FP6 – 027324

TripCom

Triple Space Communication

SPECIFIC TARGETED RESEARCH PROJECT

PRIORITY 2.4.7 Semantic-based Knowledge and Content Systems

D1.1 – State – of – the – Art and Requirements Analysis

Due date of deliverable: 30th June 2006
Actual submission date: 18th June 2006

Start date of project: 1st April 2006 Duration: 36m

Organisation name of lead contractor for this deliverable: NUIG

Short Description:

TripCom Consortium Contacts:

N°	Organisation	Country	Name	Phone N°	Fax N°	E-Mail
1	LFUI	Austria	S. Groppe	+43 512 507 6486	+43 512 507 9872	Sven.Groppe@deri.org
2	NUIG	Ireland	B. Sapkota	+353 91 495150	+353 91 495270	Brahmananda.Sapkota@deri.org
3	ONTO	Bulgaria	V. Momtchev	+359 2 9768 303	+359 2 9768 311	Vassil.Momtchev@ontotext.com
4	PROFIUM	Finland	J. Saarela	+358 9 85598 000	+358 9 855 98 002	Janne.Saarela@profium.com

Document History:

Version	Date	Changes	From	Review
V0.1	April 29, 2006	Initial outline created	NUIG	ONTO
V0.2	May 16, 2006	Initial version created	NUIG	ONTO, LFUI, PROFIUM
V0.3	June 01, 2006	First draft created	NUIG, ONTO, LFUI, PROFIUM	NUIG, ONTO
V0.4	June 03, 2006	Sections 2.2 and 3.1 updated	ONTO	NUIG
V0.5	June 07, 2006	Distributed RDF systems added	ONTO	ONTO
V0.5	June 08, 2006	Comments from FUB addressed	NUIG	ONTO
V0.6	June 14, 2006	Review and proof reading comments implemented	NUIG	ONTO

EXECUTIVE SUMMARY

Deliverable D1.1 – State-of-the-art and requirements analysis is meant to analyse and describe state-of-the-art of common storage architectures in order to design architecture of the core storage component of the TripCom infrastructure for supporting integration of heterogeneous data sources in D1.2.

This document also describes initial requirements, different storage mechanisms, and their scope and roles within TripCom infrastructure.

LIST OF ABBREVIATIONS

ANSI	American National Standards Institute
BSD	Berkeley Software Distribution
DAWG	Data Access Working Group
DBMS	Database Management Systems
ER	Entity Relationship
FOAF	Friend Of a Friend
GNU	GNU's Not Unix
GPL	GNU General Public Licence
HTTP	Hyper Text Transfer Protocol
iTQL	Interactive Tucana Query Language
JRDF	Java RDF
LAN	Local Area Network
LFUI	Leopold-Franzen-Universität Innsbruck
LGPL	GNU Lesser General Public Licence
N3	Notation 3
N3QL	N3 Query Language
NDM	Oracle Spatial Network Data Model
OASIS	Organization for the Advancement of Structured Information Standards
ORDI	Ontology Representation and Data Integration
OWL	Web Ontology Language
OWLIM	OWL In Memory
RDBMS	Relational DBMS
RDF	Resource Description Framework
RDFS	RDF Schema
RDQL	RDF Data Query Language
ROI	RDF Input/Output
SAIL	Storage And Inference Layer
SOFA	Simple Ontology Framework API
SOAP	Simple Object Access Protocol
SeRQL	Sesame RDF Query Language
SEQUEL	Structured English Query Language
SPARQL	SPARQL Protocol and RDF Query Language
SQL	Structured Query Language
TAP	The Alpiri Project
TCP	Transmission Control Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WSDL	Web Service Description Language
WSMO	Web Service Modeling Language
XML	Extensible Markup Language
YARS	Yet Another RDF Store
YARSQL	YARS Query Language

TABLE OF CONTENTS

1 INTRODUCTION

This document is concerned with storage and retrieval of information. In this document, we review a number of storage and retrieval mechanisms in order to find the scope and their roles within TripCom infrastructure. The main focii in this document will be on selected semantic and non-semantic storage systems that are either 'stand-alone' or distributed over the network of devices; and on query languages. It is the amalgamation of the constituents of these areas of technology that gives rise to the basis for storage for TripCom infrastructure. The storage systems selected for the review purpose have specific underlying data models and associated query languages. This implies that there exists a number of possible ways for storing and retrieving the information. However, the TripCom infrastructure aims at providing a language-independent storage system such that different existing storage systems can work together. In addition, a language-independent storage system for storage and retrieval of information enables integration of a multitude of existing query languages.

It is necessary to have a clear view of the data model that is going to be used for storing and retrieving the information. In the scope of TripCom, we assume that the data model being used for representing the information is the RDF data model. In order to provide abstraction for storage systems, i.e., to enable different storage systems, it is necessary to define mapping rules that maps information represented in one data model to the TripCom data model and vice-versa. Therefore, the storage and retrieval in TripCom infrastructure should work at abstract level where concrete storage of information could be distributed over different storage systems. In TripCom, the storage and retrieval of information is directly concerned with efficient storage and retrieval of information. The ability to perform queries on a semantic level would require performing reasoning to entail statements.

In the scope of TripCom, the functionality of storing and retrieving information involves dealing with large amount of data distributed across a network of storage systems. We confine the focus of this document to storage systems that are efficient, distributed, reliable, and scalable. We identify additional requirements of the TripCom infrastructure, through the review of these existing systems, and present them in this document.

The content of this document is divided into two main areas. Section two discusses the state-of-the-art in storage systems and query languages, providing an overview of different storage systems and query languages. Section three discusses requirements analysis of the TripCom infrastructure. It briefly describes a number of functional requirements as well as the non-functional requirements for the storage system.

2 STATE-OF-THE-ART

This section is concerned with techniques of storing and retrieving information for possible use in the TripCom infrastructure. A number of existing storage and query systems are discussed. We refer to these as storage systems and query languages. In the

following subsections, we study these systems topically in order to identify whether they can be utilized in the TripCom infrastructure.

2.1 Storage Systems

Existing storage systems are here categorized according the type of stored data and the method of storage. Structured data is stored as highly organized factual information – typically to allow easy indexing and rapid access. Non-semantic storage systems are studied in order to identify how they can be integrated into the TripCom infrastructure so as to enable non-Semantic systems to work with Semantic data. Semantic data is data that may be formally interpreted by a machine and used to derive new information according some logic theory defining its meaning. Semantic storage systems are further categorized into distributed and stand-alone systems. They are studied based on how they can be utilized or extended to meet the requirements of the TripCom infrastructure.

