TRIOO
Keeping the Semantics of Data Safe and Sound into Object-oriented Software

Sergio Fernández, Diego Berrueta
Fundación CTIC, C/ Ada Byron 39, Parque Científico y Tecnológico, Gijón, Asturias, Spain

Miguel García Rodríguez, Jose E. Labra
Computer Science Department, Universidad de Oviedo, Campus Los Catalanes, Oviedo, Asturias, Spain

Keywords: RDF, SPARQL, Web, Data, Persistence, Object-oriented programming.

Abstract: Data management is a key factor in any software effort. Traditional solutions, such as relational databases, are rapidly losing weight in the market towards more flexible approaches and data models due to the fact that data stores as monolithic components are not valid in many current scenarios. The World Wide Consortium proposes RDF as a suitable framework for modeling, describing and linking resources on the Web. Unfortunately the current methods to access to RDF data can be considered a kind of handcrafted work. Therefore the Trioo project aims to provide powerful and flexible methods to access RDF datasets from object-oriented programming languages, allowing the usage of this data without negative influences in object-oriented designs and trying to keep the semantics of data as accurate as possible.

1 INTRODUCTION

Software data management has experienced a notable evolution in the last few years. Relational databases are still one the most powerful and used solutions for data storage in many cases, but there are many reasons to discard it as a suitable technology: On the one hand, the emergence of new distributed architectures of software requires new approaches able to fulfil all these new requirements relying on availability and flexibility. Unfortunately federated databases do not yield the expected features and performance in some scenarios. On the other hand, it is impossible to normalise a relational schema when data schema is not clearly decided or condensed. Both issues have motivated a lot of research in the last few years. Concerning distributed storage, the software industry currently offers some quite interesting solutions: Google uses Bigtable or Pregelin many of their applications, Microsoft developed BitVault, among others. With respect to how to efficiently store schema-free data, there are other promising proposals facing this challenge: from proprietary data models, as solutions used in many of the distributed storage systems or in native object-oriented databases, to more open solutions, such as the XML-based databases.

The World Wide Consortium[1] adopted another approach, often known as the Semantic Web (Berners-Lee et al., 2001). Semantic Web aims to provide a set of technologies that would facilitate the implementation of solutions to this (and others) current problems, based on the decentralised Web architecture, the idea of a "*Web of Data*". Obviously it has some additional open issues, such as the trust and provenance of data, that do not apply to an intranet environment; but once they are solved, the "*Web of Data*" reveals as a much more powerful solution than a one. W3C's main proposal is build upon the current Web. So that means using HTTP (Fielding et al., 1999) as basic protocol for transfer information and XML as basic format for encoding documents, but enriched with the RDF technologies stack. RDF (Klyne and Carroll, 2004) (*Resource Description Framework*) provides a flexible and extensible data model, based on the concept of "triple" `(subject, predicate, object)`, that greatly simplifies data interoperability, interchange and integration across different heterogeneous sources and applications. RDF extends the linking structure of the Web to

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

Fernández S., Berrueta D., García Rodríguez M. and E. Labra J. (2010).
TRIOO - Keeping the Semantics of Data Safe and Sound into Object-oriented Software.
In *Proceedings of the 5th International Conference on Software and Data Technologies*, pages 311-320
DOI: 10.5220/0003008803110320

use URIs to name the relationship between things instead of documents (Berners-Lee et al., 2005; Sauermann and Cyganiak, 2008). This architecture is complemented with languages to model knowledge (RDFS (Brickley et al., 2004) and OWL (Schneider et al., 2009)) and languages to natively query RDF data such as SPARQL (Prud'hommeaux and Seaborne, 2008)/SPARUL (Schenk and Gearon, 2010).

Consequently RDF allows to relate data from different providers, interlinking resources from different datasources, as the "*Linked Data*" principles (Berners-Lee, 2006) dictate. The Linked Data initiative[2] is growing quite fast, and a large amount of data is already available on the Web as RDF, ready to be consumed by new applications capable of exploiting this innovative paradigm. This means a new way to integrate the Word Wide Web with business data and applications. Due to these features, RDF may be a key technology to store the data in the next generation of software applications, becoming a common layer where data can be stored and retrieved.

In this position paper we present the Trioo project (which name means something like "*triples object-oriented*"), an ongoing initiative that aims to add support to some of the most important object-oriented programming languages for managing RDF data based on those standard technologies in an efficient and flexible way. Because at the end "are objects, not triples"[3]. A key premise of the project is that software developers must not be forced to adapt their oo-design keeping the semantics of data safe and sound in the software.

This paper is structured as follows: Section 2 analyses the related work relevant to the objectives addressed by Trioo. Section 3 describes in detail these objectives and the design issues found, facing and comparing both computational models, RDF and object-oriented. Section 4 briefly depicts the ongoing implementations in the current early state of development. Finally Section 5 discusses the current conclusions, setting the basic guidelines for the work in the upcoming months.

