	Signed off version (for approval to PMT members)
V1.0	10/28/2011	Approved Version to be submitted to EU

0.3 Document data

Keywords	
Editor Address data	Name: Gaston Tagni Partner: Philips Address: High Tech Campus 34, (6-030) 5656AE Eindhoven, The Netherlands Phone: +31 40 27 49709 Fax: E-mail: gaston.tagni@philips.com
Delivery date	November 4, 2011

0.4 Distribution list

Date	Issue	E-mailer
November 2011		Loukianos.GATZOULIS@ec.europa.eu
		fp7-integrate@listas.fi.upm.es

Table of Contents

1 Introduction

INTEGRATE aims at building an infrastructure of components and tools for enabling wide-scale integration of data, information and knowledge generated within clinical trials in the area of oncology research. Such infrastructure will be based, to the maximum extent possible, on standards and well-known approaches for data integration, security and privacy management.

One of the first steps towards achieving this goal is to survey the landscape of standards and technologies stemming from different areas of technology and research relevant for building the solution envisioned by the project. With that aim in mind and within the context of the technical work package 2 (WP2) this deliverable presents a state-of-the-art survey of relevant technologies and standards. The survey intents to map the pleothora of methods, techniques, tools and standards that are thought to be relevant for different aspects of the project. More specifically, this deliverable gives an overview of several Semantic Web technologies for enabling data integration and sharing of knowledge (ontologies) among systems. The document presents an overview of the state of the art in various research topics including Ontology Representation Languages (Chapter 2), Semantic Repositories (Chapter 3), Semantic Reasoning systems (Chapter 4) and several methods for Ontology Mediation (ontology matching and merging) (Chapter 5). The type of data and information to be integrated by the INTEGRATE platform lies within the domain of Clinical research and Life Sciences and as such it must deal with data described according to standard ontologies and terminologies commonly used in this domain. Consequently, in Chapter 6 of this survey we report on several standard ontologies that are being used in the medical domain. Another crucial aspect of the INTEGRATE vision are the security and privacy aspects which are required in order to guarantee secure and validated access to the different users of the platform. Therefore, Chapter 7 reports on several security and privacy technologies that are thought to be highly relevant for the project. Finally, Chpater 8 presents the conclusions of this survey.

2 Ontology Representation Languages

2.1 Introduction

During the last decade the *Semantic Web Activity* group [1] of the W3C has proposed a series of standards, in the form of recommendations, for representing (in machine-processable form), organizing and accessing semi-structured data in the Web. The key technologies proposed so far, on top of which many others have been and are being proposed, are *RDF*, *RDFS* and *OWL*. These technologies make use of other technologies such as the *Uniform Resource Identifier* (URI) and the *eXtensible Markup Language* (XML). In a nutshell, RDF provides a framework for organizing and exchanging semi-structured data in machine-processable form that abstracts away from the specific vocabulary and syntax used for referring to the data of a subject domain. RDFS and OWL provide a common knowledge modelling language for capturing and explicitly representing the meaning of information and for organizing the knowledge of a subject domain in the form of ontologies. SPARQL provides a standard way to access semi-structured data in the Web. In this chapter we want to give a brief overview of these standard technologies as they have become relevant for building solutions that support interoperability and data integration across application domains.

This chapter is organized as follows. Sections 2.2 to 2.5 give a brief overview of key ontology representation languages developed by the Semantic Web community and sponsored by the World Wide Web Consortium (W3C) that have become standards for knowledge and metadata representation in the Web and other domains. Section 2.6 briefly introduces the SPARQL query language. Although technically not an ontology representation language, SPARQL is a key technology enabling semantic-based data integration and metadata extraction in the Web. Finally, Section 2.7 concludes the chapter with a discussion of the relevance of these technologies for the INTEGRATE project.

2.2 Resource Description Framework (RDF)

The *Resource Description Framework* (1) (RDF for short) is framework developed by the *World Wide Web Consortium* (W3C) for representing semi-structured information in the Web. It enables users to describe Web content with arbitrary vocabularies in order to associate semantics to the data through the specification of metadata in machine-processable form. RDF constitutes one of the core technologies in the Semantic Web technology stack and relies on the concept of URIs (Uniform Resource Identifier) to uniquely identify resources that need to be described. Since its conception RDF has become the W3C's standard for representing semi-structured information in the Web and other domains where semantic-based data integration is required.

Although typically referred to as a language for representing semi-structured information in the Web RDF is more precisely a Data Model. It defines a framework for modelling the data and information associated to a given domain of discourse. The basic principle behind RDF is that the elements of a domain of discourse can be described in terms of their properties and values and that such description(s) can be represented by statements or RDF triples. Each statement in the model captures the description of a *resource*, i.e. the "*thing*" one is interested in talking about or

[1] http://www.w3.org/2001/sw/

describing, and specifies the *property* used for describing the resource and its *value*. In RDF terms each statement has the form *subject-predicate-object* where the *subject* represents the resource or "thing" one wants to describe, the *predicate* represents the property used for describing the resource and the object represents the *value* of the property. Properties in RDF are used for capturing relationships between resources and resources are uniquely identified by URIs.

RDF represents (a set of) statements using directed graphs, known as *RDF graphs* in the RDF terminology. This directed graph is the abstract syntax or *graph model* of RDF (see the *RDF Concepts and Abstract Syntax* for details (2)). Moreover, RDF graphs can be associated with a name (URI) to uniquely identify a set of statements, a concept known as *RDF Named Graphs.* This mechanism extends the expressive power of RDF as it allows referring to and describing set of triples. In order for computer programs to be able to process RDF data the abstract syntax of RDF needs to be serialized using some sort of notation or language. One of the most common ones is RDF/XML that serializes an RDF graph in XML. However, other, more efficient notations also exist such as N3 (Notation 3) [2], N-Triples [3] and Turtle (3).

An important feature of RDF is that the framework makes no assumption about the semantics of a given domain nor does it prescribes a vocabulary for describing and modelling data; apart from a few language constructs used for defining statements and related RDF concepts. This allows users to use the framework to describe arbitrary domains by using vocabularies specific to the domain of discourse.

2.3 Vocabulary Description Language (RDFS)

As a general framework and Data Model for representing semi-structured information RDF does not impose any restriction on the vocabularies used for modelling the information of a given domain. Instead, users are free to use and define their own vocabularies to describe resources and the relationships among them foe any arbitrary domain. In order to define arbitrary vocabularies what is needed is a language that allows users to specify the terms in the vocabulary and their semantics.

RDFS (4), the *RDF Vocabulary Description Language*, is a W3C recommendation of a language for specifying vocabularies, i.e. sets of terms and with their associated meaning. RDFS enables users to define *concepts* (classes of resources) and *properties* that can be associated to these concepts in order to describe them. Classes and properties defined in RDFS can then be used to describe a domain of discourse using the RDF Data Model. In addition, RDFS provides semantics for specialization/generalization of classes and properties and for specifying resources as instances of classes. In other words, RDFS provides the means in the forms of language constructs to describe vocabularies, i.e. concepts and properties, which can then be used for formalizing the knowledge of an application domain.

Using terms from the predefined RDF and RDFS vocabularies users can defines classes, properties, associate properties to classes, specify that individuals in the domain are instances of certain classes, restrict the domain and range of properties, specify generalization/specialization of classes and properties, define special data structures such as containers and collections, among other constructs. One of the

[2] http://www.w3.org/DesignIssues/Notation3
[3] http://www.w3.org/TR/rdf-testcases/#ntriples

limitations of RDFS and RDF, or perhaps an advantage depending on the way one looks at it, is that the language does not provide the means for expressing *negation*, i.e. the language does not include a construct for expressing the negation of facts. This can be seen as a limitation of the language as it reduces the expressive power of RDF(S) or, as a positive feature as it makes reasoning in the language tractable.

2.4 OWL

OWL (5) stands for *Ontology Web Language* and is a the Web ontology language developed by the Web Ontology working group of the W3C that has been a W3C Recommendation since 2004. OWL is a language for representing ontologies, i.e. for specifying its concepts, relationships between concepts, individuals and axioms. It enables users to define classes (sets of objects), associate properties to classes, define relationships between classes and between individuals. Concepts in OWL can be defined based on the union, intersection and complement of other concepts. In addition, the language allows for specifying equivalence and disjoint relationships between concepts. In terms of its expressive power OWL is more expressive than RDF(S). The formal underpinning of OWL is the family of knowledge representation formalisms known as Description Logics (6).

OWL comes in three standard flavours or sub-languages, although other non-standard dialects or fragments have been proposed in different application domains and scenarios. These fragments were designed to serve different purposes depending on the needs of the application at hand. The first of these standard fragments is *OWL Lite*, which is the least expressive fragment of the three and corresponds to the DL $\mathcal{SHIF}(\mathcal{D})$. OWL Lite is meant for users in need of classification hierarchies and simple constraints. The second fragment is *OWL DL*, which corresponds to the DL $\mathcal{SHOIN}(\mathcal{D})$, is a more expressive language with good computational properties. Finally, *OWL Full* is the most expressive fragment of the three although it provides no guarantees with respect to its computational properties. Among the non-standard fragments are *OWL Horst* and *OWL DLP*. OWL Horst (7) is a fragment of OWL that is based on the extension of RDFS with rules. The language defines inference rules, called *R-entailment* rules, that are more general than those defined for RDFS, i.e. D-entailment rules. The salient characteristics of this language are that it does not enforce the Unique Name Assumption (UNA) and that its complexity is lower than that of SWRL (8). *OWL DLP* is a language whose expressive power lies in the intersection of OWL DL and Logic Programming and is less than that of OWL Lite. One of the advantages of OWL DLP is that both DL reasoners and deductive systems can be used for reasoning in the language and that the knowledge of a domain can be modelled using OWL and/or rules.

2.5 OWL 2.0

OWL 2 (9) is a revision and extension of OWL 1.0 that extends its syntax with syntactic sugar and expands its expressive power by adding asymmetric, reflexive, and disjoint properties, qualified cardinality restrictions and richer data types among other features. In the same way that OWL has three standard fragments OWL 2 offers three standard language profiles. These profiles restrict the syntax of the language in an attempt to make a trade off between efficient reasoning and expressive power. OWL 2 profiles are tailored to specific application and users needs in terms of the type of knowledge

that knowledge engineers need to model. The three profiles are *OWL 2 EL*, *OWL 2 QL* and *OWL 2 RL*.

- **OWL 2 EL**: OWL 2 EL is a subset of OWL 2 for which the basic reasoning services can be solved in polynomial time. It corresponds to the DL $\mathcal{EL}$++ (10) and it is particularly useful for expressing ontologies with large number of classes and/or properties that do not require too much expressive power. Many biomedical ontologies can be expressed with OWL 2 EL and many efficient reasoning systems for this language have been developed in recent years (see the section on Semantic Reasoners for an overview).
- **OWL 2 QL**: OWL 2 QL is tailored for applications that use large amount of instance data combined with relatively inexpressive logics and whose main reasoning service is query answering.
- **OWL 2 RL**: This is a profile aimed at applications that require scalable reasoning without losing too much expressive power. Reasoning in OWL 2 RL can be implemented using *rule-based approaches*.

2.6 SPARQL

SPARQL (*SPARQL Protocol and RDF Query Language*) (11) is a W3C Recommendation of a query language for RDF. Since its inception it has become one of the most widely used query languages for semi-structured data expressed in RDF and it is nowadays a Semantic Web standard query language. Virtually all RDF repositories nowadays provide an implementation of the language and many data repositories in the Web provide query services backed by SPARQL, the so-called SPARQL endpoints. Queries in SPARQL define an RDF triple pattern with variables over the RDF graph. A query result is a set of RDF triples satisfying the pattern and binding the free variables to RDF nodes.

2.7 Discussion and Relevance for INTEGRATE

INTEGRATE is about data integration and interoperability between systems. One way to achieve this is through the use of semantic technologies as they have been used in recent years to achieve this goal in many domains. Therefore, semantic technologies for representation and exchange of data and knowledge are important (crucial) for achieving the project goals.

3 Semantic Repositories

For (Web) data to be useful three fundamental aspects need to be addressed. First, data need to be represented in machine-processable form through the use of data representation and exchange languages. Second, data needs to be stored and managed and third, data need to be accessible to allow the generation of information and knowledge. Chapter 2 introduced several approaches for representing in machine processable form, structuring and accessing (querying) data in the Web. This Chapter is concerned with the second aspect of data management and processing, storing data and more specifically, storing and managing data structured according to RDF. Chapter 4 will cover the third aspect, namely reasoning over semi-structured data whose meaning is described according ontologies.

In the semantic Web community several approaches have been proposed and investigated for efficiently storing and querying data structured according to RDF. Many of these systems started as academic projects but have since then gain wide-spread adoption in the semantic Web community and other communities and become commercial products with enterprise-level features. Example of these systems include OpenLink Virtuoso, Ontotext's OWLIM family of repositories and Aduna's Sesame system. These systems, called semantic repositories, provide functions and services similar to those provided by traditional DBMSs with the fundamental difference that the data they manage is structured according to RDF rather than being relational data.

In this chapter we focus our attention on semantic repositories and present an overview of the state-of-the-art systems identifying their main characteristics and describing their key functionality and services provided. To this end we organize the Chapter as follows: Sections 3.1 to 3.6 gives an overview of the different semantic repositories considered in this report. The systems reported are: Virtuoso, Sesame, OWLIM, BigData, AllegroGraph, the Jena RDF Framework, RDFStore, D2R and the LarKC Platform for large-scale reasoning. Section 3.7 concludes with a discussion of the relevance of these technologies for the INTEGRATE project.

3.1 Sesame

Sesame is an open source, Java-based framework for storing, processing and querying RDF data efficiently. It was developed as a prototype application within the European project On-To-Knowledge [4] and is currently maintained by Dutch software company Aduna [5]. Over the years Sesame has become one of the defacto platforms for managing RDF metadata, a feature that is reflected by the number of other semantic technologies that make use of it. For example, many semantic technologies vendors implement the Sesame's SAIL.

3.1.1 System Architecture

Sesame's architecture follows a layered design. Each layer implements a specific set of services that can be accessed by the layers on top. A layer's functionality is in turn implemented using the services provided by the underlying layers. The architecture

[4] http://www.ontoknowledge.org
[5] http://www.openrdf.org/

was designed as to enable part of the functionality to be implemented by other external components, such as reasoners. The architecture is comprised of the following layers:

- **RDF Model:** the bottom layer is the foundation upon which the entire Sesame framework is built. This layer implements a representation or model of RDF and provides methods for handling the different RDF entities. It is similar to the Jena RDF Model, a Java-based API for managing RDF data.
- **The RIO layer:** the RDF I/O layer provides low-level methods for reading and writing RDF documents. It contains parsers for reading RDF documents expressed in different file formats and writers for writing RDF to several output formats such as RDF/XML (the XML serialization of RDF), N-Triples, etc.
- **The SAIL:** this layer, the *Storage and Inference Layer*, is an API that provides an abstraction over specific storage and inference technologies and systems. It does that by implementing a series of methods for accessing RDF storage and inference engines. By implementing the SAIL interface different storage and reasoning systems can be used within Sesame to provide these capabilities to its clients. Sesame 2 ships with two predefined SAIL implementations: one that supports in-memory storage and the other supporting on-disk storage capabilities. Thanks to this approach Sesame is an extensible platform that can be paired with different technologies for storing and reasoning over RDF data depending on the specific application requirements. For example, for applications that manage small amounts of data Sesame can be configured to use an in-memory storage component thus reducing costs and improving performance. SAILs can be stacked up to provide powerful means for handling RDF data. For instance, by stacking SAILs it is possible to implement a caching mechanisms for schema information, which is typically accessed very often and requires less space in comparison to instance data. The SAIL enables Sesame to remain DBMS-independent and hides the details of specific storage technology from the users. Examples of systems that provide a SAIL implementation and can thus be integrated and used with Sesame include OWLIM, RDFStore and Virtuoso. A description of these systems is provided in other sections of this deliverable.
- **HTTP Client:** The latest version of Sesame (v 2.x) also implements an HTTP Client that provides users with methods to access Sesame repositories running behind HTTP servers thus enabling remote access to Sesame repositories.
- **Repository API:** On top of the SAIL, RIO and HTTP Client layers sits the Repository API which is a high-level API that implements a series of methods for handling RDF data. It allows developers to have simple access to methods for reading, querying and managing RDF data. Typically, developers make use of the methods provided by the components of this layer to load, create, query and modify RDF data.
- **HTTP Server:** Sesame 2 also implements an HTTP server that enables users to set up a Sesame repository behind an HTTP Server and access it through HTTP. This functionality is implemented using Java Servlet technology and JSP (Java Server Pages).
- **User applications:** Domain-specific applications that use the Sesame framework sit on top of the Repository API.

A core component of the Sesame architecture is the SAIL interface which provides DBMS-like functionality for RDF data and schema information. Other systems and frameworks that provide similar functionality include the Jena Framework for RDF

processing and the Redland RDF Application Framework. SAIL was designed with the following principles in mind:

- To provide a basic interface for managing RDF and RDFS data
- To provide an abstraction layer over specific storage mechanisms and technologies.
- To be able to provide scalable processing of RDF data and to be able to run on low-end hardware.
- To be extendable to support new RDF-based ontology languages.

3.1.2 Features

Sesame can be extended and configured to support a variety of technologies and standards related to several aspects:

- RDF serialization and representation formats: It is compatible with several RDF serialization formats such as XML, TRIG, TRIX, Turtle, N-Triples and N3.
- Storage technology: different storage systems can be used for storing RDF data such as different RDBMS, disk-based storage, in-memory storage, etc.
- Inference: Sesame supports different forms of reasoning, i.e. reasoning according to different semantics such as RDFS, OWL, etc.
- Query results: different query results formats are supported.
- Query languages: Sesame offers support for SPARQL and SeRQL semantic query languages. Latest version (2.6.0) supports the SPARQL 1.1 Federation extension. This gives the possibility to query and integrate data from public SPARQL endpoints. The latest version also offers the SPARQL Repository which allows users to programmatically connect to SPARQL endpoints via the Repository API.
- Storage of context and provenance information.
- Support for REST-full HTTP protocol and to provide a SPARQL endpoint.
- Support for transactional access to RDF data.

3.2 Virtuoso

OpenLink Virtuoso[6] is a cross-platform universal server developed by OpenLink Software. It offers a series of data and metadata facilities such as Web server capabilities, file and database server functionality, native XML storage and an Universal Data Access Middleware. The Virtuoso Universal Server comes in two flavours: Open Source and Commercial. The open source version, distributed under the GNU General Public License, differs from the commercial version in that it does not include the Virtual Database Engine (a mechanism for accessing multiple databases transparently as if it were a single database). At the time of writing, the latest version is 6.2 which can be installed on multiple platforms including Windows, Solaris, Mac OS X and Linux.

3.2.1 Features

The following is a list of the most important features offered by Virtuoso:

[6] http://virtuoso.openlinksw.com/

- It offers transparent access to existing data sources thus reducing the cost of integrating data stemming from different sources
- Virtuoso offers Internet and Web services as well as Data Management services:
    - Internet and Web services: these include XPath query functionality. MAIL services, XSLT transformation services, HTTP 1.1, SOAP, WebDAV, Free Text Indexing among others.
    - Data Management services offered by Virtuoso include: Access to external SQL data which enables reuse of data stored in existing databases, native storage of XML-based data and RDF and a native implementation of SQL data.
- Virtuoso can be used as the backend of other semantic repository technologies such as Jena RDF Framework and Sesame. For example, Virtuoso can be accessed in Sesame through a SAIL implementation.
- One of the recent features being offered by Virtuoso is its support for *clustering and data federation*.
    - Clustering enables Virtuoso to scale to large volumes of data by supporting distributed query answering. In addition to this, clustering can improve the performance of query processing.
    - Data federation allows Virtuoso to act as a virtual server that hides the details of the location of the data from the user. This also enables reuse of existing databases and leverage of database technology such as data optimization techniques.
- Virtuoso provides support for managing *RDF named graphs* by storing RDF-based data in quads, i.e. RDF triples plus a fourth element that refers to the graph the triple belongs to. This, for example, can be used for representing context information or provenance information.
- Virtuoso also provides *anytime query answering* capabilities. The server is able to return partial results to a query rather than returning the complete results set at once. The stopping criterion can be specified by the user. This is a desirable feature for a couple of reasons:
    - First, it protects the server from denial-of-service (DoS) attacks and
    - Second, because it enables a better interaction with the user as the user does not have to wait until all the results are processed, instead results are returned incrementally.

3.2.2 Data Storage

Virtuoso implements two different approaches for storing RDF-based data. The first method uses a *native implementation* of XML and RDF-based data. The second method, referred to as *"RDF as a view"* provides a view of the underlying relational data through either Virtuoso's own relational storage functionality or federation. This enables Virtuoso to store RDF data on relational databases thus allowing Virtuoso to leverage database technology for RDF storage and query such as optimization techniques for storing, managing and querying relational data. Another advantage of using this approach is that it facilitates the integration of Virtuoso with existing databases.

3.2.3 Reasoning in Virtuoso

Although Virtuoso does not have a native inference service implemented in the architecture reasoning capabilities can be implemented on top of Virtuoso and accessed through queries. Virtuoso allows users to access inference capabilities by two means. The first approach uses *entailment and materialization* and SPARQL to query the materialized data. The second approach is based on *query re-writing* at query time. Using this approach inference is enabled at query time by inserting modifiers into the SPARQL queries. One of the advantages of not using materialization is that data loading is much faster since no pre-processing of the data is needed. No materialization is suitable for large databases where the most commonly executed queries refer to only a subset of the entire database.

The type of ontological reasoning supported by Virtuoso includes *sub-class* and *sub-property relations*, *identity by inverse-functional properties*, *owl:sameAs* and *transitive relations*.