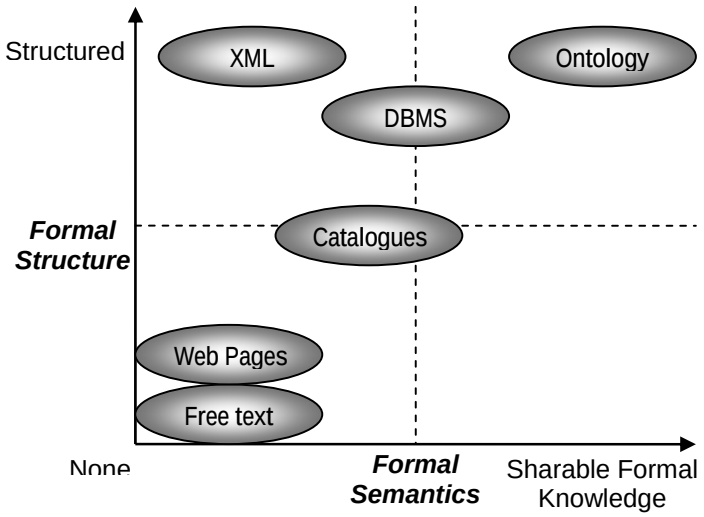

Figure 1: The different sorts of data according its structure and formal semantics [Kiryakov, 2001]

2.1.1 Non-Semantic Storage Systems

In the scope of Triple Space Computing (TripCom), storage systems that do not operate on a semantic level are considered as non-semantic storage systems. These include traditional DBMS/RDBMS and file systems formats. In this section, general APIs provided by these systems are discussed to pave the way for defining TripCom storage APIs compatible to those systems.

I. Database Management Systems

Database Management Systems (DBMS) originate back in the early sixties. A central concept and paradigm is the management of large amounts of data in a predictable and verifiable fashion, organized as collection of records. The terms of DBMS and Relational

Database Management Systems (RDBMS) are very often interchanged despite having a slightly different meaning. RDBMS are data stores to offer referential integrity between the records in different tables (using primary and foreign key relations). Developed in the seventies, the few decades' history transformed RDBMS into very sophisticated systems to control the organization, storage and retrieval of data. The most distinguishing features of relational databases are to provide:

- Modelling language used to define the schema
- Data structures highly optimized for efficient permanent storage
- Query language to read and modify the persisted data
- Transaction support to ensure data integrity despite concurrent access by multiple users

2.1.2 Semantic Storage Systems

For the purpose of this document, storage systems based on an RDF model are considered to be semantic storage systems. These include YARS, Sesame, RDFPeers, 3store, etc. As there are different approaches of maintaining storage of semantic data, we divide these systems further into two categories, namely distributed semantic storage systems and stand-alone semantic storage systems. These are discussed further in the following sub-sections.

A. Stand-alone Storage Systems

Storage systems that work on a single machine are considered (in this document) as stand-alone storage systems. These systems provide functionalities for storing, updating, and deleting information from the storage. Five stand-alone-storage systems which are relevant to TripCom are chosen for review of their features and functionalities.

I. YARS

Yet Another RDF Store (YARS) is a lightweight open-source Java implementation of an RDF index structure with a small footprint which can be embedded into an application and adapted to special application needs [Harth and Decker, 2005]. It defines and realizes a complete index structure including full-text indexes for RDF triples with context. The standard RDF Triple has the defined structure of <**S**ubject, **P**redicate, **O**bject>. In YARS, an RDF Triple is extended with the context information into the form <**S**, **P**, **O**, **C**ontext>. It uses Notation3 [Berners-Lee, 1998] as a way to encode facts and queries. The current version supports tree-shaped Datalog queries with one shared variable. The interface for interacting with YARS is HTTP (GET, PUT, and DELETE) and built upon the REST (Representational State Transfer) principle [Fielding, 2000].

License: BSD

Model Storage: YARS supports the persistence of the RDF graph serialized in Notation-3 (N3) in B+tree. There is a significant difference in the way information is stored in YARS and other semantic storage systems, which is because in YARS, information is indexed for each in combination of (S, P, O).

Supported query languages: YARS supports triple pattern and YARSQL queries. The triple pattern queries are defined in the form of ((S, P, O),C) where every parameter is constant or wildcard. The YARSQL is based on N3QL and is more extensible (see section 2.4), where the query itself is a graph and is inside the RDF data model extended with variables and grouping of graphs.

Reasoning Support: In its current implementation, YARS does not support reasoning. However, work is underway to provide support for Datalog reasoning.

Data Access Interfaces: YARS supports import and export of triples to N-Triples format. All YARS operations are accessible through a standard HTTP interface.

II. Jena

Jena is a leading Semantic Web programmers' toolkit [McBride, 2002], currently in its second generation of the product [Wilkinson et al., 2003]. There are various features improved in Jena2 and most notably: elimination of *reification storage bloat*, query optimization and configurable storage optimizations.

License: BSD

Model storage: Jena1 supports the persistence of an RDF graph in DBMS/RDBMS, and file systems. In Jena2 the support of Berkley DB (DBMS) was dropped, despite Jena1's out-performing all other RDBMS, according to the Jena developing team[1]. Three tables are used to store the RDF information: a literal table, a resource table and a statement table. There is a significant difference in the way the tables are used in Jena1 and Jena2. Jena1 utilizes a "normalized" approach, which requires every literal or resource to be stored at most at one place and later referenced via an internal identifier. The fully *normalized* approach is replaced in Jena2 with the compromise of combining the space savings of the *normalized* approach with the retrieval efficiency of *de-normalization* (i.e., no joins are needed to resolve internal identifiers). All literals and resources, whose names are not very long (configured via a length threshold; default being 256), are stored directly in the statement table. Additional optimization in Jena2 includes the usage of special statement tables to handle specific RDF or custom application specific patterns. For example, reified RDF statements no longer require the usage of 4 different records in the statement table (also known as "triple bloat") as they do in Jena1. Further optimization uses property tables to store subject-value pairs related by a particular property. It stores all instances of the property in a graph.

