



SIRMA GROUP CORP. ("СИРМА ГРУП" АД)  
IT CENTER OFFICE EXPRESS  
135 TSARIGRADSKO SHOSE, SOFIA 1784, BULGARIA  
TEL: +359 2 9768 310, FAX: +359 2 9768 311  
[HTTP://WWW.ONTOTEXT.COM](http://www.ontotext.com)



# **ORDI:**

## **Ontology Representation and Data Integration Framework**

### ***Second Generation Specification***

ver. 0.5, 30 Oct 2006

Vassil Momtchev, Atanas Kiryakov

## Contents

1	Introduction .....	3
1.1	Ontologies, Knowledge Bases, and Related Terminology .....	3
1.2	Ontology Language Layering.....	4
2	Data Model .....	6
2.1	Requirements Toward Data Model.....	6
2.2	Integration with Other Structured Data Sources .....	6
2.3	Structure of Data Model .....	7
2.4	Supported Primitive Operations.....	9
2.5	Formal Definition of Tripletset.....	9
2.6	Motivating Scenario.....	10
2.7	Relationship to Other Data Models .....	11
2.7.1	RDF and reification .....	11
2.7.2	Quadruples and Named RDF Graphs.....	12
2.7.3	Contexts .....	12
2.7.4	SPARQL and Sesame 2.0 .....	13
2.8	Serialization Formats.....	13
3	Ontology Representation .....	15
3.1	Example to Model WSML Ontology with ORDI Data Model .....	15
4	Architecture design.....	18
4.1	Architectural Requirements.....	18
4.2	Structural overview .....	18
5	Conclusion .....	21
6	References.....	22
	Appendix A .....	24

# 1 Introduction

Ontology Representation and Data Integration (ORDI) is middleware framework to allow enterprise data integration via RDF-like data model. This paper presents the second generation of the ORDI framework, which is build on top of the specification presented in [14] and its realisation in [15]. The development of the second generation of ORDI takes part in the course of projects TAO<sup>1</sup> and TripCom<sup>2</sup>. ORDI is foreseen as ontology management and data integration middleware in the several projects and products, among which: OWLIM, KIM, SemanticGov, MediaCampaign, RASCALLI.

The intended audience of this document are architects and software developers, interested in the application of new data integration methodologies, Semantic Web, or ontology management. Basic awareness of RDF, XML, and relational databases is beneficial, but not mandatory for the understanding of this document.

The rest of this section provides an introduction to several terms related to ontologies and ontology languages. Section two discusses the requirements towards the ORDI data models and defines one to meet them. Section three presents a representation of WSMML ontology, as an example of using the ORDI's model to manage ontologies defined in higher level languages. The fourth section presents the architecture of ORDI, while the fifth one provides a wrap up of the specification and discusses future plans. UML diagrams of the most important interfaces are presented on Appendix A.

## 1.1 Ontologies, Knowledge Bases, and Related Terminology

*Ontologies* can be considered as conceptual schemata, intended to represent knowledge in the most formal and re-usable way possible (similar comments are also provided in [12]). The formal ontologies (considered in the AI) are represented in logical formalisms (like OWL, [9]) which allow automatic inference over them or datasets aligned to them.

An important use of ontologies, which should be considered from a broader ontology management perspective, is that they can be used as schemata and/or "intelligent" view over information resources. Thus, they can be used for indexing, querying, and reference purposes from non-ontological datasets and systems, such as databases, document management systems and catalogue management ones. Ontologies use to represent richer semantic that allows a comprehensive interpretation of the data, i.e. inference of non-explicitly asserted statements and facts. In this way, they can improve the interoperability and the efficiency of the usage of non-ontological datasets.

*Knowledge Base* (KB) is a term with a wide usage and multiple meanings. Here we consider KB, a dataset with some formal semantics. A KB, similar to an ontology, is represented with respect to a knowledge representation (KR) formalism, which allows automatic inference. It could include multiple axioms, definitions, rules, facts, statements, and any other primitives. In contrast to the ontologies, the KBs are not intended to represent a (shared or consensual) schema, a basic theory, or a conceptualization of a domain. Thus, the ontologies are just a

---

<sup>1</sup> <http://www.tao-project.eu/>

<sup>2</sup> <http://www.tripcom.org/>

specific case of knowledge bases, developed and used as schemata. Further, it is a widely recommended, that knowledge bases, bearing specific data<sup>3</sup> are encoded with respect to one or more ontologies, which encapsulate the general knowledge and thus allow easier sharing and reuse of KBs.

Drawing the borderline between the ontology (i.e. the conceptual and consensual part of the knowledge) and the rest of the data, represented in the same formal language, is not always a trivial task. For instance, there could be an ontology about tourism, which defines the classes `Location` and `Hotel`, as well as the `locatedIn` relation between them, and the hotel attribute `category`. The definitions of the classes, relations, and attributes should clearly be a part of the ontology. The information about a particular hotel is probably not a part of the ontology, as far as it is not a matter of conceptualization and consensus, but is just a description, crafted for some (potentially specific) purpose. Then, suppose that there is a definition of the New York city – it can be argued that it is either a part of the ontology or just a description of a city.

We will use *dataset* to denote any body of structured data, as defined in Dublin Core, [10]: “A dataset is information encoded in a defined structure (for example, lists, tables, and databases), intended to be useful for direct machine processing.” RDF Dataset is used in a similar way in SPARQL, [20], to represent any combination of RDF graphs. Datasets may or may not have formal semantics – depending on the sort of schema, which defines their structure. Ontologies are schemata which define both structure and semantics. Ontologies and KB are datasets on their own, as long their formalizations are structured.

We use *tripleaset* as a name for a part or a module of an RDF Dataset. Modularization could take place design time. For instance, a knowledge engineer can design an ontology into number of parts, following one of a number of design patterns: single ontology; single ontologies with multiple modules; multiple interdependent ontologies. It is all a matter of knowledge engineering methodology and practices. Modules can also be defined later on, for the purpose of specific usage an ontology or a dataset.

