SOFA: Simple Ontology Framework API

Alexey Alishevskikh
Technology Director,
ViceVersa Technologies
Ekaterinburg, Russia
alex@viceversa.ru
alexeya@gmail.com

Ganesh Subbiah
Assistant System Engineer,
TCS Nortel-Product Test
Tata Consultancy Services Ltd,
Mumbai 400 066, India
ganesh1.s@tcs.com

Abstract

SOFA (Simple Ontology Framework API) is a Java API representing an object model of a specification of conceptualized knowledge, known as an Ontology[1]. It is intended for using by developers of the Semantic Web, Information Retrieval, Knowledge Bases applications and other ontology-driven software engineering.

SOFA provides a simplified, intuitive and highly abstract model of ontology which is independent of a specific ontology representation language and operates with ontologies on a conceptual, rather than syntactic level. It allows for SOFA-based applications to operate with ontologies described in diverse language forms and gives significant advantages in respect to the software development simplicity.

Keywords

Ontology, Semantic Web, OOP, Java, API

1 Introduction

The emergent challenges of the Semantic Web and Knowledge Management industry growth gave birth to new class of a software – the software which deals with *knowledge*. We are eye-witnesses the process of new approach coming to represent the business data which includes a *semantics* as a necessary quality of information. Traditional descriptions of data models, such as relational database schemes, are superceded by formalized specifications of explicit knowledge, known as the *ontologies*. This approach adds a semantic layer of *metadata* to the data model which causes new advanced possibilities of information classification, findability and machine processing.

During the last few years an area of ontologies using was essentially widened on the grounds of the Semantic Web initiative [2]. New needs to represent WWW content in machine-processable form with explicitly expressed semantics caused active researches and efforts reinforcement in ontology languages and design. The results of these efforts is the family of W3 Consortium standards, such as RDF[3], RDF-Schema[4] and OWL[5] languages, which provides a framework for semantical description of the web-resources and building distributed, massively scalable web-based ontologies.

Meantime, needs of processing ontological data cause another challenge – a necessity of new approaches to software design and development. Traditional data-oriented approaches in most cases don't fit for ontology-oriented software, or lead to unjustified time expenses and software development costs. The new, ontology-oriented and ontology-based software needs a specialized development framework and methodology.

Fortunately, power and flexibility of the modern object-oriented environments allow not to invent new specialized ontology software frameworks from the ground. This paper shows how the ontology software may be implemented in well-known obect-oriented programming techniques and existing development platforms using. It describes a design and implementation of the SOFA (Simple Ontology Framework API) – an attempt to implement the integral ontology software framework as an object-oriented Application Program Interface (API) for the Java platform.

The suggested software is aimed to provide the following tools for ontology applications development:

- An API of an abstract, language-independent Ontology object model to process in uniform programmatic way.
- An ontology inferencing mechanism.
- An API of ontology storage model, which allows multiple implementations for several physical storages.

- A mechanism for interoperating between distinct ontologies.
- Tools for external ontology model representation with common ontology languages.
- A mechanism for ontology validation.

The main task of an object-oriented ontology framework is to provide an ontology object model. An ontology object model is a representation of ontological concerns in a form of object-oriented programming language entities – the interfaces and classes. It gives a view on ontology from a software developer perspective and allows to manipulate (i.e. create, change and retrieve) the ontology items programmatically. Below in this section, we will consider few existing approaches to access the ontological data in software applications and see what makes the SOFA approach different.

1.1 Existing approaches to represent the ontologies in software applications

1.1.1 Ontology Model as a directed graph

It represents the ontology model as a directed graph structure, or as a set of *subject-predicate-object* triples. This is a very low-level representation which in the best way fits for machine processing purposes due to universality and computational performance. On the other hand, it is absolutely counter-intuitive and hardly suitable for the most of ontology programming tasks, so currently this model is mostly used only as an underlying layer for another, high-level ontology models.

1.1.2 Language-dependent Ontology Model

This approach assumes representing an ontology as an object model of a specific ontology language. It grew out of practical needs to parse, process and produce RDF (later OWL) files. This approach was implemented in the well-known Jena API[6], widely used for ontology software development. Other modern ontology API's are following this way as well.