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

Statement Table

Subject	Predicate	Object
mylib:doc1	dc:title	Jena 2
mylib:doc1	dc:creator	HP Labs-Bristol
mylib:doc1	dc:creator	Hewlett-Packard
mylib:doc1	dc:description	101
201	dc:title	Jena2 Persistence
201	dc:publisher	com.hp/HPLaboratories

Literals Table

Id	Value
101	The description – a very long literal that might be stored as a blob

Resource Table

Id	URI
201	hp:aResource-WithAnExtremelyLong URI

Table 1: Jena2 *de-normalized* schema [Wilkinson et al., 2003]

A specialized graph interface is available in Jena2 to manage the triples with graph level operations. The logical graph is composed by specialized graph implementations and allows add, delete and find operations. They are executed as individual operations on each specialized graph implementation and the results are combined and returned as result for entire logical graph.

Jena does not support special transactions management. All transaction problems are handled by the underlying persistent system, which may not support transactions. Therefore every Jena operation is atomic. Methods to *begin*, *commit* and *abort* explicit transactions are available in the API, but their implementation is not guaranteed.

Supported query languages: Jena1 supports triple pattern and RDQL queries. The triple pattern queries are defined in the form of (S, P, O), where every parameter is constant or wildcard. The RDQL query language is more expressive (see Section 2.2.3), so it is decomposed to a set of interrelated triple pattern queries. A major difference between Jena1 and Jena2 is the way the results are joined. In Jena1 the joins are performed in the Java layer (and are very expensive operations). Jena2 solves that problem by performing the join statements in the database. Support of SPARQL is also available.

Reasoning Support: Jena supports rule-based reasoning.

Data Access Interfaces: Jena support import and export of triples to RDF/XML, N-Triples, N3 and Turtle file formats. A separate RDF Server is used for remote access. It is released under the name Joseki and implements the W3C member submission NetAPI.

III. Sesame

Sesame is flexible architecture which allows persistent storage of RDF data and schema information and subsequent querying of that information. [Broekstra et al, 2002] It has been developed with flexibility in mind[2] to be deployed on a variety of storage systems.

[2] http://www.openrdf.org/about.jsp

A central concept in the Sesame Framework is the "repository" - the storage container of RDF statements. Repositories could be configured using the SAIL stack (sequence of SAIL layers see below) to support security, concurrent access, versioning and RDFS or custom inference.

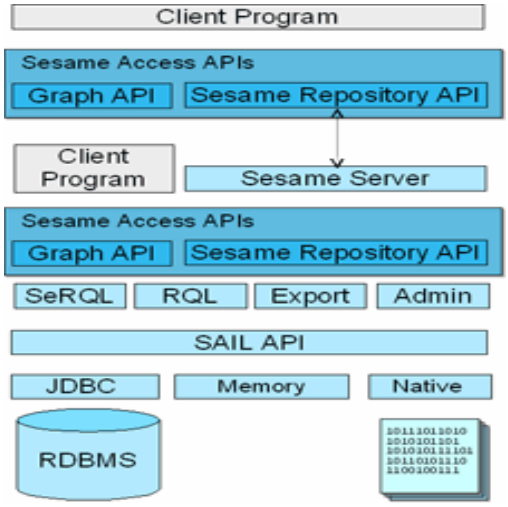

Figure 2: Sesame architecture[3]

License: GNU Lesser General Public License (LGPL)

Model storage: A core part of the Sesame architecture is the SAIL (Storage and Inference Layer) API. SAIL abstracts the persistent storage of triple data with a set of Java interfaces and provides reasoning support. There are various implementations of SAIL – most notably the MySQL, PostgresSQL, OWLIM and native SAIL indexing. Each persistent store utilizes different organization and optimizations of the triple data based on the specificities of the persistent storage. The former two are based on RDBMS and the later two utilized custom in-memory indexing (OWLIM also provides a file-based version). The in-memory implementations perform queries, inference and data modification far quicker than file-based OWLIM. CognitiveWeb announced a beta release of an implementation of the SAIL API to support Oracle 10.2g native RDFS support.[4]

Supported query languages: Sesame version 2.0 supports RQL, RDQL and SeRQL, where the former is the most powerful language. SeRQL is explicitly not meant as 'yet another' query language: its aim is to reconcile ideas from existing proposals (most prominently RQL, RDQL and N3) into a proposal that satisfies a list of key requirements. [Broekstra and Kampman, 2003] such as datayping, optional path expressions, compositionality, schema awareness. Sesame 2.X also includes a "SPARQL engine of Sesame 2"[5] and extends query language support with SPARQL. (see Section 2.2.4)

[3] http://www.openrdf.org/doc/sesame/users/ch01.html
[4] http://www.openrdf.org/morenews.jsp
[5] http://sparql.sourceforge.net/

Reasoning Support: Sesame supports RDF Schema reasoning through deduction rules. Through its SAIL API several reasoning engines can be plugged in as well. Reasoning support for a fragment of OWL-Lite is shipped with Sesame.

Data access interface: Sesame integrates the RDF Input/Output library (RIO) that supports import/export of triples to RDF/XML, N-Triple, N3 and Turtle file formats.

IV. Oracle

Oracle is the first major commercial database vendor to announce native support of RDFS (in Oracle Spatial 10g R2). According to Oracle, it has to combine all pre-existing database features to deal with structured data having the new RDF features.

License: Proprietary (not free)

Model Storage: The RDF support is build on top of the Oracle Spatial Network Data Model (NDM) in which a network or graph captures relationships between objects using connectivity. RDF graphs are modelled as a directed logical network in NDM. In this network, the subjects and objects of triples are mapped to nodes, and predicates (or properties) are mapped to links that have subject start-nodes and object end-nodes[6]. A highly-normalized approach to organize the data is used, so a key-feature is that every resource is kept in only one place. There is a general table schema (composed of the tables: `RDF_MODEL$`, `RDF_VALUE$`, RDF_NODE$, `RDF_LINK$`, and `RDF_BLANK_NODE$`) to contains all the triple information.

Supported query languages: Oracle RDF query support is limited to standard SQL syntax support by the database. New table function RDF_MATCH is introduced to allow arbitrary graph pattern, (defined as a list of triple patterns and bind variables) to be executed against the RDF data. The output of RDF_MATCH function could be combined with other structured database information. Modifications of the data are available with SQL queries execute against the normalized RDF tables. SPARQL support is expected in future.

Reasoning Support: Oracle does not support reasoning.

