A programming model for the Semantic Web

Marco Crasso, Cristian Mateos, Alejandro Zunino, and Marcelo Campo

ISISTAN Research Institute. UNICEN University. Campus Universitario, Tandil, Buenos Aires, Argentina. Tel.: +54 (2293) 439682 ext. 35. Fax.: +54 (2293) 439683
Also Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET)

Abstract. This year the concept of "Semantic Web" celebrates its tenth anniversary since it was coined by prominent researchers Tim Berners-Lee, James Hendler and Ora Lassila. To date, there are many technologies to describe, in a *machine-understandable* way, information and services available in the Web. An incipient research area is focused on solving the issues related to building applications for truly exploiting semantically-described resources. In this paper, we present an approach to upgrade a software application into a Semantic Web-enabled one. The approach builds on the Aspect-Oriented Programming (AOP) paradigm to allow developers to incorporate ontology management capabilities into an ordinary object-oriented application without modifying its source code. The paper also discusses related works and presents case studies.

1 Introduction

From its beginnings, the Web was conceived as a system of computer networks sharing linked HyperText documents. Later, applications become Web resources as well upon the arrival of Web Services technologies. Nowadays, the Semantic Web is an extension of the traditional Web, in which resources and services are described in such a way that a machine can understand it [1]. The goal of this extension is that applications become "users" of the Web. In this sense, it is expected that once every Internet resource will be properly, and semantically, described, applications will be able to compose different resources to achieve their goals, without human intervention. Accordingly, software applications needing external information or services to solve everyday problems, such as meeting or flight arrangement, supply-chain management, and so on, will reach unprecedented levels of automatism [2].

For the vision of the Semantic Web to become a reality, standards and ontologies should converge, and in turn applications should be turned from systems that assume a closed world to ones that will operate by composing Web resources. On one hand, standards play a pivotal role in developing and connecting heterogeneous functionality, specially when different providers offer and distribute their services over the Internet. In this sense, Web Services standards, such as Web Service Description Language (WSDL) and Simple Object Access Protocol (SOAP) [15], represent a big push in the right direction. WSDL is an XML format for describing the intended functionality of a Web Service by means of an interface with methods and arguments, in object-oriented terminology, and documentation in the form of textual comments. For example, a provider may describe the interface of a service operation for retrieving the temperature of a certain region as getTemperature(zip:string):double.

In the context of computer and information sciences, an ontology defines a set of representational primitives by which a domain of knowledge or discourse is modeled [5]. Ontologies are crucial for bridging and abstracting away syntactical differences, e.g. those present at WSDL documents, which may hinder the utilization of the corresponding services. To clarify the importance of ontologies, let us suppose that another provider uses getFahrenheitTempFor(regionCode:long):string to describe the signature of a service operation similar to getTemperature(zip:string):double. Clearly, these operations are syntactically different. Therefore, if either a human discoverer or a software agent looks for all services that retrieve the temperature of a certain region, they will have to infer that the aforementioned operation signatures have equivalent *semantics*. Indeed, the W3C (`www.w3.org`) encourages developers to describe ontologies by using the Ontology Web Language (OWL), an XML-based language having constructors for defining classes and properties, with cardinality, range and domain restrictions among others.

Ontologies allow providers to define each part of a service, i.e. its functionality, inputs and expected results, with an unambiguous description of its semantics. From now on, we will refer as annotation to the task of linking service descriptions with concepts represented in a machine-understandable model that explicitly defines the semantics of operations and data-types. There are different approaches to combine ontologies and standards for describing services and, in turn, build Semantic Web Services, such as OWL-S [8] and WSDL-S [12]. For the sake of exemplification, although their many differences, with these approaches developers can annotate the aforementioned services input with the ZipCode concept from Schema Web ontology[1], by extending the associated WSDL input definitions with the unique and public URI of ZipCode (`http://www.daml.org/2001/10/html/zipcode-ont#ZipCode`).

