FP6 – 027324

TripCom

Triple Space Communication

SPECIFIC TARGETED RESEARCH PROJECT

PRIORITY 2.4.7 Semantic-based Knowledge and Content Systems

D1.2 Specification of the Store Architecture and Interfaces

Vassil Momtchev, Atanas Kiryakov

Due date of deliverable: 30 September 2006
Actual submission date: 30 October 2006

Project Start Date: 1 April 2006 Duration: 36 months

Leading Contractor: LFUI

TripCom Consortium Contacts:

N°	Organisation	Country	Name	Phone N°	Fax N°	E-Mail
1	LFUI	Austria	Prof. Dr. Dieter Fensel	+43 512 507 6486	+43 512 507 9872	dieter.fensel@deri.org
2	NUIG	Ireland	Doug Foxvog	+353 91 495150	+353 91 495270	doug.foxvog@deri.org
3	USTUTT	Germany	Prof.Dr. Frank Leymann	+49 711 7816 470	+49 711 7816 472	frank.leymann@informatik.uni-stuttgart.de
4	TUW	Austria	Prof.Dr. Eva Kuhn	+43 1 58801 18512	+43 1 58801 18598	eva@complang.tuwien.ac.at
5	FUB	Germany	Prof. Dr.-Ing. Robert Tolksdorf	+49 30 838 75223	+49 30 838 75220	tolk@inf.fu-berlin.de
6	ONTO	Bulgaria	Vassil Momtchev	+359 2 9768 303	+359 2 9768 311	vassil@sirma.bg
7	PROFIUM	Finland	Dr. Janne Saarela	+358 9 85598 000	+358 9 855 98 002	janne.saarela@profium.com
8	CEFRIEL	Italy	Davide Cerri	+39 0223954324	+39 0223954524	cerri@cefriel.it
9	TID	Spain	Sara Carro Martinez	+34 983 367595	+ 34 983 367564	tripcom@tid.es

Document History:

Version	Date	Changes	From	Review
V0.1	October 10, 2006	Initial version created	ONTO	ONTO
V0.2	October 17, 2006	TripCom Storage Server section added	ONTO	ONTO
V0.3	October 18, 2006	Introduction and conclusion added	ONTO	ONTO
V0.4	October 20, 2006	Final review of the document		ONTO

EXECUTIVE SUMMARY

Deliverable D1.2 is a specification of the storage model and architecture for the infrastructure to be developed in WP1. It outlines some fundamental principles in ontology and knowledge representation and describes the functionality to be implemented in deliverable D1.3 High-Performance Storage Implementation.

The approach for integration of diverse data sources is based on a uniform structured data model. Mappings to this data model shall be used for the aggregation of concrete types of data sources, for instance, relational databases or XML documents. A new kind of generic uniform data model is presented to be used for integration of RDF, named graph, quadruple and SPARQL dataset models. The later is known to be powerful enough to enable integration of data from different relational databases. The new model maintains backward compatibility with the specifications mentioned above and introduces the concept of *tripleset*, as an identifiable set of statements.

This infrastructure will be developed on the basis of the second generation of the ORDI (Ontology Representation and Data Integration) framework, which is a joint development with the TAO[1] project. The ORDI second generation specification, [6], shall be considered complementary to this deliverable, as it provides further details and more extended discussion on two important aspects: the uniform model for representation of structured data and the store architecture.

The ORDI framework uses an ontology-language-neutral software architecture to realize the *tripleset* data model. The framework incorporates a set of tools that perform generic tasks like distribution, unified query support and representation of domain specific ontologies over the underlying data model. The ORDI framework provides flexibility by supporting *hot-spots*. The *hot-spots* allow feasible adaptation of the framework behavior to the domain-specific project requirements.

The current deliverable is of special interest to the component developers in WP2 and WP3 who need to store permanently information. Indirectly, the storage architecture will be used by all other components. It is also interesting for other researches to investigate the possibilities to implement custom domain-specific RDF-based storage systems.