2 RELATED WORK

The ongoing work described in this paper is extensively related to the work made these last years on object-relational mappings, but also to work in accessing (Semantic) Web data. On these fields, and for better understating of this analysis, it is necessary to introduce two relevant design patterns:

Active Record is a widely used design pattern to work with relational databases where each row is represented as a single object (Fowler, 2002). Therefore, ActiveRecord can be considered as an approach to access databases: each table (or view) is *wrapped* as a class, and each row as a instance object. The assumption "*one object, one row*" greatly simplifies the development of CRUD functions (Create, Read, Update and Delete) on relational databases. In fact, most of the current ORMs ("*Object-Relational Mapping*") implements this design pattern, or an adaptation of it.

Data Mapper is an adaptation of the Mapper design pattern (Fowler, 2002). Objects and relational databases have different mechanisms for structuring data, and many parts of an object, such as business logic, collections and inheritance, are not present in relational databases. The object schema and the relational schema do not need to match up. The Data Mapper is a layer of software that separates the in-memory objects from the database. Its responsibility is to transfer data between the two and also to isolate them from each other. With Data Mapper the in-memory objects do not need to know even that there is a database present; they do not need SQL interface code, and certainly no knowledge of the database schema; in fact the database schema is always ignorant of the objects that use it.

These two design patterns are probably the most relevant and used for object-data mapping, but there are many more, such as "*Table Data Gateway*" or "*Row Data Gateway*", among others. However this is not a full report about these patterns, just a brief introduction of the concepts, in order to illustrate how data access use to be implemented in modern software systems. Although the target of the Trioo project are just RDF-based stores, one of the aims of this position paper is to extract (positive and negative) conclusions from other current solutions. Therefore this state of the art will not be restricted to RDF stores, but it will cover a wider range of approaches in order to reach the best possible solution in Trioo.

2.1 Traditional Stores

In this paper the term "*traditional stores*" actually concerns traditional relational databases. There are several software solutions that solve the object-relational impedance mismatch problems by replacing direct persistence-related database

[2] http://linkeddata.org/

[3] http://harth.org/andreas/blog/2009/03/27/its-objects-not-triples/

accesses with high-level object handling functions. Some of the most relevant are (by alphabetic order):

ActiveRecord[4] is the implementation of the pattern with the same name used in the popular Web framework Ruby on Rail, which main contribution to the pattern is to relieve the original of two stunting problems: lack of associations and inheritance. By adding a simple domain language-like set of macros to describe the former and integrating the Single Table Inheritance pattern for the latter, Active Record narrows the gap of functionality between the data mapper and active record approach. It follows the principle DRY philosophy[5]. There are many other implementations for many other languages that follow a quite similar approach, such as Django Models for Python, nHydrate and Castle for C# (this last one actually an implementation built on top of NHibernate), CakePHP for PHP, or GORM for Groovy.

Hibernate originally is an implementation of the Data Mapper design pattern for the Java language, but currently is much more, providing a framework for mapping an object-oriented domain model to a traditional relational database (Bauer and King, 2004). The current version is a certified implementation of the Java Persistence API 1.0 specification (JPA (DeMichiel and Keith, 2006)), but hopefully soon it will start to support the recent version 2.0 (JSR317 (DeMichiel, 2009)). Hibernate also provides a SQL inspired language, called Hibernate Query Language (HQL), which allows SQL-like queries to be written against Hibernate's data objects. Undoubtedly, Hibernate is the most extended data mapper for the Java world. A port for the .NET framework, known as NHibernate[6], has been even carried out. Nowadays JPA has become so popular that there are several implementations, although Hibernate is still the most popular one.

iBATIS is an implementation in Java, .NET and Ruby of the Active Record design pattern, providing a persistence framework which automates the mapping between relational databases and objects. It takes the reverse approach than, for instance, Hibernate: iBatis automates the creation of POJOs (Plain Old Java Objects) from an already existed relational database.

All these solutions allow to develop persistent classes over relational databases following object-oriented idiom, including association, inheritance, polymorphism, composition, and collections management.

2.2 RDF-based Stores

The critical problem of how to efficiently query RDF datasets has been there since the origins of the Semantic Web until today (Sintek and Decker, 2002). It is true that the current technological landscape has substantially improved: there is a standard query language (SPARQL), efficient RDF stores (Virtuoso[7], Sesame[8], 4Store[9], and many others), and libraries that support all this technology. However there is still a huge gap on how all this RDF-based data is actually managed on the software systems. As described above, a RDF class is not exactly the same that a class in object-orientation, even though there are some solutions based on this approach[10]. Anyway there are some interesting attempts to be analysed:

ActiveRDF proposed a novel way to work with RDF with object-orientation on Ruby (Oren et al., 2008), based on the ActiveRecord design pattern (Fowler, 2002), and clearly influenced by the work on model-driven Web development. When it was developed, SPARQL did not have data update capabilities, so it works over proprietary interfaces (based on a pluggable architecture that allows new implementations) for each store for persisting the RDF data the query language has evolved since then, but apparently there is not any plan to support its new features.