3.3 OWLIM

OWLIM (12) is a family of semantic repository solutions. It is fundamentally a DBMS that provides functions to manage (store, query and reason over) data structured according to RDF. It is the result of the research efforts partially made within several European projects including SEKT[7], TAO[8], TripCom[9], LarKC[10] and SOA4ALL[11] among others and Ontotext AD[12]. OWLIM has been used in several domains and sectors as a basic data integration platform for heterogeneous data. The domains include telecommunications, life sciences and the publishing sector. OWLIM is implemented in Java and comes in three flavours:

- **OWLIM-Lite**: It is the new instalment of the OWLIM version previously known as SwiftOWLIM. It is an *in-memory semantic data repository* that provides efficient and relatively fast services for storing, querying and inferring RDF-based data. Reasoning is performed entirely in memory and is based on TRREE, the *Triple Reasoning and Rule Entailment Engine* while query processing and answering is provided by Sesame (see section 3.1 for a description of Sesame). OWLIM-Lite implements the Sesame SAIL interface thus allowing Sesame to access the reasoning and storage services provided by OWLIM. OWLIM-Lite was designed for applications that require management of medium-size data volumes which roughly speaking relates to amounts of about 100M RDF triples. OWLIM-Lite is distributed free of charge.
- **OWLIM-Standard Edition (SE)**: This version uses a scalable, file-based indexing mechanism and several (query and storage) optimization techniques to provide efficient query answering and storage functionality. As such it is capable of handling tens of billions RDF statements in commodity hardware and billions of explicit triples on desktop hardware. It also uses

[7] http://www.sekt-project.com/
[8] http://www.tao-project.eu/
[9] http://www.tripcom.org/
[10] http://www.larkc.eu/
[11] http://www.soa4all.eu/
[12] http://www.ontotext.com/

optimization techniques for *owl:sameAs* reasoning, query answering and retraction (deletion) of data (assertions). OWLIM-SE is distributed free for research and evaluation and under a commercial license for development purposes. Another important feature of this version is its hybrid approach to query answering that combines SPARQL 1.1 with full-text search, geo-spatial constraints and ranking of query results. As with OWLIM-Lite this version implements Sesame's SAIL interface.

- **OWLIM-Enterprise Edition (EE)**: This is the replication cluster version of OWLIM and is based on OWLIM-SE. It is distributed free for research and evaluation and under a commercial license for development purposes.

For details about the differences between the three versions the interested reader is referred to (12) and the OWLIM Website[13].

3.3.1 Reasoning in OWLIM

Reasoning in all three versions of OWLIM is based on a process called *total materialization*. Total materialization refers to the process of computing the full closure of the data in a single, atomic process which in the case of OWLIM happens at triple loading time and whenever a new triple is added to the repository. Materialization in OWLIM is a rule-based process that uses R-entailment (7) rules for deriving knowledge and has the advantage of making query processing/answering more efficient because no inference is required in order to retrieve knowledge from the repository. The main disadvantage of this process is that loading triples requires more time. The component of OWLIM that implements reasoning is called OWLIM TRREE, *Triple Reasoning and Rule Entailment Engine*.

OWLIM supports inference based on several rule sets with different levels of complexity. This allows users to compute different type of knowledge from the same underlying repository. OWLIM supports reasoning based both on standard and custom rule sets. The standard rules set natively supported by the system include:

- *RDFS* (13), using entailment rules from the RDFS schema language but without data type reasoning.
- *OWL-Horst* (7) without data type inference.
- *OWL-Max*, a combination of RDFS and a fragment of OWL-Lite (14) that can be captured in rules, i.e. inferring functional and inverse functional properties, all-different, subclass by enumeration and union, min/max cardinality constraints, etc.
- OWL 2 RL: The OWL 2 profile that can be implemented using rule engines (15).
- OWL 2 QL: In addition to this OWLIM-SE supports reasoning with OWL 2 QL (15).
- In addition to these (standard) rule sets OWLIM also supports reasoning based on user-defined, i.e. custom, rule sets which can be newly defined rules of inference or an extension/modification of the standard rule sets mentioned above. This enables users to tune the system for optimal expressiveness and performance depending on the requirements of the user application.

[13] http://www.ontotext.com/owlim

Although the reasoning paradigm is the same for the three versions of OWLIM the approaches used in OWLIM-Lite and OWLIM-SE/EE are different due to their mechanism for storing data (see next sections for a description of the data storage and management approach used in OWLIM). In the former reasoning and query evaluation is performed in memory while on the latter both are performed over a persistent storage layer.

3.3.2 Query Evaluation

Query evaluation in OWLIM is performed using Sesame's functionality and the current version (as of October 2011) supports SPARQL1.1. In addition to this, OWLIM implements a series of techniques for query evaluation that are relevant for different scenarios. These are *ranking of results*, *full-text search* and *RDF priming*. In the following we briefly describe each of them. For more details of these features the interested reader is referred to (12).

3.3.3 Data Storage and Management

Data in OWLIM is stored using two different mechanisms depending on the version used. *OWLIM-Lite stores data in memory* while the other two versions of the system use a *disk-based approach for storing, reasoning and query answering*. Adding of data to the knowledge base is handled in the same way in the three versions, i.e. upon insertion of new data OWLIM computes the statements entailed by the combination of the newly added triples and the existing data. However, deletion is handled differently in OWLIM-Lite and OWLIM-SE/EE. Deletion of data in OWLIM-Lite is done by deleting the previously generated closure and re-computing it without the statement(s) that were deleted. If deletions happen very often on a given application this is a factor to take into account when choosing this implementation of OWLIM as it is a rather inefficient approach and does not scale well. On the other hand, OWLIM-SE/EE updates the full closure incrementally by a combination of forward and backward chaining to determine which sentences must be deleted as a consequence of a deletion. This approach allows OWLIM to scale well even when data changes are frequent. Additionally, OWLIM-SE implements a notification system that allows users to get notifications when certain RDF statements are inserted/deleted. As in the case of classical RDBMS notifications are useful for applications that implement reactive behaviour.

3.4 AllegroGraph

AllegroGraph[14] is a high-performance, persistent RDF (graph) database that supports storing, querying and reasoning over RDF-based data. More generally speaking AllegroGraph is a graph database for managing graph-based data such as for example, RDF data, social networks and, in general, any kind of data that can be structured as a graph. It was developed by Franz Inc. and is distributed as a closed-source application under a commercial licence. At the time of writing this report the latest version of the system is 4.3.

[14] http://www.franz.com/agraph/

3.4.1 Architecture and Data Storage

AllegroGraph is based on a layered architecture that exposes a RESTFul protocol to enable access to clients. Figure 1 depicts the system architecture. For details of the several components the interested reader is referred to the online documentation at the AllegroGraph Website (see [14]).

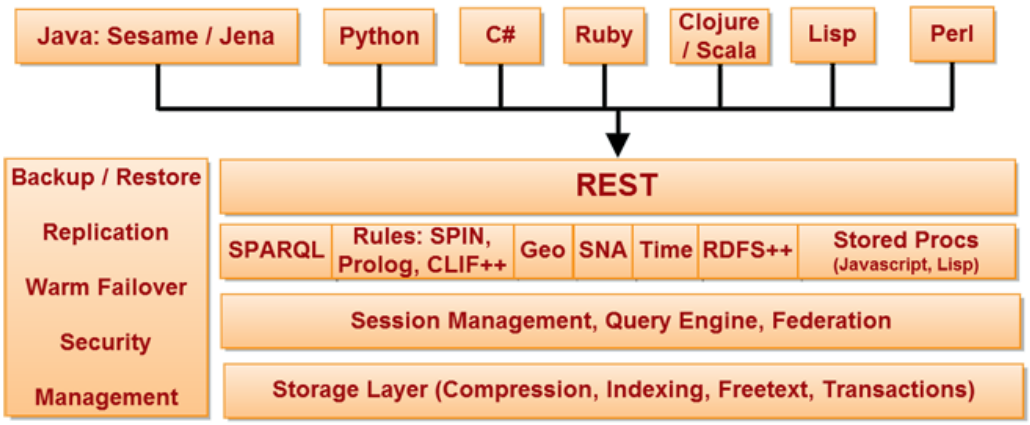

Figure 1: AllegroGraph Architecture (taken from [14])

AllegroGraph supports storage and reasoning over *RDF named graphs*, also known as *quads*, i.e. RDF triples extended with a fourth component which sometimes is referred to as the triple's ID. This gives AllegroGraph the possibility to handle different types of RDF-based metadata such as provenance information. AllegroGraph is able to handle *data stored either locally*, using either a *disk-based storage mechanism* or an *in-memory approach*, in *remote triple stores* or in *federated triple stores*. Federation gives AllegroGraph the possibility to scale to large amount of data.

AllegroGRaph implements a form of materialization called *Dynamic Materialization* which overcomes one of the disadvantages of using complete materialization, namely the high cost of maintaining the closure when updates to the knowledge base need to be performed. Simply stated, AllegroGraph does not perform materialization of data but rather optimizes SPARQL and Prolog queries. Dynamic Materialization enables AllegroGRaph to deviate from the traditional workflow implemented by many RDF stores (load triples, materialize, answer queries) and provide *linear time loading of triples*, access to query and reasoning capabilities at load time. Dynamic Materialization also enables AllegroGraph to overcome the scalability performance barrier that arises when large amount of data is loaded into the repository and materialization is performed at loading time, e.g. due to the fact that *rdfs:subClass* can be very demanding when the number of *rdfs:subClass* relations grows.

3.4.2 Reasoning and Query Answering

The system supports reasoning over RDFS++, a non-standard extension of RDFS that includes the following OWL constructs: *owl:sameAs*, *owl:inverseOf*, *owl:transitiveProperty*, *owl:hasValue*, *owl:someValuesFrom*, *owl:allValuesFrom* and

owl:oneOf. In addition to standard RDFS and non-standard RDFS++ reasoning AllegroGraph uses a Prolog engine to provide powerful reasoning functionality to reason about high-level concepts built based on complex rules and numerical processing.

Query answering in AllegroGraph is implemented by its *Twinql* SPARQL 1.1 query engine, which allows for answering queries over *RDF named graphs*. In addition to supporting standard SPARQL, queries in AllegroGraph can be defined in declarative form and then answered by using a Prolog engine. This approach enables a more powerful query answering mechanism alternative to standard SPARQL query processing.

3.4.3 Additional Features

A distinctive feature of AllegroGraph that sets it apart from other semantic repositories is its support for *geo-spatial data*, *temporal data* and *social network analysis*.

- *Geo-spatial reasoning* refers to reasoning over geo-spatial data. In particular, AllegroGraph supports reasoning over a more general type of two-dimensional data, namely, any coordinate system defined on a flat plane or a sphere.
- AllegroGraph also supports reasoning over temporal data, including time points, time intervals and datetime data.
- Finally, the graph-oriented database approach of AllegroGraph enables it to support reasoning about social interactions (or any kind of interactions that can be modelled as a network). It is useful for example for solving problems like computing the shortest path between two entities and the degrees of connectivity between two entities in a social network.

In addition to these features AllegroGraph provides many of the features desired in this type of systems. These include triple level security mechanisms, the possibility to integrate AllegroGraph within applications through the use of multiple programming interfaces, e.g. C#, Ruby, Perl, Java, Lisp and JavaScript, the possibility to deploy AllegroGraph in the cloud through its cloud-based deployment mechanism using Amazon EC2, a RESTFul protocol that clients written in several languages can use to access the system's functionality, etc.

3.5 BigData

BigData [15] is a scalable solution for persistent data storage and distributed computing that includes a *high-performance RDF database* with support for *standard RDFS reasoning* and *limited OWL reasoning* capabilities. It was developed by Systap LLC and it is distributed freely as an open-source application under GNU General Public Licence v2. BigData is also released with an evaluation/research license and supports the possibility of obtaining a commercial licence for situations where BigData needs to be embedded into another system.

3.5.1 Architecture and Data Storage

[15] http://www.systap.com/bigdata.htm

BigData provides three mechanisms for storing RDF data. The first approach stores RDF data as triples following the standard RDF data model. The second approach uses *Named Graphs* or *quads* to store data. The third and last method, called *statement-level provenance*, stores data as *triples associated with provenance information* which is technically achieved by associating an ID with each triple. Using such an ID it is possible to refer to any triple and add additional information to the triple description.

For details of the system architecture and a description of the main technologies used in the entire platform, not only the RDF database, the interested reader is referred to the online documentation found at the system's website [15].

3.5.2 Reasoning and Query Answering

The BigData RDF database supports reasoning over RDFS data and OWL Lite ontologies. It uses a hybrid approach for computing the RDFS closure that combines the so-called *eager closure approach*, a method in which the RDFS closure is computed at triple loading time and, a *lazy closure approach* or *materialization approach*, in which the closure is computed (materialized) at query time. In BigData, some of the entailment rules are applied at loading time while others are applied at query time. The idea behind this approach is to benefit from the advantages of both approaches. Although it has been reported that BigData supports the definition of custom rules for reasoning, e.g. to support social network analysis, no official documentation in this regards has been found that details such a feature.

For query answering BigData supports *distributed query processing* and *relies on Sesame 2 for SPARQL query processing* although the query evaluation process is modified to make it more efficient. More specifically, BigData uses a native rule model for evaluating SPARQL queries. Queries are parsed by Sesame 2 and then fed to BigData's SAIL implementation for transformation into the native rule model. The rule model of BigData supports conjunctive queries, filters and optional joins. As an important note concerning query answering in BigData, it has been reported in some benchmarks that BigData does not support the full set of SPARQL features.

3.6 Other RDF-based Semantic Repositories

This section gives an overview of other, less popular systems (although the Jena framework is perhaps the most popular framework for Java-based RDF processing), approaches and frameworks for storing, querying and reasoning over RDF-based data. We decided to report these systems within the same section due to the lack of documentation and third-parties benchmark results found in the literature in comparison to other mainstream systems such as OWLIM, Virtuoso, Sesame and AllegroGraph.

3.6.1 Jena RDF Framework

Jena [16] is a Java-based, open source Framework for building semantic-based applications that provides a series of API for handling RDF-based data, e.g. RDFS and

[16] http://jena.sourceforge.net/

OWL, querying such data through the SPARQL standard and reasoning over such data. It was developed in conjunction with the HP Labs Semantic Web Research [17].

The framework comprises the following technologies:

- The Jena RDF API, a *Java API for programming with RDF* that includes an RDF model, parsers, etc.
- ARQ, a *SPARQL query engine* for Jena.
- Jena SDB [18], a *persistence RDF storage* engine that uses a SQL database to store RDF data and to provide support for transactions. Jena SDB can be deployed on top of existing databases and supports many of the state-of-the-art RDBM Systems. From the practical point of view SDB can be used both within a Java-based application through the provided API and as a standalone application. In addition, SDB can be paired with the Joseki SPARQL engine to provide a SPARQL endpoint over HTTP. Query answering is based on ARQ.
- Jena TDB [19], a component of Jena that provides support for *large-scale persistent storage* and *query of RDF data*. Jena TDB stores RDF data natively rather than relying on a relational database backend. It does not support transactional database access. Query processing is based on ARQ and supports SPARQL. The latest version at the moment of writing this report is 0.8.10.
- Joseki, a *SPARQL query server* for Jena that uses the SPARQL protocol.

3.6.2 RDFStore

RDFStore[20] is a toolkit written in Perl/C for processing, storing and retrieving RDF data. Data in RDFStore can be stored using a memory-based or disk-based approach and can be retrieved locally from memory and disk and remotely through HTTP. It was designed to support RDF contexts (quads for provenance), typed and multi-lingual literals, reification and blank nodes.

The system implements RDQL[21] (*RDF Data Query Language*), which allows for querying RDF data using standard DB interfaces such as JDBC, ODBC and DBI.

3.6.3 D2R

D2R [22] is an open source application for publishing relational data, i.e. data stored and managed by an RDBMS, on the Semantic Web. The D2R server acts as a wrapper around the contents of an RDBMS and provides services for accessing its data through the SPARQL protocol. SPARQL queries are translated on the fly into SQL queries that are then evaluated using the query processing facilities of the underlying RDBMS. To achieve this the application transforms the contents of a relational database into RDF format so that the data stored in RDBMSs can be exposed as

[17] http://www.hpl.hp.com/semweb/
[18] http://openjena.org/wiki/SDB
[19] http://openjena.org/wiki/TDB
[20] http://rdfstore.sourceforge.net/
[21] http://www.w3.org/Submission/RDQL/
[22] http://www4.wiwiss.fu-berlin.de/bizer/d2rmap/D2Rmap.htm

Linked Data [23] without the need to replicate the data in a specialized RDF storage system. The transformation rules are customizable. One of the benefits of using such an approach for accessing data is that applications can access (browse and search) data stored in legacy RDBMSs with relative ease of implementation.

This type of approach to exposing relational data as RDF is sometimes referred to as a *bottom up approach* and refers to the process of transforming or mapping data stored in traditional databases into schema and instance information expressed in RDF. Similar services and tools have been developed[24].

D2R is a Java-based application that is compatible with Oracle, MySQL, PostgreSQL, Microsoft SQL Server, and any SQL-92 compatible database server.

3.7 Discussion and Relevance for INTEGRATE

The semantic repositories described in the previous sections differ in a number of important aspects. First of all, they differ on the technology they use for storing RDF data. Some of them store RDF data in some native structure while others rely on traditional database technology. Second of all, they also differ on the type of services they provide to the applications built on top of them. Some of them provide basic data storage capabilities while others implement a full-fledge data management system for RDF data that provide services such as inference over RDF data, query answering for different query formalisms and data clustering and federation. Semantic repositories also differ in terms of the license under which they are released, some being distributed as open source applications while others offer both free closed-source and commercial licenses.

From the survey presented above four basic classes of semantic repositories can be identified w.r.t. how they store data:

- Systems that store RDF data natively, i.e. by implementing a computational model of the RDF data model.
- Systems that map relational data to RDF data. These translate SPARQL queries into queries that are evaluated on relational databases and provide tools and applications for mapping the content of a relational database to RDF thus allowing to expose relational data as RDF. An example of such a system is the D2R Server.
- Systems that wrap SPARQL around existing databases and data sources. This is a common practice for exposing data stored in legacy systems (databases and other data sources) through SPARQL endpoints. The advantage of storing RDF data in RDBM systems is the possibility to leverage database technology to improve the efficiency of the overall system by benefiting from query and storage optimization techniques.
- Systems that offer a combination of these approaches, for example the OpenLink Virtuoso Universal Server gives the option to store data using a native approach or by saving RDF data into relational databases.

[23] http://linkeddata.org/
[24] http://www.w3.org/wiki/RdfAndSql

In addition the following features are identified as desirable in a semantic repository for RDF-based data:

- **Lightweight**: This is important in order to facilitate its use on single machines in order to reduce deployment costs.
- **Abstraction**: Independent of the data source and/or the underlying data storage technology used for storing the data. This enables easy reuse/integration of legacy database systems and allows for leveraging database technology
- **Efficient query processing**: Many applications both in the Semantic Web and in other domains such as the medical and life science domains require almost real time access to data through user queries. This calls for efficient methods to process large number of queries from multiple users in very short time.
- **Scalable**: semantic repositories should be scalable not only in terms of the amount of data they are able to handle efficiently but also in terms of the number of simultaneous users (queries) they can handle. The Web and other domains contain large amount of (semi)-structured data that changes and grows very rapidly. This calls for scalable solutions to store and process data.
- **Full-text search**: Many applications need to handle information stored in text form. For this efficient full-text search services are required.
- **Provenance and context information**: Ideally, semantic repositories should offer support for representing and reasoning with provenance and context information.
- **Reusable**: Easy to interface and integrate with existent tools and systems specially legacy systems.
- **Flexible deployment:** Possibility to distribute as part of a cloud-based, grid-based and cluster-based computing approaches in order to reduce costs of deployment and gain access to scalable computing facilities.

Given the plethora of systems available today and their differences in combination with application requirements and the lack of a standard procedure for characterizing semantic repositories choosing a candidate repository for a given application scenario is not an easy task. In this respect, some of the factors that need to be considered when choosing a repository solution are the following:

- The type of queries that are characteristic of the user scenario. This requires to assess the complexity of queries in terms of the number of constraints and the relations between (un)known variables.
- Frequency with which each type of query is posed (application-dependent factor).
- Type of user interaction. For example, very interactive user sessions requires sub-second response times for query answering (application-dependent factor).
- Size of data (application-dependent factor).
- Type of reasoning required. This can be determined based on the expressiveness of the ontological language (e.g. sub-properties, sub-classes, transitive properties, etc.), which in turn depends on the kind of queries the application needs to handle.

Semantic repositories are relevant for INTEGRATE as they provide one of the basic services required for implementing semantic-based data integration solutions. These

solutions call for methods for managing RDF-based information and for providing query and reasoning capabilities. The core data model being developed in INTEGRATE together with data annotated using such data model call for a storage and query solution able to accommodate the type of reasoning required by the different uses cases.

4 Automated Reasoning in the Semantic Web

One of the pillars in the technology stack that is required to help fulfill the Semantic Web vision is ability to be able to use the knowledge of a given domain, which is formally described by ontologies, to infer implicitly stated knowledge from the explicitly represented information. In the Semantic Web jargon this process is referred to as *semantic reasoning*.

Over the past several years the Semantic Web community have proposed several approaches, techniques and tools for efficient reasoning over ontological data expressed in various ontology representation languages. More recently and due to the exponential increase in the volume of information published on the Web the emphasis has shifted to designing efficient methods for scalable reasoning.

Two broad categories according to which most reasoning algorithms and tools can be categorized are *full-fledged reasoning platforms* and *classifiers*. The first refers to reasoning systems that are able to deal with description logics-based knowledge bases (ontologies) of varying degrees of expressiveness and that provide a wide range of description logics reasoning services such as subsumption checks, classification of concepts, instance checking, etc.. These systems are typically designed to be used with any kind of ontology that is expressed in the right formalism. Two commonly used logical reasoning strategies are tableaux-based methods and rule-based methods. Examples of these systems are Pellet, RacerPro and FaCT++. The other type of reasoning systems, *classifiers*, are specialized reasoning algorithms that exploit the structural characteristics of certain type of ontologies in order to provide more efficient reasoning services at the expenses of reduced number of reasoning services. They incorporate specialized optimization techniques and are based on non-standard reasoning strategies as opposed to those used by full-fledged reasoning systems. This trend started since the development of a new polynomial-time classifier for the family of Description Logics $\mathcal{EL}$ (10). Rather than developing general algorithms for reasoning with an entire family of ontologies of a given ontological language this trend seems to be aimed at focusing on specific, ontology-custom properties and developing tractable algorithms for these ontologies. The idea is to devise optimizations tailored at the specific structures of ontologies such as that exhibited by many medical and life science ontologies, for instance SNOMED CT.