[1] http://www.tao-project.eu/

1 INTRODUCTION

Enterprise application integration (EAI) is a widely recognized problem within contemporary enterprises. Many organizations operate tens or even hundreds of information systems, implemented in different periods, using different technologies, and serving an overwhelming variety of different purposes. Web Services (WS) offer an integration paradigm which solves the basic interoperability problems. Semantic Web Services employ ontologies to facilitate dynamic discovery, composition, orchestration, and data-mediation of WS.

In most of the cases, WS interchange data through messages – small bodies chunks of information are de-contextualized, to the necessary extend, and sent to the recepient. The message exchange happens in, de-facto, synchronous manner – both parties shall be online, so that more messages can be sent, providing on the necessary information on demand. A more decoupled and flexible pattern of information exchange is the one based on independent publication and retrieval. TripCom provides a novel realization of this pattern, named triple-space computing. In essence, the information is published in terms of "tuples", represented in conformance with the recent Semantic Web (Services) standards. Each party is publishing (or referring to already published) information in shared triple-spaces. WP1 deals with the storage mechanisms which underlie the triple-spaces. Such stores shall facilitate easy and efficient persistent publication of structured information in RDF. Further, for the sake of reading and retrieval, the stores shall provide flexible and powerful query evaluation.

The remainder of this section first outlines the relationship between the ORDI framework and the storage infrastructure of TripCom, then comments on the scope of the deliverable. The second section presents the unified structured data model. An overview of the knowledge store architecture is provided in section 3. The last, fourth, section concludes the document.

1.1 ORDI Framework

The Ontology Representation and Data Integration (ORDI) framework enables enterprise data integration based on RDF. The process of integration aims to overcome structural and syntactic information heterogeneities between the information stored in legacy data sources, such as relational database systems or flat-files, and modern semantic repositories. The framework facilitates efficient interoperability between ontological and non-ontological datasets, and further supports richer semantic to allow for comprehensive interpretation and analysis of the data.

The storage component in TripCom will be based on the second generation[2] of the ORDI framework. This further development of ORDI takes part in the course of two projects:

[2] First generation of ORDI was developed in the course of the project DIP

TAO[3] and TripCom. While both projects will share the core of the architecture, they will extend it in different directions:

- TAO will put emphasis on distributed repositories, annotation management, and heterogeneous queries;
- TripCom will focus on integration, though federation or consolidation, of independent structured data-sources.

ORDI is used (or is foreseen to be used) as ontology management and data integration middleware in several projects and products, among which: OWLIM, KIM, SemanticGov, MediaCampaign, RASCALLI. The ORDI specification, [6], shall be considered complementary to this deliverable, as it provides further details and extended discussion on two important aspects: the uniform, structured data-model and the store architecture.

1.2 Scope

Communication between the clients and an instance of the ORDI framework is out of the scope of this specification. The framework is neutral with respect to issues related to multi-user environments. A principle schema for implementation of remote access depends on more general questions regarding the TripCom architecture that cannot be determined at the current phase of the project. One possible approach is to have client-server communication model, where a TripCom Storage Server (TSS) instance is an autonomous service application running in a separate, dedicated system process, rather than a host process under the direct control of the user. One possible implementation is presented on Figure 1.

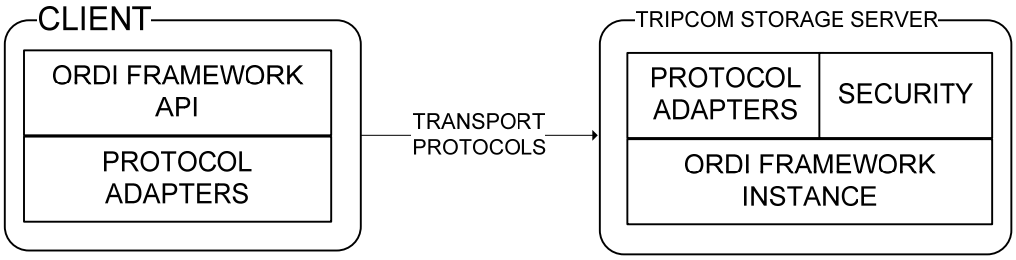

Figure 1 TSS and client connection

There is extreme heterogeneity in the available storage systems and in the features they offer. It is possible to classify them according to several *data qualia* described by Kiryakov in [2]:

- Semantics – whether data could be formally interpreted by a machine to generate new information based on a specific logic theory.