Annotating a service may be a cumbersome task, specially when it requires building ontologies from scratch and training development team members on OWL and WSDL-S and their surrounding technologies. To cope with this problem, researchers have been investigating on approaches to facilitate ontology construction along with service annotation, e.g. by automatically suggesting ontologies for a given service description in WSDL [3]. In this paper we explore the annotation problem, but from the perspective of Semantic Web applications, i.e. those applications that consume external services to accomplish their goals. In essence, the approach for turning an ordinary application into an application that automatically discovers, selects and invokes Semantic Web Services, is to annotate its business object definitions with ontologies. This is because such an application not only requires to semantically express its functional needs, but also to "talk" with Semantic Web Services through ontologies. For example, to automatically call a service expecting a ZipCode concept as input, an application must upgrade its internal representation of ZIP code, possibly a variable of type long, into an OWL class instance.

Until now, unlike the case of service annotation, there is not a consensus about how to annotate applications. In some clean cases, the approach to decorate business objects classes with special proxy objects has shown to be useful for maintaining annotations in accordance with dynamic objects states [14]. One limitation of this approach is that

[1] `http://www.schemaweb.info/schema/SchemaInfo.aspx?id=20`

it is feasible only when business objects are modeled and implemented in accordance with specific design conventions. Besides, to incorporate such special proxies to an application, its source code must be modified.

Undoubtedly, it is not always possible to refactor existing applications for accommodating business object designs and implementations or adding proxies. Therefore, this paper presents an approach that copes with the aforementioned limitations. The approach bases on the Aspect-Oriented Programming paradigm to treat the requirement of semantic annotations as a cross-cutting concern. In computer science, a cross-cutting concern is a requirement that traverses different parts of an application, and it is complex to decouple from other application components. Then, with this approach, management of semantic annotations is encapsulated in modules, called *semantic aspects*, which can be plugged in any specific point of an application, without modifying its source code. Therefore, the opportunity for transparently turning an ordinary application into a Semantic Web-powered one, comes at expenses of learning and implementing the semantic aspects.

The rest of this paper is organized as follows. Section 2 presents an overview of related work. Section 3 explains the proposed approach to annotate applications. Case studies and comparisons are shown in Section 3.1. Finally, Section 4 concludes the paper.

2 Related work

The problem of ontology-application integration has been mainly approached by translating ontologies to source code files in any object-oriented language. The basic idea is to create an application programming interface (API) that represents a given semantic model [6]. Furthermore, when a business object is programmatically modified, the API forwards the associated changes to a database of ontological annotations, a.k.a. semantic repository [14]. This approach bases on bridging semantically equivalent object constructors of Description Logic (DL) and Object Oriented (OO) systems. For example, an instance of a Java class represents an instance of a single OWL class with most of its properties maintained.

RDFReactor [14] is an approach to transform a given ontology into an Object-Oriented Java API. The approach receives its name since it accepts as input an ontology in Resource Definition Framework (RDF), the baseline for W3C ontology languages like OWL. For each business object definition found in the input ontology, RDFReactor generates a special stateless proxy object that delegates all method calls to semantic repository queries and updates. A proxy represents business objects by exposing the methods that would expose a plain object designed for representing the same business object. The proxies are responsible for querying the semantic repository when an *accessor* method is called, or for updating the repository when the call refers to a method used to control changes to a variable, i.e. a *mutator* method. To clarify this, let suppose that books are modeled as instances of a class named Book, which has associated isbn instances as well. Then, a special proxy for such an instance would set a relationship between a concept Book and another concept ISBN when the association is programmatically set, for example by calling setISBN method.

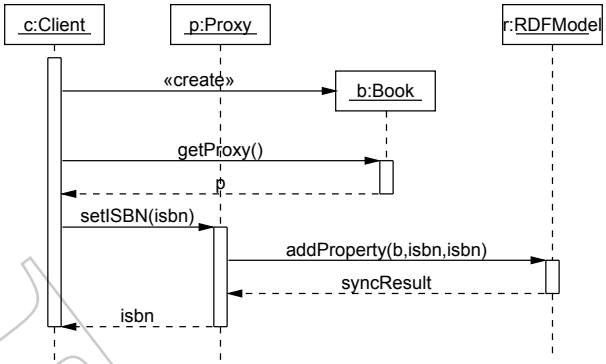

Fig. 1. RDFReactor's proxy example.