## 1.2 Ontology Language Layering

The ontology language provides a set of modelling primitives you can use to build ontology. There are a number of layers of definition that could be distinguished for an ontology language or in general a knowledge representation language:

- **Data model** determines the data-structure. For instance, the data model of RDF(S) is a graph of resources and literals connected by properties, which are also resources. The core data-model of OWL is the same as the one of RDF(S).
- **Epistemology** defines the language at a conceptual level. It determines what constructs and primitives are used to represent knowledge in this language, in accordance with some modelling paradigm and philosophy. At this level, the meta-model of the language is specified, in terms of patterns in the data model. For instance, the core epistemology of RDF(S) includes: descriptions, classes, properties, collections, reified statements. The epistemology of OWL is richer and also includes: data- and object-properties, functional and transitive properties, ontologies, etc. Epistemology is the level at which the major KR paradigms differ, although they tend to be based on

---

<sup>3</sup> Often referred as instance data, instance knowledge, A-Box, etc.

quite similar logical foundations. The following are some of the typical distinctions: concepts vs. classes; relations, properties, attributes, vs. roles; etc.

- **Vocabulary** determines what sorts of symbols are valid for composition of expressions in a language. Nowadays most of the languages are based on, or allow serialization in, strings of ASCII or UNICODE characters. The vocabulary defines the naming conventions for various primitives, defined in the data-model and the epistemology of the language.
- **Syntax** determines the structure of the valid expressions within a language. For some languages, a number of alternative syntaxes exist, such as XML and N3 syntaxes of RDF(S), or the abstract, N3, and the RDF/XML syntax of OWL.
- **Semantics** determines the meaning of the expressions made in this language. The semantics of a language is often defined in terms of couple, consisting of a mathematical model and a function, defining the correspondence between the expressions of the language and the elements of the model. Any sort of inference is performed on the basis of the semantics.

This layering is presented visually on Fig. 1 together with some samples of an RDF(S)-like language. In a number of languages, the different layers of definition are hard to separate, because some of them are not explicitly defined or separated. Often the vocabulary and syntax layers are merged, as well as the data model and the epistemology.

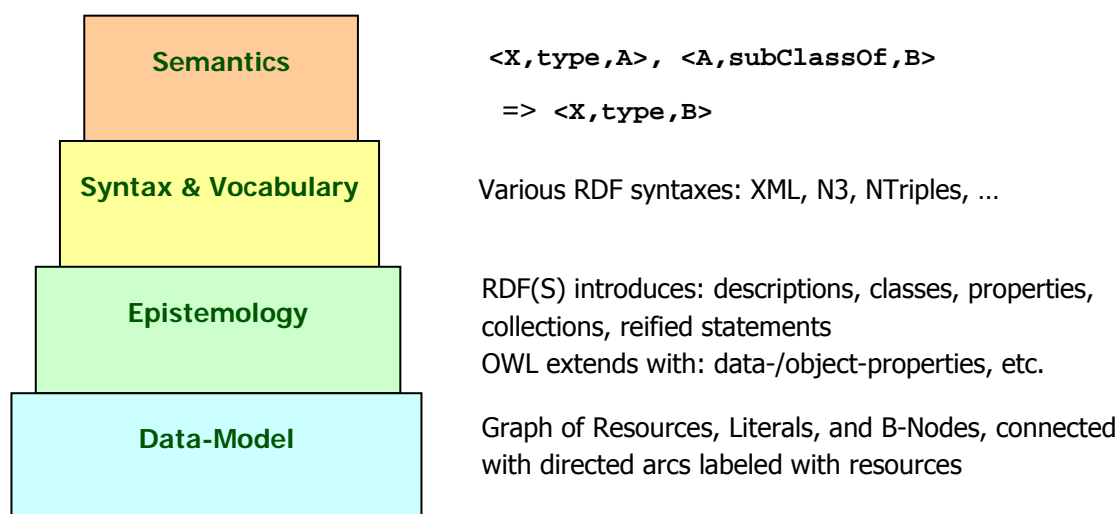


Fig. 1 *Ontology Language Layering*

## 2 Data Model

The ORDI data model, defined hereby, provides basic notions and data-structures, which are to be used for ontology and data representation. The model is in no way ontology-specific – it allows a representation of any structured data. It does not prescribe any sort of semantics and epistemology – therefore it is left to the higher levels, as well as to the specific implementations and applications, which are to impose a further interpretation over it. In other words, the data-model is neutral in terms of interpretation (or semantics) – it provides the freedom of choice of a data-source or the reasoning service to interpret its data accordingly in order to derive the desired results, i.e. to answer questions related to the existing data or to infer new data.

We ground our data representation on an RDF-compatible data model [16], since it is well-founded and detached from the semantics of the various knowledge representations, ontology, and semantic web languages used today. Later, the section describes the base requirements toward the data model and completes an overview of the extensions over the RDF model to support them.

### 2.1 Requirements Toward Data Model

ORDI's data model design is motivated by several rational goals and requirements related to existing problems in the modern Semantic Web systems. It should provide support for:

- Feasible integration of different structured data sources including RDBMS;
- It should be backward compatible with the existing RDF specifications and SPARQL query language, [20];
- 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 (e.g. WSML);
  - 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 Integration with Other Structured Data Sources

The process of integration of other structured data sources aims to overcome the structural and syntactic information heterogeneity. The integration is performed by agreement on data model level, so it does not necessary mandates semantic integration as well. We present an abstract model for data integration represented as triple (not to be mistaken with RDF triple):

$$\langle G, S, M \rangle$$

where:

- G – ORDI data model

- S – data model of a specific data source
- M – mapping from a data source to the base ORDI data model

Fig. 2 presents the scheme of layering of the data-models and mapping in case of integration of multiple data sources.

G		
M <sup>1</sup>	...	M <sup>N</sup>
S <sup>1</sup>	...	S <sup>N</sup>

Fig. 2 Layering of data models in ORDI

The proposed model is feasible for the integration of legacy data sources as relational database systems, XML or other structured documents and ensures the full decoupling between information and structural or syntactic representation. The data model mappings can differ in several ways:

- They could provide a one-way or a bi-directional conversion. In the first case ORDI can “see” the data from the data source, but there is no way of converting ORDI data into the model of the data source. The later means that this source is read-only, in with respect to its ORDI integration.
- The conversion could be complete or partial (in either of the directions).
- A mapping between data models can be combined with mappings at the higher levels of the language definition stack (Fig. 1 ): epistemology, vocabulary, and semantics. For instance, a mapping of a relational database to RDF can be designed to convert tables into OWL classes, foreign keys into object properties, and so on. In contrast, a data model mapping, which does not reuse OWL, could map them to resources, which may (or may not) be typed, as `<T1,rdf:type,M1:Table>` , where `M1:Table` is just a resource introduced by this mapping. Another example would be a mapping between `wsm1:Concept` and `owl:Class`.

