Abstract: Although the faster development of the semantic web technologies provides new scenarios to interchange and interoperate among different automated or semi automated systems, in big datasets, there are several problems that influence in a low performance. On the other hand, due to the special store mode of RDF data the inference and reasoning capabilities are more costly. In a web server environment, this fact not provides a real room to enable the semantic web vision. We are developing a new approach to semantic data inference based on the use of object oriented storage technologies.

Key-Words: Semantic Web, RDF Storage, Semantic Web Servers.

1 Introduction
The evolution of the Semantic Web has contributed to new vision on the treatment of the information. Any server can publish information on the Web defined by a schema and linked in such a way that it can be used by machines not just for display purposes, but for automation, integration and reuse of data across various applications. The new generation of web-based applications uses the semantic web technologies. RSS[10], FOAF[11] and other projects are examples of this fact. The last news of the NASA server, or the last multimedia resources using the podcast channels can be inserted inside of a web application using this technologies. The semantic web specifications are widely used. Recent studies shows as an application server can be a data provider of information based on schemas described in semantic web languages (i.e. RDF[4], OWL[5]). To retrieval the information, the SPARQL protocol has been defined. The products of the HP research about this field are the Jena Framework and the SPARQL Server “Joseki”[6].

Although the faster development of the semantic web technologies provides new scenarios to interchange and interoperate among different automated or semi automated systems, in big datasets, there are several problems that influence in a low performance. On the other hand, due to the special store mode of RDF data (triples and reification process, explained in the next section) the inference and reasoning capabilities are more costly, therefore these low-performance rest efficient in a web server environment.

In this paper, the main troubles to achieve a good-performance in a web server that provides information in RDF or other semantic high-level language (i.e. OWL) are described in the next section. In the third section we explain our solution to the inference and retrieval data in large datasets using an underlying OODBMS to store RDF. Finally, we explain our conclusion and the further work in the section four.

2 Problem Formulation
Recent studies showed the efficient problems on the semantic web application, in particular, the storage way and the inference process. The next subsections explain the different ways to store RDF and make several inferences on a big dataset.

2.2 RDF Data Structure
The RDF data structure is a directed graph. Each node represents a resource (subject) that is linked to other resource (object) thought a direct arc labeled with a third resource (predicate). This basic representation has a semantic defined by the statement “subject has a property (predicate) valued by an object”. The structure of \{<subject>,<predicate>,<object>\} with the following declaration is called “triple”:

- Subject: An RDF URI reference or a blank node.
- Predicate: An RDF URI reference.
- Object: An RDF URI reference, a literal or blank node

The figure 1 shows an example of triple
A collection of triples builds a graph that represents a RDF model. The following section explains how to store this model.

### 2.2 RDF Storage

There are three ways to store a RDF model: in memory, in a file or in a database. In any of the cases, the triple is the basic piece to store a RDF model.

#### 2.2.1 In-Memory Storage

The graph is the best structure to store RDF in memory. Each node described a resource or a literal; all resources can be linked to other nodes thought a property.

#### 2.2.2 In-File Storage

The RDF/XML Syntax specification proposes an XML syntax for encoding RDF graphs. However, the standard RDF/XML mapping is unsuitable for this since multiple XML serializations are possible for the same RDF graph, making retrieval complex.

N3 is a language that offers a compact and readable alternative to RDF’s XML syntax, keeping its expressiveness. Simple and consistent grammar, readable and URI abbreviation are some of the most outstanding features.