Empire[11] is an implementation of a large chunk of the core of JPA to provide an interface to RDF databases (currently only 4Store, Sesame, and Jena) using SPARQL, SeRQL (Broekstra and Kampman, 2003) and the supported stores' proprietary API, by a small annotation framework for tying Java beans to RDF.

JenaBean[12] persists POJOs to Jena's models[13], and back. Although it is annotation-based, requires to extend a templated class (RdfBean) to gain RDF

[4] http://ar.rubyonrails.org/

[5] Don't Repeat Yourself

[6] https://www.hibernate.org/343.html

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

[8] http://www.openrdf.org/

[9] http://4store.org/

[10] Such as the one use by Soprano (http://soprano.sourceforge.net/apidox/trunk/soprano_devel_tools.html#onto2vocabularyclass) or by Protégé (http://protegewiki.stanford.edu/index.php/JSave)

[11] http://github.com/clarkparsia/Empire

[12] http://jenabean.googlecode.com/

[13] http://jena.sourceforge.net/javadoc/com/hp/hpl/jena/rdf/model/Model.html

capabilities, which is a heavy design restriction on programming languages with single inheritance like Java.

OntoBroker originally consists of a number of languages and tools that enhance query access and inference service of the World Wide Web (Fensel et al., 1998). This is based on the use of ontologies to annotate Web documents and to provide query access and inference service that deal with the semantics of the presented information. Nowadays OntoBroker[14] has become a consolidated commercial product that provides a high performance and scalable Semantic Web reasoning middleware, supporting several standard technologies such as OWL, RDF, RDFS, SPARQL or F-logic.

Ripple is a relational, stack-based data-flow language for the Semantic Web (Shinavier, 2007), a versatile framework for traversal-based algorithms on semantic networks. The open source implementation[15] is written in Java and includes a command-line interpreter as well as a query API which only interoperates with the native API of Sesame RDF store.

SuRF[16] is another implementation of the Active Record pattern. It exposes the RDF triple sets as sets of resources and seamlessly integrates them into the object-oriented paradigm in Python, in a similar manner as ActiveRDF does for Ruby. In can work both over files or RDF stores, using SPARQL for reading where possible, but implementing proprietary interfaces for writing data to a couple of natively supported stores.

This analysis draws that many these products use the same approaches for RDF than their equivalents for other technologies. For instance, even though the *Active Record* pattern fits well for relational databases, that is not so for RDF: a RDF object is not a row, it is a set of triples. Thus RDF data model has many differences that should be taken into account for developing such a kind of tools. Therefore it would be necessary learn from other more developed technologies, but it must flee from negative influences. Many of these details will be described below in this paper.

[14]http://www.ontoprise.de/en/home/products/ontobroker/
[15]http://ripple.googlecode.com/
[16]http://surfrdf.googlecode.com/

2.3 Other Families of Stores

Data storage embraces many more technologies than those described above, such as object-oriented databases. Unfortunately object-oriented databases are still minor players with reduced market niches. And meanwhile some of the products described above have added object capabilities to relational databases. In a parallel way there are other alternatives that do not require fixed table schemes, and usually avoid join operations. It is worth highlighting the NoSQL initiative[17], that actually it is not a concrete product, but a portmanteau term for a loosely defined class of non-relational data stores that break with a long history of relational databases and ACID guarantees[18]. In addition, there are two products that are somehow interesting due their adaptability to many different stores:

EclipseLink[19] provides an extensible framework that allows Java developers to interact with various data services, including relational databases, web services, Object XML mapping (OXM), and Enterprise Information System (EIS). EclipseLink supports a number of persistence standards including Java Persistence API (JPA), Java API for XML Binding (JAXB), Java Connector Architecture (JCA) and Service Data Objects (SDO).

LINQ is a general-purpose query language proposed by Microsoft for adding native query capabilities to their .NET framework (Meijer et al., 2006). LINQ applies to all sources of information, not just relational or XML data, by implementing new data providers, even for RDF, such as the fledgling effort on LinqToRdf[20].