Figure 1 depicts the resulting classes and messages that have been produced after translating an ontology for the mentioned book domain. In the Figure, a specific business object, i.e. the b:Book, is created by a client. The approach replaces business object instantiation (lets suppose a new statement in Java) with proxy instantiation. Then, the client indirectly updates the semantic model when calling the mutator method named setISBN, a typical Java *setter* method.

Furthermore, there are a handful of tools implementing the same approach followed by RDFReactor, such as Jastor [13] and ActiveRDF [9]. A complete list of similar projects can be found in `http://semanticweb.org/wiki/Tripresso`. This Web site groups several researchers that discuss issues related to translation of ontology languages to Object-Oriented languages.

All in all, though with the approach described in the previous paragraphs developers might bootstrap any Object-Oriented application into a Semantic Web ready application, the approach has some limitations. Particularly, a major limitation is that the approach forces developers to introduce design and implementation-level changes into target applications, which is clearly an undesirable situation.

3 An AOP-based approach to annotate applications with semantics

Separation of Concerns (SoC) [11] is both a principle and a process for building software applications, which states that each constituent part of an application should be free of behaviors not inherent to its functional nature. Traditionally, SoC has been achieved by using modularity, encapsulation and information hiding techniques in the context of Object-Oriented programming, or layered designs in an architectural one. The AOP paradigm has shown to be quite effective in increasing application modularity and reducing inter-module coupling by allowing the separation of *cross-cutting* concerns, i.e. a concern that affects many parts of an application [4].

With AOP, the logic of cross-cutting concerns is encapsulated in modules called *aspects*, which can be joined to specific points of an application, called *join-points*.

Aspects inhabit a special plane, whereas application components constitute the base plane. *Weaving* is the act of activating the *aspects* of the aspects plane that are associated with join-points at the base plane. Most AOP materializations allow one of two types of weaving: dynamic and static. The difference between both types of weaving stems from the moment at which aspect code is linked to the code of its join-points. With static weaving, the code is woven at compilation time, whereas dynamic weaving allows aspects to be incorporated into running applications.

Having explained the main concepts of the AOP paradigm, we propose an approach that handles semantic annotation as a cross-cutting concern. The rationale behind this assumption is that it is out of the scope of most business object representations to manage their associated semantic model, and clearly business objects represent most application parts, at least in traditional CRUD applications.

Our approach proposes to encapsulate the code needed to interact with a semantic repository into aspects. Furthermore, every business object mutator method is conceived as a join-point. Static or dynamic weaving, indistinctly, produces business object representations capable of managing their semantic annotations without having to modify their source code. In contrast to proxy-based approaches such as [14,6], our approach is not invasive for applications, since it maintains business objects encapsulation. This is because business object classes remain untouchable, even having instance variables for holding objects states. Aggregated code blocks, i.e. those sentences needed for updating a semantic repository, are concealed by the aspects.

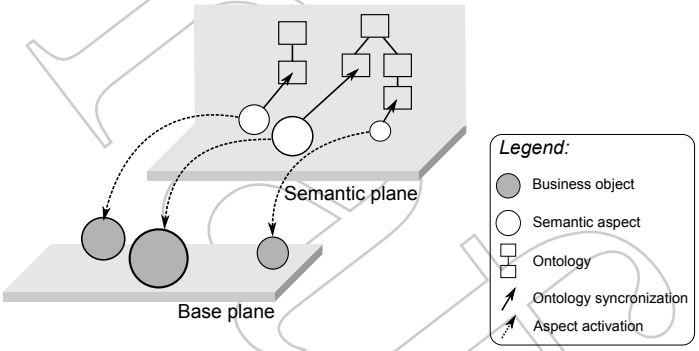

Fig. 2. Base and Semantic AOP planes.

Figure 2 depicts the conceptual overview of the approach. In the Figure, business objects constitute the base plane of an application, which are not responsible for nothing apart from modeling the application business logic. Alternatively, the modules of the semantic plane, or semantic aspects, know how to keep business objects and ontologies updated.

To systematically annotate applications business objects with the proposed approach, the next steps should be followed:

1. identify business objects,

2. map the identified business objects onto ontological elements,
3. build semantic aspects for programmatically updating ontologies,
4. identify join-points,
5. weave aspects into join-points.

