Developing Ontology Based Applications with O3L

AGOSTINO POGGI
Dipartimento di Ingegneria dell'Informazione
Università degli Studi di Parma
Viale G.P.Usberti 181/A, 43100, Parma
ITALY
Agostino.poggi@unipr.it http://www.ce.unipr.it/people/poggi

Abstract: - Ontologies have been gaining interest and their use has been spreading in different applications fields. However, their use in the realization of applications might be further increased by the availability of more usable and efficient software library for the management of ontologies. In this paper, an object-oriented software library for the management of OWL ontologies is presented. This software library, called O3L (Object-Oriented Ontology Library), provides a complete representation of ontologies compliant with OWL 2 W3C. O3L has not the goal to be use for the creation and manipulation of ontologies, but provides a simplified and efficient API for the realization of applications, that interoperate through the use of shared ontologies, and allows: i) the use of OWL individuals as data of the applications, ii) the exchange of OWL individuals between applications, iii) the reasoning about OWL individuals, and iv) the classification of OWL classes and properties. This software library has been experimented in the realization of some e-business applications showing both high effortlessness in the development of the applications and high performances in their execution.

Key-Words: OWL, object-oriented model, ontology based applications, ontology reasoning, semantic Web, Java.

1 Introduction

While there are many definitions of what an ontology is (see, for example, [12][13][21][23]), the common thread in these definitions is that an ontology is some formal description of a domain of discourse, intended for sharing among different applications, and expressed in a language that can be used for reasoning [35]. An ontology necessarily entails or embodies some sort of world view with respect to a given domain. That world view is often conceived as a set of concepts (e.g., entities, attributes and processes), their definitions and their relationships that is referred to as a conceptualization [12]. A conceptualization may be implicit (e.g., it exists only in someone's head), but can be also embodied in a piece of software.

For such reasons, ontologies are considered the most appropriate means that can be used for facilitating the interoperability between heterogeneous systems involved in commonly interested domain applications by providing a shared understanding of domain problems and a formalization that makes ontologies machine-processable.

As a consequence, ontologies have been gaining interest and have been applying not only in the so called semantic Web [4][33], but also in other different computational fields including knowledge engineering, knowledge representation, qualitative modeling, language engineering, database design, information retrieval and extraction, and knowledge management and organization. (see, for example, [5][18][19][20][24][26][32][34]).

However, their diffusion in the realization of software applications might be increased through the availability of more usable and efficient software library for their management. In fact, the current software libraries either allows only a partial representation and management of the most known and used ontologies defined in the usual ontology representation languages (e.g., OWL [8][11]) or provide a complex API that guarantees a complete representation and management of ontologies, but makes difficult to write the code that use or manipulates such ontologies.

This paper presents an object-oriented software library, called O3L (Object-Oriented Ontology Library), that provides a complete representation of ontologies compliant with OWL 2 W3C specifications [22] and a simplified and efficient

API for the realization of applications, that interoperate through the use of shared ontologies. The next section discussed the problems of the mapping of OWL ontologies into an object-oriented representation. Section three introduces the O3L object-oriented model for representing ontologies. Section four presents how the algorithms for reasoning about OWL individuals are defined. Section five discusses about the implementation of the O3L software library. Section six shows how an ontology based application can be realized with O3L. Finally section seven concludes the paper and outlines some future work.

2 From OWL to an OO Model

The Web Ontology Language (OWL) is a semantic markup language that is become the international reference means for publishing and sharing ontologies [8][11].

The mapping of an OWL ontology into an object-oriented representation can be very useful for increasing the diffusion of ontologies and semantic Web technologies [25][31]. In fact, the availability of such a representation can be the basis for the development of some flexible and efficient software libraries for the management of ontologies that allow to cope with the limits of the current software libraries and tools for the realization of ontology based applications.

The main problem of this mapping is that there are important semantic differences between OWL and an object-oriented language and so it is difficult to provide an object-oriented mapping that both minimizes the need of writing code manually and full satisfies OWL semantics.

OWL allows the definition of classes and properties as specialization of multiple classes and properties. Therefore, the object-oriented languages that provide multiple inheritance would seem to be the most suitable for representing OWL ontologies. However, the use of multiple inheritance can cause conflict because a subclass can inherits the same variable or method from different classes. These inheritance clashes are usually resolved by the subclass either redefining the conflicting variable or method for itself or by specifying which inheritance is preferred. These inheritance clashes are possible in representing OWL ontologies (e.g., when an OWL class can inherits a restriction on the same property from different classes) and so they must be managed through the manual or automatic generation of some additional code.