This chapter is organized as follows: Sections 4.1 to 4.10 gives an overview of several state-of-the-art reasoners and classifiers developed in the Semantic Web community and applied in different application domains including the Semantic Web and the Life Sciences. Then, Section 4.11 concludes the chapter with a discussion on how these technologies relate to the project and a description of the most important factors to take into consideration when choosing the right reasoner for the applications envisioned in the project.

4.1 Pellet

Pellet (16) is an OWL reasoner developed by Clark & Parsia[25] that provides reasoning services for OWL ontologies with support for the latest revision of the Ontology Web

[25] http://clarkparsia.com/pellet/

Language, OWL 2.0. It is distributed under the terms of the AGPL v3 license for open source applications and under alternative license terms for proprietary, commercial closed-source applications. Pellet is regarded as the first reasoner to support OWL-DL (14) and over the years it has become the defacto standard for implementing applications that require reasoning over OWL ontologies.

In terms of usability Pellet is implemented in Java and its reasoning services can be accessed through its own Java API or by using one of the many bindings to common programming toolkits such as Jena and the OWL API [26]. Pellet has also been integrated in the Protege ontology editor [27]. Additionally, Pellet implements the DIG interface which allows users to access the reasoner's services through HTTP requests.

4.1.1 Architecture

Pellet's architecture was designed with *extensibility* in mind and contains four main components (see Figure 2, taken from (16)). The first and most important is the *tableaux-based reasoning component* that implements consistency. A secondary related component is a *data type reasoner* that enables Pellet to reason about both built-in and derived XML Schema data types. The second important component is the *knowledge base* itself for which Pellet follows the Description Logics conventions and divides it into a *terminological part* (Tbox) and an *assertional part* (Abox). The third component, *Species Validation and Ontology Repair*, is responsible for validating and repairing ontologies to ensure that they are syntactically valid w.r.t. OWL-DL. The fourth component in the architecture is the *Abox Query Engine* that is responsible for processing conjunctive queries expressed in SPARQL or RDQL, solving them by using the reasoning component. The rest of the components include several parsers and APIs for accessing the services. For details of the architecture the interested reader is referred to (16).

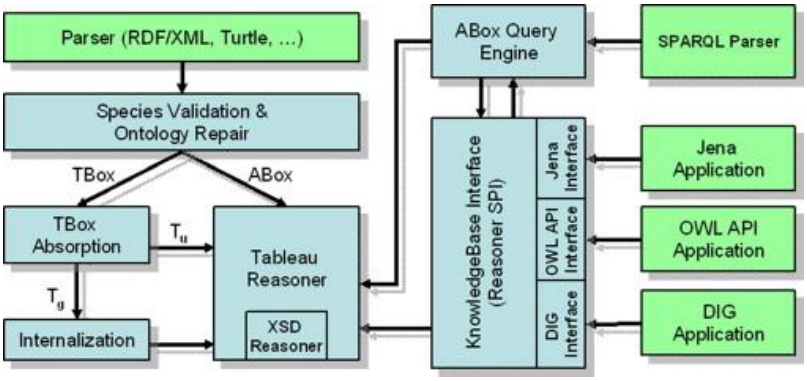

Figure 2 Overview of Pellet's architecture

4.1.2 Reasoning in Pellet

Reasoning in Pellet is based on *tableau methods*. This is in contrast with other reasoners that use *rule-based* and *hybrid* mechanisms. HermiT's (see section 4.5) reasoning is based on a similar approach named *hypertableaux*. In order to the derive

[26] http://owlapi.sourceforge.net/
[27] http://protege.stanford.edu/

the truth value of a logical formula tableaux-based methods try to build a model of the formula by exploiting its structure and by applying a series expansion rules until no more rules can be applied or, a contradiction is found. Pellet is able to reason over OWL-DL ontologies, a syntactic variant of the DL $\mathcal{SHOIN}(\mathcal{D})$.

The tableaux method implemented in Pellet was designed with extensibility in mind and is based on the application of expansion strategies of which several are used depending on the characteristics of the knowledge base. This design also allows for adding and testing new expansion strategies. Pellet implements a dynamic selection procedure for selecting the expansion strategy to use. In particular, Pellet includes the following built-in strategies:

- $\mathcal{SHOIN}$ strategy,
- Empty $\mathcal{SHN}$ strategy,
- $\mathcal{SHIN}$ strategy,
- $\mathcal{SHON}$ strategy and
- E-connection strategy. This has the advantage of being able to exploit the specific characteristics of an ontology to provide sound and complete reasoning. Pellet implements a method for reasoning with multiple ontologies based on E-connections.

As an OWL-DL reasoner Pellet implements all the basic DL reasoning services including *consistency checking, concept satisfiability*, *classification* and *realization*. Consistency checking, the process of checking whether a (set of) logical formulae is consistent, is the basic reasoning service upon which all other services are implemented. In addition, Pellet supports other non-standard services that have been identified as important for practical applications. One such service is the *explanation and debugging of ontologies*. The current version of Pellet (as of the time of writing this report) supports also reasoning over OWL 2 EL (15).

Pellet also provides support for *non-monotonic reasoning*, a form of reasoning that is capable of capturing several forms of common-sense and database reasoning, through the implementation of a non-monotonic language called $\mathcal{ALCK}$. This feature allows Pellet's users to *"turn on"* the CWA on demand at query time by means of an extension to the SPARQL language. Knowledge bases can also include, in a restricted form, a non-monotonic rule that makes use of the K operator in $\mathcal{ALCK}$ to represent non-monotonic information. Using this feature Pellet should be able to treat knowledge bases as database-like repositories.

Moreover, Pellet provides limited support for *rule-based reasoning*. This is achieved by implementing a decidable fragment of SWRL (*Semantic Web Rule Language* (17)), called $\mathcal{AL}$-Log that combines Datalog rules with DL knowledge bases allowing DL concepts to be used in the body of the rules. Pellet also provides support for reasoning with standard and user-defined XML Schema-based data types, nominals and over individuals.

4.1.3 Query Answering

Pellet supports *reasoning with individuals* (Abox reasoning) through conjunctive queries over assertions. Such queries can be issued using any of the languages supported by Pellet, namely SPARQL (18), RDQL (19) and KIF.

As with the tableau method, query answering in Pellet is implemented using a series of sub-query engines tailored to specific characteristics of the queries such as queries with no variables, queries with undistinguished variables, queries with one undistinguished variable, etc.

4.1.4 Debugging and Repair of Ontologies

An important non-standard service provided by Pellet to support Knowledge Engineering and Management tasks is ontology analysis and repair, a.k.a. *debugging of ontologies*. Pellet is able to perform a syntactic analysis of an OWL Full ontology to detect and repair fragments of the ontology that logically corresponds to OWL DL.

Another non-standard reasoning service implemented by Pellet is *axiom pinpointing and debugging*. Axiom pinpointing refers to the process of finding justifications for any arbitrary entailment. Using this technique Pellet is able to identify or pinpoint the source of inconsistencies in an ontology and extract that part from the ontology. This service is important for the design, debugging and evolution of ontologies.

4.2 Racer Pro

RACER stands for Renamed *ABox and Concept Expression Reasoner* and is a Description Logics reasoning system with support for terminological and assertional reasoning over knowledge bases specified in the DL $\mathcal{SHIQ}(\mathcal{D})$, i.e. $\mathcal{SHIQ}$ with concrete domains, extended with simple data types. In Semantic Web jargon RACER supports OWL Lite and OWL DL. In addition, RACER implements a decision procedure for satisfiability in Modal Logics. It was the first DL reasoning system to support a very expressive logic. RACER PRO is the commercial version of RACER. The system is maintained and released by Racer Systems GmbH & Co. KG [28].

RACER is distributed free for research purposes while RACER PRO is the commercial version of the system. It can be accessed programmatically through Java, C and C++ APIs and, through TCP/IP.

4.2.1 Reasoning in RACER

Reasoning in RACER (Pro) is based on a *tableau-calculi* that implements the reasoner's core reasoning service, namely Abox consistency. In addition to the standard reasoning services provided by state-of-the-art DL systems RACER implements a series of non-standard services that have been identified as important in many practical applications. These include services to *retrieve the list of concepts and individuals* in the knowledge base, *retrieval of the set of roles and sub-roles*, etc. The full list of services can be found at the reasoner's website [29].

[28] http://www.racer-systems.com
[29] http://www.racer-systems.com/products/racerpro/index.phtml

A distinctive feature of Racer is its support for *reasoning over multiple Tboxes and Aboxes*. RACER also supports knowledge base management activities by providing services to add and retract axioms from (possibly) multiple Tboxes and Aboxes. Moreover, RACER provides support for algebraic reasoning including *concrete domains over integers*, *min/max cardinality restrictions over integers* and *(in)equalities over strings*.

The architecture of Racer reflects the advances in optimization techniques for DL systems. This is manifested in the fact that Racer implements a series of optimizations for improving the performance and efficiency of reasoning.

4.2.2 RACER as a Semantic Web Reasoner

In addition to being seen as a Description Logics reasoner RACER can also be seen as a Semantic Web reasoner. As such RACER *supports reasoning over OWL Lite and OWL DL ontologies* with approximations for nominals in class expressions. RACER is also *capable of reasoning over certain extensions of OWL* such as OWL-E and, of handling rules with its implementation of SWRL (17). From this point of view RACER allows for:

- Checking the consistency of OWL ontologies.
- Computing and querying the specialization/generalization hierarchy induced by the declarations in the ontology.
- Finding synonyms for resources 9both for classes and instances).
- Retrieving extensional information by means of OWL-QL.
- Information retrieval based on incremental query answering.
- Accessing reasoning services using the DIG interface through HTTP.

A limitation of the system, however, is the *lack of support for user-defined XML data types*.

4.2.3 Query Answering in RACER

RACER support queries written in nRQL (*new Racer Query Language*), which supports *negation as failure*, *numeric constraints wrt. attribute values of different individuals*, *substring properties between string attributes*, etc. RACER's implementation of nRQL is the basis for implementing many of the features of OWL-QL.

4.3 FaCT++

FaCT++[30] (20) is a DL *tableau-based reasoning platform* that supports reasoning over $\mathcal{SHOIQ}$ DL knowledge bases and, in its latest version provides limited reasoning support for OWL 2 ($\mathcal{SROIQ}$) knowledge bases. In addition to this, FaCT++ provides support for reasoning with XML Schema data types although it lacks support for built-in primitive types. FaCT++ was originally designed as a reasoning platform for experimenting with novel tableau methods and optimization techniques and for reasoning over ontologies that use inverse roles.

[30] http://owl.man.ac.uk/factplusplus/

It is an open source project distributed under GNU LGPL. It is available as a Protege plug-in and its services can be accessed through the DIG interface and the OWL API.

4.3.1 Reasoning in FaCT++

Reasoning in FaCT++ is based on a *tableau method* that is used for implementing the core reasoning service, namely KB satisfiability. The core component of the reasoner is a satisfiability checker that allows for checking the satisfiability of a set of logical formulae. Every other reasoning service is reduced to this. The basic workflow of the reasoner loads an ontology into the knowledge base and then classifies the ontology using the satisfiability component for deciding subsumption between pairs of concepts.

The reasoner applies several optimization techniques for enabling efficient reasoning over ontologies. Many of the optimizations are well-known in the DL community and implemented by several DL reasoning systems. These are the so-called *standard optimization techniques* (21). In addition to incorporating these techniques FaCT++ implements novel optimization techniques and heuristics. These are applied at different stages of the reasoning and ontology management process and include optimization strategies used while loading data to the reasoner, strategies for optimizing satisfiability checking and those used in the classification task geared towards reducing the number of subsumption tests. For details of these optimization techniques and how they are used in FaCT++ the reader is referred to (21) and (20) respectively.

Although FaCT++ is able to reason over OWL 2 ontologies ($\mathcal{SROIQ}$ DL) its reasoning capabilities are limited. Specifically, FaCT++ can not properly handle Top/Bottom Object and Data property semantics and has partial data type support; the only supported data types are *literal*, *string*, *anyURI*, *boolean*, *float*, *double*, *integer*, *int*, *dateTime* and *nonNegativeInteger*.

4.4 KAON2

KAON2[31] is a platform for managing and reasoning over ontologies specified according to OWL DL, SWRL and F-Logic. It was developed by the *University of Manchester*, the *University of Karlsruhe* and the *Information Process Engineering (IPE) at the Research Center for Information Technologies (FZI)*. It is a continuation of the work started with the KAON system that aims at supporting reasoning over more expressive knowledge representation formalisms.

KAON2 is distributed free of charge for non-commercial purposes and under a commercial license with the name of OntoBroker OWL. The system is implemented in Java and its services can be accessed through the DIG interface. No support for the newer OWL API interface has been reported so far.

4.4.1 Reasoning in KAON2

[31] http://kaon2.semanticweb.org/

Reasoning in KAON2 is based on the work of Boris Motik (22). More specifically, KAON2 reasons by reducing a $\mathcal{SHIQ}(\mathcal{D})$ KB to a disjunctive datalog program and applying well-known and tested deductive databases techniques and resolution-based methods. In this sense KAON2 implements a reasoning paradigm that differs from the tableau-based method used in many DL reasoners.

In terms of expressiveness KAON2 supports full reasoning over OWL Lite and partial reasoning over OWL-DL ontologies as it can reason over the $\mathcal{SHIQ}(\mathcal{D})$ fragment of OWL-DL, the fragment that does not include nominals. The system is capable of answering SPARQL conjunctive queries with non-distinguished variables although full support for SPARQL reasoning is not implemented.

4.5 HermiT

HermiT [32] (23) is a Description Logics reasoning system that is based on a novel technique for reasoning with Description Logics knowledge bases called *hypertableau calculi*. HermiT was developed as part of the efforts made in order to provide scalable reasoning methods for large and complex ontologies such as the ones encountered in the Life Sciences. HermiT is able to reason over OWL Lite and OWL 2 ($\mathcal{SROIQ}$ DL) ontologies and provides support for OWL 2 data types.

In terms of usability, HermiT is an open source project written in Java and distributed under the GNU LPGL[33]. Its services can be accessed through command line or programmatically from other applications using the OWL API. It can process ontologies serialized using RDF/XML syntax, the OWL Functional Syntax, KRSS and OBO.

4.5.1 Reasoning in HermiT

Reasoning in HermiT is done using a *hypertableau calculi* (24). HermiT implements a series of optimizations to address two of the most important factors affecting the efficiency and performance of tableau-based reasoning methods. First, the hypertableau method implemented by HermiT addresses the issue of large number of models by using several optimizations that aim at reducing the number of candidate models of a given set of logical formulae. The second performance factor, the size of the candidate models, is addressed by implementing a techniques known as *anywhere blocking*, an optimization strategy that allows HermiT to limit the size of the models constructed by the hypertableau method.

Another reasoning aspect addressed by HermiT is the handling of nominals. HermiT implements a novel technique to improve the efficiency and performance of reasoning when nominals are used in the presence of number restrictions and inverse roles. In addition to these optimizations HermiT also optimizes the ontology classification task by implementing optimization strategies tailored to the way the hypertableau algorithm constructs models. The latest version of HermiT (v1.3.4) includes *support for incremental Abox reasoning*.

[32] http://hermit-reasoner.com/
[33] http://www.gnu.org/licenses/lgpl.html

HermiT is able to reason over $\mathcal{SHIQ}$ DL knowledge bases therefore covering OWL Lite. In addition to this, work is still being done in order to support the DL $\mathcal{SHOIQ}$ which will enable HermiT to reasoner over OWL DL ontologies. HermiT also has support for *DL safe rules*.

A non-standard functionality implemented by HermiT is the support for reasoning with ontologies that contain *description graphs*. Description graphs are ontological constructs that extend the expressiveness of OWL and support the representation of structured objects, i.e. objects composed by several parts interconnected in arbitrary ways (25). At the time of writing (September 2011, version 1.3.5) HermiT is the only known DL reasoner capable of supporting this modelling construct.

4.6 OWLIM TRREE

OWLIM TRREE stands for *Triple Reasoning and Rule Entailment Engine* and is the reasoner at the core of the OWLIM family of high-performance semantic repositories. The TRREE is available in two different versions depending on the OWLIM repository used which in turn depends on the specific user needs. The TRREE included in OWLIM Lite is called *SwiftTRREE* and is characterized by its *in-memory reasoning and query processing capabilities*. The other version is called *BigTRREE* and is bundled with OWLIM SE. BigTRREE is the in-disk version of TRREE that is capable of delivering high-performance reasoning over data and information stored in the file system.

4.6.1 Reasoning in TRREE

TRREE is a *rule-based engine* that performs reasoning by *forward chaining* RDF entailment rules on RDF triples with variables. The reasoning paradigm followed by the engine is *total materialization*. TRREE supports the specification of *inequality expressions* between variables in the rules body and it supports *inconsistency checks*.

The format and semantics of the inference rules used by TRREE are a slight variation of the R-entailment rules proposed in (7). The reasoner has the ability to perform reasoning with respect to a customized set of inference rules, i.e. a user-defined rule set which must include the entailment rules to use in the forward-chaining process and the axiomatic triples. In addition to user-defined rules both OWLIM Lite and SE come bundled with a predefined set of inference rules that the TRREE can use. The set of entailment rules supported by default by the TRREE are enumerated in Section 3.3.1.

4.7 CEL

CEL (26), *Classifier for $\mathcal{EL}+$,* is an OWL 2 EL reasoner based on the *polynomial-time algorithm* for classifying $\mathcal{EL}+$ ontologies; technically speaking CEL is a *description logics classifier*. The language it supports, $\mathcal{EL}+$, is a knowledge representation formalism that extends the description logic $\mathcal{EL}$ with role inclusion and posses enough expressive power to represent the type of knowledge typically found in Life Science ontologies such as SNOMED CT. The *main reasoning task supported by CEL is ontology classification*.

CEL was developed by the Technical University of Dresden and is regarded as the first classification system for $\mathcal{EL}+$ ontologies. After its conception several other systems have been proposed (cf. Snorocket). It is distributed as an open source project[34] and its reasoning services can be accessed through Protege (as plug-in), programmatically through the OWL API and DIG interfaces and through a command line interface. The input syntax is based on an extension of KRSS and ontologies can be loaded from files or entered interactively through the command line. The interface is also able to accept goal-oriented queries aimed at retrieving the subsumption relationship between a pair of concepts. Its current version, v1.0 beta, includes support for *role domain and range constraints*, *role reflexivity*, more powerful *subsumption queries*, and a weak form of *incremental classification*.

4.7.1 Reasoning in CEL

The core reasoning algorithm implemented by CEL was first described in (27) and it is based on an adaptation of the algorithm implemented for the DL $\mathcal{EL}++$ (10). Since then the algorithm has undertaken a few modifications to improve its performance (26) (27). The algorithm enables CEL to compute the subsumption hierarchy induced by an $\mathcal{EL}+$ ontology in polynomial time and has been proven to be efficient in classifying real-world ontologies expressed in $\mathcal{EL}+$.

In addition to its basic reasoning service CEL supports other services that have been identified as important for the development and use of ontology-based applications. These, implemented in version 1.0, include the following (28):

- **Partial incremental classification** is a non-standard reasoning service that allows for classifying an ontology upon modification without the need of re-classifying the entire ontology.
- **Axiom pinpointing** refers to the task of computing the (set of) justifications of a given consequence.
- **Module extraction** refers to the task of extracting a subset of an ontology given a signature, i.e. a set of concepts and roles.

Other standard DL reasoning services are also supported by CEL, namely: *checking satisfiability of concepts*, *checking the consistency of an ontology*, *checking the equivalence between two concepts*, etc.

The CEL algorithm to classify an $\mathcal{EL}+$ ontology proceeds in two sequential steps. The first step is called a *normalization* and transforms an $\mathcal{EL}+$ ontology into a normal form applying a series of syntactic modifications in linear time. Normalization occurs at loading time. In the second step, called *mapping*, CEL computes two mappings to make implicit subsumption relationships explicit. Mappings are computed by applying a series of completion rules for which efficient rule selection and application strategies have been devised.

[34] http://lat.inf.tu-dresden.de/systems/cel/

4.8 Snorocket

Snorocket[35] (29) is a high-performance, Java-based implementation of the same algorithm for classifying $\mathcal{EL}+$ ontologies implemented by the CEL reasoner (26) (see Section 4.7). It is distributed as a Protege plug-in [36] and can be accessed through the OWL API. Input ontologies can be serialized in various formats including KRSS, the SNOMED CT distribution format and the OWL 2 Functional Syntax.

The development of Snorocket was motivated by the scalability issues shared by many of the classifiers and reasoners that were not able to handle large real-world ontologies in the Life Sciences domain such as the well-known SNOMED CT ontology. Putting Snorocket into context the reasoner is part of a family of reasoners and classifiers systems that are specifically optimized for a certain family of knowledge representation formalisms, in particular those based on Description Logics, that are typically used for representing ontologies used in the Life Sciences and the medical domain such as FMA, SNOMED CT, the Gene Ontology, etc.

The reasoner, technically speaking a *description logics classifier*, provides a cross-platform reasoning system that has at its core a polynomial-time classification algorithm. Its cross-platform characteristic means that it can be more easily adopted by developers of ontology-based applications and systems; this being a drawback of the initial implementation of the algorithm by the CEL classifier.

4.8.1 Reasoning in Snorocket

The core reasoning component is an optimized implementation of the polynomial-time classification algorithm developed for the DL $\mathcal{EL}+$ by (26). Snorocket provides, however, a more efficient implementation of the aforementioned algorithm in comparison to that provided by the CEL reasoner. Compared to CEL Snorocket is characterized by its Java-based implementation, exhibiting better performance and being licensed by IHTSDO (the *International Health Terminology Standards Development Organization*).

Snorocket *partially supports reasoning over OWL 2 EL ontologies* as it lacks support for handling ontologies that include definitions of properties domain and range, same individual relations and different individuals among others. For details about the knowledge representation constructs supported by Snorocket the reader is referred to (29). Snorocket also supports *partial incremental classification* which means that the reasoner is able to incrementally classify an ontology upon insertion of statements without the need of re-classifying the entire ontology but not upon data deletion.

4.9 TrOWL

TrOWL is *a tractable reasoning infrastructure for OWL 2* (30) that uses lightweight reasoners and language transformation techniques for providing tractable reasoning over various fragments of OWL 2. TrOWL aims at supporting users in the ontology development and management processes. It was developed by *Aberdeen University*. The stable version at the time of writing this report is v0.7.