The first step refers to analyze the applications under study looking for the minimal set of business objects that need to be annotated, possibly to interact with external Web Services. Step 2 is intended for identifying to which classes or properties of an ontology should the business objects be mapped. At the next step, the implementation of semantic repository updates should be provided. This step requires to select technologies for materializing the AOP paradigm and the semantic repository, or in other words choosing among different AOP frameworks such as AspectJ or SpringAOP [7], and ontology containers, such as Jena [10]. Once proper technologies have been selected, the specific lines of codes for updating the ontology elements identified at step 2 should be encapsulated into aspects, which is done at step 3. Moreover, step 4 deals with analyzing the whole application and seeking in which parts of it the business objects are modified. Finally, the fifth step is for gluing aspects and the application.

The next sub-sections explain how to annotate particular business objects that belong to two real world applications that we are porting to the Semantic Web as an initial attempt to assess the feasibility of the proposed approach. The application domains and the employed AOP technologies are described.

3.1 Case study I

In this section the five steps previously described will be applied for annotating an application belonging to the library domain. The application has been implemented in Java. Broadly, the application manages books and their authors. Books and authors are the business objects required to be annotated, which are represented as JavaBeans classes. JavaBeans is a programming convention that states that for each instance variable there is an accessor method named using the prefix "get", and a mutator method having the "set" prefix in its signature name. For example, the Book class has an instance variable called author of type Author, and it has two methods accordingly: getAuthor() and setAuthor(Author), the accessor and mutator, respectively. The Book class has another instance variable, a String named title, and two more methods getTitle() and setTitle(String).

At the same time, books and authors are semantically represented using the next part of the library ontology:

```
<owl:Class rdf:ID="Book"/>
<owl:Class rdf:ID="Person"/>
<owl:ObjectProperty rdf:ID="isAuthorOf">
  <owl:inverseOf>
    <owl:ObjectProperty rdf:ID="hasAuthor"/>
  </owl:inverseOf>
  <rdfs:domain rdf:resource="#Person"/>
  <rdfs:range rdf:resource="#Book"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasAuthor">
  <rdfs:domain rdf:resource="#Book"/>
  <owl:inverseOf rdf:resource="#isAuthorOf"/>
  <rdfs:range rdf:resource="#Person"/>
</owl:ObjectProperty>
```

The association between books and authors for the business objects is semantically represented via the isAuthorOf property, which is defined as inverseOf the hasAuthor property, meaning that by annotating x isAuthorOf y the relationship y hasAuthor x holds.

Returning to the steps needed to employ the proposed approach, at this point business objects have been identified (step 1), and these objects have been mapped onto concrete ontology elements (step 2). With regard to step 3, the fact that the JavaBeans convention was followed allows for rapid identification of join-points, since aspects in charge of updating the semantic repository should be activated every time a *setter* method is called. To implement the aspects (step 4) for this application we have employed the well-known AspectJ support, and Jena for ontology annotations persistence. AspectJ has been designed as an extension of the Java language. One of the design drivers of AspectJ was that it should allow Java developers to intuitively incorporate aspects to their Java applications. On the other hand, Jena is a software library for manipulating ontology repositories programmatically. The next code illustrates the resulting aspect. As the reader can observe, we have used an aspects extension mechanism similar to regular class inheritance, and thus the generated aspect inherited from another aspect called SemanticAspect. This is because the boilerplate code needed for Jena initialization tasks has been abstracted for clarity reasons.

```
public aspect BookSemanticAspect extends SemanticAspect {

  pointcut setAuthor(Book b):
    target(b) && call (public * setAuthor(..));

  before(Book b): setAuthor(b) {
    if (model.getIndividual(ns + b.getISBN()) == null) {
      model.createIndividual(ns + a.getISBN(), this.baseClass);
    }
  }

  after(Book b): setAuthor(b) {
    Individual ai = model.getIndividual(ns+b.getAuthor().getSN());
    Individual bi = model.getIndividual(ns+b.getISBN());
    bi.setPropertyValue(model.getProperty(ns+"hasAuthor"),ai);
  }
}
```

Table 1. Aspect composition summary.