V. ORDI Framework

The Ontology Representation and Data Integration (ORDI) framework allows the representation and basic management (storage, retrieval, exchange) of ontologies [Kiryakov et al., 2004]. By providing a set of general ontology operations, it defines an abstraction which is a basis for higher-level tasks and functionalities (i.e. browsing, editing, evolution and mediation). The key design rationales behind the framework are:

- Ontology language neutrality

[6] RDF Support in Oracle, Oracle USA Inc.
http://www.oracle.com/technology/tech/semantic_technologies/pdf/semantic_tech_rdf_wp.pdf

- Integration of data from databases and other structured data sources
- Ontology and data modularization
- Support for distributed heterogeneous reasoners and data sources

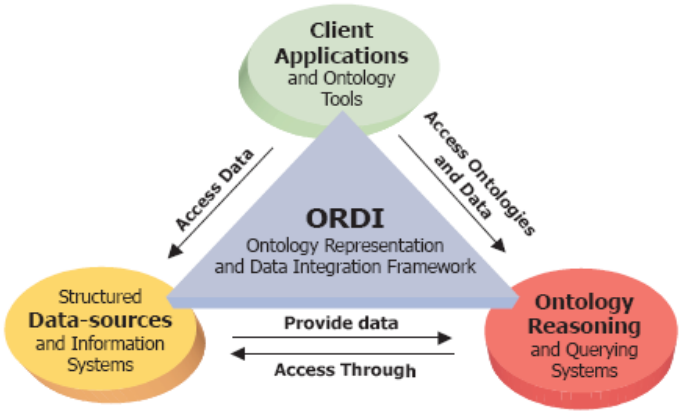

Figure 4: Conceptual model of Ontology Representation and Data Integration (ORDI) framework

The current status of the framework is definition at a conceptual level. The requirements and the major design choices are outlined and motivated. A few of the major constituents are listed below:

- A data-model, resembling the RDF model
- An theory of knowledge compliant with WSMO and allowing an easy mapping of OWL, RDFS, and ER
- The notion of a dataset is introduced to allow for the usage and management of different portions of ontologies or data, defined explicitly or as views

Reasoning Support: The ORDI framework uses Sesame for storage; therefore the reasoning support is same as that in Sesame.

VI. 3store

The following information is provided at the 3store homepage[7]. 3store [Harris and Gibbins, 2003] is an RDF [RDF_Primer, 2004] triple store, released under the GNU General Public Licence. Its purpose is to store large amounts of RDF data in a way that can be queried quickly. It was developed by the Intelligence, Agents and Multimedia research group at Southampton University, as part of the AKT Project[8]. 3store is a core C library that uses MySQL to store its raw RDF data and caches. The standards supported by 3store are SPARQL query language specification [SPARQL_RDF, 2006], SPARQL result format specification [SPARQL_XML, 2006], SPARQL protocol specification [SPARQL_Protocol, 2006] and RDQL query language note [Seaborne, 2004]. The

[7] http://threestore.sourceforge.net/index.php
[8] http://www.aktors.org/akt/

software used by 3store is MySQL database[9], Raptor RDF parser toolkit [Beckett_Raptor, 2006] and Rasqal RDF Query Library [Beckett_Rasqal, 2006].

License: GPL

Model Storage: The core part of the 3store storage system is MySQL. It is based on RDBMS for supporting persistency of large RDF graphs. 3store uses a unified storage mechanism for both classes and instances, in part due to the underlying RDF syntax of RDF Schema. 3store's uses layered architecture for query optimization that can be characterised as RDF Syntax, RDF Representation and RDBMS.

Supported Query Languages: 3store supports SPARQL, RDQL

Reasoning support: The reasoning support in 3store is limited to RDF and RDFS.

Data Access Interface: Standard HTTP Interface.

VII. Kowari

The following information is provided at the Kowari homepage[10]. Kowari is an open source, massively scalable, transaction-safe, purpose-built database for the storage, retrieval and analysis of metadata. Kowari is written in Java and licensed under the Mozilla Public License. Kowari supports Resource Description Framework (RDF) [RDF_Primer, 2004] and Web Ontology Language (OWL) [OWL, 2004] metadata.

Kowari has following features: Kowari supports Native RDF support, multiple databases (models) per server, the interactive Tucana Query Language (heading toward W3C SPARQL support in a future release), small footprint, full text search functionality and datatype support. Regarding performance and scalability, Kowari supports a large storage capacity, is optimized for metadata storage and retrieval, supports multi-processors, is independently tuned for both 64-bit and 32-bit architectures, has low memory requirements, supports on-disk joins and streamed query results. Regarding reliability, Kowari supports full transactions, allows clustering and storage level fail-over and supports permanent integrity. Regarding connectivity, Kowari supports JRDF [Newman, 2006] and uses SOAP [SOAP, 2003].

License: Mozilla Public License, Version 1.1

Model Storage: Kowari supports the persistence of an RDF graph directly on a physical device without the need of separate database systems. The graph is stored as “quads” consisting of subject, predicate, object and meta nodes. The first three items form a standard RDF statement and the meta node describes which model (context) the statement appears in.

[9] http://www.mysql.com/
[10] http://www.kowari.org/

Supported Query Languages: iTQL, SPARQL

Reasoning support: Kowari supports OWL-Lite reasoning.

Data Access Interface: Kowari provides a number of access APIs such as SOFA, JRDF, RDQL, SOAP etc.

B. Distributed Storage Systems:

I. YARS: Yet Another RDF Store (YARS) provides distributed storage and retrieval facilities. Indexing structures are optimized for retrieval of RDF statements including context (via quads) while minimising the need for joins, plus n-gram full text indexing for efficient keyword searches with wildcards. Since YARS needs to accommodate a very large amount of data, distribution of the index to a number of nodes is crucial to allow for cost-effective scalability and extensibility. It uses FOAFRealm [Kruk and Decker, 2004] for the purpose of distributing the index over a number of nodes in the network.

II. RDFPeers

RDFPeers is a distributed and scalable peer to peer RDF triple repository. The triple repository is composed of a large network of peers, which are equal and self-organize into a multi-attribute addressable network (MAAN) [Cai and Frank, 2004]. A globally known hash function to each peer is used to distribute the triples across to the addressable network. Every triple is replicated in multiple places over the peers in order to provide a reliable data persistency mechanism. Multi-attribute range queries are supported on the basis of single-attribute resolution by preserving locality; however the authors note the existence of a query selectivity breakpoint, beyond which the flooding approach (simple mechanism to route the queries by propagating it to all neighbours) becomes better then the used scheme [Sung et al, 2005]. RDFPeers is not publicly available.