[35] http://aehrc.com/hie/snorocket.html
[36] http://protegewiki.stanford.edu/wiki/Snorocket

More concretely, TrOWL is an interface that provides access to two reasoners, namely TrOWL Quill and TrOWL REL.

TrOWL Quill [37] supports reasoning over RDF-DL and OWL 2 QL ontologies. Quill stores OWL 2 QL ontologies in a database and thus is able to exploit the database query and indices technologies for efficient query processing and subsumption computation. The approach used in Quill is based on novel algorithms for query rewriting and ontology normalization. The reasoner provides standard reasoning services such as concept satisfiability, consistency checking and conjunctive query answering over OWL 2 QL ontologies. In addition, it is capable of semantically approximating more expressive ontologies using an OWL DL reasoner. It also supports negation as failure and fuzzy queries. TrOWL Quill is the engine that supports ONTOSEARCH2's search and query engine.

The second reasoner included in the TrOWL platform is REL[38]. REL supports reasoning over OWL 2 EL ontologies by implementing an optimized version of the algorithm for reasoning over $\mathcal{EL}+$ ontologies (10) (the one implemented by the CEL classifier) and is capable of supporting tractable TBox reasoning for OWL 2 EL ontologies. This reasoner is used as the basis for implementing the syntactic transformations used for transforming OWL 2 ontologies into specific fragments with desirable computational properties. REL also includes a conjunctive query engine for OWL EL ontologies. The reasoning services provided by REL are *Tbox classification* and *conjunctive query answering* over $\mathcal{EL}+$ ontologies. Together with Snorocket they represent two optimized implementations of the CEL algorithm.

TrOWL also provides support for *full DL reasoning* by relying upon existing DL reasoners such as Pellet and FaCT++; these underlying reasoners are incorporated into the TrOWL infrastructure as plug-ins. The idea behind this approach is to provide optimized reasoning services for different applications depending on the application requirements.

TrOWL uses syntactic and semantic language transformations for transforming OWL 2 ontologies to specific fragments. For example, for transforming OWL 2 ontologies into OWL 2 QL ontologies TrOWL makes use of a technique known as *Semantic Approximation* that transforms an OWL DL ontology into an OWL Lite ontology. More concretely this technique is characterized by the use of a reasoner in order to guarantee that every axiom in the approximated ontology is valid wrt. the source ontology. For transforming OWL 2 ontologies into OWL 2 EL TrOWL uses a *soundness-preserving approximate reasoning* method. This transformation is syntactic in comparison to the previous one. TrOWL also uses transformation strategies based on forgetting (31) (32).

[37] http://kt.abdn.ac.uk/wiki/Projects/Quill
[38] http://kt.abdn.ac.uk/wiki/Projects/REL

4.10 The Large Knowledge Collider (LarKC)

The Large Knowledge Collider (LarKC[39] (33)) is a FP7 European project whose aim is to develop a platform for large-scale semantic reasoning in the Web. Its main contribution is a *platform for massive, distributed, large-scale incomplete reasoning over Web data*. LarKC tackles scalability using a combination of different approaches that include reducing the reasoning search space by identifying and selecting relevant sources of data and information that can be used in the reasoning process, by transforming data from one format to another, by inferring new implicitly stated knowledge from the explicitly stated one and by using a meta-reasoning paradigm to decide whether more computations is needed. Each of these approaches uses techniques from different areas such as Cognitive Psychology, Machine Learning, Information Retrieval, etc.

In the broadest sense, reasoning in LarKC is achieved by executing a workflow of activities each of which tackles a separate aspect of the global reasoning process. More concretely, LarKC models reasoning in terms of five activities: *Identify*, *Select*, *Transform*, *Reason* and *Decide*. *Identify* is responsible for identifying sources of information relevant for the reasoning process. The *Select* task aims at selecting from all the data retrieved by the Identify task the subset of data and information that can be used in the reasoning process. The *Transform* task is responsible for making transformations to the data in order to accommodate different data formats and representation languages. The *Reason* task is where the actual reasoning happens. LarKC does not imposes a specific type of reasoning, which means that both logic-based and non-logic-based approaches can be implemented. Finally, the *Decide* task is where decisions are made in order to decide whether more time is required to produce an answer. The reasoning process in LarKC is triggered by a SPARQL query and ends when the expected answers are returned to the user.

LarKC implements the workflow mentioned above using a pluggable architecture of components (plug-ins). Each plug-in type implements one of the five tasks described before. Users are free to implement their own plug-ins and to build/reuse workflows in order to solve specific reasoning problems, i.e. answer specific SPARQL queries. User applications are built reusing and/or implementing new workflows which in turn can be based on newly developed plug-ins or existing ones. Besides providing support for building and executing workflows and plug-ins the LarKC platform also provides additional services required for implementing each of the five aspects of a reasoning process. These include parallelization strategies, a data layer that supports storage, retrieval and lightweight inference over large volumes of data and that automates the exchange of RDF data, caching of data, etc.

LarKC is implemented in Java and is distributed under an Apache 2.0 license [40]. It can be deployed in consumer machines as well as in a cluster configuration.

4.11 Discussion and Relevance for INTEGRATE

Reasoning systems can be characterized according to several characteristics. A characterization of reasoners is important to support users in choosing a reasoner for a given application scenario. Several characterizations have been proposed in the

[39] http://www.larkc.eu
[40] http://sourceforge.net/projects/larkc/

literature, see (34)(35)(36)(37) for an example of some of these works. From these characterization frameworks it is possible to identify several factors that need to be taken into account when deciding which reasoner to use. The following is a summary of the most important features to consider elaborated based on the aforementioned results:

- ***Required expressiveness***, e.g. RDFS, OWL fragments, OWL 2 Profile, etc.
- ***Reasoning services*** required, these include:
    - Standard (logic-based) description logics reasoning services such as *instance retrieval* (*conjunctive query answering*), *consistency check*, *subsumption reasoning*, *concept satisfiability*, *classification,* etc.
    - Non-standard services such as ontology management, evolution and debugging.
- ***Type of reasoning*** required such as for example *temporal reasoning*, *non-monotonic reasoning*, *geo-spatial reasoning*, *anytime reasoning*, *approximate reasoning*, *probabilistic reasoning*, etc.
- ***Correctness*** of the reasoner, i.e. soundness and completeness. It is important to notice that these properties depend on the type of application at hand. For example, in some application scenarios incomplete reasoning is acceptable if a small number of answers is returned in less time.
- ***Scalability*** of the reasoner both in terms of number of simultaneous users (queries) the reasoner is able to handle and the size of the data (ontologies, knowledge bases) the reasoner is able to reasone over; in the semantic Web this is typically measured in terms of RDF statements. If the specific application scenario requires large amount of data (large ontologies) then the scalability of the reasoner, i.e. the capacity of the reasoner to handle large volume of data efficiently is important. In this case, a selection criteria could be the ability of the reasoner to cache answers and/or to be able to distribute data across multiple compute nodes and distribute the process of answering queries. Another important feature to look for in a reasoning platform in this case would be the support for anytime or incremental reasoning, i.e. the ability to provide a trade-off between computational resources such as time or memory for some reasoning property such as completeness. If, on the other hand, the application is used by multiple users simultaneously then the ability of the reasoning platform to distribute the load is an important criterion to consider.
- ***Reasoning performance***, which canbe measured in terms of the resources required for carrying out each the reasoning services supported by the system.
- Ability to ***integrate with existing semantic repositories***
- The ***access mechanisms*** supported. For example through HTTP suing the DIG or OWLink interfaces.
- The ***query protocol*** supported. Given its wide-spread usage, SPARQL should be supported.

Table 1 summarizes some of the most salient characteristics of the reasoners included in this survey.

One of the conclusions drawn from this survey of reasoners is that there is no system that outperforms the rest in all the features presented above. Each of the reasoners considered in this survey have its strengths and weaknesses. For example, a known issue in KAON2 is its inability to handle large numbers of cardinality restrictions.

Snorocket does not support individuals, specification of domain and range constraints. A limitation of the CEL classifier is its lack of a proper interface and reasoning system around it that allows for easier integration into user applications and consequently widespread adoption by the community. A first step into making this reasoner more usable has been the implementation of the OWL API that eases the process of integrating the reasoner into ontology management tools such as Protege. Ultimately, the desicion of which system to use for a given application scenario depends very much on a carefull analysis of the aforemetioned factors and on a systematic evaluation of the performance of the system using data from the domain at hand. Also important is the type of reasoning task that is required from a reasoner. If, for example, the user application requires only classification services then a classifier such as CEL or Snorocket is preferred over a more general reasoner such as Pellet or FaCT++ due to their used of optmization techniques specifically tailored for certain types of ontologies, in this case life science ontologies. On the other hand, if more reasoning services are required such as consistency checking and instance retrieval (query answering) then more general reasoners are preferred.

For INTEGRATE the relevance of lightweight reasoners (classifiers) such as CEL and Snorocket is important as they have been shown to outperform other more general reasoners such as Pellet, RacerPro, etc. wrt. the classification tasks. Because these classifiers exploit the structure of certain types of ontologies they are able to use optimization techniques for classifying large ontologies such as the ones used in the Life Sciences. This is very important for INTEGRATE as the type of ontologies relevant for the project share many of the charactersitics of the ontologies for which these reasoners perform well. For details of the benchmarks and evaluation results the readeris referred to deliverable D2.2.

	Pellet	***FaCT++***	***RacerPro***	***HermiT***	***TRREE***	***KAON2***	***Snorocket***	***CEL***	***TrOWL***
Reasoning	Tableau	tableau	tableau	tableau	Rule-based	resolution	completion rules	Completion rules	DL reasoning
Sound	yes	yes	yes	yes	yes	yes	yes	yes	yes
Complete	yes	yes	yes	yes	yes	yes	N/A	N/A	yes
Rules	SWRL (AL-Log)	no	SWRL	SWRL (DL safe)	OWL-Horst OWL 2 QL OWL 2 RL OWL Max Custom rules	SWRL	no	no	No
Justifications	yes	no	yes	no	no	no	no	yes	no
Abox Reas.	yes	yes	yes	yes	yes	yes	no	No	yes
Expressive P.	SHOIN(D) SROIQ(D)	SHOIQ SROIQ(D)	SHIQ(D)	SROIQ SHOIQ(D)	OWL 2QL/RL OWL Horst OWL Max	SHIQ(D) F-Logic	EL+ (OWL2 EL)	EL+ (OWL2 EL)	RDF-DL OWL 2 QL/RL
Non-monotonic	yes	no	no	no	no	no	no	no	No
Queries	SPARQL RDQL		nRQL OWL QL		SPARQL	SPARQL			
API	OWLAPI Jena, DIG	OWLAPI DIG	OWLAPI OWLink DIG	OWLAPI		DIG			

Table 1 Reasoners characteristics

5 Ontology Mediation, Alignment and Merging

5.1 Introduction

Ontology mediation is the process concerned with *discovering, specifying and exploiting the semantic correspondences between ontologies*. A correspondence reflects a relationship between entities of two ontologies; the term entity is used for referring to the concepts, relations among these concepts and instances of the ontologies involved in the mediation process. In other words, aligning, mediating or matching two ontologies refers to the process of translating the terminology of one ontology into the terminology of a second (target) ontology in such a way that the translation respects or agrees with the target ontology's semantics. In the context of an ontology-based application three types of ontology mediation processes are typically required (38):

- **Ontology Mapping:** this refers to the process of *identifying and representing the semantic correspondences between two ontologies*. It deals mostly with the representation or specification of the mappings between ontologies. The process takes as input two ontologies and produces an ontology mapping, i.e. a collection of correspondences between the entities of both ontologies. The problem of finding and specifying the mapping between two ontologies is sometimes referred to as the *ontology matching problem*. Following the characterization in (38) the ontology mapping task consists of the following steps:
    - **Finding alignments between ontology entities:** This refers to the process of *finding the correspondences between the entities* (concepts, properties/roles and instances) of two ontologies. This process is known as *ontology alignment* and several methods, techniques for (semi) automatic alignment of vocabularies and ontologies have been proposed in the recent years.
    - **Specification of the alignments:** This refers to the process of *specifying the mappings between ontologies* so as to enable exploitation of the semantic overlap between two ontologies. Several methods and approaches exist for specifying these mappings.
    - **Exploitation of the mapping:** This step consists in *using the mappings* for enabling reuse and knowledge sharing among applications across domains. For example, the mapping between two ontologies could be used to implement a search engine that search for information in multiple heterogeneous databases.
- **Ontology Merging:** This process consists in taking a set of ontologies and producing a new ontology that is the union of the source ontologies and captures both the correspondences and discrepancies among the ontologies.
- **Ontology Integration:** Ontology integration can be seen as the process of creating an ontology by composing other existing ontologies possibly describing different domains (39); in contrast to ontology mapping where usually the ontologies describe the same domain.

5.1.1 Classification of Methods and Techniques

The literature of ontology mediation is rich in methods for addressing different aspects of the ontology mediation task. They make use of techniques and algorithms from different areas. For example, ontology mediation benefits from the techniques developed in the area of *Linguistics*, *Natural Language Processing* (NLP) (see e.g. the methods proposed by (40)(41)(42)(43)), and *Machine Learning* (44) among others. More recently, new methods for ontology mediation have been proposed that make use of reasoning techniques from the area of *Automated Reasoning* in Artificial Intelligence and Semantic Web research. These methods differ in many aspects including the kind of information they use for producing the alignment (structural, linguistic, semantic, etc.), the computational resources used for producing the alignment (e.g. memory and time), the quality of results (e.g. precision, recall, etc.) and the type of correspondences or mappings they are able to discover, among other aspects. For example, the works of (45)(46)(47) use probability and statistical methods to produce an alignment but they are limited by their inability to detect semantic differences. The approaches presented in (48)(49)(50)(51) consider many other informational aspects but they fail to support the alignment of large-scale ontologies.

In order to organize and help the users make sense of all the methods available a classification or categorization is required. Broadly speaking ontology alignment techniques (or simply *matchers*) can be classified into *elementary* and *composite techniques*. *Elementary matchers* are individual matchers from which other, more complex, *composite matchers* that combine different techniques can be built. One approach to classifying elementary matching methods was proposed by (52). This work builds upon the previous work of the authors as well as other researchers and introduces a multi-layer classification schema. Each layer classifies ontology matching techniques according to a different criterion. Three layers are identified: the *granularity and input interpretation* layer, the *basic techniques* layer and the *type of input* layer. The first one classifies methods according the granularity of the matchers and the way they interpret the input. The third layer classifies methods according to the type of input used by the matchers. Finally, the second layer identifies several classes of techniques and classifies them according to the categories of the other two layers. Based on this classification criteria the authors identify several classes of ontology matching methods. For details of this classification and example of methods the reader is referred to (52).

Many other classification frameworks have been proposed in the literature, such as the one proposed in (53) but, in general all of them refer to the same categories and classes identified in (52) although they do so by using a different nomenclature. For example, ontology mapping methods can also be classified into *instance-based* and *schema-based* methods depending on the type of data they use. These categories correspond to the *extensional* and *structural* techniques introduced in (52). Sometimes methods are classified into *lexical methods* and *structural methods*. The former rely on the lexical characteristics of terms to determine the degree of similarity between them. The latter rely on the structural information of terms, including the relations with other terms such as through ancestors and descendants. These correspond to the *internal* and *relational* of (52) respectively.

5.1.2 Organization of this Survey

Over the years several methods for ontology alignment have been proposed. In this survey we concentrate our attention into ontology mapping systems that have proven

to be successful over the recent years and, in particular, that have been used and tested in applications of the medical and Life Sciences domains as this is the context of the INTEGRATE project. The approach for selecting the systems consisted in exploring the latest experiments conducted in the context of the *Ontology Alignment Evaluation Initiative* [41] and choosing those systems that have participated several times and/or that have been used for mapping medical ontologies. In addition we survey those systems that over the years have become defacto standard in the community due to their success.

The goal of this survey is to identify candidate systems for reuse and to identify techniques and strategies that could be reused and/or extended as part of the solution envisioned by the project. Techniques and algorithms that have been tested on ontologies relevant to INTEGRATE are particularly interesting. We are also interested in technical details such as whether the systems are free or distributed through commercial licences and whether they support integration with other systems. Each system is described in terms of the technologies and techniques used for finding mappings between ontologies, the type of correspondences derived, the type of background knowledge used by the system to assist the alignment process (if any) and several other more general features. Another goal of this survey is to identify the factors that need to be taken into account when choosing an ontology mapping solution.

Although experimental results are important for choosing the appropriate system in this deliverable we refrain from reporting on the details of benchmarks and experiments aimed at evaluating the performance of the systems presented in this survey. These will be reported in Deliverable 2.2.

This chapter is organized as follows: Sections 5.2 to 5.11 gives an overview of several ontology matching and alignment systems. We conclude the chapter with a discussion of the relevance of these systems to the project and several factors to be considered when choosing mapping algorithms and systems.

5.2 ASMOV

ASMOV (54) is an ontology mapping algorithm that computes an alignment between two input ontologies. The alignment consists of *equivalence relations* between the entities of the ontologies; no other type of correspondence is computed. Although equivalence relations are an important asset other complementary correspondences such as *sub class* and *sub property relations* are also important in certain applications. ASMOV assumes the ontologies to be aligned are expressed in OWL DL.

ASMOV takes as input two ontologies to be mapped, an *optional reference alignment* and applies an iterative process that consists of two main phases: a *similarity calculation phase* and a *semantic verification phase*. The output of the similarity calculation phase is a matrix that contains information about the similarity between different entities in the ontologies. From this matrix a pre-alignment is extracted and evaluated and refined by the semantic verification process from which the final alignment can be extracted. The iterative algorithm finishes when either the resulting alignment is repeated or when the same alignment matrix is obtained.

[41] http://oaei.ontologymatching.org/

1. **Similarity Calculation:** In this phase ASMOV computes a similarity matrix that contains information about the similarity among the entities of the ontologies involved in the mapping process. The algorithm computes a (total) similarity measure as a weighted average over four similarity measures. The first one is a *lexical similarity measure* that takes into account lexical information and uses string comparison methods and external resources such as thesaurus. The second measure is a *relational measure of similarity* that takes into account structural information such as the specialization/generalization hierarchy of the ontology. The third one, a *restriction similarity measure*, also takes into account structural information to determine the similarity of entities based on the constraints used in the definition of concepts and properties. The fourth measure is an *extensional similarity measure* that uses data about instances to determine the similarity between classes and between properties. In addition, the algorithm can make use of a reference alignment that can be used to overwrite the weighted similarity measure computed by the algorithm. The total similarity measure computed by the algorithm corresponds to the measure provided by the reference alignment (if present) or to the weighted average over the other four measures. The computation of this total similarity measure is partially done at pre-processing time (computation of the lexical measure) and during the similarity calculations phase of the algorithm.
2. **Semantic Verification:** In this phase the algorithm tries to semantically verify the pre-alignments by removing those correspondences that are not supported or are less likely to be satisfiable by the axioms in the ontologies. This is done by considering different kinds of inferences made from the correspondences in the pre-alignment.

ASMOV is implemented in Java and uses a standard thesaurus adapter API to facilitate the integration of the tool with different thesauri in different domains. ASMOV's performance and accuracy have been evaluated using the Ontology Alignment Evaluation Initiative (OAEI) benchmark suite in multiple occasions. For details of benchmark results the interested reader is referred to Deliverable 2.2 and the OAEI's website.

5.3 Eff2Match

Eff2Match (55) is an ontology mapping method. The algorithm implements an iterative matching process that uses different type of information, including lexical and structural information in order to iteratively expand an initial set of matching entities. The type of correspondence between entities computed by the algorithm is entity equivalence (similarity between concepts and between properties). The algorithm starts with two ontologies to be matched and iteratively produces a set of correspondences between entities of the two ontologies. The following are the key steps of the algorithm.

1. **Anchor generation:** The first step of the algorithm consists in generating pairs of similar entities using an exact string matching method using the entities names and labels.
2. **Candidate generation:** In this step Eff2Match tries to find candidate similar entities to those for which no similar entities were found in the previous step. Candidate entities are computed using Vector Space Models in conjunction with information about an entity's name, an entity's ancestors and descendants

(in the case of concepts) and an entity's domain and range (in the case of properties). For each unmatched entity in the source ontology a set of candidate entities is computed using the VSM-based similarity measure.

3. **Anchor expansion:** In this step the algorithm tries to expand the set of equivalent entities found in the previous step by using terminological alignment methods. In particular, the algorithm relies on WordNet to find equivalent entities (concepts and properties) based on the synonymy between their labels. In addition to this, the Eff2Match uses a novel techniques called *Informative Word Matching* that determines the similarity between entities based on the similarity between an entity label and the label of the target entity's ancestor.
4. **Iterative boosting:** The last step of the algorithm consists in extending the set of matching entities using the initial anchor set computed in the first step and an expanded anchor set. The expanded anchor set is the union of the initial anchor set and the anchor set produced by the anchor expansion step. The algorithm tries to find matches for those entities for which no match was found between them and the candidates set. Structural information of the ontologies such as the number of ancestors/descendants of a given entity is exploited by the algorithm in order to discover correspondences between entities.

Eff2Match was implemented in Java and is distributed as a single jar file that can be embedded into one's application. No front-end or GUI is provided with the system.

5.4 RiMOM

RiMOM [42] (56) is a framework for ontology alignment that combines multiple mapping strategies and dynamically selects which strategy to apply based on several types of ontological information derived from the input ontologies. It extends a previous version of the tool and combines multiple alignment strategies with a dynamic strategy selection mechanism that can detect which strategy best suits a specific alignment task.

RiMOM was developed based on the observation that not all alignment methods are equally suitable, i.e. accurate and efficient, for every ontology. Multiple experiments in the field of ontology alignment and mediation have shown that there is no single method or strategy that outperforms the rest across every single ontology. Instead, depending on the features of an ontology certain strategies perform better than others; see the OAEI's website for experimental results. Based on this observation and on experiments done with a previous version of the tool the authors in (56) propose a ontology alignment method that combines multiple strategies and selects the strategy to apply depending on several ontological features. In this sense the aim of this system is to serve as a framework for ontology mapping where multiple alignment strategies can be used and where the accuracy of the mapping method is improved by carefully selecting at runtime the strategies to use depending on the characteristics of the input ontologies.