OWL ontologies can be represented also by using object-oriented languages that do not provide multiple inheritance. For example, some previous approaches coped with this problem by using Java interfaces [3][16]. This solution only partially solves the problem because interfaces allow the definition of class variables and methods, while instance variables and methods code must be provided by the classes implementing the interfaces. Therefore, the representation of OWL ontologies requires the manual or automatic generation of a large amount of additional code.

Another problem of representing OWL classes and properties with classes of an object-oriented language is the mapping of OWL class and property names into class names of the object-oriented language. In fact, the most known object-oriented languages have restrictions on the syntax of class names different from the ones imposed by the OWL language. In this case, the solution is to: i) change the OWL class and property names on the basis of the restrictions of the target language (e.g., trading-price may be changed to trading_price for defining a Java or C++ class) and ii) avoid the introduction of name conflicts (e.g., trading-price and trading+price cannot be both changed into trading_price).

A solution for avoiding the previous problems, is the decomposition of inheritance into the more basic mechanisms of object composition and message forwarding [10]. Therefore, for example, an OWL class contains (the references to) its super classes, does not inherit their features, but can get/modify them through the methods provided by the super classes. Moreover, as done in other approaches, the problem of representing an OWL ontology is separated from the problem of acting and reasoning on it. This solution allows the definition of a very simple OWL ontology model based on few classes, that define the variables for maintaining the components of a particular kind of OWL resource and implement the methods for getting and setting their values. Therefore, an OWL ontology is described by a set of instances: one of them contains the general information about the ontology and the others describe its classes, properties and individuals.. Moreover, this solution avoids the problem of mapping OWL resource names in admissible identifiers of the used object-oriented language, because the name of an OWL resource

become a value that is stored into a variable of the object representing such a resource.

Different recent works followed this approach [15][25][30][36][38]. In particular, the OWL API [25] is the reference software library for the latest OWL 2 specifications [22] and provides all the functionality for creating, examining and modifying OWL 2 ontologies; moreover, it also offers a selection of parsers, renderers, and interfaces to the most known ontology reasoners.

The rest of the paper presents another object-oriented software library, called O3L (Object-Oriented Ontology Library), that, in a similar way of the OWL API, provides a complete representation of ontologies compliant with OWL 2 W3C specifications. This library is not an alternative to the OWL API because has not the goal to be use by applications for the creation and manipulation of ontologies, but provides (respect to the OWL API) a simplified and efficient API for the realization of applications that interoperate through the use of shared ontologies.

3 O3L Ontology Model

The O3L ontology model provides a representation of ontologies, described in the OWL 2 format, that completely encoded the information about the relationships among OWL classes and properties.

This model is based on five different elements: *OwlOntology*, *OwlClass*, *OwlDataProperty*, *OwlObjectProperty* and *OwlIndividual*. This model is based on the assumption that the applications that use one or more ontologies through the O3L software library do not need to modify such ontologies, but can create and then modify, delete and exchange some OWL individuals built on the basis of the resources defined in such ontologies. Therefore the information that are encoded in the O3L representation of the resources of an OWL ontology are not the ones declared by the OWL axioms of such ontology, but are the ones inferred by a complete reasoning on the OWL ontology (e.g., the direct ancestors of an OWL class are not the OWL classes of which it is declared as a subclass, but the ones that directly subsume such an OWL class).

An OWL ontology is represented by the *OwlOntology* model that contains information about: i) the URI representing the ontology identifier and the information about the version of the ontology and about the previous versions of such an ontology that are compatible with the current version, and ii) all the classes, properties and individuals that are defined or referred in such an ontology (see table 1).

Ontology identifier
Ontology version
Ontology backward compatible versions
Imported ontologies
Classes
Properties
Individuals

Table 1. Owl ontology variables.

An OWL class is represented by the *OwlClass* model that contains information about: i) the URI representing the class identifier, ii) the URI identifying the ontology where the class is defined, iii) the direct ancestors and descendent of the class, and, finally, iv) the equivalent and disjoint classes (see table 2).

Class identifier
Ontology where the class is defined
Direct ancestor classes
Direct descendent classes
Equivalent classes
Disjoint classes

Table 2. Owl class variables.

Property identifier
Ontology where the property is defined
Direct ancestor properties
Direct descendent properties
Equivalent properties
Disjoint properties

Table 3. Owl property variables.

While the information of OWL annotation and ontology properties is directly encoded in the representation of the OWL resources to which they are related, OWL datatype and object properties are respectively represented by the *OwlDataProperty OwlObjectProperty* models that contain information about: i) the URI representing the property identifier, ii) the URI identifying the ontology where the property is defined, iii) the direct ancestors and descendent of the property, and, finally, iv) the equivalent and disjoint properties (see table 3).