3 BRINGING TRIOO TO THE ARENA

The Trioo project aims to develop a technology enabling to easily use RDF data directly in some object-oriented programming languages, not only to persist that data in RDF stores, but also to consume data available on the Web and to expose the business model of the application as *Linked Data*. It is true that some languages start to offer new generic mechanisms closer to the object-orientation that to the query

[17]http://nosql-database.org/
[18]Atomicity, Consistency, Isolation, and Durability
[19]http://www.eclipse.org/eclipselink
[20]http://linqtordf.googlecode.com/

technology, such as the above described Microsoft's LINQ; but they still fail to offer more natural ways to access the data, from an object-oriented perspective. This is not a new challenge in computer science, because the software industry already offers some solutions since some years ago, for instance Hibernate for relational databases. But going deeply into the problem, it is clear that there is not any satisfactory solution to work with RDF, because they all strongly rely on design patterns, such as Active Record, too close to the entity-relational model. And the RDF model has many special features and semantics that should be treated as such. Therefore providing a technology capable of managing RDF data in a object-oriented way is a appealing scientific challenge that, from our point of view, may potentially have a great impact on the usage of the Semantic Web to provide innovative software solutions to real world problems.

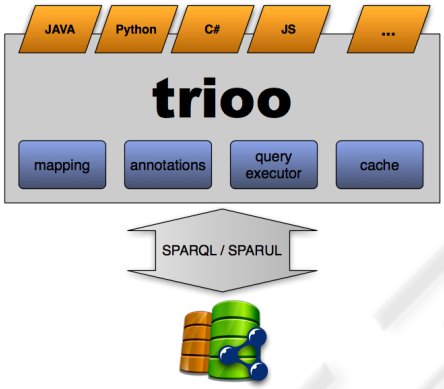

Figure 1: Trioo overview.

Ideally with Trioo developers will be not need to deepen in detail on the RDF data model, abstracting on how the query language internally works. However Trioo would not only take care about data mapping, but also about the typing inference where possible. And probably a cache mechanism would be also required, in any case such a kind of implementation details are not so relevant for the scope of this paper.

3.1 Computational Models

Both computational models (RDF and OO) share many concepts. But at the same time they present many substantial differences on their semantics.

In the object-orientation there is no a unique computational (Logozzo, 2004), there are more than one with several implementations. Usually object-oriented languages are divided into two big families (Evins, 1994): class-based and prototype-based programming languages.

Class-based programming languages use a computational model where objects are grouped on sets with equal structure and behaviour as instances of the same class (Booch, 1993). Thus there is a heavy link between an object and its class on instantiation, and it is not possible to create and use an object without previously define its static structure and behaviour on a class.

Prototype-based programming languages aim to be more faithful to the object computational model, where it is only necessary to have objects, no additional abstractions such as classes are required (Blaschek, 1994; Noble et al., 1999; Borning, 1986). The common behaviour is defined by what is known as "*trait objects*" (Ungar et al., 1991). The shared structure is represented as *prototype objects*, which inherit their behaviour from the *trait objects*. Therefore the state is always defined always as objects. The creation of objects (actually *instance objects*) is done using cloning those prototype objects.

So both approaches of object-orientation must be taken into account in Trioo.

RDF semantics is further complicated. It is an assertional language intended to be used to express propositions using precise formal vocabularies (Hayes and McBride, 2004). The semantics of the language is specified using model-theory, a technique which assumes that the language refers to a "world", and describes the minimal conditions that a "world" must satisfy in order to be an interpretation which makes a expression in the language true. The utility of a formal semantic theory is to provide a mechanism to determine valid inference processes, i.e. when the truth is preserved. Through the notions of satisfaction, entailment and validity, RDF semantics gives a rigorous definition of the meaning of a RDF graph. Actually, the RDF semantics specification defines four normative notions of entailment for RDF graphs: Simple, RDF, RDFS, and Datatype entailment. Simple-entailment captures the semantics of RDF graphs when no attention is paid to the particular meaning of any of the names in the graph (this is precisely the semantics supported by the current version of SPARQL). To obtain a complete meaning of an RDF graph written using a particular vocabulary, it is necessary to add further semantic conditions to interpret particular resources and typed literals in the graph. This way, RDF-entailment provides support for basic entity interpretation using the RDF vocabulary (typing, literals, basic structures,

etc.). Furthermore, D-entailment imposes additional conditions on the treatment of datatypes (for instance, for the use of externally defined datatypes identified by a particular URI reference). And finally RDFS Vocabulary extends RDF to include a larger vocabulary with more complex semantic constraints, such as classes, subclasses, domains, ranges and so on (Brickley et al., 2004). RDF graphs (serialised using RDF/XML syntax) are also the normative representation of OWL ontologies for exchange purposes. However, Trioo is not intended to support OWL semantics for now. A full compatibility with OWL would introduce a lot of complexity, as it will require interpreting OWL vocabulary and axioms, and providing support for DL reasoning.

Hence there are many details on these computational models (object-oriented and RDF) that would need to be analysed in detail in order to archive a real integration:

3.1.1 Typing

Typing is an intriguing challenge, since both approaches implementing the object-oriented computational model do it differently. For class-based languages the typing is build upon a hierarchy model of types based on classes (inheritance is defined at that class level, building a hierarchy system of classes). Nevertheless prototype-based languages provide a faithfully implementation of the object-oriented model: the inheritance relationship between objects is actually a derivation relationship, consequently "*instance of*" relationship is no necessary any longer; ideally this derivation could be from more than one object, although commonly this feature is restricted to a single reference in many of the implementations of the prototype-based computational model, such as Python. Typing on RDF works like the pure prototype-based model: each RDF object could have several types, with no restrictions. Figure 2 illustrates these differences on typing. Other concepts, such as inheritance and consistency, only appears with RDFS and OWL. Following section describes how type inference works.