III. Edutella

The author of the project defines it as a multi-staged effort to scope, specify, architect and implement an RDF-based metadata infrastructure for JXTA [Edutella, 2006]. The first stage is to provide infrastructure to exchange educational resources. JXTA is open source peer to peer platform initially started by Sun Microsystems and now released under Apache Software License. It provides the basic protocols including peer discovery, peer groups, peer pipes, and peer monitors to cover the P2P functionality. Edutella offers three basic services: a Query Service to search the available information; a Replication service to ensure workload balance and data availability and persistency; and a Mapping, Mediation and Clustering service to translate the data and ensure interoperability between the heterogeneous peers [Nejd, 2002]. The project administrator indicated that no recent development was done.

2.2 Feature Comparison

The systems studied and discussed in the previous section suggest that the main purpose of these systems is to facilitate storage and retrieval of persistent RDF graphs. However, they differ with each other in terms of the features they provide. The table below summarizes their features.

Storage System	Features				
	Storage Model	**Data Model**	**Reasoning**	**Provenance**	**Distribution**
YARS	B+ Tree	Quad	---	YES	YES
Jena	RDBMS, File	Triple	Rule Based	---	---
Sesame	Abstract/SAIL	Triple	Rule Based	---	---
Oracle*	NDM	Triple	---	---	---
ORDI	Abstract	Triple	---	---	---
3store	RDBMS	Triple	---	---	---
Kowari	AVL Tree	Quad	DL	YES	---
RDFPeers	RDBMS?	Triple	---	---	YES
Edutella	RDBMS?	Triple	---	---	YES

Table 2: Features of different RDF storage systems

2.3 Query Languages

This section is dedicated to study and discuss existing query languages associated with the storage systems discussed in Section 2.1. The purpose of this study is to determine which of these query languages can be used or extended to satisfy the requirements of the TripCom infrastructure. In the following subsections some of the various query languages are discussed in general and their APIs in particular.

2.3.1 SQL

SQL (Structured Query Language) is a standard language for accessing and manipulating non-semantic storage systems. It was originally developed to expose a relational data model via a declarative language as part of IBM's System/R project and is still often pronounced SEQUEL due to its original name "Structured English Query Language" .

SQL was adopted as an ANSI standard in 1986 and an ISO standard the following year. There are many extensions and variations of these standards, many of which are of a proprietary nature.

SQL operates on the concepts of tables and columns which have been derived from an Entity-Relationship (ER) data model. However, SQL evaluation is not aware of the data model semantics such as a property 'composer' being a sub-property of 'creator'. Such semantics are encoded in the applications that issue SQL commands. Any changes in data model semantics thus effectively require changes in application code.

* The interest here is on the RDF storage support in Oracle.

SQL as a query language has some properties which have helped SQL gain wide popularity. First, being a declarative query language by nature i.e. a query expresses what is sought instead of saying how the information is sought. This makes it easier for software vendors to develop optimized algorithms to evaluate a given SQL query expression. Second, SQL provides closure. As SQL queries operate on tables with columns and rows, they also return tables which consist of columns and rows. Queries can thus be easily nested which makes it easy for developers to construct complex queries from simple ones.

2.3.2 N3QL

N3QL is an implementation of an N3-based query language for RDF [Berners-Lee, 2004]. N3 offers an extension to the RDF data model with primitives for variables and grouping of graphs. Variables in N3 are denoted using an initial question mark "?". Within N3, RDF subgraphs can become the subject or object of a statement, using "{ }". N3 represents lists of nodes using "( )".

N3QL treats RDF as data and provides queries with triple patterns and constraints over a single RDF model. The target usage is for scripting and for experimentation in information modeling languages. The language is derived from Notation3 [Berners-Lee, 1998] and RDQL [Seaborne, 2004].

The purpose of N3QL is as a model-level access mechanism that is higher level than an RDF API. The N3QL query provides a way in which the programmer can write a more declarative statement of what is wanted, and have the system retrieve it.

In N3QL the basic syntactic N3 primitives are denoted as following: URIs are denoted with brackets < >, Literals are denotes with quotes " ", where as a blank node identifier is denoted with _:. In addition, a number of syntactic shortcuts are available in N3QL, for example "." represents another predicate and object for the same subject. @prefix is used to introduce namespaces.

2.3.3 RDQL

RDQL is SQL-like RDF query languages derived from SquishQL and rdfDB [Miller, et al, 2002] and utilizes N3 syntax for basic syntactic primitives. An RDQL query is composed of 5 possible types of clause (See Figure 5):

- SELECT – identifies the output variables.
- FROM – selects the model by URI.
- WHERE – define the graph pattern.
- AND – specify additional Boolean expressions.
- USING – introduce namespace prefix and its full URI

```
SELECT ?family , ?given
WHERE  (?vcard  vcard:FN "John Smith") AND
  (?vcard  vcard:N  ?name) AND
  (?name   vcard:Family  ?family) AND
```

```
  (?name   vcard:Given  ?given) AND
  USING  vcard FOR <http://www.w3.org/2001/vcard-rdf/3.0#>
```

Figure 5: RDQL query [Seaborne, 2004]

The graph pattern is expressed as a collection of triple patterns, which may be linked via shared variables. All the triple patterns are interpreted conjunctively (i.e. the graph pattern must match all the triple patterns to be added in the result set). The RDF graph is queried as pure data model. If the implementation supports inference, all inferred and asserted triples will be returned with no distinction.

RDQL has number of implementation and is supported by various RDF stores like Jena, Sesame, 3store, Redland and other. It is due to be replaced by SPARQL and its extended features. RDQL is a subset of SPARQL with the exception of several differences including the syntax of triple patterns and constraints. The USING clause is also dropped in SPARQL.

2.3.4 SPARQL

SPARQL (see [SPARQL_RDF, 2006], [SPARQL_XML, 2006] and [SPARQL_RDF_Protocol, 2006]) is a query language for retrieving information from RDF graphs stored in semantic storage systems. SPARQL will be increasingly important as query language for RDF as it has recently reached a candidate recommendation of the W3C. The outline query model is graph patterns expressed by simple triple patterns. It does not use rules and is not path based.