Finally, OWL individuals are represented by the *OwlIndividual* model that contains information about: i) the URI representing the individual identifier, ii) the URI identifying the ontology where the individual is defined, iii) the classes to which the individual belongs, iv) the equivalent and different individuals, and v) the property-values pairs (facts) describing the individual (see table 4).

Individual identifier
Ontology where the individual is defined
Individual classes
Equivalent individuals
Different individuals
Facts

Table 4. Owl individual variables.

4 Reasoning about individuals

While it is not necessary to provide a support for reasoning about OWL classes and properties, given that an application cannot modify them and so their relationships are encoded in their representations, it is necessary to provide some algorithms for reasoning about OWL individuals. Such algorithms allow to check: i) the membership of OWL individuals to OWL classes, ii) if OWL individuals satisfy OWL properties and data types constraints, and iii) if OWL individuals are either equivalent to or different from other OWL individuals.

These algorithms cannot be implemented by only using the information associated with OWL classes and properties by the O3L models that have been introduced in the previous section, but can be implemented by transforming the declarative representation of the constraints defining OWL classes and properties in procedural code. Therefore, the O3L model is enriched by associating an operation with each OWL class, property and individual. While the operation associated with OWL classes and properties checks if the definition of the OWL individuals satisfy their constraints, the operation associated with OWL individuals checks if the OWL individuals satisfy the constraints of the OWL classes and properties that concur to their definition.

This paper, does not introduce a complete description of how the different OWL expressions are transformed into procedural code, but shows how they can be represented by formulas, expressed through set theory and logical expressions, that can be easily encoded in some procedural code. Therefore, let be:

- i and l respectively the individual to be checked and an its literal to be checked;
- ce and CE respectively a class expression and a set of class expressions;
- dr and DR respectively a data range and a set of data ranges;
- dt a data type;
- fl and FL respectively a pair and a set of pairs: facet and literal;
- fl(lt) is true if the literal, lt, satisfies the constraint defined by the pair facet and literal fl;
- ope and OPE respectively an object property expression and a set of object property expressions;
- dpe and DPE respectively a data property expression and a set of data property expressions;
- ie and IE respectively a named / anonymous individual and a set of named / anonymous individuals;
- lt and LT respectively a literal and a set of literals;
- V(ope, i) the set of individuals connected with the individual i through the object property expression ope;
- V(dpe,i) the set of literals connected with the individual i through the data property expression dpe;
- C(OPE, i) the set of individuals connected with the individual i through the chain of object property expressions OPE.

The constraints applied by OWL class expressions on an OWL individual can be expressed through a set of formulas based on set membership expressions as shown in table 5.

In a similar way, the constraints applied by OWL data ranges to the literals, that represent the facts of an OWL individual, can be expressed through some set membership expressions. In particular, the

constraints of OWL data type restrictions, are represented by the conjunction between a set membership expression and another expression that checks if the literal satisfies the constraint defined by a facet – literal pair (see table 6).

subClassOf(CE)	$\forall e \in CE : i \in e$
intersectionOf(CE)	$\forall e \in CE : i \in e$
unionOf(CE)	$\exists e \in CE : i \in e$
complementOf(ce)	$i \notin ce$
oneOf(IE)	$i \in IE$

Table 5. OWL class expressions.

intersectionOf(DR)	$\forall e \in DR : l \in e$
unionOf(DR)	$\exists e \in CE : l \in e$
complementOf(dr)	$l \notin dr$
oneOf(DR)	$l \in DR$
DatatypeRestriction(dt, FL)	$l \in dt \wedge (\forall e \in FL : e(l))$

Table 6. Data range expressions.

someValuesFrom(dpe, dr)	$\exists e \in V(dpe, i) : e \in dr$
allValuesFrom(dpe, dr)	$\forall e \in V(dpe, i) : e \in dr$
hasValue(dpe, lt)	$lt \in V(dpe, i)$
minCardinality(dpe, n)	$\lvert V(dpe, i) \rvert \geq n$
minCardinality(dpe, n, dr)	$\lvert V(dpe, i) \cap dr \rvert \geq n$
maxCardinality(dpe, n)	$\lvert V(dpe, i) \rvert \leq n$
maxCardinality(dpe, n, dr)	$\lvert V(dpe, i) \cap dr \rvert \leq n$
exactCardinality(dpe, n)	$\lvert V(dpe, i) \rvert = n$
exactCardinality(dpe, n, dr)	$\lvert V(dpe, i) \cap dr \rvert = n$

Table 7. Data property restrictions.

The constraints applied by OWL data and object property restrictions on an OWL individual can be described through some set membership and intersection expressions and through some numeric equality and relational expressions applied to the values of the facts describing the OWL individual (see tables 7 and 8).