3.1.2 Type Inference

Type inference is a feature to automatically deduce the type of a value in a programming language, both at compilation-time and run-time depending on the concrete language. It is a common feature in dynamically typed programming languages, but also in some strongly statically ones, such as C#. On programming

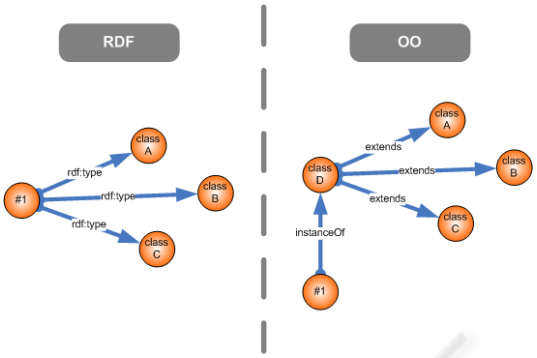

Figure 2: Differences on typing.

that inference affects to the type of variables (or attributes in the case of object-oriented programming). By contrast, in RDF relationships are declared using properties. And those properties are global, i.e. the same can be used to link different objects. In RDFS (and OWL) each property can declare both a domain and a range: the type that respectively the subject and the value may have. Therefore given a property P with a domain restriction D and a range restriction R, with a triple (a,P,b) it would be inferred that a is of type D and b of type R. Needless to say that support this behaviour would be difficult on statically typed programming language, but it would be achievable in dynamic languages.

3.2 Expressiveness

As commented above, OWL would be not supported by Trioo. However, Horst proved that entailment for RDFS is decidable, NP-complete, and in P if the target graph does not contain blank nodes. Therefore probably it would more realistic just to use the RDFS semantics (ter Horst, 2005). Moreover, this level of expressiveness could be achieved without perform a full reasoning process, because it could be vastly implemented directly inside the object model (in fact, there is already an implementation in Python of this concept[21]).

3.3 Integrity Constraints

Integrity constraints are used to ensure accuracy and consistency of data. Those constraints are directly ensured by the engines for other families of stores (for instance, by the entity-relationship model). However, detecting constraint violations is out of the scope of RDF Schema and OWL due to the Open World Assumption adoption. Under OWA, a statement cannot

[21] http://www.ivan-herman.net/Misc/PythonStuff/ RDFS-Closure/

be inferred to be false on the basis of a failure to prove, i.e. if something is not said, it is not possible to know whether it is true or false. As a consequence, everything may be true unless proven otherwise. The OWA assumes incomplete information by default and it is really useful for knowledge reusability in contexts such as Linked Data, limiting the the kinds of deductions just to those that follow from statements that are known to be true. However, with respect to Integrity Constraints, OWL and RDF Schema consistency checking only detects logical inconsistencies in the data, but missing information does not cause an inconsistency. Consider the following RDF graph:

```
foaf:name  rdfs:domain  foaf:Person .
:sergio    foaf:name    "Sergio" .
```

Under OWA, the domain restriction implies that `:Sergio` is a `foaf:Person`, but there is no violation of the domain of the property. Missing values of data do not cause inconsistencies, rather new information of data is inferred from the application of the RDF Schema entailment rules. These inferences may be counterintuitive for users who need data validation, like in some real scenarios and applications. As other frameworks for managing object-oriented domain models with data repositories, Trioo is envisioned to provide support for integrity and validation over RDF data. In order to check these inconsistencies, Trioo must explore how to introduce the Closed World Assumption alternatively to OWA. CWA interpretation means that an assertion is considered false if it is not explicitly known whether it is true or false. This semantics is usually supported by classical logical programming systems implementing Negation as Failure (NAF). There are several ways to come up with Integrity Constraints from the OWL and RDF Schema axioms of an ontology (Sirin and Tao, 2009; Motik and Rosati, 2007). One of the most interesting approaches is the translation of OWL axioms to SPARQL queries (following the entailment regime defined by Sirin et al. (Sirin and Parsia, 2007)). SPARQL is equivalent to non-recursive datalog and NAF may be encoded using the well-known OPTIONAL/FILTER/!BOUND pattern. The Pellet Integrity Constraint Validator[22] implements this approach, automatically translating OWL axioms into ASK SPARQL queries, so an OWL ontology can be used to validate RDF data integrity. The previous domain restriction of the `foaf:name` property may be written as the following integrity constraint:

```
ASK WHERE {
  ?person  foaf:name  ?name .
  OPTIONAL { ?person2  rdf:type foaf:Person }
  FILTER ( bound(?person2) )
  FILTER ( ?person = ?person2 )
}
```

