Future Generation Computer Systems 26 (2010) 389–399

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

OSLN: An Object-Oriented Semantic Link Network language for complex object description and operation

Xiaoping Sun

China Knowledge Grid Research Group, Key Lab of Intelligent Information Processing, Institute of Computing Technology, Chinese Academy of Sciences, 100190, Beijing, China

ARTICLE INFO

Article history:
Received 3 March 2009
Received in revised form
1 June 2009
Accepted 24 July 2009
Available online 3 August 2009

Keywords:
Semantic Web
Semantic Link Network
Representation language
Object-Oriented languages

ABSTRACT

As the semantic data grows rapidly on the Web, we need flexible and powerful tools to describe and manage complex data, information and knowledge structures on the Web. Basic structural semantic information of classes, instances, properties and relationships can be described using Semantic Web languages. More and more applications need to describe and manage objects with complex structures, operations and interactions on the Web. In this paper, we introduce an Object-Oriented Semantic Link Network language OSLN that can be used to define complex objects with rich object-oriented semantics on the Web. In OSLN, objects are the basic semantic elements with internal members and functions that are declared to express attributes and semantic processes. Semantic links are defined to describe semantic relationships among objects. Many important features from traditional object-oriented programming languages are incorporated into OSLN, allowing users to write semantic programs for not only describing complex structures of objects but also defining object operations and manipulations. OSLN enables users to write semantic scripts like using traditional programming languages, which will improve both user experiences and application areas of the Semantic Web technologies.

1. Introduction

As the Web grows rapidly, semantic contents can be utilized over a large-scale networked environment to implement more intelligent information services. Semantic Web technologies have become the standard for describing formal semantic content on the Web [1]. RDF and RDFS are the basic elements underlying the protocol stack of the Semantic Web technologies. In RDF, basic elements of semantic information are represented by triples consisting of subject, predicate and object. RDF triples can be used to describe semantic relationships between subjects and objects. RDFS extends RDF by defining a formal vocabulary consisting of class and property. The OWL language incorporates Description Logic into RDF to describe more rich semantics such as transitivity and symmetry [2]. In OWL, TBox contains schema that describes the basic semantic structure based on class semantics. ABox contains assertions that are defined by deriving instances from the elements of TBox. Reasoning tasks can be performed on TBox and ABox to test whether an object is subsumed by another.

Both RDFS and OWL are carefully designed based on RDF to mainly capture the class–instance relationship semantics. The key feature of RDF is that the property, class and instance are defined independently so that they can be defined and used at different sites on the Web. The open-world assumption is adopted in the logic framework of RDFS and OWL languages so that semantic objects can be easily integrated on the Web. Relationship semantics can be easily captured using triples of RDFS and OWL. However, when describing objects with complex internal structures and operations, users often face many difficulties.

First, RDFS and OWL do not give a syntactical bound and a semantic bound for defining a class with its own properties. For example, when defining a property P for a class A, one has to use *rdf:subclassOf* on a *owl:Restriction* element, where the *owl:Restriction* element will relate class A to P that is defined elsewhere using an *owl:ObjectProperty* element. This semantically correct construction however does not explicitly indicate that property P belongs to class A, rather, it means that class A is a subclass of objects that have P property. Moreover, when another class B is derived from A, P is also inherited completely to B through A. This will incur difficulties when B actually needs to use a different interpretation of P, because there is no explicit overriding mechanism for B to redefine P. The only way is to define a new Property P' for class B.

Second, structures of class and individuals are difficult to define by RDFS and OWL languages. There are too limited operators for users to define and control the internal structure of a class. For example, there is no direct way to define a dynamic set or an array where objects will be inserted or removed in the future operations.

Third, it is hard to define reference relations between classes and between individuals. The reference relation here means that the value of a property P_1 of class A (or an individual) is taken from

E-mail address: expensun@yahoo.com.cn.

0167-739X/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2009.07.007

a property P_2 of another class B (or an individual). It is because properties are defined outside the class and there is no direct way to refer to a property of a specific class or an individual.

The above problems can be solved by defining new elements and structures based on RDF. We need a framework that encapsulates expressions and manipulations of those complex semantic structures into a pattern that is easy to use. In this paper, we present an Object-Oriented Semantic Link Network language (OSLN) that is built to describe complex objects as well as their operations. OSLN is not a new Semantic Web language. It can be viewed as a framework that helps developers writing Semantic Web scripts to describe complex objects and operations.

To help describing complex semantic structures, OSLN adopts many useful semantic elements from traditional Object-Oriented Programming (OOP) languages such as C++ and JAVA. Class–instance relationships in OOP languages are used for code reuse, which greatly eases the programming process for developers. Semantics of OOP languages are defined for code compiling and running but not for semantic interpretation of codes themselves, which is quite different from Semantic Web languages that have a model-theory-based semantic interpretation schema. Traditional OOP languages have many useful structures and operators such as encapsulation, overriding, and polymorphism that should be very useful also for Semantic Web languages. These operators and features not only can be used for code organization and reusing, but also can be used for directly representing semantics of contents. Encapsulation will give a clear bound for objects that have internal properties. Overriding and polymorphism will provide flexible semantic object manipulations and integrations. OSLN incorporates those structures and operators from OOP languages to enable developers write Semantic Web programs more easily and more expressively.