The constraints applied by OWL data and object property expressions on an OWL individual can be described through some set membership and intersection expressions and through some numeric equality and relational expressions applied to the values of the facts describing the OWL individual (see tables 9 and 10).

someValuesFrom(ope, ce)	$\exists e \in V(ope, i) : e \in ce$
allValuesFrom(ope, ce)	$\forall e \in V(ope, i) : e \in ce$
hasValue(ope, ie)	$ie \in V(ope, i)$
hasSelf(ope)	$i \in V(ope, i)$
minCardinality(ope, n)	$\lvert V(ope, i) \rvert \geq n$
minCardinality(ope, n, ce)	$\lvert V(ope, i) \cap ce \rvert \geq n$
maxCardinality(ope, n)	$\lvert V(ope, i) \rvert \leq n$
maxCardinality(ope, n, ce)	$\lvert V(ope, i) \cap ce \rvert \leq n$
exactCardinality(ope, n)	$\lvert V(ope, i) \rvert = n$
exactCardinality(ope, n, ce)	$\lvert V(ope, i) \cap ce \rvert = n$

Table 8. Object property restrictions.

subPropertyOf(dpe_1, dpe_2)	$V(dpe_1, i) \subseteq V(dpe_2, i)$
propertyDomain(dpe, ce)	$i \in ce$
propertyRange(dpe, dr)	$\forall e \in V(dpe, i) : e \in dr$
functionalProperty(dpe)	$\lvert V(dpe, i) \rvert \leq 1$

Table 9. Data property expressions.

subPropertyOf(ope_1, ope_2)	$V(ope_1, i) \subseteq V(ope_2, i)$
subPropertyOf(PropertyChain(OPE), ope)	$C(OPE, i) \subseteq V(ope, i)$
propertyDomain(ope, ce)	$i \in ce$
propertyRange(ope, ce)	$\forall e \in V(ope, i) : e \in ce$
inverseProperties(ope_1, ope_2)	$\forall e \in V(ope_1, i) : i \in V(ope_2, e)$
functionalProperty(ope)	$\lvert V(ope, i) \rvert \leq 1$
inverseFunctionalProperty(ope)	$\forall e_1 \in V(ope, i) : (\forall e_2 \in I : e_1 \notin V(ope, e_2))$
reflexiveProperty(ope)	$i \in V(ope)$
irreflexiveProperty(ope)	$i \notin V(ope)$
symmetricProperty(ope)	$\forall e \in V(ope, i) : i \in V(ope, e)$
transitiveProperty(ope)	$\forall e \in V(ope, i) : V(ope, e) \subseteq V(ope, i)$

Table 10. Object property expressions.