[3] http://www.tao-project.eu/

- Structure – if data is formatted using specific structure interpretable by a machine. It relates, most often, to the performance of the data access.
- Schema – whether there is system information which determines the structure or the semantics of other data.

The requirements identified in D1.1 “State-of-the-art and requirement analysis” limit the scope of integrated storage types to structured data sources. Further, they are categorized to database systems (structured, non-semantic, non-schema), knowledge bases (structured, semantics, non-schema) and ontologies (structured, semantics, schema). Special attention is paid to the integration of database systems, because of their wide commercial adoption and their impact over the industry in the past few decades.

2 STORAGE MODEL

This section defines the uniform structured storage model of TripCom. Under the term "storage model" we consider the incorporation of the allowed primitive operations with the underlying data model. The storage model that we propose is an extension of the RDF data-model. To cope with the requirements, listed in the first sub-section, we need two extensions: Named Graphs and Triplesets. The extensions of the RDF model are first introduced informally. Then we continue with a motivating example, followed by a more formal definition of the tripleset model. An extended and more formal presentation of the model is available in section 2 of the ORDI specification, [6]; one can find there also comments on its links with other data models.

2.1 Objectives

Several rational goals and requirements, identified in the TripCom project, motivate the design of the proposed data model. It should provide support for:

- Feasible integration of different structured data sources including RDBMS;
- Backward compatibility with the existing RDF specifications and the SPARQL query language, [8];
- Transactional operations over the model;
- Efficient processing and storage of meta-data or context information;
- Grouping statements into manageable groups for the purposes of:
    - Definition of access right and signing;
    - Management of sets of statements which correspond to single construct in a higher level language;
    - Transaction tracking and management;
- Easy management of data from several sources within one and the same repository or computational environment. Such are the cases of having data imported from different files (e.g. several ontologies).

2.2 Tripleset Model Introduction

The RDF data model [3] is well-founded core technology in the Semantic Web community. It is detached from the semantics of various knowledge representation formalisms and used in high number of semantic storage systems [9]. We ground our data model to an extension of the RDF data model. To maintain backward compatibility with RDF [3], SPARQL datasets [8] and other RDF-based specifications, [3], the novel notion of *triplesets* is introduced. To define it in a more formal way, we describe several related terms.

The main modelling block in RDF is the *statement* – a triple **<Subject, Predicate, Object>**, where: **Subject** is the resource, which is being described; **Predicate** is a

resource, which determines the type of the relationship; and **`Object`** is a resource or a literal, which represents the "value" of the attribute. A set of RDF triples can be seen as a graph, where resources and literals are nodes and each statement is represented by an arc, labelled with the predicate and directed from the subject to the object.

Named Graphs (NG) were introduced in [5], as a mechanism for referring to and describing RDF graphs. The name of each NG is an URI, which can be used to represent the graph; an RDF description for this URI is considered to represent metadata about the underlying graph. The set of triples which are part of a particular NG shall be explicitly defined, i.e. it may not be derived from the RDF graph. Named graphs can be modelled by quadruples **`<Subject, Predicate, Object, NG_Name>`**, where the first three elements model an RDF statement and the last one represents the name of the NG the statement belongs to. Let us call this quadruples "*contextualized triples*" or "*contextualized statements*".