RiMOM takes as input two ontologies to be matched and follows five steps in order to produce a mapping of the input ontologies. The key steps in the process are the following:

[42] http://keg.cs.tsinghua.edu.cn/project/RiMOM/

1. **Pre-processing:** The first step consists in computing two ontology similarity measures, one based on structural information and the other based on an entity's label. These measures will be used for selecting the appropriate strategy in subsequent steps.
2. **Linguistic-based ontology matching:** The second step of the algorithm consists in computing the similarity between ontology entities using multiple linguistic-based similarity measures including standard metrics such as edit distance and vector distance. A strategy selection method is applied in order to select what linguistic-based strategy to employ.
3. **Similarity combination:** In the next step the algorithm combines the results produced in the linguistic-based ontology matching step. As in the previous step a selection method is applied in this step to determine the appropriate weights for each similarity measure to be combined.
4. **Similarity propagation:** In this step the algorithm computes multiple structural similarity measures and applies a strategy selection method.
5. **Alignment generation and refinement:** The final step of the algorithm consists in refining and returning the alignment found between the ontologies.

RiMOM applies three strategy selection methods for selecting the appropriate strategies to use at three different steps. One method is used for selecting the right linguistic-based strategy to use. Another method is applied when combining multiple similarity measures and a third method is used when computing structure-based similarity measures. In the linguistic-based ontology mapping step two thresholds (which can be tuned by the user) based on the label and structure similarity measures are used for selecting the right strategy. For similarity combination RiMOM computes several weights and then uses them in combination with the label and structure similarity measures computed in step one to decide whether to rely on a label-based similarity or on vector distance-based similarity. Finally, the decision of what type of structural information to use in the similarity propagation phase is based on the structure similarity measure computed in the first step. Comparing this value to a threshold, which can be tuned, helps deciding what strategy to use.

RiMOM is implemented in Java and is available for public download [43]. For details on evaluation and benchmark the reader is referred to the section on Benchmark of Ontology Mapping Tools.

5.5 SAMBO

SAMBO [44] (53) is a system for aligning and merging biomedical ontologies. It is based on a framework developed to support the semi-automatic alignment and merge of biomedical ontologies and for comparing alignment strategies. It is characterized by the use of multiple matchers, each of which can implement a specific alignment strategy, a method for combining and filtering the results obtained from these matchers and an interactive process that allows users to participate in the ontology mediation process by accepting and discarding alignments proposed by the systems implementing the framework. The alignments computed by the framework are based on the computation of similarity measures between the entities of the source ontologies. The framework can exploit several kind of ontological information and

[43] http://keg.cs.tsinghua.edu.cn/project/RiMOM
[44] http://www.ida.liu.se/~iislab/projects/SAMBO

supports the use of strategies based on *structural information*, *linguistic or lexical information*, *constraint-based information*, *instance-based information* and strategies that use auxiliary information such as dictionary, domain thesauri, etc. The output of an alignment task is the set of similarity relations between the terms of the input ontologies.

SAMBO takes as input two ontologies expressed in OWL and can output a new ontology that is the result of merging the input ontologies or, a set of alignments between the input ontologies. The system uses a terminological matching algorithm that implements a linguistic-based strategy that computes the similarity between terms (concepts or relations) using *n-gram*, *edit distance* and *word-based* techniques. To improve the accuracy of the algorithm SAMBO uses WordNet as an external source of information. In addition, SAMBO implements a structural-based alignment strategy that determines the similarity between ontology entities based on the specialization (*is-a*) and *part-of* hierarchies. SAMBO also uses the UMLS' Metathesaurus to determine the similarity between concepts based on the synonym relation between the concepts in the ontology and those in the metathesaurus. SAMBO also allows for the combination of strategies and enables users to choose the strategies to be used. Each matcher (strategy) suggests an alignment and suggestions can be analyzed using weights or based on the similarity value of the matcher that produces the suggestion. SAMBO currently uses alignment strategies that derive similarity and is-a relations only; i.e. correspondences that reflect other types of relations such as disjoint relations are not computed.

Using the alignments computed by the alignment process SAMBO can merge ontologies by applying the alignments and inferring new knowledge that is consistent with the ontologies and alignment.

5.6 AgreementMaker

AgreementMaker [45] (57) is an ontology and schema matching system developed and maintained by the *University of Illinois at Chicago* in the *Advances in Information Systems Laboratory* (ADVIS Lab). AgreementMaker was designed and developed as an efficient solution to the problem of aligning large real-world ontologies and schemas and has been applied in different domains that include geospatial data (58), environmental data (59) and biomedical data (60).

What differentiate AgreementMaker from other similar systems is that AgreementMaker is not only an ontology and schema matching system but it is more generally a *platform for evaluating and comparing multiple alignment methods*. The system is built around a flexible architecture that allows users to incorporate new matching methods and strategies and test their performance against other methods.

Another distinctive feature of AgreementMaker, and one that also sets it apart from other ontology mapping systems, is its user interface whose design was driven by the requirements of domain experts. It provides a visual environment not only for finding mappings between ontologies according to several alignment methods but also for evaluating, comparing and combining different alignments as well as the performance of these methods. Users can configure and run multiple methods and visualize the

[45] http://agreementmaker.org/

results of the alignments. The GUI also provides a mechanism for visualizing sub trees of the source and target ontologies along with the corresponding mappings. Users can also specify previously computed mappings as input to a new matching task. Also, several mappings can be combined either manually or automatically. In addition to its comprehensive user interface, the system incorporates several mapping methods that vary along multiple dimensions and produce mappings based on different strategies, e.g. by using schema and/or instance information, by using manual or (semi) automatic strategies, by considering different features of the ontologies, etc.

In terms of its architectural design the system is composed of three layers. Each layer contains a matcher that implements a specific mapping algorithm and produces a set of mappings by extending the functionality of the matchers in the layers above. In general, every matcher follows a generic architecture that consists of a module for computing the similarity between the entities of the input ontologies (concepts and properties) and a module for selecting the best mappings. The first module takes as input the two ontologies and produces two matrices, one for concepts and the other for properties, containing information about the similarity between concepts and properties of the input ontologies. The mapping selection module then selects the best mappings based on a threshold value and on the cardinality of the mappings. Specific matchers are specializations of this generic architecture. The first layer of the architecture contains a matcher that is responsible for producing the similarity matrices based on several concept features such as name, label, comments and instances. The second layer's matcher improves on the results of the first layer's matcher by refining the mappings using conceptual or structural information. Finally, the third layer's matcher combines the results of the matchers in the layers above in order to produce the final mapping. This architectural design of both the system and the individual matchers allows AgreementMaker to compose matchers in a serial or parallel fashion. Because of its flexibility and modular approach the architecture enables users to develop and plug-in their own matchers.

In order to evaluate the quality of the alignment produced by the system AgreementMaker can make *use of a reference alignment* and compare the results of the different methods to that "gold standard" and produce a measure of relative quality. However, when a reference alignment is not available the system can determine the quality of the alignment using other metrics, in particular local and global metrics. For details on these metrics the reader is referred to (57).

The different native matching algorithms implemented in AgreementMaker make use of various alignment strategies. AgreementMaker uses both syntactic and structural methods for finding mappings between the ontologies. Among the syntactic methods the system uses a *Base Similarity Matcher*, a *Parametric String Matcher* and a *Vector-based Multi-word Matcher*. As for the structural matchers the system includes a *Descendants Similarity Inheritance Matcher* and a *Siblings Similarity Contribution Matcher*. For details about the specifics of these methods the reader is referred to (57).

In terms of usability AgreementMaker supports several input and output formats including XML, N3, RDFS, OWL as input formats and XML and Excel as output formats. The system also supports the development and integration of new matchers through a Java API that provides the basic means for developing and plugging in new matchers. The current version (0.23 at the moment of writing) is publicly available through a free-of-charge registration procedure.

5.7 BLOOMS

BLOOMS (61) is method for mapping ontologies that uses simple lexical mapping methods and couples them with a novel global similarity computation techniques that takes advantage of the semantic variability of edges (62). BLOOMS was developed as a component of a much larger ontology extension system called Auxesia. It was designed as a fully automatic system for finding alignment between biomedical ontologies with *precision over recall in mind*, although it can also be used for mapping ontologies in other domains.

At its core BLOOMS is composed of a *lexical matching component* and a *global similarity component*. The first component is responsible for computing the similarity between concepts and is composed by two other modules, an *exact matcher* and a *partial matcher*. The exact matcher determines the similarity between two concepts by using a simple string matching technique whereas the partial match component computes the similarity between concepts by considering the different parts of a concept's textual description. The second component of BLOOM is a module that implements a similarity propagation technique called *semantic broadcast*. This technique is based on the idea that concepts whose relatives (ancestors and descendants) are similar should also be similar and relies on the availability of a semantic similarity measure between concepts. Using an average measure of the semantic similarity between two concepts, the sum of similarities between the concept's relatives and the lexical similarity computed by the lexical component BLOOMS determines a final similarity measure between concepts.

Alignments in BLOOMS are produced by each of the three matching modules, i.e. the exact, partial and semantic broadcast matching modules, using different thresholds for determining when a match is considered an alignment.

5.8 AOAS

AOAS (63) (64) (65) stands for *Anatomy Ontology Alignment System* and is the result of the efforts made at the U*.S. National Library of Medicine* to develop domain knowledge-based techniques for aligning large anatomical ontologies. It is a domain-specific ontology matching system for anatomical entities, i.e. for anatomy concepts and relations.

The AOA system uses an automatic, *rule-based alignment method* that relies on *schema-level information* and *domain-specific lexical information* to find mappings between concepts. The mappings found by the system represent *equivalence relations* between the entities. AOAS uses a *direct alignment method* for finding mappings between ontologies. In ontology alignment and mediation, a direct alignment method finds mappings between two ontologies by computing the similarity between concepts (or entities in general) without relying on a reference ontology as it is the case with *indirect alignment methods* where mappings are indirectly computed based on direct mappings with a reference ontology. In particular, the AOA system computes the similarity between concepts using *domain-specific lexical techniques* in combination with *structural and semantic techniques* to validate the mappings. The process of determining the lexical similarity between two concepts is reduced to deciding an exact match between the terms.

The alignment process used by the system involves two key (sub) processes. The first one is a *lexical mapping* process that finds mappings between concepts across ontologies by using lexical techniques to map concepts based on their names and synonyms, if present. To support the lexical matching of concepts *AOA can make use of an external source*. For example, in (65) the UMLS was used for resolving synonyms between concepts in the process of aligning the GALEN and FMA ontologies. The second process used in the alignment task is called *structural validation* and consists in validating the mappings produced by the lexical mapping by computing a structural similarity measure. To validate mappings AOA uses shared relations to other aligned (equivalent) concepts, such as *is-a* and *part-of* relations. The system considers both explicit relations and implicit relations extracted through *augmentation* and *reasoning techniques*. For example, if concepts A and B are lexically aligned and both are in a *is-a* relationship with concepts C and D respectively and, C and D are correctly mapped then the mapping between A and B is considered valid. For details of these techniques the interested reader is referred to (65). Augmentation is particularly interested when aligning biomedical ontologies because it allows to extract knowledge that is embedded in concept names, a feature present in many biomedical ontologies and more generally, in ontologies rich in descriptions, comments and annotations.

The AOA system was used in (64) to automatically align the anatomy part of the *NCI thesaurus* [46] and the *Adult Mouse Anatomy ontology (MA)* [47]. The work in (63) is particularly interesting because it proposes a hybrid system for ontology alignment that combines direct and indirect alignment methods to align anatomical entities. The direct method works as described before (AOAS's) while an indirect method finds mappings between the MA and NCI thesaurus by using the FMA ontology as a reference ontology. An indirect mapping is produced on the basis of two direct mappings between concepts of MA and FMA and concepts of the NCI thesaurus and FMA. For details the interested reader is referred to (63).

A benchmark of AOAS was conducted during the OAEI 2006 campaign while the hybrid system presented in (63) was evaluated in OAEI 2007. For details of the evaluation the reader is referred to Deliverable 2.2.

5.9 FALCON-AO

Falcon-AO [48] (66) (67) is an ontology alignment system developed by the *Department of Computer Science and Engineering* of the *Southeast University in Nanjing*. It is an open source project implemented in Java and distributed under the Apache 2.0 license. Although it is publicly available for download at the time of writing it was not possible to get access to the project's homepage due to technical problems on the team's website.

Falcon-AO's architecture is composed of a few key components. The first component is the *model pool* which is responsible for parsing the input ontologies and preparing the ontologies for alignment by adding/removing statements based on the inference capabilities of the ontology languages. The second component is the *matcher library* which contains a series of matching algorithms. More concretely, the library

[46] http://cancer.gov/cancerinfo/terminologyresources
[47] http://www.informatics.jax.org/searches/AMA_form.shtml
[48] http://ws.nju.edu.cn/falcon-ao/

contains two *linguistics-based matchers* (LMOs) that use linguistics-based algorithms to compute the similarity between words using lexical comparisons and statistical analysis. These rely on the availability of concept descriptions (entities descriptions in general) such as names, labels and comments. A third matching algorithm used in Falcon-AO is a *partition-based matching algorithm* (or PBM) used for finding blocks of mappings; a technique that aims at improving the performance when aligning large-scale ontologies. The fourth matcher implements the *graph matching for ontologies* algorithm (or GMO) and uses a graph-based approach to align two ontologies. GMO computes the similarity between two ontology entities based on structural information associated to these entities, which in turn is computed based on the similarity between RDF statements involving the two entities. GMO is used in conjunction with the LMOs and the PBM. It takes as input an external alignment produced by the LMOs, the blocks identified by the PBM and an existing alignment of the built-in constructs of ontology languages. The output of GMO is an alignment that extends the alignment produced by the other matchers. These two alignments are later on integrated to produce the final alignment. For details of the algorithm the reader is referred to (68). Another key architectural component is the *central* controller that is in charge of executing the alignment workflow, e.g. executing matchers and combining similarity measures. The *alignment set* component is responsible for evaluating the generated alignment against reference alignments. For details of the architecture the reader is referred to (67).

The alignment workflow in Falcon-AO proceeds as follows. Given two ontologies to be aligned the system first prepares both ontologies for alignment by adding/removing statements based on the inference capabilities of the ontology languages. This step is called *model construction*. The next step, called *matcher execution*, consists in running the LMOs and PBM to produce part of what is called an *external alignment*. The other part of this external alignment is comprised by an *existing alignment* that contains mappings between the built-in constructs of different ontology languages (RDF, RDFS, OWL). Using this external alignment GMO tries to generate additional mappings. The final step, called *similarity combination*, is to integrate the two alignments using the *linguistic* and *structural comparability measures*. More concretely, GMO requires an external component to evaluate the quality of the computed alignment. The approach used in Falcon-AO is to use two measures to evaluate the quality or reliability of a mapping. The first one is a *linguistic comparability measure* that quantifies the similarity of two ontologies based on their lexical similarity and the second one, is a *structural comparability measure* that quantifies the similarity of two ontologies based on how similar their structures are. The alignment produced by Falcon-AO *captures equivalence relations* only.

5.10 SOBOM

SOBOM [49] (69) is an ontology matching system that uses a novel approach to ontology matching based on *sub-ontology extraction* to find alignments between two input ontologies. It was designed as a general purpose system to automatically match lightweight ontologies structured according to an *is-a* hierarchy. SOBOM was developed by the *School of Computer Science and Technology* of the *Harbin Institute of Technology* in Harbin, China.

[49] http://mlg.hit.edu.cn:8080/Ontology/Download.jsp

SOBOM views an ontology from two different point of views: on the one hand, SOBOM treats an ontology as composed of an *is-a* hierarchy only. On the other hand, an ontology is seen as comprised of different types of relations among concepts. Using this two-fold view of an ontology the system is able to find anchors (pairs of equivalent concepts between the ontologies), extract sub-ontologies from the two ontologies being aligned and align relations between the ontologies. Correspondences computed by SOBOM represent *equivalence relations* between ontology entities.

The architecture of the system is composed of four modules. The first one, called *anchor generator*, is a module that finds correspondences (anchors) between concepts in the source and target ontologies by using linguistic techniques and a concept's local information, e.g. instances (if present), textual information (label, comments, etc.) and structural information. The second component of the architecture is called *Semantic Inductive Similarity Flooding* (SISF) and implements an algorithm for propagating similarity relations from the anchors to the sub-ontologies. The third component is a *relation matcher* that finds alignments between ontology relations. The final component is called *Sub-ontology Extractor* and is used for extracting sub-ontologies from the source and target ontologies.

SOBOM executes the following workflow to find an alignment between the source and target ontologies. The first step consists in extracting anchors and sub-ontologies. Using the anchors found by the *anchor generator* the system extracts a set of sub-ontologies from each ontology. The second step consists in ranking each set of sub-ontologies and then aligning sub-ontologies using SISF. Next, the system tries to find an alignment between the relations in the ontologies and the individuals using as input the output produced by SISF. The final step consists in grouping the alignments obtained in the previous steps and generating the final alignment. Ontologies can be specified in RDF and OWL. It is important to notice that SOBOM does not use background knowledge to align ontologies a feature that needs to be considered when analyzing benchmark results.

The system was developed in Java and is publicly available for downloads although at the time of writing the system was unavailable due to connection errors on the server side.

5.11 NBJLM System

NBJLM (70) is a multi-strategy (or hybrid system), dynamic ontology matching system written in Java. It has been jointly developed by the *Nankai University* and the *Military Transportation University*.

NBJLM combines similarity of literal concepts and semantic structure to discover correspondences between pairs of concepts and properties. The correspondences represent *equivalence and subsumption relations* between the entities of the source and target ontologies. More specifically, ontology entities are aligned based on a function that computes the similarity between two entities in terms of two similarity measures. The first is a *literal concept similarity measure* and the second is a *structural similarity measure*. The *literal concept similarity measure* takes into account polysemy and synonym of words and uses WordNet as an external resource to determine the degree of similarity between two concepts using their names. The *structural similarity measure* on the other hand uses the *literal concept similarity*

measure values, the structure of the ontology and the hypernym and hyponym of words to compute the degree of similarity between two concepts. To help speed up the computation of mappings the system uses parallelization techniques.

The system is publicly available for downloads at the group's website [50] although at the time of writing the system was unavailable due to connection errors on the server side.

5.12 Discussion and Relevance for INTEGRATE

Ontologies in the Biomedical domain and the Life Sciences pose important challenges to the mediation and alignment tasks due to their distinctive features. First, biomedical ontologies tend to be very large in terms of the number of concepts they contain. SNOMED CT for example contains roughly 340,000 concepts and the Gene Ontology about 50,000 concepts. A second feature of this type of ontologies is the range of relation types they contain. Most of these ontologies contain a limited set of relation types, e.g. is-a, part-of, etc. and this can affect negatively the performance of more general-purpose mapping systems that take advantage of the rich structure of ontologies. A third feature is the vast amount of textual information these ontologies posses. This comes in the form of concept names, synonyms, labels, comments, etc. Although rich textual descriptions can be exploited by alignment methods based on lexical strategies the high degree of ambiguity found in these ontologies also poses a challenge for these methods. The distinctive characteristics of biomedical ontologies have also been acknowledged by the knowledge representation and reasoning community which have proposed in recent years novel mechanisms and techniques for representing and reasoning over this type of ontologies.

The characteristics of this type of ontologies call for methods that can deal with large number of concepts and in some cases relatively shallow terminologies. Table 2 shows for each system in this survey the *type of semantic correspondences* it is able to compute, e.g. *equivalence relations* (=) or *subsumption relations* (<=), whether the system is able to associate a *confidence value* to each correspondence thus allowing users to judge the accuracy of the alignments produced and whether the system uses *background knowledge*.

System	Correspondence Type	Confidence	Background Knowledge
ASMOV	=, <=	Yes	Yes
Falcon-AO	=	Yes	No
AgrMaker	=	Yes	No
SAMBO	=	No	Yes
BLOOMS	=	No	No
RiMOM	=	No	No
Eff2Match	=	Yes	No
AOAS	=	No	Yes
NBJLM	=, <=	No	Yes
SOBOM	=	Yes	No

Table 2 Characteristics of Ontology Matching Systems

[50] http://www.brsbox.com/OAEI2010

The SAMBO system is particularly relevant to INTEGRATE due to its focus on alignment strategies that are applicable to ontologies found in the biomedical domain. It is also relevant due to its underlying framework that can in principle be extended and/or used for implementing and testing new alignment strategies in the biomedical domain. BLOOMS, SAMBO, AOAS and AgreementMaker are particularly relevant for the project because they have been applied in alignment tasks in the Life Sciences and Medical domains as well.

As for the disadvantages and limitations of the BLOOMS system the current version (as of September 2011) does not take into account spelling variants and does not make use of external resources to complement the mapping algorithm. Moreover, the only type of correspondence between entities computed by the system is an equivalence relation. From a practical point of view the major disadvantage is that the system was not designed as a standalone application but rather as part of a bigger system. This could hinders its reusability outside the host system.

Perhaps the major disadvantage of the AOA system is its domain-specific nature that makes it not suitable for other domains, i.e. for domains where the data contains no anatomical entities. However, although the reusability of the system is low some of its underlying mechanisms and strategies could be reused and possibly adapted to other application domains. One key idea used in the system is the structural validation mechanism that uses the structure of the ontologies to validate the lexical mappings. Another key concept is the exploitation of domain-specific information for discovering lexical mappings between concepts and, the use of such information throughout the entire alignment process.

In the case of Eff2Match its main drawback is the limited support for finding correspondences between entities. As of August 2011 the system is not able to discover equivalence relations between concepts and properties. The authors report in (55) their plans to extend the functionality of the tool in order to be able to compute other types of correspondences such as non-equivalence relations, specialization relations and part-of, a very common type of relation found in many biomedical ontologies. However, no concrete changes have been made to the tool or reported since then.

One of the disadvantages of RiMOM is its inefficiency when dealing with large ontologies.

5.12.1 Choosing the Right System

The task of choosing an appropriate ontology mediation tool or alignment method mainly depends on the requirements of the application at hand. Each application will impose certain constraints on the type of system needed based on the characteristics of the application and the type of data used. From the survey conducted in this work and the description of the systems presented in the previous sections it is possible to identify several key factors that need to be taken into account when deciding on an ontology mediation technique or system, e.g. merging or alignment tool.

- **Accuracy of the mappings**: The set of mappings produced by an ontology matching method are typically evaluated with respect to their accuracy to ensure that the mappings produced are correct. The most common approach to

evaluate the correctness of mappings is by measuring their precision and recall.