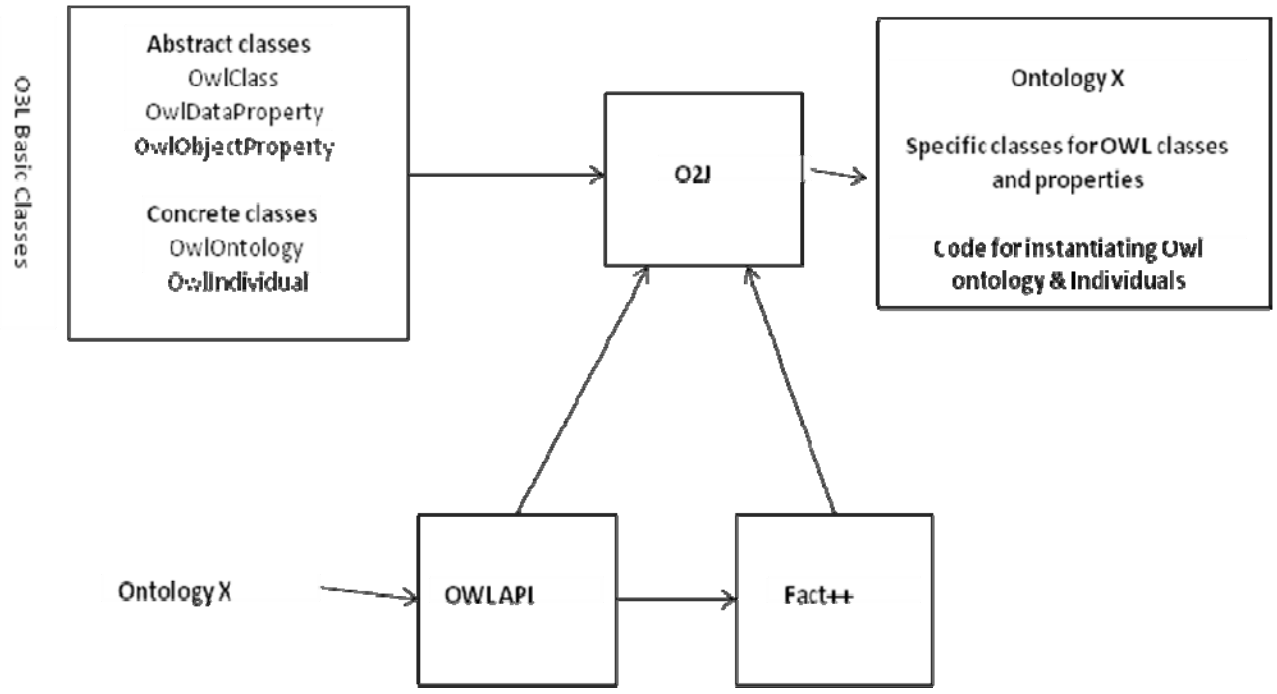

Figure 1. O3L ontology Java classes creation process.

5 Implementing the O3L Model

The O3L model has been implemented taking advantage of the Java programming language.

This implementation is based on five main Java classes: two concrete classes, *OwlOntology* and *OwlIndividual*, for the representation of OWL ontologies and individuals, and three abstract classes, *OwlClass*, *OwlDataProperty* and *OwlObjectProperty*, that are extended by the concrete classes for the representation of OWL classes and properties.

The *OwlOntology* class defines the methods for getting the information of the corresponding OWL ontology that are defined in the O3L ontology model and for getting the instances of the Java classes implementing the OWL classes, properties and individuals of such an OWL ontology.

The *OwlClass*, *OwlDataProperty* and *OwlObjectProperty* abstract classes, besides defining the methods for getting the information of the corresponding OWL entities that are defined in the class and data/object property O3L models, add an abstract method, called *satisfy*, whose goal is to check if an OWL individual satisfies the constraints of the represented OWL class or property.

Finally, the *OwlIndividual* class, besides defining the methods for manipulating the information of the corresponding OWL individual that are defined by the O3L individual model, adds a method, called *satisfy*, whose goal is to check if the represented OWL individual satisfies the constraints of all the OWL classes and properties concurring to the definition of such an OWL individual.

The code of the *satisfy* method of the *OwlIndividual* class does not depend on the specific OWL individual because it simply calls the code of the satisfy methods of the Java classes implementing the OWL classes and properties involved in the definition of such an OWL individual.

Of course, the definition and implementation of the previous five Java classes do not conclude the work for representing an OWL ontology with the O3L software because it is necessary both to implement all the concrete Java classes representing the OWL classes and properties of the ontology and to instantiate (with the necessary data) the *OwlOntology* and *OwlIndividual* Java classes for representing the OWL ontology and its OWL individuals.

This work is not delegated to the application developers, but is automatically performed by an O3L software tool called O2J (OWL to Java). This tool has been implemented by taking advantage of the OWL API [25] and of the FACT++ reasoner [9]. It is done because:

- the OWL API provides a complete access to the information of an OWL ontology without the need of managing both the different formats in

which it can be represented (e.g., RDF and Manchester formats) and the different kinds of repository where it can be maintained (e.g., file systems and database management systems);
– the FACT++ reasoner can manipulate OWL API ontology representations and allows a complete classification of OWL classes useful for the generation of the O3L Java classes representing an OWL ontology.

The process, executed by the O2J tool for building the Java classes of a particular OWL ontology, is based on the following five steps (see figure 1 for a graphical representation of the process):

1. an intermediate object-oriented representation of the OWL ontology is built taking advantage of the OWL API;
2. all the explicit relationships among OWL classes, properties and individuals are checked and all the implicit relationships among OWL classes, properties and individuals are found taking advantage of the FACT++ reasoner applied to the OWL API representation of the OWL ontology;
3. the Java classes for the OWL classes and properties of the ontology are created and filled with the information defined by the corresponding O3L models and acquired by the OWL API ontology representation, and by the results obtained by the FACT++ reasoner;
4. the code for instantiating the *OwlOntology* and *OwlIndividual* classes for the current OWL ontology and for its individuals is generated on the basis of the OWL API ontology representation;
5. the code of the *satisfy* methods of the Java classes representing the OWL classes and properties of the current ontology is generated taking advantage of the OWL API ontology representation.

The Java classes generated by the O2J tool are grouped into Java packages: each package corresponds to an ontology and its name corresponds to the ontology URI. As it is written in a previous section, the name of the Java classes might not always correspond to the name of the corresponding OWL entities because of the different constraints imposed by the OWL and Java syntax. Therefore, the name of such classes is automatically generated by the O2J tool without taking into account the name of the corresponding OWL entity. However, it does not make complex the use of the O3L Java ontology representation because the relationships between the entities of an ontology are mapped by using the URI of the corresponding OWL entity and so it is possible use the name of the OWL entities for accessing the corresponding O3L Java classes.