Triplesets represent groups (or sets) of contextualized triples. Each triple can be considered as a tag, which can be associated with a specific triple. Triplesets are independent from the NG – triples from different named graphs can coexist in one and the same tripleset. In contrast to the NGs, the semantics of the triplesets impose linking of or association triples, instead of copying them.

2.3 Motivating Example

We believe that one appropriate usage of triplesets will be to efficiently support external contextualized information as defined in [12]. There is a high number of motivating examples to justify the requirement to support extensive meta-data. Imagine an RDF repository which holds the CRM-related information of a company. This information comes from different sources, which are modelled as named graphs: a CRM database (**`ng1`**), which keeps information about customers, contact persons, locations, total amount of purchases, customer types/levels; Yellow-Pages-like catalogue (**`ng2`**), providing contact information about organizations; a public companies database (**`ng3`**), providing data about the management of the companies and their financial reports (e.g. annual turnover).

Suppose that access control shall be implemented, so that different people (having accounts associated with different roles, etc.) have access to different parts of this database. The information visible to the different users is encoded in different triplesets.

- Sales manager (any tripleset): no constraints;
- Account manager (**`ts1`**): has access to all the CRM information, except the financial figures from **`ng3`**;
- Sales assistant (**`ts2`**): has access to the contact information only.

The association of the contextualized triples, with the corresponding triplesets is indicated by means of tags on Figure 2.

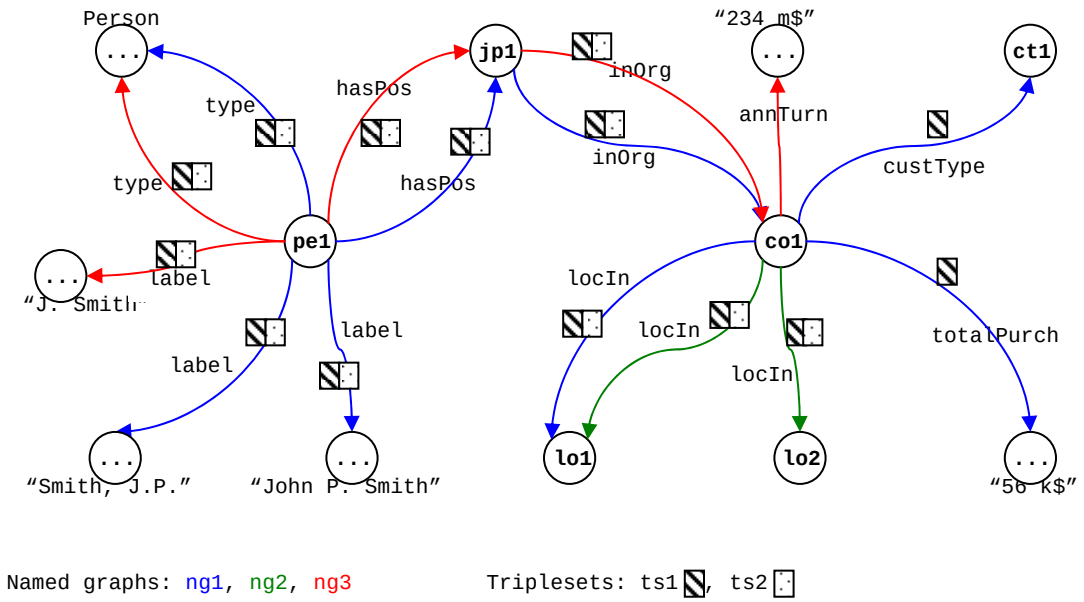

Figure 2 CRM data from different sources and having different access types

2.4 Formal Definition

Named graphs are formally introduced in [12] as a pair **<N,G>**, where **N** is a URI, representing the name of **NG**, and **G** is an RDF graph. We define the functions **name(NG)=N** and **rdfgraph(NG)=G**. Often, NG are used for modelling provenance and other sorts of contextual information in RDF. We will use the terms "named graph" and "context" interchangeably in this specification.