[22] http://clarkparsia.com/pellet/icv

3.4 Target Languages

The success of the project strongly depends largely on the achieved impact. And the impact is directly linked to the programming language chosen, both for its popularity and its technical features. With respect to the popularity of programming languages the TIOBE Programming Community index gives an indication of it[23]. On the top of this ranking is the Java programming language (Gosling et al., 2005) since 2001. Java is general-purpose, concurrent, and class-based object-oriented programming language. It has a simpler object model and fewer low-level facilities, such as direct access to memory. Java has single inheritance, but conceptually that is not entirely true since the inclusion of Interfaces (reference types without implementation). Java applications are compiled to an intermediate language (known as"bytecode") that runs on a virtual machine. The `java.lang.reflect` package provides introspection capabilities that allows to examine Java applications at run-time; for providing some computational reflection capabilities the package simulates them with an implementation of the Proxy pattern (Gamma et al., 1994), known as "Dynamic Proxy Classes". Additionally to all these features of the language, Java would a good candidate for implementing Trioo because all the good Semantic Web tools implemented in Java: Jena, OWL API, Pellet, etc.

However dynamic programming languages are closer to RDF, because actually they implement a prototype-based model. Probably the most pure and complete of these languages could be Self (Ungar and Smith, 1987). Self is a prototype-based dynamic object-oriented programming language, environment, and virtual machine centred around the principles of simplicity, uniformity, concreteness, and liveness. Unfortunately it is less usable than other modern languages. Due to this fact, and also taking into account popularity criteria, Python would be one of the most interesting choices for provide the second reference implementation of Trioo. Python is a prototype-based general-purpose and dynamic typed programming language (van Rossum, 1989), designed with the philosophy of emphasise the code readability. It offers a modern and versatile computational model, supporting several paradigms such as object-oriented and structured programming.

[23] Available online at http://www.tiobe.com/index.php/tiobe_index, retrieved by February of 2010.

Python also includes some important features, such as introspection, structural reflexion (the union of the latter two is called metaprogramming), multiple inheritance and generative programming, converting its computational model in one of the most usable and versatile between modern programming languages.

But, apart from the commitment of initially apply Trioo on these two widely used object-oriented programming languages, Trioo aims to offer much more, covering other scenarios that would be very interesting to complete this research work. For instance, the powerful dynamic features available on JavaScript offer an interesting scenario where more conclusions could be extracted. For instance, given the trend to use AJAX (Asynchronous JavaScript and XML) for improving the user experience on Web interfaces, together with the W3C's effort to enrich the markup with technologies such as RDFa (Adida et al., 2008), it could convert Trioo in a suitable technology to provide support for developing semantically-enriched user interfaces on (X)HTML due the interesting dynamic features of the JavaScript language. Additionally, the support for dynamic typing included in C# 4.0 (Hejlsberg, 2008) open an interesting area of applicability in the Microsoft's .NET framework. These, and others, lines open amazing opportunities on unexplored research fields in many of the potential target languages where Trioo could be applied and/or extended.

3.5 Mechanism

How Trioo would be applied on all target languages is a quite important question that may not be answered yet for all of them. Although probably such mechanism would need to be supplemented with a external configuration file (XML, YAML or whatever suitable format). A priori there are some requirements that should be taken into account: simplicity (it should follow the DRY philosophy, reducing as possible the repetition of information), non intrusive (the mechanism used should not modify the original design, e.g. inheritance would not be a suitable construction when the language only supports single inheritance), legible (it should not dirty the readability of source code), accessible at run-time (due to the intrinsically dynamic features of the project, the mechanism chosen should offer that meta-information at run-time), customisable (such mechanism should support to be parametrise, in order to allow implementations to be customisable when would be needed). All these requirements could be covered with annotations, that are supported by the two first chosen target languages. Annotations were introduced in the version 5.0 of the Java programming language (Bracha, 2004). They are a special form of syntactic metadata that can be added to Java source code and can be reflectively retrieved at run-time. On the contrary, annotations in Python are syntactic sugar that actually provides a generic implementation of the decorator pattern (Gamma et al., 1994), enhancing the action of the function or method they decorate. In Python prior to version 3, decorators only apply to functions and methods; however class decorators are supported from Python 3.0 (Winter, 2010).

3.6 Persistence

It is obvious that, in this concrete moment when W3C is running a working group[24] to extend SPARQL technology, including some of the features that the community has identified as both desirable and important for interoperability based on experience with the initial version of the standard (Kjernsmo and Passant, 2009), formally known as SPARQL 1.1. For instance, the new properties path (Seaborne, 2010) converts SPARQL in a more powerful query language. Henceforth all these new features should be exploited by new tools.