6 Realizing Applications with O3L

An O3L based application is centered on a set of predefined ontologies, that define the domain of the application, and on another ontology, called working memory, that imports the previous ontologies and does not contain OWL classes and properties, but also maintains the set of OWL individuals representing the most relevant data of the application.

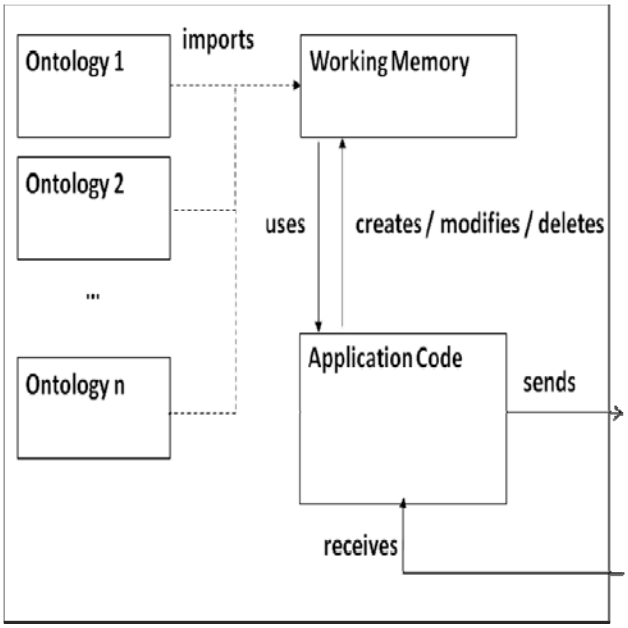

Figure 2. Structure of an O3L application.

While the application cannot modify the predefined ontologies, it can modify the working memory. In particular, the application can (see figure 2):

– initialize the working memory with a set of OWL individuals;
– dynamically generate and eliminate new OWL individuals;
– dynamically manipulate the OWL individuals of the working memory taking also advantage of O3L reasoning support;
– dynamically acquire and transmit OWL individuals from/to some other applications.

An O3L application can also use data that are not OWL individuals, but the use of OWL individuals is at least recommended when such data correspond to entities defined in the domain model of the

application and is necessary when the processing of such kinds of data requires the use of ontology reasoning algorithms and when they are exchanged with another application that should interoperate with such an application thanks to the sharing of some ontologies.

The development process of an O3L based application may be based on the usual steps of any software engineering development process. In such a process, two of the main tasks are the identification of the domain requirements and the design (or reuse) of the ontologies needed for the representation of such a domain.

Of course, the reuse of a predefined ontology that presumably might be used by other applications should be preferred to the design of a custom ontology because it may simplify the interoperability with such application. Moreover, the reuse of a predefined ontology should be preferred even if such a predefined ontology either covers a larger domain that the one covered by the developing application or could be used in such application only after an adjustment of its domain model.

7 Conclusion

This paper presented an object-oriented software library for the management of OWL ontologies. called O3L (Object-Oriented Ontology Library) [28]. This software library provides a complete representation of ontologies compliant with OWL 2 W3C [11]. O3L has not the goal to be use for the creation and manipulation of ontologies, but provides a simplified and efficient API for the realization of applications that interoperate through the use of shared ontologies.

The problem of providing a Java representation of OWL ontologies in not new and different solutions have been proposed.

These solutions can be divided in two groups: the first group includes low level APIs that directly manipulate the ontology (see, for example, [25][30]) and the second includes high level APIs that hide certain parts of the model disabling the user (or program) to add new classes and properties to the model at run time (see, for example, [9][15][16]).

Moreover, while the solutions of the first group provide a complete representation of OWL ontologies and support reasoning on them, the solutions of the second group does not provide a complete representation of OWL ontologies and does not support reasoning on them.

O3L belongs to the last group: it uses a compact representation of OWL ontologies, but support a complete reasoning on OWL individuals. Moreover, O3L provides the information on the relationships among the OWL classes and properties without the need of any reasoning algorithm because such information is directly encoded in the Java representation of the OWL classes and properties.

O3L derives from OWLET [27]. In a similar way to what done by O3L, OWLET maps OWL ontology in a Java representation. This mapping is only possible for OWL 1 DL ontologies and is based on a more complex object-oriented model that, however, allows a dynamic reasoning on all the ontologies resources (i.e., classes, properties and individuals). Therefore, at the same way of the OWL API, it can be also used for the developing of new ontologies.