In this methodology, an ontology model exactly matches the model of a specific language. Building blocks and other syntax constructions of a language are straightforwardly implemented as elements (objects and classes) of the object model. It gives full control of every aspect of an ontology described with a given language and it is necessary for needs of processing ontologies in a context of a specific language syntax. Also it guarantees exact conformance of the object model with a resulted representation. However, this approach has few disadvantages:

- The ontology model rigidly depends on a specific ontology representation language, what restrict an area of use instances of this model. When a language changed, the API must be changed in accordance with latest alterations.
- The ontology model abounds of details related with a language and has low level of abstraction. So, it is often too complex and counter-intuitive for the most of ontology programming tasks.

1.1.3 Ontologies and data object models (the Kazuki approach)

The original approach to bind the ontologies with application data has been implemented in the Kazuki project[7]. In Kazuki, an ontology (an OWL file) is used as a source for auto-generating the interfaces of an object model of data in a problem domain. An ontology is mapped into a business data model and all further manipulations are performed on that model. Therefore, this technology makes sense for traditional software development and cannot be considered as a specialized ontology framework.

1.2 Conceptual Ontology Model: the SOFA approach

A different approach is implemented in design of the SOFA ontology model. SOFA represents a highly abstract view on ontology which is independent from specific languages, storage or transmission techniques, and other particular factors. This view is summarized in the Conceptual Ontology Model – a high-level, language-neutral and simplified representation of an ontology. This model is a basis of SOFA object-oriented API and it is straightforwardly implemented in API as a hierarchy of Java interfaces. When programming with the SOFA API, a developer operates among conceptual knowledge concerns – such as concepts, instances and relations. Such level of model abstraction allows to develop ontology software which doesn't depend on particular ontology representation form. Several different ontology description formats may be used to represent an ontology model. It allows to consider the SOFA model as an intermediate for different ontology languages.

2 Conceptual Model of an Abstract Ontology

The Ontology Conceptual Model reflects a vision of the ontology domain as it is implemented in SOFA Ontology Object Model API. This vision comes from practical needs of ontology software development and from common modern approaches and standards.

2.1 Relations to OWL and other ontology languages

The SOFA ontology model is conceptually consistent with the certain subset of W3C OWL language. It means that every SOFA model is able to be interpreted in terms of the OWL language and represented in form of the OWL language expressions. If those expressions are parsed back to SOFA, they will produce the same ontology model as initial one. But it is not guaranteed that any arbitrary OWL ontology is able to be processed with the SOFA model without losses of certain OWL-specific aspects.

Other ontology languages may be used for the SOFA model representation to the extent of the SOFA model may be interpreted with their expressional capabilities.

2.2 Elements of a Conceptual model

2.2.1 Things

The *Thing* is a root notion of the SOFA conceptual model. This is a logical meaning ontology member which encapsulates knowledge about a specific individual within an area of interest. This knowledge is conceptualized as a set of statements declaring some facts about a subject represented of the given Thing – its relationships with another Things or with actual datatype values. A Thing usually is an *instance* of one or more ontology *Concepts*, what declares its membership in a certain group of the similar Things sharing some common charactersistics. The notion of the Thing is semantically equivalent to the same notion in the OWL ontology.

2.2.2 Ontologies

Whereas Things represent the individual knowledge items, an *Ontology* is a representation of knowledge about an area of interest as a whole. It plays a role of a knowledge repository that responsible for Things creation, storing, retrieving and removing. From another perspective, an Ontology is a Thing in itself, that allows for declaring some statements about a whole knowledge domain represented by the Ontology.

The notion of the Ontology is semantically equivalent to the OWL ontology.