As already mentioned, we call *contextualized triple* the quadruple:

CT=<S,P,O,NG>

Where **S**, **P**, and **O** are respectively the subject, predicate, and object of an RDF triple and **NG** is a named graph. We define the functions **namedgraph(CT)=NG** and **triple(CT)=<S,P,O>**.

We define *dataset* as a set of contextualized triples. A dataset represents an RDF multi-graph, i.e. such graph where two contextualized triples can link one and the same node, with one and the same predicate, and differ only by their **NG** component. Suppose that **DS** is such dataset, we define the following functions:

- **namedgraphs(DS)=SNG**, where **SNG** is a set of resources, such that **NG** is member of **SNG** iff there exists a contextualized triple **CT** in **DS**, such that **namedgraph(CT)=NG**;
- **triples(DS)=TRS**, is a set of triples, where the triple **T** is a member of **TRS** iff there exists a contextualized triple **CT** in **DS**, such that **triple(CT)=T**. **TRS** is a normal set where each triple appears once (instead of multi-set).

This definition of *dataset* is almost equivalent to the definition of RDF dataset in SPARQL. The set of named graphs of an ORDI dataset defines a SPARQL dataset, where:

- There is no default graph – this can be solved by the introduction of a special purpose URI, which is used for encoding of the default graphs. Another options is to adopt **`rdf:nil`** for this purpose;
- There is no guarantee that full named graphs are included in the dataset – an ORDI dataset contains only those statements of a named graph which appear as contextualized triples in the set, while in principle, there could be other triples which also belong to this named graph.

Tripleset is a named dataset, defined as a pair:

`TS=<TSI,DS>`

Where **`DS`** is a dataset and **`TSI`** is an URI, which represents the identifier (or the name) of **`TS`**. We define the functions **`name(TS)=TSI`** and **`dataset(TS)=DS`**. The following operations are defined for a tripleset **`TS1`** and a contextualized triple **`CT`**:

- **`TS2=associate(CT, TS1)`**, where **`dataset(TS2)= dataset(TS1)+{CT}`**;
- **`TS2=disassociate(CT, TS1)`**, where **`dataset(TS2)= dataset(TS1)-{QD}`**.

2.5 Primitive Operations Supported by the Storage Model

There are five supported primitive operations that ensure the full backward compatibility of ORDI data model with the triple **`<S,P,O>`** and quadruple **`<S,P,O,C>`** models.

- **`AddStatement(<S, P, O, C>)`** Asserts new statement to the data model. If the statement already exists, then no modification over the model is done. To stay strictly in the triple model, a default value is used to assert facts into the *default context* or the graph which has no name.
- **`RemoveStatement(<S, P, O, C>)`** Removes one or more statements from the data model. Wildcards are allowed to match any value for a specific element.
- **`AssignToTripleSet(<S, P, O, C>, {TS1,…,TSn})`** Assigns tripleset list to one or more statements. Wildcards are allowed for the statement pattern to match any value. If no statements are matched, no modification is performed.
- **`UnassignFromTripleSet(<S, P, O, C>, {TS1,…,TSn})`** Removes the association between statements and triplesets. Wildcards are allowed for the statement pattern to match any value.
- **`GetStatements(<S, P, O, C, TS>)`** Returns set of statements, based on pattern match. Wildcards are allowed as all parameters.

For more information and the precise definition of the interfaces, please refer to the ORDI technical specification, [6].

3 CONCEPTUAL ARCHITECTURE

The conceptual architecture section provides an overview of the main features and functionalities supported by the framework.

Section 3.1 outlines the key design principles used to define the ORDI framework on conceptual level and describes several core technical aspects of the framework features. Later on, sections 3.2 and 3.3 clarify the composition and the interactions between the system components. To get more detailed information about the full framework specification and definition of interfaces, please refer to ORDI technical specification, [6].