O3L has been experimented in the realization of some e-business applications showing both a high effortlessness in the realization of applications and high performance in their execution.

Current work is devoted to the integration of the O3L library in the JADE multi-agent development framework [1][2][14] and the extension of O3L for providing a Java bean like representation for OWL individuals.

The work on the integration of the O3L library in the JADE multi-agent development framework, besides allowing the direct use of the large repository of available OWL ontologies into JADE multi-agent applications, has the goal of using the O3L ontology reasoning tools for both enhancing the cooperation among agents and the realization of more flexible applications through a semantic composition of agent tasks. Regarding the last point, we are also working on the O3L library to simplify the integration between agent-based and Web services.

The work on the extension of O3L for providing a Java bean like representation for OWL individuals has the goal of adding a further simplification for the realization of O3L based applications by modeling and then implementing OWL individuals as Java beans where each couple of getter and setter methods allows the access to the values of an OWL property that define the OWL individual.

In fact, this kind of representation allows a programmer to realize the software code that manipulate and OWL individual by accessing and modifying the values of a property concurring in the definition of such an individual through the corresponding getter and setter methods.

This solution has the advantage of reducing the length of the application code by avoiding the burden of finding the values of such a property among all the property values defining the OWL

individual (i.e., current implementation manages such values through a Java Map; therefore, before manipulating the values associate with a property is necessary to check if there is an entry for the property in the Map and then retrieve it from the Map).

References:

[1] Bellifemine, F., Poggi, A., Rimassa, G. Developing multi agent systems with a FIPA-compliant agent framework. Software - Practice & Experience, 31:103-128, 2001.

[2] Bellifemine, F., Caire, G., Poggi, A., & Rimassa, G. JADE: a Software Framework for Developing Multi-Agent Applications. Lessons Learned. Information and Software Technology Journal, 50:10-21. 2008.

[3] Bergenti, B., Poggi, A., Tomaiuolo, M., Turci. P. An Ontology Support for Semantic Aware Agents. In. G. Carbonell and J. Siekmann (Eds.). Agent-Oriented Information Systems III, pp. 140-153, Springer, Berlin, Germany, 2006.

[4] Berners-Lee, T., Hendler, J., Lassila, O. The Semantic Web. Scientific American, 284(5):34–43, 2001.

[5] Borges, A.M., Gil, R., Corniel, M., Contreras, L., Borges, R.F. Towards a Study Opportunities Recommender System in Ontological Principles-based on Semantic Web Environment. WSEAS Transactions on Computers, 2(8):279-291, 2009.

[6] Calero, C., Ruiz, F., Piattini, M. Ontologies for Software Engineering and Software Technology, Springer, Berlin, Germany, 2006.

[7] Connolly, D., van Harmelen, F., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Stein, L.A. DAML+OIL Reference Description, W3C Note. 2001. Available from http://www.w3.org/TR/daml+oil-reference.

[8] Dean, M., Schreiber, G. OWL Web Ontology Language Reference, W3C Recommendation 10 February 2004, 2004. Available from http://www.w3.org/TR/owl-ref/.

[9] FACT++ software Web Site. Available from http://code.google.com/p/factplusplus/.

[10] Frohlich, P.H. Inheritance decomposed. In Proc. of the Inheritance Workshop at ECOOP 2002, Malaga, Spain, 2002.

[11] Golbreich, C., Wallace, E.K. Web Ontology Language: New Features and Rationale, W3C Working Draft, 2008. Available from http://www.w3.org/TR/owl2-new-features/.

[12] Gruber, T.R. A translation approach to portable ontologies. Knowledge Acquisition, 5(2):199-220, 1993.

[13] Guarino N., Formal Ontology in Information Systems. In Proc. of FOIS'98, Trento, Italy, pp. 3-15, IOS Press, 1998.

[14] JADE software Web site. 2009. Available from http://jade.tilab.com.

[15] Jastor software Web Site. 2009. Available from http://jastor.sourceforge.net.

[16] Kalyanpur, A., Pastor, D., Battle, S., Padget, J. Automatic mapping of owl ontologies into java. In Proc. of the Sixteenth International Conference on Software Engineering & Knowledge Engineering (SEKE'2004), pp. 98-103, Banff, Canada, 2004.

[17] Knublauch, H. Ontology-driven software development in the context of the semantic web: an example scenario with protegé/OWL. First International Workshop on the Model-Driven Semantic Web (MDSW), 2004.