There is no requirement for a formal declarative approach for definition of mappings within ORDI, but maybe considered as future extension.

### 2.3 Structure of Data Model

ORDI data model is realized as a directed labelled multi-graph, designed in accordance with previously enlisted requirements and goals. In order to be backward compatible with RDF, SPARQL and other RDF-based specifications, a new kind of information is introduced to the RDF graph, [16]. The *Tripletset* model is a simple extension to RDF graph to allow the efficient and natural association of meta-data to the statements. It is a new element in the RDF statement, previously expressed as triple or quadruple, to describe the association between the statement and an identifiable group of statements. This new term is introduced to distinguish the new model from several similar, already existing, RDF extensions and the terms associated with them:

- Context, as defined in several RDF APIs like Sesame 2.0, YARS and others;
- Datasets, in the way they are defined in SPARQL;

- Named-graphs, in the way they are introduced at <http://www.w3.org/2004/03/trix/>, implemented in Jena and used in SPARQL.

For more information to explain the difference between the models refer to section 2.5.

The atomic entity of the ORDI data model is a quintuple, composed by the RDF data type primitives – URI, blank node and literal, as follows:

$$\langle s, p, o, c, \{TS1, \dots, TS_n\} \rangle$$

where:

- *s* – subject of a statement; of type URI or blank node;
- *p* – predicate of a statement; of type URI;
- *o* – object of a statement; of type URI, blank node or literal;
- *c* – context of a statement; of type URI or blank node;
- $\{TS1, \dots, TS_n\}$  – unordered set of identifiers of the triplesets to which the statement is associated; of type URI or blank node;

The following is true for the triplesets model:

- Each contextualized triple  $\langle s, p, o, c \rangle$  could belong to multiple triplesets;
- The triples are not “owned” by the contexts; a triple can be disassociated from a triplesets without being deleted;
- When a triple is associated with several triplesets it should be counted as a single statement, e.g. a single arc in the graph;
- A triplesets can contain triples from different contexts;
- Each triplesets can be converted to a multi-set of triples, as one triple can correspond to multiple contextualized triples, belonging to the triplesets.

The diagram on Fig. 3 depicts an entity-relationship diagram of the relationships amongst the major elements in the ORDI data model.

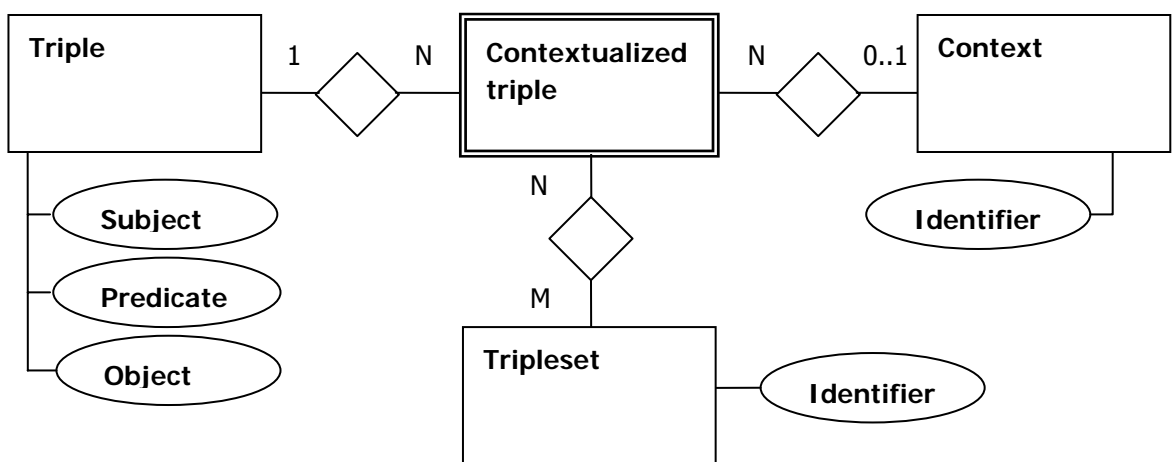


Fig. 3 Entity-Relationship diagram of ORDI data model

## 2.4 Supported Primitive Operations

There are five supported primitive operations, to ensure the full backward compatibility of ORDI data model with the triple  $\langle s, p, o \rangle$  and quadruple  $\langle s, p, o, c \rangle$  models.

- Add new statement

`AddStatement(<S, P, O, C>)`

Assert new statement to the data model. If the statement already exists, then no modification over the model is done. To stay strictly to the triple model a default value is used to assert facts into the *default context* or the graph which has no name.

- Remove existing statements

`RemoveStatement(<S, P, O, C>)`

Remove one or more statements from the data model. Wildcards are allowed to match any value for a specific element.

- Assign statements to tripleset(s)

`AssignToTripleSet(<S, P, O, C>, {TS1,...,TSn})`

Assign 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.

- Un-assign statements from tripleset(s)

`UnassignFromTripleSet(<S, P, O, C>, {TS1,...,TSn})`

Remove the association between statements and triplesets. Wildcards are allowed for the statement pattern to match any value.

- Retrieve statements

`GetStatements(<S, P, O, C, TS>)`

Return set of statements, based on pattern match. Wildcards are allowed for all the parameters.

## 2.5 Formal Definition of Tripleset

This section extends the above introduction to the model, but means of a more formal definition of triplesets, instead of quintuples.

*Named graphs* (NG) are introduced in [5] as a pair  $\langle n, g \rangle$  where  $n$  is URI, representing the name of NG, and  $g$  is an RDF graph. We define the functions  $name(NG)=n$  and  $rdfg(NG)=g$ . [5] puts some constraints on the interpretation of the graphs, which are not necessary for ORDI, nor they appear in SPARQL, we stick to the unconstrained definition in SPARQL, [20]. Having said this, we will use "named graph" and "context" interchangeably in this specification. We call *contextualized triple* the quadruple

$CT = \langle s, p, o, NG \rangle$

where  $s$ ,  $p$ , and  $o$  are respectively subject, predicate, and object of an RDF triple and  $NG$  is a named graph. We define the functions  $namedgraph(CT)=NG$  and  $triple(CT)=\langle s, p, o \rangle$ .