A SPARQL query contains a set of triple patterns, where triple patterns are like RDF triples except that each term may be a variable. A basic graph pattern matches a subgraph of the RDF data when RDF terms from that subgraph may be substituted for the variables and the result is RDF graph equivalent to the subgraph. The expressiveness of SPARQL is powerful: it is a query language equivalent in expressive power to Relational Algebra (Angles and Gutierrez, 2008). Therefore, in a similar way that was previously done for relational databases (Cyganiak, 2005), it would be necessary to additionally describe a transformation from object-oriented programming languages into the algebra of SPARQL.

Consequently the innovative approach of the Trioo project would be to provide a purely standards-based implementation for store RDF data. So that means only using SPARQL 1.1, not implement proprietary interfaces, allowing developers to be independent from a concrete RDF store. Obviously, this purist approach would be supplemented with the additional possibility of using specific dialects of SPARQL[25], in a similar way as Hibernate does, in order to maximise the performance where would be possible, but at the same time without favouring the

[24] http://www.w3.org/2009/05/sparql-phase-II-charter

[25] SPARQL proprietary extensions are widely implemented by many vendors.

vendor lock-in allowing developers to switch to other RDF store just changing a configuration line.

4 IMPLEMENTATIONS

Currently[26] the Trioo project is immersed in developing the two reference implementations in the two programming languages selected above (see Section 3.4). The analysis carried out in this paper establishes a solid foundation that greatly facilitated the design and implementation on both programming languages. An alpha prototype has been developed in Java with a functional version of some of the core features. The philosophy followed for the development of the Python version is far more ambitious, since it aims to provide RDF support in a more *natural* way, trying to exploit the full power of the prototype-based computational model. Therefore it is still too early to assess the results. In a nutshell, unfortunately both are far from being full functional implementations according the criteria described in this paper, and they still would require more effort on development. Of course, needless to say that when they would be ready for a real usage, all these implementations will be released as open source.

5 CONCLUSIONS AND FUTURE WORK

This paper has described the ongoing work carried out by the Trioo project to leverage the power of the RDF data model into object-oriented programming languages, trying to keep the semantics of data safe and sound into the applications. But the final aim is not only to persist data on RDF stores, but also to allow consuming and publishing RDF data linked on the Web.

The analysis accomplished on Section 3 shows that it is possible to integrate RDF data model into object-oriented computational models with a good enough level of commitment between both semantics. The prototype-based model fits best with RDF than the class-based, but that should not be an impediment to also implement Trioo on class-based programming languages. And since the approach followed to perform this analysis has been completely language independent, Trioo could be potentially applied to any object-oriented programming language.

Trioo gives rise to the necessity to have software tools for accessing RDF data from software applications. The learning curve of some of the current RDF tools is not always accessible to all developers, while they require advanced knowledge of both the data model and query language. And, as with other technologies, developers should not need to be aware of all the details about how their applications store the data. Therefore Trioo should abstract developers of such details. For the moment the results of the two current implementations are still promising. As these implementations progress to a more mature level of development, many of the current open questions will have a more clear answer. The lessons learned from these implementations will serve as feedback for the current groups working on the related standards involved in Trioo, in such a way that all these standard technologies can really serve to fulfil the objectives set forth in the project initially.

[26] At March 5th, 2010

REFERENCES

Adida, B., Birbeck, M., McCarron, S., and Pemberton, S. (2008). RDFa in XHTML: Syntax and Processing. Recommendation, W3C.

Angles, R. and Gutierrez, C. (2008). The Expressive Power of SPARQL. In *Proceedings of the 7th International Semantic Web Conference (ISWC2008)*.

Bauer, C. and King, G. (2004). *Hibernate in Action*. Manning Publications.

Berners-Lee, T. (2006). Linked Data: design issues.

Berners-Lee, T., Fielding, R., and Masinter, L. (2005). RFC3886: Uniform Resource Identifier (URI): Generic Syntax. Request For Comments, The Internet Society.

Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The Semantic Web: a new form of Web content that is meaningful to computers will unleash a revolution of new possibilities. *Scientific American*.

Blaschek, G. (1994). *Object-Oriented Programming with Prototypes*. Springer-Verlag New York, Inc.

Booch, G. (1993). *Object-Oriented Analysis and Design with Applications*. Addison-Wesley.

Borning, A. H. (1986). Classes Versus Prototypes in Object-Oriented Languages. In *Proceedings of the ACM/IEEE Fall Joint Computer Conference*, pages 30–40.

Bracha, G. (2004). JSR 175: A Metadata Facility for the Java Programming Language. Java Specification Request, Sun Microsystems.

Brickley, D., Guha, R., and McBride, B. (2004). RDF Vocabulary Description Language 1.0: RDF Schema. Recommendation, W3C.

Broekstra, J. and Kampman, A. (2003). SeRQL: a second generation RDF query language. In *Proceedings of SWAD-Europe Workshop on Semantic Web Storage and Retrieval*.