[18] Kozaki, K, Hayashi, Y., Sasajima, M., Tarumi, S., Mizoguch, R. Understanding Semantic Web Applications. In K. Aberer, K. Choi, N. Noy, D. Allemang, K. Lee, L. Nixon, J. Golbeck, P. Mika, D. Maynard, R. Mizoguchi, G. Schreiber, P. Cudré-Mauroux (Eds.). The Semantic Web -Lecture Notes in Computer Science, Vol. 4825, pp. 524-539, Springer, Berlin, Germany, 2008.

[19] Lopez de Vergara, , J.E., Guerrero, A., Villagra, V.A., Berrocal, J. Ontology-Based Network Management: Study Cases and Lessons Learned. Journal Network System Management, 17:234–254, 2009.

[20] Masuoka, R., Labrou, Y., Parsia, B., Sirin, E. Ontology-enabled pervasive computing applications. IEEE Intelligent Systems, 18(5):68-72, 2003.

[21] McGuinness, D.L. Ontologies Come of Age. In D. Fensel, J. Hendler, H. Lieberman and W. Wahlster (Eds.). The Semantic Web: Why, What, and How, pp. 171-194, MIT Press, Cambridge, MA, 2001.

[22] Motik, B., Patel-Schneider, P.F., Parsia, B. Web Ontology Language: Structural Specification and Functional-Style Syntax, W3C Working Draft, 2008. Available from http://www.w3.org/TR/owl2-syntax/.

[23] Noy, N.F. Semantic integration: a survey of ontology-based approaches. SIGMOD Record. 33(4):65-70, 2004.

[24] Oberle, D., Eberhart, A., Staab, S., Volz, R. Developing and managing software components in an ontology-based application

server. In Proc. of the 5th ACM/IFIP/USENIX International Conference on Middleware, pp. 459-477, Toronto, Canada, 2004.
[25] OWL API software Web Site. Available from http://owlapi.sourceforge.net/.
[26] Pan, T.J.,, Zheng, L.N., Zhang, H.J., Fang, C.B., Lou, J., Shao, Z. Combining Web Services Toward Innovative Design of Agile Virtual Enterprise Supported by Web 3.0. WSEAS Transactions on Communications, 1(8):81-91, 2009.
[27] Poggi, A., OWLET: An Object-Oriented Environment for OWL Ontology Management. In Proc. of the 11th WSEAS International Conference on COMPUTERS, pp 44-49, Agios Nikolaos, Greece, 2007.
[28] Poggi, A., O3L: An OWL Object-Oriented Library for the Realization of Ontology Based Applications. In Proc. of the 13th WSEAS International Conference on COMPUTERS, pp. 340-345, Rhodes, Greece, 2009.
[29] Prieto, A.E., Lozano-Tello, A. Use of Ontologies as Representation Support of Workflows Oriented to Administrative Management. Journal Network System Management, 17: 309–325, 2009.
[30] Protegé OWL API Web Site. Available from http://protege.stanford.edu/plugins/owl/api/.
[31] Puleston, C., Parsia, B., Cunningham, J. Integrating Object-Oriented and Ontological Representations: A Case Study. In A. Sheth, S. Staab, M. Dean, M. Paolucci, D. Maynard, T. Finin, K. Thirunarayan (Eds.). The Semantic Web - ISWC 2008, pp. 140-153, Springer, Berlin, Germany, 2008.
[32] Salguero, A., Araque, F., Delgado, C. Ontology based framework for data integration. WSEAS Transactions on Information Science & Applications, 6(5):953-962, 2008.
[33] Shadbolt, N., Hall, W., Berners-Lee, T. The Semantic Web Revisited. IEEE Intelligent Systems, 21(3):96-101, 2006.
[34] Tran, T., Peter Haase, P., Lewen, H., Muñoz-García, O., Gómez-Pérez, A., Studer, R. Lifecycle-Support in Architectures for Ontology-Based Information Systems. In K. Aberer, K. Choi, N. Noy, D. Allemang, K. Lee, L. Nixon, J. Golbeck, P. Mika, D. Maynard, R. Mizoguchi, G. Schreiber, P. Cudré-Mauroux (Eds.). The Semantic Web - Lecture Notes in Computer Science, Vol. 4825, pp. 508-522, Springer, Berlin, Germany, 2008.
[35] Uschold, M, Gruninger, M. Ontologies: Principles, methods and applications. Knowledge Engineering Review, 11:93-136, 1996.
[36] Völkel M., Sure, Y. RDFReactor -- From Ontologies to Programmatic Data Access. In Proc. of the Jena User Conference,. Bristol, UK, 2006.
[37] Welty. C. Ontology research. AI Magazine, 24(3):11-12, 2003.
[38] Zimmermann, M. Owl2Java - A Java Code Generator for OWL, 2009. Available from http://www.incunabulum.de/projects/owl2java..