We briefly introduce SPARQL by a simple example. The following example is taken from [SPARQL_RDF, 2006] and shows a simple SPARQL query (see Figure 6) to find the title of a book from the information in the given RDF graph. The query consists of two parts, the SELECT clause and the WHERE clause. The SELECT clause identifies the variables to appear in the query results, and the WHERE clause has one triple pattern.

```
SELECT ?title
WHERE
{
  <http://example.org/book/book1>
    <http://purl.org/dc/elements/1.1/title>
     ?title .
}
```

Figure 6: Simple SPARQL Query

The data (see Figure 7) of the query contains only one triple, which associates `<http://example.org/book/book1>` with `<http://purl.org/dc/elements/1.1/title>` and `"SPARQL Tutorial"`:

```
<http://example.org/book/book1>
  <http://purl.org/dc/elements/1.1/title>
    "SPARQL Tutorial" .
```

Figure 7: Data of the SPARQL query of Figure 6

The query result is the title “SPARQL Tutorial”.

Title
"SPARQL Tutorial"

Figure 8: Result of the SPARQL query of **Fehler! Verweisquelle konnte nicht gefunden werden.** with input of Figure 7.

Furthermore, we cannot express nested queries in SPARQL. These are important e.g. in the context of queries on views, which can be evaluated by one nested query, where the inner query represents the view and the outer query represents the query on the view. Note that views are often used in a *mapping layer* (sometimes also called *mediation layer*) to express the concrete transformation from one format to another format.

There is a possibility to construct RDF data by a SPARQL query. Nevertheless, there is no possibility to insert, update or delete RDF data in existing RDF graphs

Furthermore, compared to SQL, aggregate functions like count, max, min et etc to group results are not supported.

2.3.5 TAP

TAP/GetData [Guha and McCool, 2003] is targeted at Semantic Search, which is a small subset of query answering. TAP introduces a query primitive GetData which is a very simple mechanism to retrieve the property value of a given resource. For example, execution of *GetData* (*mailto:janne.saarela@profium.com, worksFor*) might return <http://www.profium.com>.

The GetData requests are sent via a SOAP message to a URI which identifies an RDF graph. This means the TAP system can effectively manage named graphs which associate a fourth information item to otherwise known triples of the RDF data model.

TAP promotes the idea of having simple primitives to query semantic data instead of trying to express complex queries in one expression.

Any service providing support for TAP queries can decide whether they only provide access to ground 3-tuples/triples or also to inferred triples. A service may thus decide upon the semantics they support in the inferred triples.

2.3.6 SeRQL

SeRQL (Sesame RDF Query Language) is a RDF/RDFS query language developed by Aduna as part of Sesame. It combines the features of existing query languages (e.g., RQL, RDQL, N3) and adds some of its own [Broekstra and Kampman, 2003] making it a hybrid language. It is one of the most expressive query languages available for RDF repositories. It draws on experiences with RQL and RDQL as well as syntax

specifications such as N3. The CONSTRUCT clause provided by SeRQL allows it to perform a limited form of graph transformation.

The design of SeRQL has been based on the RDF specifications with respect to support for several features such as *language* and *datatype* facets to literals, for which SeRQL has functions to query them explicitly. However, SeRQL currently has no support for datatype-aware comparison.

2.4 Quadruples Query Language

The RDF specification does not define the notion of context. [Guha et al, 2004] argue that applications usually require context to store various kinds of metadata for a given set of RDF statements. The interpretation of context is, however, arguable and varies from one application to another. In an information integration use case, for example, the context is the URI of the file or repository from which a triple originated.

[MacGregor and Ko, 2003] reported on an application that stored information about ships and their position and argued that quads are a suitable formalism to capture context. One of the fundamental necessities in open distributed environments like TripCom is to capture provenance in order to judge the quality of the data. The provenance can be captured through the context information. Apart from this, contexts are useful in other application scenarios as well, such as versioning or access control.

The quads are supported in YARSQL [Harth and Decker, 2005]. Two sets of N3QL are used for expressing a quad in a query. It introduces two namespaces namely `ql`[11] and `yars`[12] for such an extension. The first set of language extensions, namely the predicates `ql:where` and `ql:select` allows the formulation of queries. A query consists of a `ql:where` clause which comprises a set of statements that can contain variables. An optional `ql:select` clause determines the format of the result set.

The second set of language extensions define YARS-specific query primitives and are listed below.

- The `yars:context` predicate denotes that the subgraph grouped in the subject is occurring in the context provided as the object. The `yars:context` predicate enables us to express our notion of context within the RDF data model.
- The `yars:keyword` predicate allows the specification of a keyword-containment requirement for subjects.
- The predicate `yars:prefix` can be used to specify that a variable in the subject has to match on the prefix denoted in the object.

[11] http://www.w3.org/2004/12/ql#
[12] http://sw.deri.org/2004/06/yars#

- The predicate `yars:count` is used to query occurrence counts for statements quoted as subject. The object is a variable that is bound to the occurrence count of the access pattern during query evaluation.

3 REQUIREMENTS ANALYSIS

3.1 Functional Requirements

The fundamental functional requirements of TripCom infrastructure consists of data persistency, atomicity, data integrity and data consistency. In addition, TripCom infrastructure should provide support for context representation in order to enable data integration and provenance tracking. These functional requirements are discussed in the following subsections.

3.1.1 Data Persistency

Information stored in a repository should remain there until it is removed by its owner. It should be ensured that a large amount of stored data exists independent of any process using that data similar to that in file systems. Data must be stored persistently over a long period of time in a safe manner.

3.1.2 ACID - Atomicity, Consistency, Isolation and Durability

In order to manage transactions, relational databases have been standardised to support so-called XA transactions which support ACID properties of a transaction. Atomicity, Consistency, Isolation and Durability are often regarded as desirable features of a transaction:

- Atomic – a transaction may be composed of multiple parts which will never be partly done. The transaction will complete them all or none of them.
- Consistency – a transaction once finished will leave the transacted system in a consistent state
- Isolation – the results of the transaction will be made available to other users of the system only when the transaction has been committed.
- Durability – the results of the transaction persist so that they can be recovered by any authorized user of the system at a later time.

For a semantic store ACID transactions are highly desirable. Typically a graph that consists of set of triples expresses semantics which are useful only if they are all available. Discarding one triple from a graph at insertion time is thus not wanted and ACID transactions can be used to make sure the whole graph or nothing at all enters a semantic store.

ACID transactions can be implemented in a centralized repository with a one-phase commit (1PC) protocol while a distributed system needs a two-phase commit (2PC) protocol in order to let all transaction participants agree on the transaction commit phase.