3.1 Design Principles

The ORDI framework has been designed according to the following key functional design rationales:

- Support of datasets by the data model and provide extensive support of meta-information;
- Allow for different implementations of the underlying storage and inference modules and simultaneous parallel work;
- Allow for wrapping and integration of different structured data sources;
- Support distribution of the tripleset data model to multiple storage locations;
- Provide virtual super-store, as a view to a cluster of distributed stores, by consolidating or stacking multiple repositories;
- Generalize the existing storage systems and further reuse already implemented functionality, well-known APIs and standards;
- Provide possibilities for extension and replacement of components by third-party organizations.

In order to provide the needed flexibility and possibility for further extension and customization, the approach to develop hot-spot adaptable framework, [7], is undertaken. Hot-spots are regarded as specialized places where the adaptation of the specified project requirements occurs. Hot-spots are typically implemented via behavioral template method pattern [1], which relies on the “Hollywood principle” - "Don’t call us, we’ll call you." [4]. The ORDI framework identifies several key hot-spots, so they are partially implemented in the components, to offer them and allow the further customization of the behavior.

3.2 Components and APIs

The ORDI conceptual model consists of a collection of loosely-coupled components, described by generic APIs. An overview of the main responsibilities and features for every component is provided next.

RDF Model is a Java API and reference implementation enabling the representation of basic notions for modelling the RDF statements elements. Generic interfaces like Value, Resource and URI are reused from the Sesame 2.0 alpha API [11], thus compatibility with the high-performance RIO parser implementation is ensured.

Store Model is a generic API specifying different types of storage interfaces and the offered functionality. It defines, on an abstract level, the compatibility and functionality of all defined components and available hot-spots in the system. The following concepts are introduced:

- Stackable storage interfaces to allow the layering of multiple storage interfaces;
- Storage interfaces to support generic queries, where one of the specializations is the SPARQL support;
- Triple storage model interface to perform operations over triple models, where one of the specializations is the tripleset data model.

Store Management is a light-weight reference implementation of a registry to maintain the store adapter interfaces. The component is decoupled from concrete store functionalities, so as to support any further extensions of the store model.

SPARQL Engine component is responsible for processing SPARQL queries into trees of conjunctive triple patterns and to execute them against arbitrary stores. SPARQL Engine does not incorporate knowledge about semantics, so all requests are performed on data model level. To support semantics, the queried store has to supports materialized inference.

Security stub is a component providing a hook to implement session based security. So far, there are no TripCom specific security requirements toward the ORDI framework, but if later on the requirements evolve, we can consider the extension of the security stub.

For more technical information such as the exact component interface definitions, please refer to the ORDI technical specification [6].

3.3 Hot-Spots

Definition of hot-spots in a framework is an iterative process, which is hard to be fully predicted in the design phase. We determined the most likely hot-spots in the system, but based on further user requirements or specification, their number may be extended. For more detailed information over the template classes and hook methods related to a hot-spot, please refer to the ORDI technical documentation, [6].

The **Store model hot-spot** is used to supplement additional storage models to the framework, besides the standard tripleset. Typical candidate example for the store model hot-spot is a WSMO storage model that is layered on top of a tripleset model.

The **Store adapter hot-spot** is used to supplement adapters to support various data storage systems such as RDF-stores, RDBMS, structured files or RDF-repositories with incorporated reasoner support. The store adapters are responsible to communicate with specific data source APIs. Further, they perform required transformation between the

extended hot-spot store model and the adapted data source model. An example of a store adapter hot-spot is an implementation supporting tripleset model over a legacy database.

The **Security hot-spot** allows specification of custom security mechanisms.

3.4 Example Application of the ORDI Framework

To understand the benefits of the ORDI framework, we provide a simple example to evaluate its applicability to general use cases. A domain specific storage model, which is efficiently deductible to tripleset model, is introduced. Such domain specific model could be the Triple Space API, which is defined in WP3, or a WSMO repository interface for WP4 needs. A new domain-specific model has to be specified in the framework as an extension of store model hot-spot. The reference implementation has to incorporate the domain-specific requirements and lower the model to that of a tripleset for every write operation, and the opposite way for the read operations. After the domain specific model is fully implemented, the upper layers may reuse the full functionality offered by the framework.