- **The type of correspondences computed by the mapping algorithm**: Every ontology matching algorithm produces one type of correspondence between ontology entities, namely an *equivalence relation*. However, equivalence relations are not the only type of entity relationship that is relevant. Some applications require other types of relations to be discovered such as subsumption, disjointness and part of, among others. A few of the systems presented in the previous sections are able to discover, in addition to equivalence relations, subsumption relations as well.
- **The format in which the mappings are expressed**: In order to facilitate the use of the mappings discovered between ontologies they need to be represented in a form that enables reuse. RDF-based formats are very common these days and an ontology of mappins is typically used.
- **The licence under which the system is distributed**: This is a relevant technical aspect of the system specially for projects and/or systems that follow open source practices.
- **The input format of the ontologies**: The application requirements will determine the format in which the ontologies are represented and based on this the choice of system may change.
- **The external resources the system may require or accept**: It is a plus that the system can exploit them but if the system can only work when external resources are available, e.g. only mapping ontologies if WordNet is available, could be a drawback.
- **Computational resources required**: These are factors that need to be determined empirically using ontologies from the application domain.
- **Alignment with a confidence value**: Some of the systems surveyed in this work are able to assign a confidence value to the alignments produced. This is meant to assist users in deciding the validity of the alignments and to help them choose among alignments produced by different tools.

6 Ontologies for the Life Sciences

6.1 MedDRA

MedDRA[51] (*Medical Dictionary for Regulatory Activities*) is a medical vocabulary developed by the IFPMA-International Federation of Pharmaceutical Manufacturers and Associations that refers to diseases, diagnoses and reactions and results to classify information related with adverse events associated to the use of biopharma and other medical products on humans. The structure of this dataset is hierarchical, i.e., concepts are "owned" by a sequence of predecessor terms; one term could be preceded by more than one father-term. The hierarchy contains 6 levels and approximately 57,700 concepts (according to the version of 2002).

The hierarchy of the core dataset is composed by six levels:

- System Organ Class, SOC.
- High Level Group Term, HLGT.
- High Level Term, HLT.
- Preferred Term, PT.
- Lowerest Level Term, LLT.

A purchase license is required if it is intended to be used within a commercial software, although, true not-for-profit organizations qualify for a Basic subscription. For instance, an educational institution, or a direct patient care provider as a hospital, use MedDRA as a reference tool.

6.1.1 Use within the INTEGRATE project

MedDRA has all the virtues and benefits of being a taxonomy extremely well defined, which facilitates its management and comprehension. Nevertheless, it is only oriented to work with diseases, which may not be enough to represent all the possible concepts needed in the project.

6.2 SNOMED-CT

SNOMED-CT[52] (*Systematized Nomenclature of Medicine – Clinical Terms*) is considered the most important clinical terminology due to its precision and highly comprehension data. It was developed by the College of American Pathologists (CAP) and it is a property of the International Health Terminology Standards Development Organization (IHTSDO). This taxonomy allows its users to tag, index and store clinical information; facilitating the correct management of medical media. Its usability has been an important help point to everyone working with electronic medical record systems (EMRs); becoming adopted as the standard clinical terminology for many institutions, as for instance, the NHS of England.

SNOMED-CT consists of over a million medical concepts. These concepts are arranged in a *is-a* hierarchy and can have multiple parents and roles. SNOMED-CT

[51] www.meddramsso.com
[52] *http://www.ihtsdo.org/*

contains roughly 730,000 descriptions and 920,000 relations. The hierarchy is arranged in 33 levels. Each concept has associated the next properties:

- **Fully Specified Name:** A unique way to name and denominate the concept
- **Preferred Term:** The common phrase/term used by clinics to name the concept
- **Synonym:** Additional phrases/terms that could represent the concept

SNOMED-CT is free for research purposes. There are also a series of free and useful web-based taxonomy browsers on the internet, being especially interesting the SNOMED-CT core browser[53] developed by the Virginia-Maryland Regional College as well as the one facilitated by the U.S. National Cancer Institute's Term Browser[54].

6.2.1 Use within the INTEGRATE project

The large size of the taxonomy with its large number of concepts is one of its great advantages. In addition, if it is necessary to define a new concept which does not exist yet on the taxonomy, SNOMED-CT allows its users to "create" new concepts by using existing concepts; this process receives the name of *"Post-coordination"*. On the other hand, the large size and the irregular structure of the taxonomy suppose also disadvantages because it makes it more complex and disperse, which could make its management and its comprehension to be more difficult.

6.3 LOINC

LOINC (*Logical Observation Identifiers Names and Codes*)[55] was developed by the Regenstrief Institute to provide a definitive standard for identifying clinical information in electronic reports. The main goal of LOINC is to facilitate the exchange, management and pooling of results/outcomes and research on clinical care. The LOINC database provides a set of universal names and ID codes for identifying laboratory and clinical test results in the context of existing HL7, ASTM E1238, and CEN TC251 observation report messages. The latest version (2010) contains roughly 65,000 concepts organized in a flat hierarchy, i.e. with no levels.

LOINC codes are principally intended to identify test results and clinical observations and have been expanded to include nursing diagnosis, nursing interventions, outcomes classification and patient care data. Other fields in the LOINC message can transmit, for example, the identity of the source laboratory or other special details about the sample.

LOINC provides a Windows-based mapping utility called the **RELMA**[56] to facilitate searches through the LOINC database and to assist efforts to map local codes to LOINC codes. Also is available as a Microsoft Access database file and a web search application[57].

[53] http://snomed.vetmed.vt.edu/sct/menu.cfm
[54] http://nciterms.nci.nih.gov/ncitbrowser/pages/vocabulary.jsf?dictionary=SNOMED%20Clinical%20Terms
[55] http://loinc.org
[56] http://loinc.org/relma
[57] http://search.loinc.org/

6.3.1 Use within the INTEGRATE project

LOINC is the best way to represent clinical results because it allows users to define all the relative fields, for instance, components, scales or procedures of measure. Nevertheless, it has the inconvenient of being overly specific focusing only in clinical experiments. Moreover, the benefits of a taxonomy are lost as it does not have a hierarchical structure.

6.4 MeSH

The *Medical Subject Headings* (MeSH)[58] is a vocabulary for the purpose of indexing journal articles and books in the Life Sciences, it can also serve as a thesaurus that facilitates searching. It was developed by the United States National Library of Medicine (NLM).

This taxonomy is composed by descriptors. The latest version of MeSH (2007) contains about 26.000 descriptors and everyone has a list of synonym and a list of qualifiers; the total number of qualifiers according to the latest version is 83 while the number of synonyms or supplements is 139,000. These qualifiers are a subheading which can be added to descriptors to narrow down the topic and provide semantic information for classifying the descriptors. The descriptors or subject headings are arranged in a hierarchy of 12 levels. A given descriptor may appear at several places in the hierarchical tree. The tree locations carry systematic labels known as tree numbers, and consequently one descriptor can carry several tree numbers. Every descriptor also carries a unique alphanumerical ID that will not change.

MeSH can be browsed and downloaded free of charge on the Internet through PubMed. The yearly printed version was discontinued in 2007 and MeSH is now available online only. Originally in English, MeSH has been translated into numerous other languages and allows retrieval of documents from different languages. It has all the virtues of a taxonomy extremely well defined with its 6 exact hierarchy levels which facilitates its management and comprehension. However, it is only oriented to diseases which may not be enough to represent all the possible concepts to represent in the project.

6.4.1 Use within the INTEGRATE project

It has a brief and concise structure which makes it an ontology extremely well defined and with scarce dispersion. However, this simplified structure may suppose the lack of many needed concepts. In addition, the nature of the MeSH terminology (indexing MEDLINE publications) is not directly related with the INTEGRATE project needs.

6.5 ICD-10

ICD-10[59] stands for *International Statistical Classification of Diseases and Related Health Problems* (10th revision). It is a vocabulary specifically designed for the encoding and classification of all the information relative to external causes of injury or

[58] http://www.nlm.nih.gov/mesh/
[59] http://www.who.int/classifications/icd/en/

diseases. It was developed by the World Health Organization (WHO) and contains approximately 14,400 codes although it can be expanded to 16,000. Therefore, it is used to classify data in a wide variety of health-situation conditions and to manage and administrate every possible health record form. ICD is revised periodically and is currently in its 10th edition which dates from 1992.
The ICD-10 Procedure Coding System has a seven-character alphanumeric code structure. Each of these characters has 34 possible values: Ten digits (0-9) and 24 letters (A-H, J-N and P-Z). The first character of the procedure specifies the section (for instance 2-Placement or 7-Osteopathic), the second have a consistent meaning within the section, the third specifies the type of procedure performed, and the rest of the characters specify useful additional information. Therefore, ICD-10 is a medical terminology that provides codes to classify diseases, signs, symptoms or abnormal findings and which allows users to assign to every health condition a unique codification and category. ICD-10 contains no hierarchy levels.

6.5.1 Use within the INTEGRATE project

ICD-10 has the benefits of being a vocabulary extremely well defined and which is used worldwide nowadays. Nevertheless, it is only oriented to work with diseases and it is not a taxonomy but a vocabulary or terminology.

6.6 UMLS

The Unified Medical Language System [60] (UMLS) is a compilation of many of the most important medical vocabularies and biomedical taxonomies that exist nowadays. It was developed by the US National Library of Medicine. It puts at disposal to the medical environment multiple sources of knowledge plus a set of related software applications. The current version contains roughly five million concepts of which roughly one billion of them are biomedical concepts. Its main purpose is to facilitate the comprehension and understanding of the broad spectrum of items in the medical domain and, for it, UMLS facilitates a series of vocabularies that seek to provide some cohesion to the medical field.
This compendium of vocabularies is currently applied in tasks ranging from the systematic exchange of clinical information until the uniform definition of patient records, being also very useful in the development of tools for extracting relevant information from the existing medical literature.
UMLS is composed of three knowledge sources: the metathesaurus, the semantic network and the Lexicon, being the first one the base of the UMLS. It is a subset composed of a series of concepts that have their own characteristics and that are related to each other by different types of relationships. It is composed of around one hundred vocabularies, being the most important of them SNOMED-CT, MeSH, ICD-10 and LOINC.

6.6.1 Use within the INTEGRATE project

The principal disadvantage is that it is not exactly a taxonomy, it's just a compendium of taxonomies. On the other hand, the large number of concepts may be an advantage because it would not be necessary to define new concepts in order to

[60] http://www.nlm.nih.gov/research/umls/

represent all the needed concepts, however, this enormous quantity of elements, would suppose many management problems.

7 Security and Privacy Standards

7.1 Identity Management

7.1.1 SAML

The *Security Assertion Markup Language*[61] (71) defines an XML-based protocol, making it possible to exchange authorisation and authentication data between one or more security domains. This exchange is done by using signed assertions containing identity information. The entity that provides the assertions is called the asserting party while the relying party is the entity that consumes and verifies the assertions. A level of trust is required between the assertion providers and the relying parties. The current version of SAML is 2.0 which is a combination of three predecessor identity federation standards: SAML 1.1, ID-FF 1.2 and Shibboleth. This resulted in SAML 2.0 not being compatible with SAML 1.1.

SAML mainly focusses on solving the problem of web browser single sign-on (72). For this it offers a single sign-on profile. In the case of single sign-on the assertion provider takes the role of Identity Provider (IdP) and the relying party the role of Service Provider (SP). When the user visits a protected Service Provider, the SP redirects the user to the IdP with an authentication request (SP-initiated web Single Sign-On (SSO) opposed to IdP-initiated web SSO). Once the user has successfully authenticated himself (assuming he/she is already registered to the IdP), a signed assertion is generated by the IdP that contains all the information that is requested by the SP to authenticate and authorise the user. This assertion is send back to the SP that will use it to determine whether the user is allowed to access the service. SAML does not define how a user should authenticate himself on the IdP. It leaves the responsibility completely to the security domains [62].

Next to the single sign-on profile, SAML offers some other useful profiles (72): Single Logout, Attribute Exchange, Name Identifier Mapping, Assertion Query/Request profile.

SAML is widely adopted thanks to its focus on federation enablement and exchange of asserted information (it is the industry standard). Shibboleth is a well known implementation of the SAML protocol.

7.1.2 Liberty Alliance standards

The *Liberty Alliance* [63], currently transitioning to the *Kantara Initiative*[64], is an effort of more than 150 organisations that try to establish open standards, guidelines and best practices for identity management. The main keywords in the project are Federated Identity, Single Sign-On (SSO), Global Logout, Circle of Trust and Web Services. The Liberty Alliance Project is categorised in three main components explained in the following three paragraphs.

[61] http://en.wikipedia.org/wiki/Security_Assertion_Markup_Language
[62] http://en.wikipedia.org/wiki/Security_Assertion_Markup_Language
[63] http://projectliberty.org/liberty/specifications__1/
[64] http://kantarainitiative.org/

The **Liberty Identity Federation Framework** (ID-FF) specifies core protocols (SSO, Single Logout (SLO), Federation, Name Registration), schemata, bindings (HTTP and SOAP) and concrete profiles (Browser/Artifact, Browser/Post, etc.) that allow implementers to create a standardised, multi-vendor, identity federation network. ID-FF is an extension of SAML and served as input for SAML 2.0.

The **Liberty Identity Web Services Framework** (ID-WSF) consists of a set of schemata, bindings (SOAP, PAOS), protocols (Discovery and Interaction) and profiles (Security Mechanisms, etc.) for providing a basic framework of identity services, such as interoperable identity services, identity service description and discovery.

The **Liberty Identity Service Interface Specifications** (ID-SIS) utilize the ID-WSF and ID-FF to provide networked identity services, such as contacts, presence detection or wallet services that depend on networked identity. ID-SIS enables interoperable identity services such as personal identity profile service, alert service, calendar service, wallet service, contacts service, geo-location service, presence service and so on.

The Liberty Alliance standards are not commonly used for identity management solutions.

7.1.3 WS-*

WS-* is a collective noun for a variety of specifications associated with web services. These specifications form together the basic framework for web services build on the first-generation standards of SOAP, WSDL and UDDI. This association does not mean they are developed by a main standard body, the specifications are maintained by a diverse set of bodies or entities. The WS-* specifications are not strictly disjunct, they may complement, overlap, and compete with each other.

In the domain of security and especially identity management, five specifications of the WS-* are important: WS-Security, WS-Trust, WS-SecureConversation, WS-Federation and WS-SecurePolicy.

WS-Security [65] defines how web service messages can be exchanged in a secure way by guarding the integrity, confidentiality and the sender's identity of the messages. To enforce this, WS-Security uses XML signature (for integrity), XML encryption (for confidentiality) and various security token formats, like SAML, Kerberos, X.509 (for sender authentication), to provide end-to-end security.

WS-Trust [66] is an extension of WS-Security providing methods for issuing, renewing and validating security tokens and providing ways to establish, assess the presence of, and broker trust relationships. Using the extensions defined in WS-Trust, applications can participate in secure communication designed to work within the web service framework. A main concept in WS-Trust is the Security Token Service (STS), this is a special web service that issues security tokens conforming to the WS-Security specification. In Figure 3, each arrow represents a possible communication path

[65] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss.
[66] http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/os/ws-trust-1.4-spec-os.pdf

between the participants. Each participant has its own policies which combine to determine the security tokens and associated claims required to communicate along a particular path.

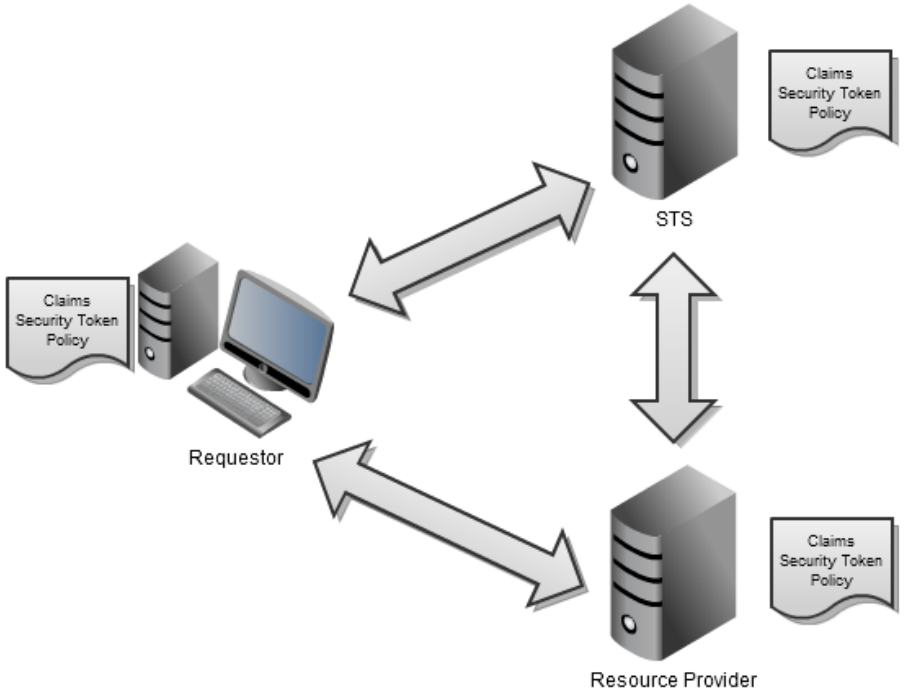

Figure 3 Security Token Service

WS-SecureConversation [67] defines extensions that include session key derivation and security context establishment/sharing. This allows contexts to be established and potentially more efficient keys or new key material to be exchanged, thereby increasing the overall performance and security of the subsequent exchanges.

WS-SecurePolicy [68] extends the WS-Security, WS-Trust and WS-SecureConversation security protocols by offering a specification for policy-based web services .WS-SecurePolicy defines XML based policies that are called security policy assertions. These policies allow web services to express their constraints and requirements.

WS-Federation [69] defines federation mechanisms by extending WS-Trust, WS-Security and WS-SecurePolicy. A fundamental goal of WS-Federation is to simplify the development of federated services through cross-realm communication and management of federation services by re-using the WS-Trust Security Token Service model and protocol.

7.1.4 OpenID

[67] http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.4/os/ws-secureconversation-1.4-spec-os.pdf
[68] http://docs.oasis-open.org/ws-sx/ws-securitypolicy/v1.3/os/ws-securitypolicy-1.3-spec-os.pdf
[69] http://docs.oasis-open.org/wsfed/federation/v1.2/os/ws-federation-1.2-spec-os.pdf

OpenID (see the OpenID foundation's website fot he specifications [70]) is a lightweight HTTP-based protocol for Single Sign-On and attribute exchange. The OpenID specification is widely adapted and implemented by internet companies that have large user bases like Google, Yahoo, WordPress and Facebook.

OpenID Authentication (see the specification at its website [71]) uses only standard HTTP(S) requests and responses, no special capabilities are required from the user client. OpenID is not tied to the use of cookies or any other specific mechanism of consumer or OpenID Provider session management. The OpenID Authentication protocol messages are a defined fixed set of key-value pairs which are included in the HTTP(S) requests as HTTP parameters.

The main OpenID authentication component has some extensions:

- *OpenID Simple Registration* [72] allows very light-weight profile exchange.
- *OpenID Attribute Exchange* [73] defines how to exchange identity information between endpoints through attributes.
- *OpenID Authentication Policy* [74] allows a consumer to request that particular authentication policies are applied by the OpenID provider when authenticating an end user.

OpenID does not define explicit trust between the OpenID authentication provider and the consumer. OpenID only ensures identification, that the given person is the one he claimed to be during his previous visit. It enables a consumer to re-identify the user but gives no guarantee about the user itself because of the lack trust between consumer and provider. Anyone can use any provider to identify himself or even set up his/her own one. If trust is required, a consumer could require the user to use a specific provider, i.e. Google.

7.1.5 PKIX

PKIX is a standard, specified by the IETF's Public-Key Infrastructure working group, describing a public key infrastructure. It specifies public key certificates, certificate revocation lists, attribute certificates, and a certification path validation algorithm. PKIX is a derivation of the X.509 (73) standard in order to adapt it to the more specific domain of internet standards. The term X.509 certificate usually refers to the IETF's PKIX Certificate and CRL Profile of the X.509 v3 certificate standard.

X.509 public key certificates bind a certain identity to a public key. It identifies the holder and can be used to authenticate him. Public key certificates typically last for a long time (several years). Authorisation information can be placed in a public key certificate through the available X.509 extension mechanism. It is usually not desirable though to place authorisation information in a public key certificate. Authorisation information typically has a short lifetime and the authority issuing public key certificates usually is not authoritative for the authorisation information. Authorisation information can be separated by putting it in an attribute certificate. An attribute certificate is a

[70] http://openid.net/developers/specs/
[71] http://openid.net/specs/openid-authentication-2_0.html
[72] http://openid.net/specs/openid-simple-registration-extension-1_0.html
[73] http://openid.net/specs/openid-attribute-exchange-1_0.html
[74] http://openid.net/specs/openid-provider-authentication-policy-extension-1_0.html

digitally signed identity with a set of attributes. It does not have a public key and typically does not last as long as a public key certificate. The PKIX standard is mainly used for authentication though and less commonly for authorization purposes (74).

The main building blocks in the PKIX system are Central authorities (CA's), these are entities that can issue certificates. Each of these entities contains a signed certificate(s) with public/private key-pair. A CA signs every new issued certificate with his private key and includes a reference to his own certificate. In this way every certificate can be verified by using the public key of the CA that issued it. An issued certificate signed by a CA can also be used as base certificate of a new CA. It is clear that in this way a structured system can be build of CA's and end entities (base certificates that are not used as base for a CA). These structures can range from very simple, hierarchical topologies to complex topologies such as mesh architectures.

The certification path validation algorithm [75] is the algorithm which verifies that a given certificate path is valid under a given PKI. A path starts with the end entity and proceeds through a number of intermediate certificates up to a trusted root certificate, typically issued by a trusted certification authority. Path validation is necessary for a relying party to make an informed trust decision when presented with any certificate that is not already explicitly trusted.

The revocation of certificates is specified in the Certificate Revocation List (CRL) profile. The main concept behind CRL is the generation of a blacklist containing a list of all revoked certificates.

Although PKIX certificates are widely spread there are some issues. As explained before PKIX is a derivation of X.509, a standard from 1988, some of the design decisions are not compatible with the current widely adopted computer (internet) standards. The CRL system for example is not a good design solution to revoke certificates; it slows down the overall performance of the PKI infrastructure. Because of this many of the X.509 implementations turn off the revocation checks. Another example is that the X.509 specification is over-functional and underspecified and the normative information is spread across many documents from different standardization bodies.