We call *dataset* a set of contextualized triples. The dataset represents an RDF multi-graph, i.e. such graph where two contextualized statements can link one and the same nodes, with one

and the same predicates, and differ only by their  $ng$  component. Suppose  $ds$  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 in  $ds$  exists a contextualized statement  $cs$ , such that  $namedgraph(cs) = ng$ ;
- $triples(ds) = trs$ , is a set of triples, where the triple  $t$  is member of  $trs$  iff in  $ds$  exists a contextualized statement  $cs$ , such that  $triple(cs) = t$ .  $trs$  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 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 the named graph which appear as contextualized triples in the set, while, in principle, there could be other triples which also belong to the named graph.

*Tripletset* is a named dataset, defined as a pair

$$TS = \langle TSI, DS \rangle$$

where  $ds$  is an 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  $ts1$  and a contextualized triple  $ct$ :

- $ts2 = associate(ct, ts1)$ , where  $dataset(ts2) = dataset(ts1) + \{ct\}$ ;
- $ts2 = disassociate(ct, ts1)$ , where  $dataset(ts2) = dataset(ts1) - \{ct\}$ .

## 2.6 Motivating Scenario

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 organisations; a public companies database ( $ng3$ ), providing data about the management of the companies and their financial reports (e.g. annual turnover).

Fig. 4 depicts a snapshot from the RDF graph, where  $pe1$  is the contact person for customer  $co1$ , which has offices in locations  $lo1$  and  $lo2$ . As we can see, some of the triples come from more than one data-source; for instance, those defining the position of  $pe1$  within  $co1$ . On the other hand,  $ng1$  and  $ng3$  provide different variations of the name of the person. Further,  $n1$  and  $n2$  "agree" that  $co1$  has an office in  $lo1$ , but only  $n2$  "knows" that it has also office in  $lo2$ . Note that the corresponding arcs are duplicated, so, if one of them gets deleted, the other remains there to state that this fact is still true according to a different source.

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 roles, is encoded in different tripletsets.

- Sales manager (any tripleset): no constraints;
- Account manager ( $\tau_{s1}$ ): has access to all the CRM information, except the financial figures from  $ng3$ ;
- Sales assistant ( $\tau_{s2}$ ): has access to the contact information only.

The association of the contextualized triples, with the corresponding triplesets is indicated by means of tags on Fig. 4 .

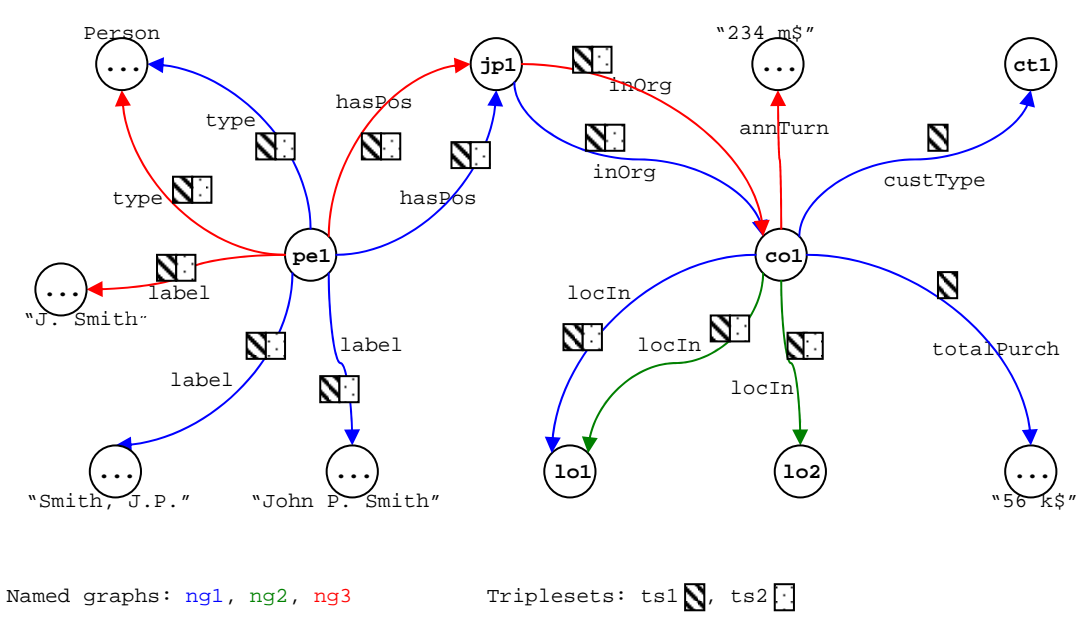


Fig. 4 CRM data from different sources and having different access types

## 2.7 Relationship to Other Data Models

ORDI’s data model was defined to generalize several of the most popular RDF-related data models.

### 2.7.1 RDF and reification

ORDI model may be regarded as extension and super-set of the W3C recommended RDF model [16]. Any valid RDF model is also a valid ORDI model. Thus, any semantics that could be expressed in RDF graph is possible to be expressed in ORDI as well. RDF offers two quite cumbersome and inefficient mechanism to state information for other statements.

Reification, as defined in [16], allows identifying single triple, so assertion of new facts about it is further possible. The approach, despite its compliance with the W3C recommendation for RDF, is very inefficient, because to convert triple into reified statement and identified it, four additional statements are necessary. Let’s take for example any arbitrary triple  $\langle subj1, pred1, obj1 \rangle$  and identify it by  $reif\_id1$ :

Triple	subj1 pred1 obj1 .
--------	--------------------

Reified triple	<pre>reif_id1 rdf:type rdf:Statement . reif_id1 rdf:subject subj1 . reid_id1 rdf:predicate pred1 . reif_id1 rdf:object obj1 .</pre>
----------------	---

Table 1 Reification of a triple, [16]

Another approach to identify a statement and further assert facts about it is the *object-oriented context* is described by MacGregor, [17]. The identification is based on additional new resource in the RDF graph with semantics – instantiation of contextualized reference to the real object. The new resource encapsulates the so called *object-oriented context*.