Join-Point	AspectJ
class: Book method: setAuthor	pointcut setAuthor(Book b): target(b) && call (public * setAuthor(..))
class: Book method: setTitle	pointcut setTitle(Book b): target(b) && call (public * setTitle(..))

To sum up, two instance variables, namely author and title, of the book business object have been mapped onto a semantic model in OWL. Then, proper synchronization mechanisms to mirror object changes onto a semantic repository have been implemented using AspectJ and Jena. Table 1 summarizes the application join-points and their corresponding aspects.

3.2 Case study II

We employed the proposed approach with another case study, namely a warehouse management Java application that uses a relational database through JDBC to persist warehouse article data. One peculiarity of the application's design is that business objects are represented by a single Warehouse object, which updates and queries the database. From a semantic perspective, we have employed an ontology having a class named Article, which has a property named hasQuantity. The range of this property is an integer, and its domain is an instance of the OWL class named Article. Therefore, when an article's quantity is modified in the Warehouse object (i.e. within the Java code), the associated hasQuantity property in the corresponding article is updated in the semantic repository.

To implement the semantic aspects we have employed the SpringAOP framework. This framework allows developers to implement aspects as any other ordinary Java method, but according to specific signatures defined by certain hook methods. Concretely, we have implemented the required hook method afterReturning as follows:

```
public void afterReturning(Object o, Method m, Object[] args,
                           Object target) throws Throwable {
  String code = String.valueOf(args[0]);
  if (model.getIndividual(ns  + code) == null) {
    model.createIndividual(ns+code,baseClass);
  }
  long quantity = Long.parseLong( String.valueOf(args[1]) );
  Individual i = model.getIndividual(ns + code);
  Property p = model.getDatatypeProperty(ns + "hasQuantity");
  i.setPropertyValue(p, model.createLiteral(quantity) );
}
```

As the reader can see, this application design is not as clean as that of the case study presented previously, since business objects and mutators were not evident. Here, business objects have been represented using the primary key of a relational table, i.e. in a conventional, database-like style. Moreover, one class has mutators, but no code convention was followed to name them. The next code presents for example the incArticle mutator method:

```
class Warehouse{
    public void incArticle(String code,long quantity) throws Exception {
        conn.setAutoCommit(false);
        PreparedStatement stmt = conn.prepareStatement(
        "UPDATE ARTICLE SET QUANTITY=? WHERE CODE=?"
        );
        stmt.setLong(1,quantity);
        stmt.setString(2,code);
        stmt.executeUpdate();
        conn.commit();
    }
}
```

Then, the semantic aspect should be activated every time the incArticle method successfully returnes. Having identified business objects, mapping them onto ontology elements, implementing the semantic aspects, and detecting join-points, the missing step was to configure the SpringAOP framework for weaving the aspect and the join-point at run-time. The following XML code belongs to the SpringAOP configuration file:

```
<!– After advise declaration –>
<bean id="myAfterAdvice" class="MySemanticAfterAdvise"/>
```

```
<!– Proxy with interceptors stack declaration –>
<bean id="businesslogicbean" class="#aop#.ProxyFactoryBean">
  <property name="proxyInterfaces">
    <value>Warehouse</value>
  </property>
  <property name="target">
    <ref local="beanTarget"/>
  </property>
  <property name="interceptorNames">
    <list>
        <value>myAfterAdvisor</value>
    </list>
  </property>
</bean>

<!– Join point declaration –>
<bean id="myAfterAdvisor" class="#aop#.RegexpMethodPointcutAdvisor">
  <property name="advice">
    <ref local="myAfterAdvice"/>
  </property>
  <property name="pattern">
    <value>Warehouse.incArticle</value>
  </property>
</bean>
```

4 Conclusions and future research possibilities

This paper presented a novel approach to incorporate ontology management into conventional applications. The most important difference among related approaches is that application code is not modified. In most cases, running applications and ontologies may be integrated by re-launching the former ones. The main limitation of the approach, however, is that application developers should be trained in AOP concepts and technologies.