For instance, the case shown on Figure 3 demonstrates the integration with OWLIM[4] semantic repository, which supports nearly full OWL Lite semantics [10].

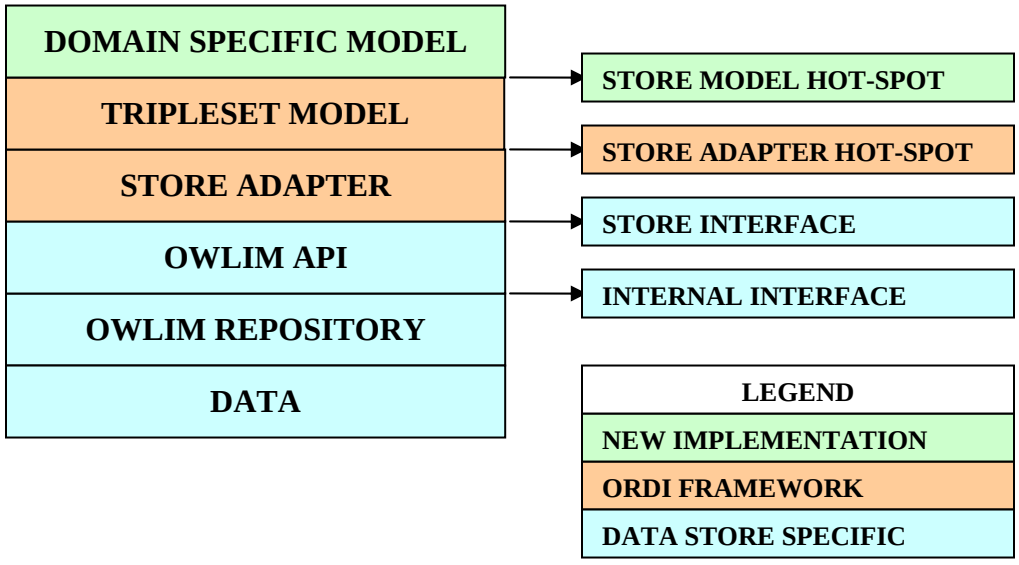

Figure 3 Layering of ORDI store models;

[4] OWLIM – http://www.ontotext.com/owlim/

4 CONCLUSION AND FURTHER DEVELOPEMENT

The D1.2 deliverable "Specification of the Store Architecture and Interfaces" was presented in the terms of underlying storage model and architecture realized as a framework. The tripleset model presents an extension of the RDF model with Named Graphs and Triplesets. It is a generalization of several popular data models and is able to serve the full range of data integration and ontology management tasks. ORDI is an extensible framework, adaptable for wide range storage related routines. It defines the basic APIs to specify the storage models, data sources and supported transaction operation over the model.

The storage specification represents a middleware architecture. In its core there is a set of interfaces, which determine the interoperability patterns of a repository, enabling management (storage, modification, retrieval) of the tripleset model. The central interface there, named **`Store`**, is designed to allow for multiple implementations and extensions of stores. Wrappers for data management systems (e.g. RDBMS) shall be developed as implementations of **`Store`**. Higher-levels of modelling (e.g. concrete ontology languages as WSML) shall be implemented as extensions of the **`Store`** interfaces.