Triples	<pre>subj1 pred1 obj1 .</pre>
Object-oriented context	<pre>subj1 pred1 ref_obj1 ref_obj1 rdf:value obj1</pre>

Table 2 Contextualization using object-oriented context

Schockaert suggests [18] that despite the elegance of the approach, it could not be easily applied in cases of provenance tracking and graph signing.

As bottom-line we see that the RDF data model does not support way to identify statements and further define context. There are two inefficient approaches to do it on semantic level using reification and *object-oriented context*, but high-performance implementation of the both approaches is highly-disputable. A consequence to the described above problems, the developers of several well-known storage implementations like Jena 2, [2], and Oracle Spatial Network Data Model, [1], suggest algorithms to optimize the persistence of reified statements and support of contexts by using quadruples internally.

### 2.7.2 Quadruples and Named RDF Graphs

MacGregor suggests another approach to efficiently link context to statement, by extending the RDF data model with a fourth element, [17]. The quadruple are commonly referred as quads and are represented as:  $\langle s, p, o, c \rangle$ . The semantics of the fourth element is not strictly defined and has different meaning in the various quadruple stores. Some authors suggest to be used as data source, identifier of a statement or model id, thus it is named abstractly *context*. Named RDF graph data models defined by Carroll, [5], refines stricter semantic over the fourth element by defining it as *RDF graph name*, which could not be of type blank node. The named RDF graphs are designed to be disjoint from each other and define that the blank-nodes could not be shared across different graphs.

Although not explicitly stated, their philosophy is that, when joined, those form an RDF multi-graph, i.e. when one and the same triple appears in two graphs, it should be counted and manipulated as two separate arcs. On the contrary,

### 2.7.3 Contexts

Tolle, [19], summarizes the different approaches to use the context according the expressed features to:

- *Internal context* or semantically related to the information;
- *External context* or semantically unrelated to the information like author, source, transaction, timestamp.

Depending on the different strategies Tolle, [19], suggests that there is no universal approach to capture all needed features. As conclusion the quads stores allow the feasible support of one kind of meta-data only.

Context is also the term used in Sesame 2.0, but with a much more specific semantics, as discussed below. In [22], external context is modelled with named graphs.

#### 2.7.4 SPARQL and Sesame 2.0

In SPARQL, RDF Dataset is defined as

$$\{ \mathcal{G}, (\langle U1 \rangle, \mathcal{G}_1), (\langle U2 \rangle, \mathcal{G}_2), \dots (\langle Un \rangle, \mathcal{G}_n) \}$$

where  $\mathcal{G}$  and each  $\mathcal{G}_i$  are graphs, and each  $\langle u_i \rangle$  is a distinct an IRI<sup>4</sup>.  $\mathcal{G}$  is called the default graph. The pairs  $(\langle u_i \rangle, \mathcal{G}_i)$  are called named graphs. As we see, SPARQL adds the notion of "default graph" which is not present in [5]. Sesame 2.0 implements a model identical to the one of SPARQL, but with a minor deviation in the terminology:  $\mathcal{G}_i$  are called "contexts", and the default one is referred as "null context". The Triplet model is a generalization of this one, as long, as it allows for "tagging" of the contextualized triples within each of the graphs in the dataset.

## 2.8 Serialization Formats

Creation of new serialization syntax would require completely new parsers. Thus, we find more appropriate the adoption of an existing standard. The existing standards for serialization of named graphs are obvious candidates: TriG, [2], and TriX, [4].

TriX stands for "RDF Triples in XML" – it is a straightforward XML syntax for representation of RDF triples, which may or may not be part of named graphs. Although being straightforward this format is not necessarily the most readable or practical one. An example follows:

```
<graphset xmlns="http://jena.sourceforge.net/TriX/">
  <graph>
    <id>binfo</id>
    <triple>
      <uri>http://example.org/aBook</uri>
      <uri>http://purl.org/dc/elements/1.1/title</uri>
      <typedLiteral datatype=http://www.w3.org/1999/02/22-rdf-syntax-
ns#XMLLiteral
      >&lt;ex:title xmlns:ex="http://example.org/"&gt;A
GoodBook&lt;/ex:title&gt;</typedLiteral>
    </triple>
    <triple>
      <uri>http://example.org/aBook</uri>
      <uri>http://www.w3.org/2000/01/rdf-schema#comment</uri>
      <plainLiteral xml:lang="en">This is a really good book!</plainLiteral>
    </triple>
    <triple>
      <id>binfo</id>
      <uri>http://example.org/source</uri>
      <uri>http://example.org/book-description.rdf</uri>
    </triple>
  </graph>
</graphset>
```

---

<sup>4</sup> IRI is the internationalized form of URI, <http://www.ietf.org/rfc/rfc3987.txt>

TriG represents an extension of the Turtle syntax for RDF, where each graph can be separated from the others by means curly brackets, as demonstrated below:

```
:G1 { _:Monica ex:name "Monica Murphy" .
      _:Monica ex:email <mailto:monica@murphy.org> .
      G1 pr:disallowedUsage pr:Marketing }
```

Out of the two standards Trig is obviously more compact and readable; it is also fairly close to the syntax of SPARQL. An option which would allow smooth extension of the TriG format is that triplesets are recorded as in comments after the corresponding triple. This way, a triple-set aware parser will be able to load them, while those will be ignored otherwise. A sample snippet follows:

```
:G1 { _:Monica ex:name "Monica Murphy" . #!{ex:ts1, ex:ts2}
      _:Monica ex:email <mailto:monica@murphy.org> .
      G1 pr:disallowedUsage pr:Marketing }
```

The syntax for serialization of tripleset models requires further specification.

### 3 Ontology Representation

Ontology representation deals with semantic interpretation of the data-model layer. ORDI framework is most important for the back-end and middleware ontology infrastructure, which should provide ORDI-compliant services, interfaces, and behavior. It is regarded as a framework, model, and specification to be used as a basis for the development of middleware components and this way to assure data-model and interlingua interface components. It is important to emphasize that ORDI is neutral to the semantics of the ontologies. It only facilitates the representation and basic management of ontologies as well as basic level of integration with other data-sources.