7.1.6 CAS

CAS (see the *Central Authentication Service* documentation, version 1.0 at Jasig's website [76]) is a centralized ticked based single sign-on HTTP-based protocol. Unlike most of the Single Sign-On systems, CAS lacks the use of attributes.

When a user visits a website requiring CAS authentication, the site redirects the user to a CAS server. This is the central component in the CAS architecture and is responsible for authenticating users, issuing security tickets and validating these tickets. Once the user has successfully authenticated him/herself, the CAS server redirects the user back to the website passing along with a security ticket. The website validates this security ticket by contacting the CAS server. If the validation succeeds, the user is successfully authenticated on the website.

[75] http://en.wikipedia.org/wiki/Certification_path_validation_algorithm
[76] http://www.jasig.org/cas/protocol

CAS 2.0 added support for proxiable credentials which allows a user to be authenticated on back-end services of a website through proxy tickets.

7.1.7 Kerberos

Kerberos (75) can be seen as the first widely distributed Single Sign-On system. It is a ticket oriented system that allows a user to authenticate him/herself in a non-secure network domain. The authentication is mutual, both the user and the server verify each other's identity. It is based on symmetric key cryptography and requires a trusted third party. Kerberos offers sufficient protection facilities against eavesdropping and replay attacks. The user credentials are not communicated over the untrusted network. The Kerberos system demands that the users are trusted else the Kerberos system is at risk of being compromised.

7.1.8 Discussion and Relevance for INTEGRATE

The INTEGRATE platform will provide a range of different services that demand from a user some kind of identification. Implementing on each service a local identification component is not sustainable in an extendable platform. A central identification management system provides a better solution. This system should implement one or more of the standards that were explained in the previous sections to be compliant to the general industry solutions.

7.2 Authorization

7.2.1 Attribute-based Authorization

In an attribute-based authorisation model, identity information on a subject is exchanged from one site to another site in support of some action. The identity information exchanged will rather be information on some characteristic of the subject relevant to the action performed (i.e. the role of the subject in a given scenario), opposed to information exchanged during authentication on who, when and how the subject authenticated. To exchange this authorisation information the same languages can be used as for exchanging authentication attributes (i.e. SAML, WS-*, Liberty, X.509).

7.2.2 Oauth

Oauth (76) is an open standard for authorisation. It allows a resource owner to grant access to his/her private resources on one site (which is called the server), to another site (called client) without the need to share his/her personal credentials. Instead OAuth uses shared tokens which can be limited in time and/or scope. As such the resource owner can give different clients different levels of access. The OAuth protocol messages are a defined fixed set of key-value pairs which are included in the HTTP(S) requests as HTTP parameters.

7.2.3 XACML

XACML [77] (full name: *eXtensible Access Control Markup Language*) is a XML based declarative access control policy language defining both a policy, decision request and decision response language. It is based on the Attribute Based Access Control (ABAC) model which incorporates Role Based Access Control (RBAC). Currently version 2.0 of XACML is adopted, the planned 3.0 version (which is currently in first draft) will support a broad range of new features [78]. The following paragraphs use version 2.0 as reference unless otherwise stated.

Central in the XACML architecture stand two main components: the Policy Enforcement Point (PEP) and the Policy Decision Point (PDP). The PEP intercepts access requests by a subject on one or more protected resources on a server. After the PEP has intercepted such an access request, it generates a decision request based on the attributes of the subject, the resource in question, the performed action and other information pertaining to the request. This decision request is send to the PDP, a component responsible for making access decisions. For making an access decision the PDP interprets the incoming decision request and searches for policies (known to the PDP) that apply to the request. The search for matching policies can be an overhead in the XACML system, because for each decision request, every policy available to the system is checked (no indexing mechanism available). After an access decision is made, based on the access rules in the policies, the PDP generates a decision response and sends it back to the PEP. The PEP uses this response to decide if access is granted to the protected resource or not.

The XACML policy language is designed to offer great interoperability with different platforms and extensibility. The smallest element in the language is the rule element. Each rule targets a set of decision requests to which it is intended to apply. The rule target is expressed in form of a logical expression on attributes of the request. A rule also has a condition element that refines even more the set of decision requests that are targeted. If for the given target and condition a set of decision requests is found, the rule responses with a defined effect value that permits or denies access to the selected set of decision requests. Rules can not exists on there own, they have to be combined in a policy element. This policy element groups rules and combines the access decisions (permit or deny), that were evaluated for the targeted set of decision requests, to a general policy access decision using a combining algorithm. A policy element can target a set of decision requests. Finally there is an optional third element called policyset. This element can group policy elements and/or other policysets. Like the policy element it has a combining algorithm to generate a general policyset access decision. The policyset can target a set of decision requests. Policies and Policysets can contain obligations, which contain a set of operations that must be performed by the PEP in conjunction with an authorisation decision**.**

XACML 3.0 adds a new important concept to the policy language: delegation. It permits some users to create policies of limited duration to delegate certain capabilities to others.

7.2.4 PERMIS

[77] http://www.oasis-open.org/committees/xacml/
[78] http://www.webfarmr.eu/2010/07/enhancements-and-new-features-in-xacml-3-axiomatics/

PERMIS (77) (*PrivilEge and Role Management Infrastructure Standards*) is an authorisation infrastructure that is based on two underlying technologies: role based access control (RBAC) and Policy based Management. RBAC allows to group all users who use the infrastructure into roles (or attributes). Each role (or attribute) is associated with a collection of privileges. A user's membership of a role will allow the user to exercise the privileges associated with the role. The policy based management is a collection of rules that specify the authorization criteria for the users.

7.2.5 Ponder

Ponder (78) is a declarative, object-oriented language for specifying different types of policies, grouping policies into roles and relationships and finally defining configurations of roles and relationships as management structures. Ponder can be used to specify security policies with role-based access control, as well as general-purpose management policies. Ponder2 [79] is the successor of the Ponder policy language. It corrects some of the main disadvantages of Ponder making the framework easier to deploy, make it more accessible for small devices and improve the collaboration/interaction/federation between the policy execution components.

Key concepts of the language include:

- Domains, which provide a means of grouping resources to which policies apply and can be used to partition the resources in a large system according to geographical boundaries, resource type, responsibility and authority or for the convenience of human managers. Membership of a domain is explicit and not defined in terms of a predicate on resource attributes. A domain does not encapsulate the resources it contains but merely holds references to resource interfaces.
- Roles, which group the policies relating to a position in an organisation (like service administrator, service operator, etc.)
- Relationships, which define interactions between roles and management structures to define a configuration of roles and relationships pertaining to an organisational unit such as a department.

Ponder supports, as said before, an extensible range of policy types.

- Authorisation policies are essentially security policies related to access-control and specify what activities a subject is permitted or forbidden to do to a set of target resources.
- Obligation policies specify what activities a subject must do to a set of target resources and define the duties of the policy subject. Obligation policies are triggered by events and are normally interpreted by a manager agent at the subject.
- Refrain policies specify what a subject must refrain from doing and are similar to negative authorisation policies but are interpreted by the subject.
- Delegation policies specify which actions subjects are allowed to delegate to others. A delegation policy thus specifies an authorisation to delegate.
- Composite policies are used to group a set of related policy specifications within a syntactic scope with shared declarations in order to simplify the policy

[79] http://ponder2.net/

specification task for large distributed systems. Four types of composite policies are provided: groups, roles, relationships and management structures.

7.2.6 Cassandra

Cassandra (79) is a role-based policy specification for access control in a distributed system in which the expressiveness (and the computational complexity) can be tuned according to need by choosing an appropriate constraint domain. It is designed to satisfy complex policy requirements and at the same time be simple enough to express access control semantics that can be defined in a formal way (trade-off between expressiveness and decidability). This means that, depending on the application's requirements, a suitable constraint domain can be used in the Cassandra system without the need of changing the semantics.

The language is based on a clear mathematical foundation, more precisely Datalog with constraints (subset of Prolog) and uses five special predicates to express a wide range of policies including role hierarchy, role delegation, separation of duties, cascading revocation, automatic credential discovery and trust negotiation. It is policy neutral meaning it can express subtle variants of well-known policy idioms. The formal specification for Cassandra does not only include policy language semantics but also operational semantics. A goal-oriented distributed policy evaluation algorithm is used to guarantee termination.

7.2.7 Discussion and Relevance for INTEGRATE

Strict policies are defined by official medical instances to ensure that the confidential medical data can be accessed and manipulated in a secure way. It is clear that a well defined access control system is essential in the INTEGRATE platform. This ensures that a user only can see and manipulate the resources that he is allowed to. To be compliant to the industry standards, INTEGRATE should use one or more of the standards discussed above.

7.3 Data Transfer Security

7.3.1 TLS/SSL

A reliable protocol for securing data transfer is SSL (*Secure Socket Layer* [80]) together with his successor TLS (*Transport Layer Security* (80)). They both offer services to secure the packets of the application layer of the OSI-model [81].These services provide authentication, message integrity and confidentiality over an insecure network. TLS (SSL) is commonly used in the HTTPS protocol and for tunnelling in VPN.

To set up a secure connection the protocol uses several steps called the handshake procedure. First the server and client that want to exchange sensitive data over TLS (SSL) negotiate the cipher suite (this are the cypher and hash algorithms) that will be used by both parties as base for encryption and hashing of the data. After this step the server sends a digital certificate containing the server name, the CA that signed the certificate (to verify the certificate can be trusted) and the public encryption key of the

[80] http://www.mozilla.org/projects/security/pki/nss/ssl/draft302.txt
[81] http://en.wikipedia.org/wiki/OSI_model

server. After the client has checked the validity of the certificate, the client encrypts a random number with the public key of the server and sends it to the server (which can only be decrypted by the server's private key). Finally both client and server generate session keys, using the random number as base, that will be used for the secured connection. The client and server can now exchange the sensitive data in a secure way.

7.3.2 IPSec

Next to the well-known SSL-TLS standards there is another protocol that is specified for securing data that needs to be transmitted. IPSec (Internet Protocol Security) (81) secures the packets of the network layer of the OSI-model by encrypting and/or authenticating the IP packages of this layer. This allows network applications to communicate in a secure way on a network without the need of implementing extra network software in their application stack. This application independence is not possible in TLS-SSL systems. The drawback of this is that IPSec is more complex to deploy than TLS-SSL.

IPSec has two main modes of operation:

- **Transport mode**: in this mode only the payload (the data you transfer) of the IP packet is encrypted and/or authenticated.
- **Tunnel model**: here the entire IP packet is encrypted and/or authenticated. It is then encapsulated into a completely new IP packet containing a new IP header. This operation is commonly used in VPN.

The encryption of small packages in the network generates a large computational overhead which can be a problem in real-time systems.

7.3.3 Discussion and Relevance for INTEGRATE

The different services of the INTEGRATE platform exchange resources with each other, over a possible unsecure exchange channel. To guard confidentiality and integrity of the exchanged resources, a secure system like SSL/TLS or IPSec needs to be provided.

7.4 De-identification

When publishing data about individuals, also called microdata, it is important that no sensitive information is revealed about them. Microdata is typically stored in a table, and each record (row) corresponds to one individual. Each record has a number of attributes, which can be divided into the following three categories:

- **Explicit identifiers** are attributes that clearly identify individuals (e.g., social security number, address, name).
- **Quasi-identifiers** are attributes whose values when taken together can potentially identify an individual (e.g., zip-code, birthdate, gender).
- **Sensitive attributes** are attributes that are considered sensitive (e.g. disease, salary).

To avoid the identification of records in microdata, the uniquely identifying information (explicit identifiers) are removed from the table. However, the quasi-identifiers,which

are a set of non-sensitive attributes, can be used for identifying an individual if linked to other public records. These quasi-identifier attributes should be de-identified in the dataset in order to prevent the sensitive information of individuals from being disclosed. So, microdata should be sanitised on two levels before it is published: firstly all explicit identifiers should be removed and secondly all the quasi-identifier attributes should be transformed in such a way that the disclosure risk is limited while maximising the usefulness of the data (82).

There are two types of information disclosure: identity disclosure and attribute disclosure. **Identity disclosure** occurs when in the released data a record is linked to an individual. **Attribute disclosure** occurs when new sensitive information can be extracted from the data about some indivuals. When an identity is revealed, corresponding sensitive attributes are disclosed, often causing attribute disclosure (82).

7.4.1 Transformation Techniques

The process of modifying the original microdata into a form that limits disclosure is called anonymisation. For this, several techniques can be used.

- **Generalisation** consists in substituting the values of a given attribute with more general values. Generalisation allows the data to be modeled in generalisation hierarchies in which the leaves are the actual values of the attribute and the precision of a value is reduced by each move to one higher level. A de-identification algorithm needs to deal with this hierarchical nature of the attributes. For a quasi-identifier consisting of multiple attributes, each with its own domain (e.g. age, zip code, salary), the domain generalisation hierarchies of the individual attributes can be combined to form a multi-attribute generalisation lattice (83).
- **Suppression** removes data from the table to anonymise it. It can be applied at the level of a row by removing a whole record, at the level of an attribute by obscuring all values of an attribute or at the level of a cell by obscuring the attribute value for a specific attribute and record. Removing all values of an attribute is equivalent to generalisation to the root node of the hierarchy.
- **Global recoding** consists in aggregating the values observed in an attribute causing several values of that attribute to be collapsed into a single one.
- **PostRAndomisation Method (PRAM)** is a method to protect categorical attributes (e.g., blood type) from disclosure. By using a known probability mechanisms, the values of a categorical attribute are changed to a new values, which may or may not be different from the original. It is basically a form of deliberate misclassification, wherefore it will be difficult to identify records (with certainty) as corresponding to certain individuals. Since the probability mechanism that is used when applying PRAM is known, characteristics of the true data can still be estimated from the perturbed data file.
- **Microaggregation** aggregates records into groups. Instead of releasing the actual values of sensitive attributes, the mean of the group to which the observation belongs is released. The confidentiality of individual data subjects is protected by ensuring that each group has at least a minimum number of observations.

- **Top- and bottom coding** are particular global recoding techniques. Top coding on numirical attributes groups together all top values. It basically sets an upper limit on all published values of that attribute. Bottom coding groups together all bottom values.
- **Slicing** partitions data vertically into groups of correlated attributes (columns). It then horizontally partitions the data into groups of records. Within each group of records, the rows in each column are randomly permutated to hide the linking between the different columns. Generalisation is often applied to columns to make them satisfy k-anonymity.

The above list is not exclusive, other, lesser discussed, techniques exist such as adding-noise, data-swapping (specific for categorical data), re-sampling (specific for numerical data).

7.4.2 K-anonimity

K-anonymity transforms a data set containing personal information making it difficult to determine the identity of individuals in the data set. Each record in a k-anonymised data set is similar to at least k-1 other records on a given set of quasi-identifiers. Such a group of k records is called an equivalence class. K-anonymity is reached by using transformation techniques such as generalisation, global recoding and suppression. The maximum probability of a record being re-identified would then be 1/k. A higher value of k implies a lower probability of re-identification but also more distortion to the data, and hence greater information loss due to k-anonymization. Excessive anonymisation can make disclosed data less useful to the recipient. Some analysis can become impossible or it can produce biased and incorrect results (84).

By applying generalisation and suppression a generalisation T can be derived from a microdata set V. If such a generalisation satisfies k-anonymity, if it does remove more data then necessary (does not enforce more suppression then allowed) and if there is not another generalisation wich is less general, then the generalisation T is k-minimal (85). Different algorithms exist to find such a k-minimal generalisation. An extensive study comparing several of those algorithms (Optimal Lattice Anonymisation, Incognito (86), Datafly, Samarati) has been made (87).

As a record is similar, for a given set of quasi-identifiers, to at least k-1 other records, no identity can be linked to groups of less than k individuals preventing identity disclosure. However, information can be leaked through attribute disclosure if an equivalence class lacks diversity in the value of a sensitive attribute.

7.4.3 Diversity

l-Diversity tries to provide stronger privacy guarantees than k-anonymity by making sure that sensitive attributes are diverse enough in the groups, equivalence classes, created by k-anonymity. An equivalence class has l-diversity if the dataset contains at least *l* "well-represented values on the sensitive attribute. If all the equivalence classes in a table, created by anonymising some non-sensitive attributes in a dataset, have l-diversity, then the table is said to have l-diversity. It ensures that for a positive disclosure an attacker needs at least l-1 pieces of damaging background knowledge to eliminate l-1 possible sensitive values. By setting this parameter l, the data publisher

can determine how much protection is provided against background knowledge of sensitive values, even if the publisher doesn't know this background knowledge (88).

There exists different variants of l-diversity, based on the interpretation given to the term "well-represented".

- **Distinct l-diversity** is the most general form of l-diversity and foresees that there are at least *l* distinct values for the sensitive attributes in any possible equivalence class.
- **Entropy l-diversity** suggests that there are enough different sensitive values for each equivalence class and that these different sensitive values should be distributed evenly enough. The entropy of the distribution of sensitive values in each equivalence class is at least log(l).
- **Recursive (c, l)-diversity** restricts the less frequent values to not appear too rarely and similarly the most frequent values to not appear too frequently.

l-Diversity possesses the monotonicity property. If a generalisation of a table T (T*) satisfies l-diversity, then any generalisation T** of T* will aslo satisfy l-diversity. Because k-anonymity also possesses the monotonicity property it is possible to modify k-anonymity algorithms (e.g. Incognito, Samarati) to calculate the minimal generalisation. Instead of checking for k-anonymity every time a generalisation is tested, you would then test for l-diversity.

l-Diversity has a few limitations, namely:

- It can be unnecessary to achieve. A single sensitive attribute can have values that have a very different degree of sensitivity. An HIV positive result for example is much more rare than an HIV negative result, meaning that one would not mind being known as having an HIV-negative result as most of the population probably has this result. Therefore it would be unnecessary to achieve 2-diversity for an equivalence class that contains only negative records.
- l-Diversity can be difficult to achieve. To achieve for example distinct l-diversity for a population of 100000 records and 0,2% HIV positives (100000 * 0,2% = 20) there can be at most 20 equivalence classes. This means large generalizations, which will cause a lot of information loss (82).
- l-Diversity also isn't sufficient to prevent attribute disclosure (new information about an individual can be revealed).
    - As l-diversity doesn't consider the overall distribution of a sensitive attribute, a sensitive attribute can be disclosed when the overall distribution of a sensitive attribute is skewed (**skewness attack**). When there is in example, in the HIV case, an equivalence class with an equal number of positive and negative records then anyone in that class has a 50% possibility of being positive.
    - When the values of a sensitive attribute in an equivalence class are distinct, but semantically similar, attributes can also be disclosed (**similartiy attack**). An equivalence class for example could have all kinds of stomach-related disease values for the sensitive attribute "disease". Even though an attacker doesn't know which specific disease somebody in that equivalence class might have, he can still infer the fact that is is stomach related.

7.4.4 t-Closeness

t-Closeness tries to overcome the limitations and shortcomings of l-diversity and k-anonymity. T-Closeness requires that the distribution (**P**) of a sensitive attribute in an equivalence class is *close* to the distribution (**Q**) of that attribute in the overall table. A threshold *t* determines the maximum distance between these two distributions P and Q. In other words P and Q have to be within a certain proximity/closeness (*t*) of each other. A table is said to have t-closeness if all equivalence classes have t-closeness. The t parameter enables one to trade off between utility and privacy of the released data. If *t* is too small, then the useful information about the correlation between the quasi-identifier attributes and the sensitive attributes will be lost, hereby removing some information from the released data. On the other hand if *t* is a bit too big, this correlation can enable attribute disclosure.

The distance between the distributions P and Q is expressed by using the *Earth Mover's Distance* (EMD) [82]. The EMD is based on the minimal amount of work needed to transform one distribution to another by moving distribution mass between each other. EMD captures semantic distance (i.e., ground distance) well. EMD extends the notion of a distance between single elements to that of a distance between sets, or distributions of elements (89).

7.4.5 Distribution-based Microdata Anonimization

k-Anonymity, l-diversity and t-closeness achieve privacy through record grouping. They essentially specify constraints on the sensitive attribute distribution of a group of records. The anonymisation problem is thus to transform the original microdata table to groups so that the sensitive attribute distribution of each group satisfies the privacy goal. However there may not always exist groupings that satisfy the privacy goal and the data owner has no control over how the final grouping will look. As such there is a conflict between the desirable grouping for microdata analysis (i.e. by age group) of the data owner and the grouping for privacy protection. This is caused by the fact that these anonymisation approaches tightly couple grouping with the achieval of privacy goals (90).

Distribution transformation tries to cleary decouple grouping from privacy goals by modifying the sensitive attribute values in arbitrary record groups so that it is only possible to infer from the anonymised data that for each record its sensitive attribute in the group follows the target distribution and nothing more.

The basic intend is to generalise and permute the sensitive attribute values, leaving quasi-identifiers unchanged. This allows different target distributions (e.g., P1 and P2) for different groups (e.g., g1 and g2), making it possible that both distributions P1 and P2 allow l-diversity or t-closeness for their respective groups but are different such that each is closer to the original distribution. This reduces data distortion in the resulting table. Introducing fake values, for the sensitive attribute into the different groups, can yield even better results. The addition of fake values can reduce the level of generalisations needed to match the target distribution. However, the introduction of

[82] http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/RUBNER/emd.htm

fake values needs to happen in a controlled way, as adding too many fake values can influence the generalisation too much, which would result in a too distorted result table.

7.4.6 Discussion and Relevance for INTEGRATE

De-identification and pseudonymisation play a central role in the INTEGRATE platform. All sensitive privacy information of a patient should be keep hidden for unauthorised users. Using t-closeness provides a secure way of sharing the patient resources without revealing the identity of the patient.

8 Conclusion

This document reported on a survey of the state-of-the-art standards relevant to INTEGRATE. The survey covered several technologies in the form of methods, techniques, languages, ontologies and computer software systems from a variety of research and technology areas that included the *Semantic Web*, *Privacy and Security* and the *Biomedical and Life Science* domains.