3.1.3 Context Representation

The semantic web described by Tim Berners-Lee [Berners-Lee et al, 2001] suggested a universal knowledge representation language to state machine processable proofs.

[Stoermer et al, 2005] propose that the real information systems are context-dependent, despite that in general RDF statements are context-free. The context is associated with a list of statements logically clustered in a subgraph or partitioned in the overall RDF graph. Noncontextual knowledge systems allow the storage of semantically contradictory statements whose provenance can not be verified. It is possible (and very probable) for further generated knowledge to be semantically incorrect.

Requirements for context representation:

- Efficient storage of contextual information
- Possibilities to work with only specific contexts or subsets of the knowledge based on the contexts
- Ability to specify inter-contextual relationships
    - Subcontexts
    - Incompatible contexts
    - Multi-contextual reasoning (e.g. X is an employee during 2004 in the Energy Department's context and an employee during 2005 in the BP corporate context).

3.1.4 Protocols/Adapters

The TripCom storage infrastructure must support multiple storage implementations in a flexible way. There is no single RDF model storage representation that is universally suitable for arbitrary reasoning and persistence use case scenarios. The revised RDF stores utilize different and incompatible programming interfaces, which support different formal query languages. The problem is very similar to the DBMS prior the standardization of the ODBC protocol for unified data access.

The storage infrastructure must provide:

- Common triple model and interfaces to isolate the different supported API, protocols and data models of the stores. This isolation layer has to provide specific adapter implementations for several major RDF store APIs and to hide the underlying protocol details.
- A unified query language to search the triple data will be required. Besides the common model, a unified search mechanism will be needed to allow full abstraction from the underlying data sources' specific interface details.
- Access to non-triple based data sources. A major part of the available information is stored in legacy relational database systems or other proprietary file formats. A key factor for the success of the TripCom project is the integration and availability of the legacy information to the common triple model. An adapter to work with non-triple based data sources will be required.

3.2 Non-Functional Requirements

3.2.1 Trust and Security

When issuing queries to a server it may be necessary for the issuer to say which content he/she trusts. This trust issue can be partly technically addressed by encoding information in the query about the provenance of information. In RDF terms, the provenance is often associated to graphs via a URI which is either the URL from where the 3-tuples/triples have been retrieved or a URI which encodes information about the author or creator of the information.

So-called named graphs [Carroll, 2005] [Watkins, 2006] address this trust issue by letting users of RDF type information decide upon which graph they wish to issue the query.

Digital signatures for RDF graphs have not been standardised, although algorithms for such signatures have been presented [e.g. Carroll, 2003].

3.2.2 Data distribution, Monitoring, Replication and Reliability

To be able to scale, store and retrieve very large amounts of data quickly, TripCom needs data distribution including load balancing, monitoring of the distributed nodes, reliability to make sure the data is available all the time, and replication of up-to-date data to accommodate for node failures. Data has to be distributed globally on nodes interconnected via the Internet, where possible. In some cases distribution of data storage in a LAN setting is desirable because of ease of maintenance and speed – network data transfer over the Internet is expensive.

I. Data distribution

The TripCom infrastructure should be able to handle data distribution at both LAN and Internet settings of the storages systems. The storage and retrieval functions need to be decentralized to prevent a single point of failure. Even when some of the nodes are no longer available, the rest of the system should be able to operate on a best-effort basis.
A crucial question is how to distribute the data. In other words, how to assign resources/triples to the other systems?

II. Monitoring

As TripCom infrastructure operates in a distributed environment, the storage system should be able to monitor its own health. In addition, it should be able to monitor the load and uptime of nodes in order to detect failures and to repair the system. It is also crucial to ensure that the information is propagated when updating any of the repositories in the network.

III. Data replication

To protect against data non-availability as a result of node failures due to network outages or hardware failure, data has to be replicated in different places to prevent data loss. Data replication is also required to support distribution. Additional replication may be needed when one part of the database becomes unavailable to maintain sufficient redundancy.

IV. Data maintenance

Multiple copies of data must be kept in agreement with each other. Whenever a triple is modified or deleted, all copies of that triple must be similarly modified either at that time or when the next request for the out-of-date information occurs. Whenever a part of the TripleSpace reconnects to the system, changes on each of the temporarily disconnected sides must propagate to the other side.

3.2.3 Mediation, Mapping or Reasoning

Mediation and mapping is often used to express the same concept.

An application must often access data in a data format F_{transf} that is different from the format F_{orig}, which is used to store the data itself in the storage system. Applications deal with different data formats; for example, in the scenarios of data integration, schema evolution, or in bilateral situations, where two applications exchange data.

In these scenarios, the user can define a *view* in the mediation/mapping layer in one storage system so that an application can access the data in the data format F_{transf} of another storage system. The view definition specifies how the storage system transforms stored data from the format F_{orig} into the format F_{transf} when the user queries the view.

In order to support queries on views, the used language and the query and storage system have to support nested queries, as queries on views are typically expressed as nested queries, where the inner query represents the view and the outer query represents the query on the view.

Although SPARQL does not support nested queries, one effort of the TripCom project could be to investigate how to extend SPARQL so that nested queries are supported. Another could be to investigate how to extend SPARQL to support insertion, update and deletion of RDF data in existing RDF graphs. The results can be the input for SPARQL 2 for the DAWG W3C Working Group, of which LFUI is a member.

Whenever modifications on views which affect source data are supported, such view definitions should only express one-to-one mappings. In general for views formulated in complex query languages, modifications of views which affect the source data are not possible.

Reasoning is typically used to express quite different functionalities like query answering, consistency tests, satisfiability tests, and query containment tests. Whereas query answering is required for retrieving data specified by user-defined queries and for mediation, consistency tests check whether or not the existence of instances of a certain class of the ontology is possible (i.e. there are no contradictions in the specification of the ontology), a containment test may be used in caches and a satisfiability test may be used for optimization issues. The query answering functionality is typically already supported

by the query and storage system. It depends on the further functionality of the system developed by the TripCom project whether or not other tests, e.g. of consistency, satisfiability or containment, are required and should be supported by the query and storage system or have to be developed by the TripCom project itself.

4 CONCLUSIONS

This deliverable presents a number of storage systems, both semantic and non-semantic including their support for reasoning. It is the amalgamation of the constituents of these areas of technology that gives rise to the basis for storage for TripCom infrastructure. This document discusses the state-of-the-art in storage systems and query languages and provides requirements analysis for storage for TripCom infrastructure. The requirements are discussed, as are functional and non-functional requirements. Moving forward, the content of this document will be used as a description of the base technologies that will contribute to the development of the innovative storage system of the TripCom infrastructure.