The rest of this section provides an example of modeling WSMML ontology into RDF and explains how the tripleset model can be used for overcoming some of the shortcomings of the WSMML-to-RDF mapping, [8].

#### 3.1 Example to Model WSMML Ontology with ORDI Data Model

Web Service Modeling Language, [7], is an ontology language to formalize the Web Service Modeling Ontology, [21]. The section suggests a simple logical model build on top of ORDI data model to provide flexible ontology representation using domain specific interface. WSMML is frame-based language with human readable interface.

```

1 wsmmlVariant _"http://www.wsmo.org/wsmml/wsmml-syntax/wsmml-flight"
2 namespace {_"http://example.org/bookOntology#",
3   dc _"http://purl.org/dc/elements/1.1/" }
4 ontology _"http://example.org/amazonOntology"
5 nonFunctionalProperties
6   dc#title hasValue "Example Book ontology"
7   dc#description hasValue "Example ontology about books and shopping
8   carts"
9 endNonFunctionalProperties
10 concept book
11   title ofType _string
12   hasAuthor ofType author
13 concept author subConceptOf person
14   authorOf inverseOf(hasAuthor) ofType book
15 concept cart
16   nonFunctionalProperties
17     dc#description hasValue "A shopping cart has exactly one id
18     and zero or more items, which are books."
19   endNonFunctionalProperties
20   id ofType (1) _string
21   items ofType book
22 instance crimeAndPunishment memberOf book
23   title hasValue "Crime and Punishment"
24   hasAuthor hasValue Dostoyevsky
25
26 relation authorship(impliesType author, impliesType document)
27   nonFunctionalProperties
28     dc#relation hasValue authorshipFromAuthor
29   endNonFunctionalProperties
30
31 axiom authorshipFromAuthor
32   definedBy
33     authorship(?x,?y) :- ?x[authorOf hasValue ?y] memberOf author

```

Table 3 Example Listing of WSMML, [7]

A mapping schema between WSML and RDF is presented in [8] and is used to translate the WSML frame based model to RDF data model. The authors outline two important issues related to lack of contextualization in RDF model, [8]:

- WSML allows meta-modeling of entities, where every type has a different set of non functional properties.
- The WSML/RDF schema describes the functional and non-functional properties in the same way. They are distinguished as non-functional properties are marked as `owl:AnnotationProperty`. Table 4 shows the lost information.

<pre>// wsml source data instance instA   nonFunctionalProperties     dc:title hasValue "Instance A"   endNonFunctionalProperties   dc:title hasValue "Instance B"</pre>	<pre>// resulting RDF triples instA dc:title "Instance A" . instA dc:title "Instance B" . dc:title rdf:type owl:AnnotationProperty .</pre>
--	--

Table 4 Mapping Between WSML and RDF Expressed in N3 format

Many real application use cases require a more sophisticated management of the triples than a static knowledge base. Obviously the updates of the existing information are not possible if the source or the last update date of a document could not be determined.

A possible solution of the problem may be realized using the quads model. Several possible approaches exist, but each of them has serious drawbacks. The first revised one is to use the fourth context element as identifier of the triple.

Subject	Predicate	Object	Statement ID
instA	dc:title	"Instance A"	c:statement1
instA	dc:title	"Instance B"	c:statement2
dc:title	rdf:type	owl:AnnotationProperty	c:statement3
c:statement1	rdf:type	wsml:NFP	c:statement4
c:statement2	rdf:type	wsml:FP	c:statement5
c:statement1	wsml:source	http://wsmo.org	c:statement6
c:statement2	wsml:source	http://wsmo.org	c:statement7
c:statement3	wsml:source	http://wsmo.org	c:statement8

Table 5 The example from Table 4 extended with meta-data

If example from Table 5 is extended with additional meta-data like the last update date or another kind, the number of explicit asserted statements will get at least doubled. Even much optimized implementation will suffer major performance loss.

Another approach is to use the fourth element as *named graph* identifier, [5], and to model a nesting hierarchy of *named graphs*: each statement belongs to a single named graph, that is a leaf in the NG-hierarchy; as with many OO-languages, triple membership is propagated upwards on the hierarchy (the triples belonging to a NG, belong also to its parents) and meta-information is inherited downwards (the properties of the parent NG are spread to its children). However this approach is impractical when the complex hierarchy structure has to be synchronized with the nearly unpredictable changing meta-data.

Much more natural way to solve the described problem is to use the ORDI data model, which preserves the efficiency of the quadruple data model if used for the same complexity storage

tasks and in the same time offers a great flexibility tackling complex application use case scenarios.

Subject	Predicate	Object	NG	Triplesets
instA	dc:title	"Instance A"	http://wsmo.org	ts:updateDate1; wsml:InstanceNFP
instA	dc:title	"Instance B"	http://wsmo.org	ts:updateDate1; wsml:InstanceFP
ts:updateDate1	rdf:type	LastUpdate		
ts:updateDate1	rdf:valueof	"28/10/2006"		
instA	dc:title	"Instance B"	http://wsmo.org	ts:updateDate1; wsml:ConceptNFP

Table 6 Result after WSMML to ORDI data model mapping

The above examples express the document source as *named graph*. This eliminates the potential problem for a cross-reference of unnumbered anonymous identifier across multiple documents and the need to create universally unique identifier [7], because the blank nodes in different *named graphs* are disjoint, [5]. This way, every application specific meta-data can be persisted in a very efficient way.

As conclusion, ORDI data model offers extreme flexibility by maintaining the full backward compatibility to RDF W3C recommendation, [16], and *named graphs*, [2]. The users of ORDI data model that do not handle complex application scenarios may continue to use one of the above models, without significant performance penalties and on the other hand keep their option for future extension toward more complex scenarios to require efficient support of meta-information.

## 4 Architecture design

Architecture design covers the main aspects of the reference implementation of the ORDI data model and the ontology representation principles described in the previous sections. The section provides a high-level overview of the design with a short summary for the different parts of the system with their functionality. Some technical aspects are covered in order to fully motivate the choice of concrete design decisions.