From the Semantic Web research area we focused our attention on four areas of interest that contribute technologies for addressing different aspects of semantic data and knowledge management. These included *ontology representation and query languages*, *semantic repositories*, *automated reasoners* and *ontology mediation*. One of the key conlusions drawn from the survey of these technologies is the lack of standards in many areas which could be attributed to the relative young age of the semantic Web community. Although the W3C sponsors several standards in the area of ontology languages and core semantic technologies the story is rather different when it comes to semantic repositories, automated reasoners and ontology mediation techniques. Although the area is timing with techniques, methods and approaches for addressing different aspects of knowledge management the tools and systems implementing these are far from becoming standards. It is important to notice, however, that many of these tools have become the defacto standard for building semantic solutions in the semantic community and outside of it as well. This is specially reflected in the area of semantic repositories and reasoners where systems such as OWLIM, Virtuoso, Sesame and AllegroGraph have positioned themsleves as the four defacto standards for managing RDF-based data. In a similar way, reasoners such as Pellet, RACER (Pro) and FaCT++ have become the defacto choice for reasoning with ontologies. In the area of semantic reasoning it is also important to highlight the newly-developed family of efficient classifiers for dealing with ontologies in the Life Sciences. These differentiate themselves from the “classic”, general-purpose reasoners such as Pellet, FaCT++, HermiT, RACER (Pro), etc., in that they exploit the characteristics of the type of ontologies found in the Life Sciences and Medical domains in order to provide a more efficient classification service albeit with the lack of other important reasoning services. Finally, the landscape of *ontology mediation, alignment and merging* shares the same characteristic in terms of the lack of standards. Although techniques and methods abound systems and tools have not become standards; although some of them do share more popularity among different communities. Another important conlusion drawn from this survey is with respect to the selection of tools and systems. The conlusion is that no system (reasoners, matchers and repositories) seems to outperform all the other systems, although many have positioned themselves obe level higher than the rest in terms of features. Consequently, although a series of characteristics were identified as the main factors influencing the selection of systems the ultimate decision will depend on a carefull analysis of the application requirements, the characteristics of the tools and an empirical analysis of the performance of the tools. In this respect, the selection of systems willbe done based on an analysis of the performance of the systems which will be reported in Deliverable 2.2.

In the field of Security and Privacy this survey focused on four main areas, namely *identity management*, *authorization*, *data transfer security* and *de-identification*. With respect to identity management the main conclusion is that implementing a local

identification component in each service is not sustainable in an extendable platform. Instead, a better approach is a central identification management system. With respect to authorization, it is clear that a well defined access control system is essential in the INTEGRATE platform. This ensures that a user only can see and manipulate the resources that he is allowed to. To be compliant to the industry standards, INTEGRATE should use one or more of the standards discussed above. In what regards to data transfer security, the main conclusion is that in order to guard confidentiality and integrity of the exchanged resources, a secure system like SSL/TLS or IPSec needs to be provided within the INTEGRATE platform. Finally, de-identification and pseudonymisation play a central role in the INTEGRATE platform. All sensitive privacy information of a patient should be keep hidden for unauthorised users. Using t-closeness provides a secure way of sharing the patient resources without revealing the identity of the patient

Finally, this document also surveyed the landscpae of ontologies relevant for the project, covering standard ontologies such as SNOMED CT, LOINC, MeDRA, MeSH, ICD-10 and UMLS. All of them cover different areas within the biomedical domain and promise to serve as the basis for building the solution envisioned by the project.

9 Bibliography

1. **F. Manola, E. Miller.** RDF Primer. *W3C.* [Online] February 10, 2004. http://www.w3.org/TR/rdf-primer/.
2. **G. Klyne, J. Carroll.** Resource Description Framework (RDF): Concepts and Abstract Syntax. *W3C.* [Online] February 10, 2004. http://www.w3.org/TR/rdf-concepts/.
3. **D. Beckett, T. Berners-Lee.** Turtle - Terse RDF Triple Language. *W3C.* [Online] March 28, 2011. http://www.w3.org/TeamSubmission/turtle/.
4. **D. Brickley, R.V. Guha.** RDF Vocabulary Description Language 1.0: RDF Schema. *W3C.* [Online] February 10, 2004. http://www.w3.org/TR/rdf-schema/.
5. **M. Dean, G. Schreiber.** OWL Web Ontology Language Reference. *W3C.* [Online] February 10, 2004.
6. **F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P. Patel-Schneider.** *The Description Logic Handbook. Theory, Implementation and Applications.* s.l. : Cambridge University Press, 2003.
7. *Combining RDF and Part of OWL with Rules: Semantics, Decidability, Complexity.* **Horst, H.J. ter.** Galway, Ireland : s.n., 2005. ISWC. Vol. LNCS 3729, pp. 668–684.
8. *OWLRules: A Proposal and Prototype Implementation.* **I. Horrocks, P. Patel-Schneider, S. Bechhofer, D. Tsarkov.** 2005, Journal of Web Semantics, pp. 23-40.
9. **P. Hitzler, M. Krotzsch, B. Parsia, P. Patel-Schneider, S. Rudolph.** OWL 2 Web Ontology Language Primer. *W3C.* [Online] October 29, 2009. http://www.w3.org/TR/owl-primer/.
10. *Pushing the EL envelope.* **F. Baader, S. Brandt, C. Lutz.** 2005. 19th International Joint Conference on Artificial Intelligence (IJCAI).
11. **E. Prud'hommeaux, A. Seaborne.** SPARQL Query Language for RDF. *W3C.* [Online] January 15, 2008. http://www.w3.org/TR/rdf-sparql-query/.
12. *OWLIM: A family of scalable semantic repositories.* **B. Bishop, A. Kiryakov, D. Ognyanoff, I. Peikov, Z. Tashev, R. Velkov.** 1, 2011, Semantic Web – Interoperability, Usability, Applicability, Vol. 2, pp. 33-42.
13. **D. Brickley, R.V. Guha.** RDF Vocabulary Description Language 1.0: RDF Schema. *World Wide Web Consortium.* [Online] February 10, 2004. http://www.w3.org/TR/rdf-schema/.
14. **S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P. F. Patel-Schneider, L.A. Stein.** OWL Web Ontology Language Reference. *World Wide Web Consortium.* [Online] February 10, 2004. http://www.w3.org/TR/owl-ref/.
15. **Motik, B., et al.** OWL 2 Web Ontology Language Profiles. *World Wide Web Consortium.* [Online] October 27, 2009. http://www.w3.org/TR/owl2-profiles/.
16. *Pellet: A practical OWL-DL reasoner.* **E. Sirin, B. Parsia, B. Cuenca Grau, A. Kalyanpur, Y. Katz.** 2, 2007, Web Semantics: Science, Services and Agents on the World Wide Web, Vol. 5, pp. 51-53.
17. **I. Horrocks, P.F. Pate-Schneider, H. Boley, S. Tabet, B. Grosof, M. Dean.** SWRL: A Semantic Web Rule Language Combining OWL and RuleML. *World Wide Web Consortium.* [Online] May 21, 2004. http://www.w3.org/Submission/SWRL/.
18. **E.Prud'hommeaux, A Seaborne.** SPARQL Query Language for RDF. *World Wide Web Consortium.* [Online] January 15, 2008. http://www.w3.org/TR/rdf-sparql-query/.
19. **Seaborne, A.** RDQL - A Query Language for RDF. *World Wide Web Consortium.* [Online] January 9, 2004. http://www.w3.org/Submission/RDQL/.

20. *FaCT++ description logic reasoner: System description.* **D. Tsarkov, I. Horrocks.** 2006. Int. Joint Conference on Automated Reasoning (IJCAR). pp. 292-297.
21. **Horrocks, I.** *The Description Logic Handbook: Theory, Implementation and Applications.* [ed.] D. Calvanese, D. McGuinness, D. Nardi, P. F. Patel-Schneider F. Baader. s.l. : Cambridge University Press, 2002.
22. **Motik, B.** *Reasoning in Description Logics using Resolution and Deductive Databases.* s.l. : PhD Thesis, 2006.
23. *HermiT: A Highly-Efficient OWL Reasoner.* **R. Shearer, B. Motik, I. Horrocks.** 2008. 5th Int. Workshop on OWL: Experiences and Directions.
24. *Hypertableau Reasoning for Description Logics.* **B. Motik, R. Shearer, I. Horrocks.** 2009, Journal of Artificial Intelligence Research, Vol. 36, pp. 165-228.
25. *Structured Objects in OWL: Representation and Reasoning.* **B. Motik, B. Cuenca Grau, U. Sattler.** 2008. 17th International WWW Conference. pp. 555–564.
26. *CEL: A Polynomial-time Reasoner for Life Science Ontologies.* **F. Baader, C. Lutz, B. Suntisrivaraporn.** [ed.] N. Shankar U. Furbach. s.l. : Springer-Verlag, 2006. IJCAR. Vol. LNAI 4130, pp. 287-291.
27. *Is tractable reasoning in extensions of the description logic EL useful in practice?* **F. Baader, C. Lutz, B. Suntisrivaraporn.** 2005. International Workshop on Methods for Modalities (M4M-05).
28. *Reintroducing CEL as an OWL 2 EL Reasoner.* **J. Mendez, B. Suntisrivaraporn.** 2009. 22nd International Workshop on Description Logics. Vol. 477.
29. *Fast Classification in Protege: Snorocket as an OWL2 EL Reasoner.* **M. Lawley, C. Bousquet.** 2010. Australasian Ontology Workshop.
30. *TrOWL: Tractable OWL 2 Reasoning Infrastructure.* **E. Thomas, J. Pan, Y. Ren.** 2010. Extended Semantic Web Conference (ESWC).
31. *Forgetting in managing rules and ontologies.* **T. Eiter, G. Ianni, R. Schindlauer, H. Tompits.** s.l. : IEEE Computer Society, 2006. IEEE/WIC/ACM International Conference on Web Intelligence (WI). pp. 411-419.
32. *Forgetting in DL-Lite.* **Z. Wang, K. Wang, R. Topor, J. Pan.** 2008. 5th European Semantic Web Conference (ESWC).
33. *Towards LarKC: a Platform for Web-scale Reasoning.* **D. Fensel, F. van Harmelen, B. Andersson, P. Brennan, H. Cunningham, E. Della Valle, F. Fischer, Z. Huang and A. Kiryakov, T. Lee, L. School, V. Tresp, S. Wesner, M. Witbrock, N. Zhong.** 2008. Proceedings of the IEEE International Conference on Semantic Computing (ICSC).
34. *Comparison of Reasoners for large Ontologies in the OWL 2 EL Profile.* **K. Dentler, R. Cornet, A. Ten Teije, N. De Keizer.** s.l. : IOS Press, 2011, Semantic Web Journal, Vol. 1, pp. 1-5.
35. *Benchmarking OWL reasoners.* **J. Bock, P. Haase, Q. Ji, and R. Volz.** 2008. Workshop on Advancing Reasoning on the Web: Scalability and Commonsense.
36. **Liebig, T.** *Reasoning with OWL - System Support and Insights.* Ulm University. 2006. TR-2006-04.
37. *Framework for an Automated Comparison of Description Logic Reasoners.* **T. Gardiner, D. Tsarkov, I. Horrocks.** s.l. : Springer, 2006. Proceedings of the International Semantic Web Conference (ISWC). Vol. 4273, pp. 654–667.
38. **J. de Bruijn, M. Ehrig, C. Feier, F. Martin-Recuerda, F. Scharffe, M. Weiten.** Ontology Mediation, Merging and Alignment. [book auth.] R. Studer, P. Warren J. Davies. *Semantic Web Technologies: Trends and Research in Ontology-based Systems.* s.l. : John Wiley & Sons, 2006.
39. *Ontology Mapping: The State of the Art.* **Y. Kalfoglou, M. Schorlemmer.** Dagstuhl Seminar Proceedings. Semantic Interoperability and Integration.

40. *SMART: Automated Support for Ontology Merging and Aligning.* **N.F. Noy, M. Musen.** 1999. Proceedings of the 12th Workshop on Knowledge Acquisition, Modelling and Management (KAW).
41. *PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment.* **N.F. Noy, M. Musen.** 2000. Proceedings of the 17th Interational Conference on Artificial Intelligence (AAAI).
42. *PROMPTDIFF: A Fixed-Point Algorithm for Comparing Ontology Versions.* **N.F. Noy, M. Musen.** 2002. Proceedings of the 18th National Conference on Artificial Intelligence (AAAI).
43. *Ontology Merging for Federated Ontologies on the Semantic Web.* **A. Maedche, G. Stumme.** 2001. Proceedings of the International Workshop for Foundations of Models for Information Integration (FMII).
44. *Learning to map between ontologies on the semantic web.* **A. Doan, J. Madhavan, P. Domingos, A. Halevy.** 2002. Proceedings of the 11th International World Wide Web Conference (WWW).
45. *A proposal for word sense disambiguation using conceptual distance.* **E. Agirre, G. Rigau.** 1997. Amsterdam Studies in the Theory and History of Linguistic Science Series. pp. 161-172.
46. *Sentence similarity based on semantic nets and corpus statistics.* **Y. Li, D. McLean, Z. Bandar, J. O'Shea, K. Crockett.** 8, 2006, IEEE Transactions on Knowledge and Data Engineering, Vol. 18, pp. 1138-1150.
47. *Using information content to evaluate semantic similarity in a taxonomy.* **Resnik, P.** 1995. International Joint Conference on Artificial Intelligence (IJCAI). Vol. 14, pp. 448-453.
48. *Ontology matching with semantic verification.* **Y. Jean-Mary, E. Shironoshita, M. Kabuka.** 2009, Web Semantics: Science, Services and Agents on the World Wide Web, pp. 235-251.
49. *Similarity Computation by Ontology Merging System: DKPOM.* **Fahad, M. A. Q. Muhammad.** 2009, Computer, Control and Communication, pp. 17-18.
50. *Correlation between gene expression and GO semantic similarity.* **J. Sevilla, V. Segura, A. Podhorski, E. Guruceaga, J. Mato, L. Martinez-Cruz, F. Corrales, A. Rubio.** 4, 2005, IEEE/ACM Transactions on Computational Biology and Bioinformatics, Vol. 2.
51. *Using Bayesian decision for ontology mapping.* **J. Tang, J. Li, B. Liang, X. Huang, Y. Li, K. Wang.** 2006, Journal of Web Semantics: Science, Services and Agents on the World Wide Web, pp. 243–262.
52. **J. Euzenat, P. Shvaiko.** *Ontology Matching.* s.l. : Springer-Verlag, 2007.
53. *SAMBO - A System for Aligning and Merging Biomedical Ontologies.* **P. Lambrix, H. Tan.** 3, 2006, Journal of Web Semantics, Special issue on Semantic Web for the Life Sciences, Vol. 4, pp. 196-206.
54. *Ontology matching with semantic verification.* **Y. R. Jean-Marya, E. P. Shironoshitaa, M. R. Kabuka.** 3, 2009, Journal of Web Semantics: Science, Services and Agents on the World Wide Web, Vol. 7, pp. 235-251.
55. *Eff2Match Results for OAEI 2010.* **W. Wei Khong Chua, K. Jung-Jae.** ISWC Workshop on Ontology Matching. pp. 150-158.
56. *RiMOM: A Dynamic Multistrategy Ontology Alignment Framework.* **J. Li, J. Tang, Y. Li, Q. Luo.** 2009, IEEE Transactions on Knowledge and Data Engineering, Vol. 21.
57. *AgreementMaker: efficient matching for large real-world schemas and ontologies.* **I. F. Cruz, F. Antonelli, C. Stroe.** 2, 2009, Proceedings of the VLDB Endowment, Vol. 2, pp. 1586-1589.

58. *A Visual Tool for Ontology Alignment to Enable Geospatial Interoperability.* **I. F. Cruz, W. Sunna, N. Makar, S. Bathala.** 3, 2007, Journal of Visual Languages and Computing, Vol. 18, pp. 230-254.
59. *Handling Semantic Heterogeneities using Declarative Agreements.* **I. F. Cruz, A. Rajendran, W. Sunna, N. Wiegand.** 2002. International ACM GIS Symposium. pp. 168-174.
60. *Using the AgreementMaker to Align Ontologies for the OAEI Campaign 2007.* **W. Sunna, I. F. Cruz.** 2007. Proceedings of the International Workshop on Ontology Matching (CEUR-WS). Vol. 304, pp. 133-138.
61. *BLOOMS on AgreementMaker results for OAEI 2010.* **C. Pesquita, C. Stroe, I. F. Cruz, F. Couto.** 2010. Proceedings of the 5th International Workshop on Ontology Matching.
62. *BLOOMS on AgreementMaker: results for OAEI 2010.* **C. Pesquita, C. Stroe, I. F. Cruz, F. M. Couto.** 2010. Proceedings of the Fifth International Workshop on Ontology Matching.
63. *Hybrid Alignment Strategy for Anatomical Ontologies: Results of the 2007 Ontology Alignment Contest.* **S. Zhang, O. Bodenreider.** 2007. Proceedings of the 2nd International Workshop on Ontology Matching. Vol. 304.
64. *Of mice and men: Aligning mouse and human anatomies.* **O. Bodenreider, T.F. Hayamizu, M. Ringwald, S. de Coronado, S. Zhang.** 2005. Proceedings of AMIA Symposium. pp. 61-65.
65. *NLM Anatomical Ontology Alignment System: Results of the 2006 Ontology Alignment Contest.* **S. Zhang, O. Bodenreider.** 2006. Proceedings of the OAEI Campaign. pp. 145-156.
66. *Falcon-AO: Aligning Ontologies with Falcon.* **N. Jian, W. Hu, G. Cheng, Y. Qu.** 2005. Proceedings of K-CAP 2005 Workshop on Integrating Ontologies.
67. *Falcon-AO: A practical ontology matching system.* **W. Hu, Y. Qu.** 3, 2008, Journal of Web Semantics: Science, Services and Agents on the World Wide Web, Vol. 6, pp. 237-239.
68. *GMO: A Graph Matching for Ontologies.* **W. Hu, N. Jian, Y. Qu, Y. Wang.** 2005. Proceedings of Workshop on Integrating Ontologies (K-CAP). pp. 43-50.
69. *Alignment Results of SOBOM for OAEI 2010.* **P. Xu, Y. Wang, L. Cheng, T. Zang.** 2010. The Fifth International Workshop on Ontology Matching. pp. 202-211.
70. *Results of NBJLM for OAEI 2010.* **S. Wang, G. Wang, X. Liu.** 2010. The Fifth International Workshop on Ontology Matching. pp. 186-194.
71. **S. Cantor, J. Kemp, R. Philpot, E. Maler.** OASIS. Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0. [Online] 2005. http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf.
72. **J. Hughes, S. Cantor, J. Hodges, F. Hirsch, P. Mishra, R. Philpott, E. Maler.** OASIS. Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0. [Online] http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf.
73. **D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, W. Polk.** RFC 5280. Internet X.509 Public Key Infrastructure Certificate and CRL Profile. *IETF.* [Online] http://tools.ietf.org/html/rfc5280.
74. **S. Farrell, R. Housley.** RFC 3281. An Internet Attribute Certificate Profile for Authorization. *IETF.* [Online] http://www.ietf.org/rfc/rfc3281.txt.
75. **J. Kohl, C. Neuman.** RFC 1510. The Kerberos Network Authentication Service (V5). *IETF.* [Online] 1993. http://tools.ietf.org/html/rfc1510.
76. **Hammer-Lahav, E.** RFC 5849. OAuth (Open Authorization). *IETF.* [Online] 2010. http://tools.ietf.org/html/rfc5849.

77. *PERMIS: a modular authorization infrastructure.* **D. W. Chadwick, G. Zhao, S. Otenko, R. Laborde, L. Su, T. Anh Nguyen.** 11, 2008, Concurrency and Computation: Practice and Experience, Vol. 20.
78. **N. Damianou, N. Dulay, E. Lupu, M. Sloman.** *Ponder: A Language for Specifying Security and Management Policies for Distributed Systems (v2.3).* s.l. : Imperial College of Science, Technology and Medicine, 2000.
79. **Becker, M. Y.** *Cassandra: Fexible trust management and its application to electronic health records.* s.l. : University of Cambridge, Computer Laboratory, 2005.
80. **T. Dierks, E. Rescorla.** RFC 5246. The Transport Layer Security (TLS) Protocol Version 1.2. *IETF.* [Online] 2008. http://tools.ietf.org/html/rfc5246.
81. **S. Kent, K. Seo.** RFC 4301. Security Architecture for the Internet Protocol. *IETF.* [Online] 2005. http://tools.ietf.org/html/rfc4301.
82. **N. Li, T. Li, S. Venkatasubramanian.** *t-Closeness: Privacy Beyond k-Anonymity and L-Diversity.* s.l. : Center for Education and Research Information Assurance and Security, Purdue University, 2007.
83. *An efficient hash-based algorithm for minimal k-anonymity.* **X. Sun, M. Li, H. Wang, A. Plank.** 2008. Proceedings of the thirty-first Australasian conference on Computer science. pp. 101-107.
84. *Protecting Privacy Using k-Anonymity.* **K. El Emam, F. Kamal Dankar.** 5, 2008, Journal of the American Medical Informatics Association, Vol. 15, pp. 627–637.
85. *k-Anonymity.* **V. Ciriani, S. De Capitani di Vimercati, S. Foresti, P. Samarati.** s.l. : Springer-Verlag, 2007. Secure Data Management in Decentralized Systems.
86. *Incognito: efficient full-domain K-anonymity.* **K. LeFevre, D. J. DeWitt, R. Ramakrishnan.** 2005. ACM SIGMOD international conference on Management of data. pp. 49-60.
87. *A globally optimal k-anonymity method for the de-identification of health data.* **K. El Emam, F. Kamal Dankar, R. Issa, E. Jonker, D. Amyot, E. Cogo, J. Corriveau, M. Walker, S. Chowdhury, R. Vaillancourt, T. Roffey, J. Bottomley.** 5, 2009, Journal of the American Medical Informatics Association, Vol. 16, pp. 670–682.
88. *l-diversity: Privacy beyond k-anonymity.* **A. Machanavajjhala, D. Kifer, J. Gehrke, M. Venkitasubramaniam.** 1, 2007, ACM Transactions on Knowledge Discovery from Data, Vol. 1.
89. **Ninghui Li, Tiancheng Li, Suresh Venkatasubramanian.** *t-Closeness: Privacy Beyond k-Anonymity and l-Diversity.* 2007.
90. *Distribution based microdata anonymization.* **N. Koudas, D. Srivastava, T. Yu, Q. Zhang.** 2009. Proceedings of the VLDB Endowment. Vol. 2, pp. 958-969.