5 REFERENCES

[Beckett_Raptor, 2006] D. Beckett, Raptor RDF Parser Toolkit, 2006, available at: http://librdf.org/raptor/.

[Beckett_Rasqal, 2006] D. Beckett, Rasqal RDF Query Library, 2006, available at: http://librdf.org/rasqal/.

[Berners-Lee, 1998] T. Berners-Lee,. An RDF language for the Semantic Web. W3C, 1998. Available at: http://www.w3.org/DesignIssues/Notation3.html

[Berners-Lee, 2004] T. Berners-Lee,. N3QL - RDF Data Query Language, W3C, July 2004. Available at: http://www.w3.org/DesignIssues/N3QL.html

[Broekstra and Kampman, 2003] J. Broekstra and A. Kampman,. SeRQL: A Second Generation RDF Query Language. User manual, Aduna, 2003. Available at: http://sesame.aduna.biz/publications/SeRQLmanual.html.

[Broekstra, et al., 2002] J Broekstra et al., Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema

[Cai and Frank, 2004] M. Cail and M. Frank., RDFPeers: A Scalable Distributed RDF Repository based on A Structured PeertoPeer Network

[Carroll et al., 2005] J. Carroll et al., Named Graphs, Provenance and Trust. HP Technical report. HPL-2004-57R1. available at: http://www.hpl.hp.com/techreports/2004/HPL-2004-57R1.html

[Carroll, 2003] J. Carroll., Signing RDF Graphs. Proceedings of the Semantic Web Conference.

[Edutella, 2006] http://edutella.jxta.org/

[Fielding, 2000] R. T. Fielding: Architectural styles and the design of network-based software architectures, PhD Thesis, University of California, Irvine, 2000.

[Ganesan et al, 2004] P. Ganesan, M. Bawa, H. Garcia-Molina., Online Balancing of Range Partitioned Data With Applications to Peer-to-Peer Systems, In Proceeding of Very Large Database Systems (VLDB), 2004.

[Guha and McCool, 2003] R.Guha, and R. McCool,. TAP: A Semantic Web Test-bed. Journal of Web Semantics, Volume 1, Issue 1, December 2003.

[Guha et al, 2004] R. V. Guha, R. McCool and R. Fikes, Contexts for the Semantic Web. In Proceedings of the 3rd International Semantic Web Conference, November, 2004.

[Harris and Gibbins, 2003] S. Harris and N. Gibbins (2003). 3store: Efficient Bulk RDF Storage. Technical Report 7970. University of Southampton, 2003.

[Harth and Decker, 2005] A. Harth and S. Decker,. Optimized Index Structures for Querying RDF from the Web. In Proceedings of the 3rd Latin American Web Congress, IEEE Press, November 2005.

[Kiryakov et al., 2004] A. Kiryakov, D. Ognyanov and V. Kirov (2004). An Ontology Representation and Data Integration (ORDI) Framework. WP2 of DIP Project, 2004. Available at: https://bscw.dip.deri.ie/bscw/bscw.cgi/0/3012

[Kiryakov, 2001] Ontologies for Knowledge Management, 2001

[Kruk and Decker, 2005] S. R. Kruk and S. Decker,. Semantic Social Collaborative Filtering with FOAFRealm. In Proceedings of Semantic Desktop Workshop, November 2005.

[MacGregor and Ko, 2003] Robert M. MacGregor and In-Young Ko., Representing Contextualized Data using Semantic Web Tools. In Proceedings of the 1st International Workshop on Practical and Scalable Semantic Systems, October, 2003.

[McBride 2002] MCBride., Jena. IEEE Internet Computing, 2002.

[Miller, et al, 2002] Libby Miller, Andy Seaborne, Alberto Reggiori; Three Implementations of SquishQL, a Simple RDF Query Language, ISWC2002

[Nejd, 2002] EDUTELLA: A P2P Networking Infrastructure Based on RDF, Wolfgang Nejdl, Boris Wolf, Changtao Qu, Stefan Decker, Michael Sintek Ambjorn Naeve, Mikael Nilsson, Matthias Palmer , Tore Risch, WWW2002, Honolulu, Hawaii, USA., 2002.

[Newman, 2006] A. Newman, JRDF (Java RDF), 2006, available at: http://jrdf.sourceforge.net/.

[OWL, 2004] OWL Web Ontology Language Reference, W3C Recommendation, February, 2004.

[RDF_Primer, 2004] RDF Primer, W3C Recommendation, February, 2004.

[Seaborne, 2004] A. Seaborne, RDQL – A Query Language for RDF. W3C Member Submission, W3C, 2004. Available at: http://www.w3.org/Submission/RDQL/

[SOAP, 2003] SOAP Version 1.2 Part 0: Primer, W3C Recommendation, June, 2003.

[SPARQL_RDF, 2006] SPARQL Query Language for RDF, W3C Candidate Recommendation, 6 April 2006.

[SPARQL_RDF_Protocol, 2006] SPARQL Protocol for RDF, W3C Candidate Recommendation, 6 April 2006.

[SPARQL_XML, 2006] SPARQL Query Results XML Format, W3C Candidate Recommendation, 6 April 2006.

[Stoermer et al, 2005] RDF and Contexts: Use of SPARQL and Named Graphs to Achieve Contextualization.

[Stoica et al, 2001] I Stoica, R. Morris, D. Karger, F. M. Kaashoek and H. Balakrishnan., Chord: A Scalable Peer-to-Peer Lookup Service for Internet, In Proceedings of the ACM SIGCOMM ’01 Conference, August, 2001.

[Sung et al, 2005] A Survey of Data Management in Peer-to-Peer Systems, L. G. Alex Sung, Nabeel Ahmed, Rolando Blanco, Herman Li, Mohamed Ali Soliman, and David Hadaller, 2005.

[Watkins and Nicole, 2006] E. R. Watkins and D. A. Nicole., Named Graphs as a Mechanism for Reasoning about Provenance. In Proceedings of the 8th Asia-Pacific Web Conference (APWeb’06), January, 2006.

[Wilkinson et al, 2003] K. Wilkinson, C. Sayers, H. Kuno and D. Reynolds, Efficient RDF Storage and Retrieval in Jena2, 2003.