### 4.1 Architectural Requirements

ORDI's architecture should cover the following basic requirements:

- Allow for different implementations of the underlying storage and inference modules;
- Allow for wrapping and integration of different structured data sources;
- Distribution of the data model to multiple storage locations:
  - Mechanism for efficient data copy between data store instances, to make possible implementation of different schemata for replication;
  - Stacking, consolidating, and "multiplexing" universal data model repositories by providing virtual super store.

### 4.2 Structural overview

The structural overview section targets to clarify all the different functional and data storage components. ORDI framework is designed mainly with extensibility and flexibility in mind. It provides a generic API to define domain- or application-specific models and to integrate them with the existing functionality.

The architecture is defined as an extension of the Sesame<sup>5</sup> 2.0, Alpha-3, architecture. Sesame is one of the most mature, popular, and efficient, RDF management frameworks. This approach allows for re-use of mature RDF-management infrastructure and better interoperability with it. The architecture of ORDI framework is shown on Figure 4.

---

<sup>5</sup> <http://www.openrdf.org>

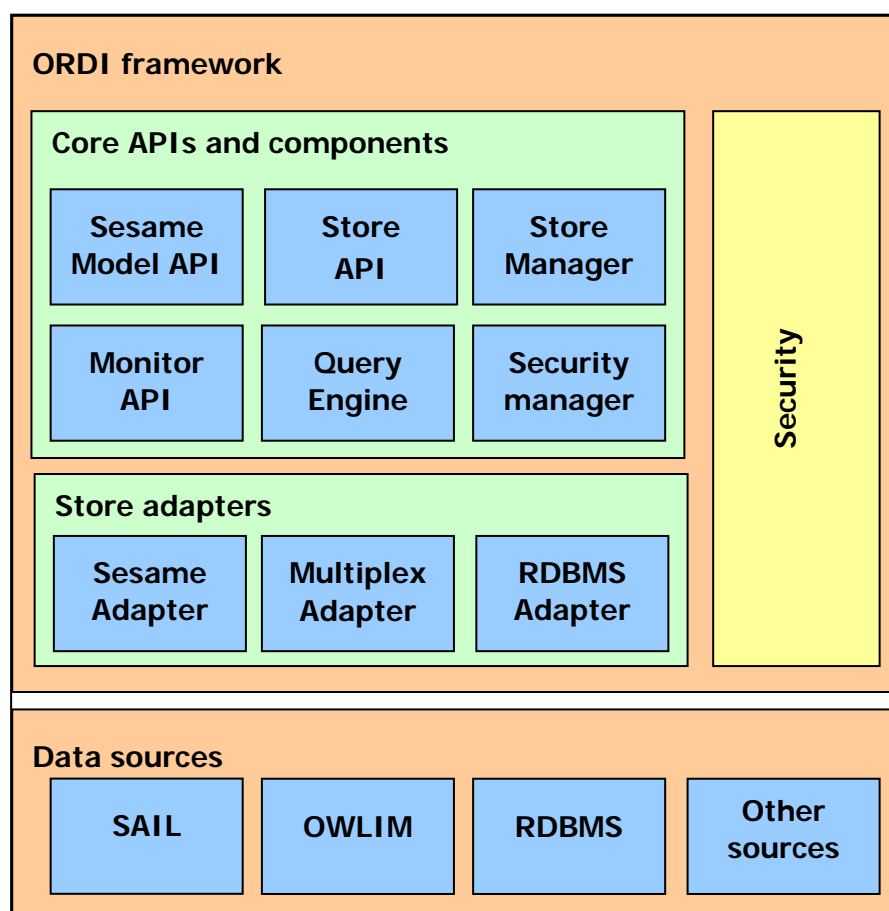


Fig. 5 Architecture of ORDI framework

ORDI framework is composed by core APIs and components and store adapters. The core components and APIs define the main functionality of the framework by introducing set of Java interfaces and implementations to interact with them:

- `Sesame Model` is a lightweight API. It is responsible to describe generic concepts like RDF statement, resource, literal, blank node and URI; for more information check **Error! Reference source not found..** ORDI framework reutilizes the generic packages `org.openrdf.model` part of Sesame in order to ensure compatibility with the high performance RIO parsers implementations.
- `store API` describes how the different framework components to interact with the functionality offered by the contained stores. A design principle is to allow no components to interact with implementation outside of its component scope.
- `store Manager` is some sort of data store registry. It provides basic operation to manipulate the available configured data stores, based on the functionality they implement
- `Monitor API` is still undefined. The requirements towards it are to provide sophisticated mechanism to trace internal system functionality and to optimize the operation over the framework.
- `Query Engine` process SPARQL based queries and creates an executable conjunction of patterns to be matched against any arbitrary data store.

- `Security Manager` is component responsible to define the security across the different system parts. Its functionality is still undefined, because of lack of significant requirements

The ORDI store adapters realize the defined interface by the Store API. They could be divided to two main groups concerning the added functionality to the framework:

- Adapters to provide data. They ensure a mapping between an interface part of the Store API and another one supported by third-party persistent store. Such adapters are the Sesame adapter to map the `TripleSet` data model to SAIL, or RDBMS adapter to transform legacy database information data set to ontology, based on specified schema. The mapping depending on the implementation could be complete or partial; and supported in one or the both direction.
- Adapters to provide new functionality. They work over adapter to provide data to introduce new functionality. Example for functional adapter is the Multiplex Adapter that implements the `MultiTripleSetModel` to union the information located in several tripleset stores.

Appendix A represents three UML diagrams of the most important interfaces in the ORDI architecture.

## 5 Conclusion

The specification of the second generation of the Ontology Representation and Data Integration (ORDI) framework was presented in terms of underlying data-model and architecture. To be able to serve the full range of data integration and ontology management tasks ORDI is based on an extension of the RDF model which includes support for: contextualized triples (quadruple or triples in named graphs) and triplesets, which represent named groups of such triples. This model represents an extension of several other models which support named graphs, most notably, the one of the SPARQL query language.

The architecture is specified in terms of APIs, which determine the major components and define the interfaces for both usage and extension of an implementation of ORDI. The design is similar to the one of the Sesame RDF database – the central `store` interface, and its successors provide methods for storage, retrieval and modification of data represented with respect to ORDI's data model. Store interfaces can be stacked on top of each other, to allow for flexible extension of the functionality. The extended `store` interfaces provide more specific functionality: tripleset support, aggregation of multiple data stores (`MultiTripleSetStore`), etc. The methods which modify the data are separated in a set of interface related to `Transaction`.