Another important feature of OSLN is that it uses a set of operators from Semantic Link Network (SLN) [3,22] language to implement semantic reference among objects in OSLN. In the SLN language, relation triples are used to represent semantic relationships between objects. A set of predefined relation operators is provided in SLN including *imply*, *cause–effect*, *reference* and so on. These semantic operators are different from those relationships in RDFS and OWL in that SLN explicitly gives each operator a specific concrete semantic meaning, rather than giving some properties of relationships (such as symmetry, reflexive, and reversible). From this aspect, SLN can be viewed as a specific pattern of RDF triples. OSLN uses the *reference* link from SLN to represent a reference relationship between two objects. Other semantic link operators can also be used to describe relationships between a set of semantic codes and another set of semantic codes.

The principle design rationale behind OSLN is to provide programmers an easy-to-use platform to help them transfer from traditional OOP language environments to Semantic Web languages. Using OSLN, developers can express formal semantics content and control the content dynamically using semantic codes. We will introduce the details of OSLN by starting with an example.

2. Application example

To illustrate the using scenarios of the OSLN language, let us first consider an application that provides information services for the maintenance and the protections of Chinese ancient buildings. In this application, we need to describe structures of ancient buildings as well as the information of repairing and protection processes of ancient builds. It is difficult to use Semantic Web languages to describe those complex structures. Database systems are also too rigid to support recording semi-structured data. We need a flexible, easy-to-use, and extensible language platform for semi-structural knowledge representation and semantic queries. It is important that the language can be easily grasped by non-computer-professional users.

The structures of an ancient building should be described formally. Different building parts and their relationships inside the building need to be explicitly represented. There are also many related background knowledge data for the building and its internal parts. Repairing knowledge and repairing cases need to be documented for future querying and sharing. Moreover, many structures of data cannot be defined in advance because they depend on the data and information used during the building repairing process. Knowledge construction and representation are rather ad hoc. Knowledge described in previous work can be augmented, reduced, or updated by new repairing processes. Finally, this application runs on the Web for the distributed knowledge sharing and management.

In the following examples, we show how to use OSLN to define basic objects of Chinese ancient buildings and manipulate the semantic data.

2.1. Object definition

In this application, many resources and information of building structures are often descriptive texts from which one has to draw formal semantic contents. For example, a description of girders will be like this: *A traditional Chinese girder in a building, also named Fang, is a main horizontal support beam of wood. The cross section of most girders in North China is of rectangle shape. Girders of buildings in Song Dynasty is of square section and girders in Ming Dynasty often have a rectangle section with the ratio of length to width being 3:2. In south China, however, girders of ancient buildings often have a section of circle.*

The basic element of the OSLN language is the object. To give a more formal description, we need to abstract objects from above descriptions. We first define an object *BuildingObject* for representing the general concept of building objects. Then, we define an object *Girder* that is derived from the object *BuildingObject*. We also define an object *GirderOfSong* for the girders of the Song Dynasty and an object *GirderOfMing* for the girders of the Min Dynasty.

As shown in Fig. 1, the object *BuildingObject* declares four public members of string types. Public members can be accessed by other objects. A member with a delimiter TRANS will be inherited by the sub-classing objects. For example, in the Fig. 1 the member declarations of *function*, *name*, *date*, and *location* in the object *BuildingObject* will be inherited by the object *Girder* and the sub-classing objects of *Girder*. But the value of the public members will not necessarily be inherited unless the TRANS is used to delimit assignment operator " =". For example, the value of the member *function* will be also inherited by the object *Girder*. Values of other members such as *name*, *date*, and *location* will not be inherited by the sub-classing objects. Thus, the member *name* in the object *Girder* will not have the value "*Building Ware*".

The object *Girder* is derived from the object *BuildingObject*. The member *name* derived from *BuildingObject* is assigned with a new value {"Girder" | "Fang"}, which means that either "Girder" or "Fang" is the value of the member. The shape of section of girders needs to be described. It is defined as a property member *section* in the object *Girder*. A member *section* will be assigned with different values according to the value of the member *location* and *date*. IF-THEN clauses are used to assign different values to the member *section* according to the *date* and the *location* of girders. Note that the value of *section* defined in *Girder* will be also inherited by the object *GirderOfSong* and the object *GirderOfMing*. Since *date* in *GirderOfSong* is set to *Ming*, the member *section* will return *square* when the girder belongs to a *north China* building. Thus, we do not have to redefine the member *section* for *GirderOfSong* and *GirderOfMing*.