Cyganiak, R. (2005). A relational algebra for SPARQL.

DeMichiel, L. (2009). JSR 317: Java Persistence API, Version 2.0. Java Specification Request, Sun Microsystems.

DeMichiel, L. and Keith, M. (2006). JSR 220: Enterprise JavaBeans 3.0. Java Specification Request, Sun Microsystems.

Evins, M. (1994). Objects Without Classes. *Computer IEEE*, Volume 27:104–109.

Fensel, D., Decker, S., Erdmann, M., and Studer, R. (1998). Ontobroker: The Very High Idea. In *Proceedings of the 11th International Florida Artificial Intelligence Research Symposium (FLAIRS98)*.

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. (1999). RFC2616: Hypertext Transfer Protocol HTTP/1.1. Technical report, The Internet Society.

Fowler, M. (2002). *Patterns of Enterprise Application Architecture*. Addison-Wesley.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional.

Gosling, J., Joy, B., Steele, G., and Bracha, G. (2005). *Java(TM) Language Specification, The (3rd Edition) (Java (Addison-Wesley))*. Addison-Wesley Professional.

Hayes, P. and McBride, B. (2004). RDF Semantics. Recommendation, W3C.

Hejlsberg, A. (2008). The Future of C#. In *the Microsoft's Professional Developers Conference (PDC)*.

Kjernsmo, K. and Passant, A. (2009). SPARQL New Features and Rationale. Working Draft, W3C.

Klyne, G. and Carroll, J. J. (2004). Resource Description Framework (RDF): Concepts and Abstract Syntax. Recommendation, W3C.

Logozzo, F. (2004). Separate Compositional Analysis of Class-based Object-oriented Languages. In *Proceedings of the 10th International Conference Algebraic Methodology and Software Technology (AMAST 2004)*.

Meijer, E., Beckman, B., and Bierman, G. (2006). LINQ: reconciling object, relations and XML in the .NET framework. In *ACM SIGMOD International Conference on Management of Data*, Chicago, Illinois.

Motik, B. and Rosati, R. (2007). A Faithful Integration of Description Logics with Logic Programming. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI 2007)*, pages 477–482.

Noble, J., Taivalsaari, A., and Moore, I. (1999). *Prototype-Based Programming: Concepts, Languages and Applications*. Springer Publishing Company.

Oren, E., Heitmann, B., and Decker, S. (2008). ActiveRDF: embedding Semantic Web data into object-oriented languages. *Journal of Web Semantics*, 6:191–202.

Prud'hommeaux, E. and Seaborne, A. (2008). SPARQL Query Language for RDF. Recommendation, W3C.

Sauermann, L. and Cyganiak, R. (2008). Cool URIs for the Semantic Web. Interest Group Note, W3C.

Schenk, S. and Gearon, P. (2010). SPARQL 1.1 Update. Working Draft, W3C.

Schneider, M., Carroll, J., Herman, I., and Patel-Schneider, P. F. (2009). OWL 2 Web Ontology Language, RDF-Based Semantics. Recommendation, W3C.

Seaborne, A. (2010). SPARQL 1.1 Property Paths. Working Draft, W3C.

Shinavier, J. (2007). Ripple: Functional Programs as Linked Data. In *Proceedings of the 3rd international workshop on Scripting for the Semantic Web (SFSW2007), co-located with the 4th European Semantic Web Conference (ESWC2007)*.

Sintek, M. and Decker, S. (2002). TRIPLE—A Query, Inference and Transformation Language for the Semantic Web. In *Proceedings of the 1st International Conference on Semantic Web*, volume Volume 2342/2002, pages 364–378, Sardinia, Italy. Springer Berlin / Heidelberg.

Sirin, E. and Parsia, B. (2007). SPARQL-DL: SPARQL Query for OWL-DL. In *Proceeddings of the 3rd OWL Experiences and Directions Workshop (OWLED 2007)*.

Sirin, E. and Tao, J. (2009). Towards Integrity Constraints in OWL. In *Proceedings of the 6th International Workshop on OWL: Experiences and Directions (OWLED 2009)*.

ter Horst, H. J. (2005). Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary. *Journal of Web Semantics*, Vol. 3:pp. 79–115.

Ungar, D., Chambers, C., Chang, B.-W., and Hlzle, U. (1991). Organizing programs without classes. *LISP and Symbolic Computation*, Volume 4:223–242.

Ungar, D. and Smith, R. B. (1987). SELF: The Power of Simplicity. In *Proceedings of The International Conference on Object Oriented Programming, Systems, Languages and Applications (OOPSLA'87)*, pages 227–241.

van Rossum, G. (1989). The Python programming language. http://www.python.org/.

Winter, C. (2010). PEP 3129 – Class Decorators. Python Enhancement Proposal, Python Software Foundation.