This specification is subject of development in the following directions:

- The syntaxes for serialization of triplesets should be determined;
- The interpretation and usage of triplesets in SPARQL should be elaborated, as well as the support for different query formalisms;
- The reasoner integration schemata should be discussed;
- The architecture should provide more concrete specification about the behaviour of a multi-store, aggregating data from several existing ones;
- Several models for distribution need to be specified.

## 6 References

- [1] Alexander, N; Ravada, S. (2005) *RDF Object Type and Reification in Oracle*. [http://download-uk.oracle.com/otndocs/tech/semantic\\_web/pdf/rdf\\_reification.pdf](http://download-uk.oracle.com/otndocs/tech/semantic_web/pdf/rdf_reification.pdf)
- [2] Bizer, C. (2005) *The TriG Syntax*. <http://sites.wiwiw.fu-berlin.de/suhl/bizer/TriG/>
- [3] Bizer, C; Cyganiak, R; Watkins R. (2005) *NG4J - Named Graphs API for Jena V0.4*. <http://www.wiwiw.fu-berlin.de/suhl/bizer/ng4j/>
- [4] Carroll, J. J; Stickler, P. (2003) *RDF Triples in XML*. <http://www.hpl.hp.com/techreports/2003/HPL-2003-268.pdf>
- [5] Carroll, J. J; Bizer, B; Hayes, P; Stickler, P. (2005) *Named Graphs, Provenance and Trust*. WWW2005, <http://www2005.org/cdrom/docs/p613.pdf>
- [6] de Bruijn, J; Lausen, H; Krummenacher, R; Polleres, A; Predoiu, L; Kifer, M; Fensel, D. (2005) *D16.1v0.21 The Web Service Modeling Language WSML*. WSML Final Draft 5 October 2005, DERI. <http://www.wsmo.org/TR/d16/d16.1/v0.21/>.
- [7] de Bruijn, J; Kopecky, J; Krummenacher, R. (2005) *WSML - a Language Framework for Semantic Web Services*. <http://www.w3.org/2004/12/rules-ws/paper/44/>
- [8] de Bruijn, J. (ed.); Kopecky, J; Krummenacher, R. (2006) *WSML/RDF*. Deliverable d32v0.1. WSML Working Draft 15 February 2006. <http://www.wsmo.org/TR/d32/v0.1/20060215/>
- [9] Dean, M; Schreiber, G. – editors; Bechhofer, S; van Harmelen, F; Hendler, J; Horrocks, I.; McGuinness, D. L; Patel-Schneider, P. F.; Stein, L. A. (2004) *OWL Web Ontology Language Reference*. W3C Recommendation 10 February 2004. <http://www.w3.org/TR/owl-ref/>
- [10] DCMI Usage Board. (2003) *DCMI Type Vocabulary*. <http://dublincore.org/documents/2003/11/19/dcmi-type-vocabulary/>
- [11] Dollin, C; McBride, B. (2003) *Jena2 reification API proposal*. <http://jena.sourceforge.net/reify-api.html>
- [12] Gruber, T. R. (1993) *A translation approach to portable ontologies*. Knowledge Acquisition, 5(2):199-220, 1993. [http://ksl-web.stanford.edu/KSL\\_Abstracts/KSL-92-71.html](http://ksl-web.stanford.edu/KSL_Abstracts/KSL-92-71.html)
- [13] Hayes, P. (2004) *RDF Semantics*. W3C Recommendation 10 Feb. 2004. <http://www.w3.org/TR/2004/REC-rdf-mt-20040210/>
- [14] Kiryakov, A; Ognyanov, D; Kirov, V. (2004) *An Ontology Representation and Data Integration (ORDI) Framework*. DIP project deliverable D2.2. <http://dip.semanticweb.org>
- [15] Kiryakov, A; Ognyanov, D. (2006) *An Ontology Representation and Data Integration (ORDI) Framework Implementation*. DIP project deliverable D2.3. ver. 0.4, Jun 2006. <http://www.ontotext.com/ordi/v0.4/FactSheet.html>
- [16] 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/>
- [17] MacGregor, R; Ko, I.-Y. (2003) *Representing Contextualized Data using Semantic Web Tools*. Proc. of the 1<sup>st</sup> International Workshop on Practical and Scalable Semantic Systems; available at <http://km.aifb.uni-karlsruhe.de/ws/pss03/proceedings/macgregor-et-al.pdf>
- [18] Schockaert, S. (2006) *RDF Reification: Problems, Opportunities and Alternatives*. <http://wise.vub.ac.be:8668/space/Steven+Schockaert+WIS+Page/rdf-position-paper.pdf>
- [19] Tolle, K. (2004) *Understanding data by their context using RDF*. Int. Conf. on Advances in Intelligent Systems: Theory and Applications(AISTA04), Luxemburg (2004). <http://www.dbis.informatik.uni-frankfurt.de/~tolle/Publications/2004/AISTA04.pdf>
- [20] 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/>

- [21] Roman, D; Lausen, H; Keller, U. (eds.) 2005. *Web Service Modeling Ontology (WSMO)*, WSMO deliverable D2v1.2. WSMO Final Draft, 13 April 2005. <http://www.wsmo.org/TR/d2/v1.2/>
- [22] Stoermer, H.; Palmisano, I; Redavid, D; Iannone, L; Bouquet, P; Semeraro, G. (2006) *RDF and Contexts: Use of SPARQL and Named Graphs to achieve Contextualization*. Jena User Conference, May 10-11, 2006, Bristol.
- [23] Watkins, E. R. and Nicole, D. A. (2006) *Named Graphs as a Mechanism for Reasoning about Provenance*. Lecture Notes in Computer Science Frontiers of WWW Research and Development - APWeb 2006: 8th Asia-Pacific Web Conference, Harbin, China, January 16-18, 2006. Proceedings 3841 pp. 943-948.

## Appendix A

UML diagrams represent as follows: the Sesame Model package; the Store-related interfaces; and the Transaction-related interfaces.