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix : <http://www.yo-on.com/oxd#> .
:Elisa dc:creator :PaperX ;
:PaperX dc:title "OntoSecurity" ;
```

#### 2.2.3 Jena RDF Storage

Jena is a framework for Semantic Web Development, is widely used, and it’s an open-source project, implemented in Java. The Jena architecture provides an abstract RDF model to manage an internal graph that store the RDF model. The applications typically interact with an abstract Model which translates higher-level operations into low-level operations with triples stored in an RDF Graph.

The Jena database subsystem implements persistence for RDF graphs using a relational database through a JDBC connection.

The experience with Jena1 (first version), exposes that the uses of a denormalized relational schema reduce response times [1]. The current version of Jena (second version, called Jena2) trades-off space for time. Both resource URI and simple literal values are stored directly in the statement table (stmt table). In order to distinguish literals and URIs, columns values are encoded with a prefix (such as Uv::, Lv::) that indicates witch the kind of value. If the length of a literal value exceeds a threshold, the literal value is stored in a separated table (long lit table). By storing values directly in the statement table it is possible to perform many find operations without a join. Although the size of the statement table is a problem, Jena2 provides several options to reduce it, such as, compress namespaces (by defining a prefix in the long uri table and using these prefix like a reference to the namespace), to store the long values only once (by deriving to the long lit table) and the use of the property tables.

#### 2.2.4 Sesame RDF Storage

Sesame is an open source RDF database with support for RDF Schema inferencing and querying information.

The progress in the development of Sesame has showed some RDF store features. The performance of Sesame together with object-relational DBMS has proved in several studies [3]. All agree in the same conclusion: The performance is very low, if the database system creates a table whenever a new class or property is added, therefore in scenarios where the schema changes often, the RDF graph’s direct mapping are not valid in the RDF store over object-relational DBMS.

The study realized with PostgreSQL in [3] uses a different approach (similar to Jena) in witch all RDF statements are inserted in a single table with three columns: “Subject, Predicate, Object”. In scenarios where the schema changes often, this approach is better than the previous.

#### 2.2.5 RDF in Oracle

The Sesame’s study reflects the fall down performance using RDF persistence over ODBMS. The Oracle approach to store RDF utilizes a new object type (SDO_RDF_TRIPLE_S) for storing RDF data. The RDF object type is built on top of the Oracle Spatial Network Data Model (NDM) [2]. NDM is Oracle’s optimal solution for storing, managing, and analysing networks or graphs in the database. Triples are parsed and stored in the system as entries in the NDM node$ and link$ tables. Nodes in the RDF network are uniquely stored and are reused when encountered in incoming triples. A key feature of RDF storage in Oracle is that nodes are stored only once – regardless of the number of times they participate in triples. Besides, only one new triple is stored for each reification - Oracle uses XML DB DBUri to reference the reified triple in the
database. This approach uses less storage and reduces performance overhead for querying. However is a complex model to manage rdf information, the user of the Oracle RDF objects might be realized all low-level operations. Besides, not yet there is a persistent framework to manage the RDF information store in Oracle.

2.3 Inference problem in large DataSets
The main problem in semantic data inferences over RDF is the efficiency to access the data stored on relational based repositories. This kind of problem can be detected clearly when you are running systems like Jena or Sesame with OWL using big amount of data for example, base knowledge from Opencyc, these systems usually do the same mistake: The did not implement an efficient storage mechanism. Some of them work just in memory and some others make the mistake of overflow the network connection by sending too much queries to the RDBMS.

3 Problem Solution
Our solution is based on the application of some basic technologies of object oriented software engineering. We can improve the performance of inferences over big amount of data using an object oriented database, because it implements “on demand materialization” and some concepts as “class extent”.

3.1 Object Oriented Data Bases
An OODBMS (Object Oriented Database Management System) is the son of the couple: object oriented programming plus database management principles. An ODBMS makes database objects appear as programming language objects in one or more object programming languages. The Object Oriented Database Manifesto [7] specifically lists the following features as mandatory for a system to support before it can be called an OODBMS: complex objects, object identity, encapsulation, types and classes, class or type hierarchies, overriding, overloading and late binding, computational completeness, extensibility, persistence, secondary storage management, concurrency, recovery and an ad hoc query facility.

Some of these characteristics are extensions of a typical object oriented language such a C++ o Java. These characteristics are very useful for retrieve knowledge in RDF format. One of the most important features of an OODBMS is that accessing objects in the database is done in a transparent manner such that interaction with persistent objects is no different from interacting with in-memory objects. This is very different from using an RDBMSs in that there is no need to interact via a query sub-language like SQL nor is there a reason to use a Call Level Interface such as ODBC, ADO or JDBC.

3.2 Why ODBMS are betters?
Consider an ODBMS when you have a need for high performance on complex data. Generally, an ODBMS is a good choice when you have all three factors: business need, high performance, and complex data. There are some disadvantages of using OOBMS, but they not so important in the case of using a OODBMS for storing RDF semantic data.

Caching Architecture: It allows large datasets to be situated close to the application, reducing the network and data access latencies that adversely impact applications operating against other databases.

Transparent Persistence: Depending on the language you are using, (almost any object oriented language: Java, C++, Smalltalk, etc) its support complete object models in the database, the databases delivers transparent persistence, eliminating any need for the object-relational mapping code that typically comprises anywhere between 25% - 70% of the coding effort in applications that use relational databases.

Class Hierarchy: Data in the real world is usually has hierarchical characteristics. The ever popular Employee example used in most RDBMS texts is easier to describe in an OODBMS than in an RDBMS. An Employee can be a Manager or not, this is usually done in an RDBMS by having a type identifier field or creating another table which uses foreign keys to indicate the relationship between Managers and Employees. In an OODBMS, the Employee class is simply a parent class of the Manager class.

No Impedence Mismatch: In a typical application that uses an object oriented programming language and an RDBMS, a significant amount of time is usually spent mapping tables to objects and back. There are also various problems that can occur when the atomic types in the database do not map cleanly to the atomic types in the programming language and vice versa. This "impedance mismatch" is completely avoided when using an OODBMS.

3.3 Persistence Framework.
Even if you want to use a RDBMS in spite of an object oriented one. You may want to get some of the object oriented advantages, this is possible using a persistence framework.

A persistence framework moves the program data in its most natural form (in memory objects) to and from a permanent data store the database. The persistence framework manages the database and the mapping.
between the database and the objects. There are many persistence framework (both Open Source and Commercial) in the market. Persistence framework simplifies the development process.

There are concepts in the relational database model that are similar to those in the object database model. A relation or table in a relational database can be considered to be analogous to a class in an object database. A tuple is similar to an instance of a class but is different in that it has attributes but no behaviors. A column in a tuple is similar to a class attribute except that a column can hold only primitive data types while a class attribute can hold data of any type. Finally classes have methods which are computationally complete (meaning that general purpose control and computational structures are provided [8]) while relational databases typically do not have computationally complete programming capabilities although some stored procedure languages come close.

Our framework is based on the one defined by [9], and is developed as a group of design patterns. The patterns that conforms the persistence framework are:

**Virtual Agent (intelligent reference):** is some kind of Proxy pattern whose main objective is enable us to manage the objects on the RDBMS in the same way as they were objects in memory.

**Broker:** the Broker design pattern has responsibility of materialize from the RDBMS, the objects that are needed by a semantic query (figure 2). The broker is responsible of the “on-demand slow materialization”, this way of doing permit that only the objects which are really necessary are stored in “real memory”. This method is implemented directly in the OODBMS also, and is one of the best advantages compared with the Jena materialization method used now.

![Logical Model](image)

Figure 2 – Broker desing pattern

4 Conclusion

Once we have study the data about the problems suffered by Jena when you try to use a fast mechanism. Studying the main features of the object oriented databases and the persistent need, we are now developing two prototypes, one based on the object oriented database called “FastObjects” and other based on the persistence framework described in [9]. This will enable us to validate the efficiency improvement using object oriented technologies to access semantic data.

References