In the near future, we will use performance metrics for evaluating different technologies for implementing the approach. The goal of this task is to have evidence about the overhead introduced by the semantic aspects over ordinary applications. Besides, we are employing the proposed approach with the reference Web application provided by Oracle to illustrate how developers can apply various Java Enterprise Edition technologies for implementing Web Services.

We are planning to extend the proposed approach in several directions. First, one line of research will investigate on facilitating semantic aspects generation. In parallel, we are designing heuristics for assisting developers in performing the second step of the proposed approach, which deals with mapping identified business objects onto ontological elements. In this sense, we will test the hypothesis that OO business objects specifications contain information –e.g. the names and comments of classes and methods– that is useful for guiding the ontology mapping process. Lastly, we will investigate the applicability of the extended approach in the context of SWAM [2], a Prolog-based programming language that allows the construction of mobile agents that are able to interact with RDF-described Web services and resources. Although SWAM is aimed at exploiting separation of concerns concepts, to a certain extent the language still forces users to mix agent code with the code in charge of interacting with semantic resources.

References

1. Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic Web. *Scientific American*, 284(5):34–43, May 2001.
2. Marco Crasso, Cristian Mateos, Alejandro Zunino, and Marcelo Campo. SWAM: A logic-based mobile agent programming language for the Semantic Web. *Expert Systems with Applications*, 38:1723–1737, March 2011.
3. Marco Crasso, Alejandro Zunino, and Marcelo Campo. Combining document classification and ontology alignment for semantically enriching Web Services. *New Generation Computing*, 28:371–403, 2010.
4. Juan Enriquez, Graciela Vidal, and Sandra Casas. Design configurable aspects to connecting business rules with Spring. In Francisco V. Cipolla Ficarra, Carlos de Castro Lozano, Mauricio Pérez Jiménez, Emma Nicol, Andreas Kratky, and Miguel Cipolla-Ficarra, editors, *Advances in New Technologies, Interactive Interfaces and Communicability - 1st International Conference (ADNTIIC 2010), Huerta Grande, Argentina*, volume 6616 of *Lecture Notes in Computer Science*, pages 92–101. Springer, 2011.
5. Tom Gruber. Ontology. In *Encyclopedia of Database Systems*, pages 304–307. Springer-Verlag New York, Inc., 2008.
6. Aditya Kalyanpur, Daniel Jiménez Pastor, Steve Battle, and Julian A. Padget. Automatic mapping of owl ontologies into java. In Frank Maurer and Günther Ruhe, editors, *SEKE*, pages 98–103, 2004.
7. Gary Mak, Josh Long, and Daniel Rubio. Spring AOP and AspectJ support. In *Spring Recipes*, pages 117–158. Apress, 2010.
8. David Martin, Mark Burstein, Drew Mcdermott, Sheila Mcilraith, Massimo Paolucci, Katia Sycara, Deborah L. Mcguinness, Evren Sirin, and Naveen Srinivasan. Bringing semantics to Web Services with owl-s. *World Wide Web*, 10(3):243–277, 2007.
9. Eyal Oren, Benjamin Heitmann, and Stefan Decker. ActiveRDF: Embedding Semantic Web data into object-oriented languages. *Web Semantics*, 6:191–202, September 2008.
10. Dave Reynolds. Jena 2 inference support. `http://jena.sourceforge.net` (last accessed June 2011), 2011.
11. Chris Richardson. Untangling enterprise Java. *Queue*, 4(5):36–44, 2006.
12. Kaarthik Sivashanmugam, Kunal Verma, Amit P. Sheth, and John A. Miller. Adding semantics to Web Services standards. In *The 2003 International Conference on Web Services*, pages 395–401, Las Vegas, NV, USA, September 2003. CSREA Press.
13. Ben Szekely and Joe Betz. Jastor: Typesafe, ontology driven RDF access from Java. `http://jastor.sourceforge.net` (last accessed June 2011), 2011.
14. Max Völkel. RDFReactor – From Ontologies to Programatic Data Access. In *Proc. of the Jena User Conference 2006*. HP Bristol, 2006.
15. Sanjiva Weerawarana, Francisco Curbera, Frank Leymann, Tony Storey, and Donald F. Ferguson. *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.