The knowledge store architecture is specified on the basis of the second generation of the ORDI framework. Its specification, [6], further details on the storage model and the architecture. The specification of ORDI will be updated in the course of its development, the short plans for which can be summarized as follows:

- **ORDI v0.4.1** (Nov 2006): a more stable and mature release including SPARQL support; the WSML/RDF mapping will be synchronized with the next version of its specification; Developer's Guide to be included.
- **ORDI SG API** (Dec 2006): the API for the second generation of ORDI will be developed in compliance with Sesame 2.0;
- **First version of ORDI SG,** v.0.5 (Jan 2007): the API will be implemented on top of OWLIM, to deliver the first functional prototype. At this stage we will have to deal with: further specification of a serialization format of tripleset data model; extension of the SPARQL query language via extended function; and better generalization of the query language formalisms.
- **wsmo4rdf v0.1** (Feb 2007): an implementation of the WSMO/RDF mapping as an extension of ORDI SG v.0.5; it will cast ORDI v.0.4;
- **wsmo4rdf v0.2** (Mar 2007): the second version of the wsmo4rdf extension will implement scalable WSML-Core reasoning;

The most notable subjects of middle- and long-term development are: an adapter for integration of RDBMS and a multi-store implementation allowing for data consolidation.

5 ACRONYMS

CRM.....Customer Relationship Management

EAI.....Enterprise Application Integration

NG.....Named Graph

ORDI.....Ontology Representation and Data Integration

TSC.....Triple Space Computing

URI.....Uniform Resource Identifier

WSML.....Web Service Modeling Language

WSMO.....Web Service Modeling Ontology

XML.....Extensible Mark-up Language

6 RELEVANT WEB SITES

ORDI – website	http://www.ontotext.com/ordi/
OWLIM – high performance semantic repository	http://www.ontotext.com/owlim/
TripCom – Triple Space Computing	http://www.tripcom.org
WSMO – Web Service Modeling Ontology	http://www.wsmo.org
TAO – project	http://www.tao-project.eu/

7 REFERENCES

[1] Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, Design Patterns, Addison Wesley, Reading, MA, 1995.

[2] Kiryakov, A; Ognyanov, D; Kirov, V. (2004) An Ontology Representation and Data Integration (ORDI) Framework. DIP project deliverable D2.2. http://dip.semanticweb.org

[3] Klyne, G; Carroll, J. J. 2004. Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C recommendation 10 Feb, 2004. http://www.w3.org/TR/rdf-concepts/

[4] Larman, C. (2002). Applying UML and patterns an introduction to object-oriented analysis and design and the unified process. Upper Saddle River, NJ, Prentice Hall PTR.

[5] MacGregor, R; Ko, I.-Y. Representing Contextualized Data using Semantic Web Tools. Proc. of the 1st International Workshop on Practical and Scalable Semantic Systems; available at http://km.aifb.uni-karlsruhe.de/ws/psss03/proceedings/macgregor-et-al.pdf

[6] Momtchev, V; Kiryakov, A. 2006. ORDI: Ontology Representation and Data Integration Framework. Second Generation Specification. Ontotext Lab, Oct 2006. http://www.ontotext.com/ordi/v0.5/ORDI_SG_Specification.pdf

[7] Pree, W. (1994). Meta patterns - a means for capturing the essentials of reusable object-oriented design. in M. Tokoro and R. Pareschi (eds), Springer-Verlag, proceedings of the ECOOP, Bologna, Italy: 150-162.

[8] Prud'hommeaux, E; Seaborne, A. 2006. SPARQL Query Language for RDF. W3C Working Draft 4 October 2006. http://www.w3.org/TR/2006/WD-rdf-sparql-query-20061004/

[9] Sapkota B.; Momtchev V; Groppe S; Saarela J. 2006. D1.1 State – of – the – Art and Requirements Analysishttp://www.tripcom.org/docs/del/TripCom-D11.pdf

[10] SwiftOWLIM: System Documentation ver. 2.8.4, 16 Sept. 2006 (Distrib. update 30 Sept.); available at: http://www.ontotext.com/owlim/OWLIMSysDoc.pdf

[11] System documentation for Sesame 2.0 (DRAFT); available at http://www.openrdf.org/doc/sesame2/system/

[12] Tolle, K. Understanding data by their context using RDF. Available at: http://www.dbis.informatik.uni-frankfurt.de/~tolle/Publications/2004/AISTA04.pdf