2.2.3 Concepts

Concepts are the hierarchical categories of Things that grouped together because they share some common properties. In a context of an ontology, they form a classification system, or a *taxonomy* of the items in a knowledge domain. The Concepts playing role of the classes in object-oriented programming languages, that is they define a common semantical category and a generic pattern for the Things of a specific type. The Concepts can be organized in a specialization hierarchy using *subconcept* axioms. Any Concept may be refined with the subconcepts representing more specialized and narrowed notions. A multiple inheritance is allowed, i.e. a Concept can be the direct subconcept of more than one superconcepts. From another perspective, every Concept is a Thing in itself. It allows to manipulate the whole categories as individual ontology members and declare the statements on the Concepts. The notion of the SOFA Concept is semantically equivalent to the notion of the Class in the OWL ontology.

2.2.4 Relations

The notion of *Relation* represents a specification of relationship between ontology members. This specification defines a type of relationship by setting certain constraints on subjects and objects of statements with the given Relation. These constraints include: a) *a domain* of the Relation specifying the Concepts, instances of which are allowed to be the subjects of the Relation, and b) *a range* of the Relation specifying the Concepts or data types, instances of which are allowed to be the objects (targets) of the Relation. The Relations can be organized in a specialization hierarchy, that is any Relation may have the *subrelations* defining more specialized relationship types. A specific Relation can be declared as a transitive Relation, symmetric Relation, or as an inversion of another Relation. These attributes together with subrelation axioms playing role in reasoning and inferencing the implied facts about the ontology members. The inferencing principles are reviewed below in the text.

In addition to generic constraints defined by a Relation specification, the individual Concepts may put the local *restrictions* on using the Relation in domain of their instances. They include a *cardinality restriction* which limits a maximal and minimal number of statements set on an instance with the given Relation and a *value restriction* limiting the acceptable objects of statements by set of predefined values. Every Relation is a Thing in itself. In such a way, it is possible to treat the specifications of relationship as individual ontology members and declare the statements on them.

The notion of Relation is semantically equivalent to the notion of the Object Property or Datatype Property in the OWL ontology. However, SOFA makes no explicit distinction of these two types of relations – for SOFA, this is only a question of a Relation range. The notion of Restriction is equivalent to the OWL Restriction excepting the ability of OWL restrictions to refine a range of a property.

2.2.5 Datatype values

The datatype values (literals) are the actual data items of a specific data type. In spite of the fact that they are the ontology members, they have no logical meaning in a context of an ontology and they are not among the Things. The datatype values are allowed to be the objects (targets) of statements with Relations having a corresponding datatype specification as their range.

As the SOFA ontology model is designed for the Java language, the datatype values are treated as the Java objects. Virtually any Java class may be specified as a Relation range. It gives a way to bind an application data with ontologies.

2.3 Reasoning and inferencing

The SOFA model is not a static information model. It uses explicitly stated initial facts for inferencing of sets of implied facts about subjects of an ontology. Inferencing logic is defined by specific relation attributes and built-in rules.

2.3.1 Relation attributes

Relations may have the following attributes, playing role in inferencing:

Transitivity: If the relation R is *transitive*, then statements $\{x\ R\ y\}$ and $\{y\ R\ z\}$ cause an implicit statement $\{x\ R\ z\}$.

Symmetry: If the relation R is *symmetric*, then statement $\{x\ R\ y\}$ causes an implicit statement $\{y\ R\ x\}$ and vice versa.

Inversion: If the relation $\bar{R}$ is *an inversion of* the relation R, then statement $\{x\ \bar{R}\ y\}$ causes an implicit statement $\{y\ R\ x\}$ and vice versa.

2.3.2 Built-in inferencing rules

Instances inheritance: If the Thing T is an instance of the Concept C , it is also an instance of all ancestor Concepts of the C.

Subconcepts transitivity: If the Concept C is a subconcept of the Concept C', it is also a subconcept of all ancestor Concepts of the C'.

Subrelations transitivity: If the Relation R is a subrelation of the Relation R' it is also a subrelation of all ancestor Relations of the R'.

Domain concepts inheritance: If the Concept C is a domain concept of Relation R then all subconcepts of C are also the domain concepts of the R.

Range concepts inheritance: If the Concept C is a range concept of the Relation R, then all subconcepts of C are also the range concepts of the R.

Relations generalization: If the Thing T has a statement with the Relation R and with the object O, it also has a set of implicit statements with all ancestor Relations of R and with the same object O.

2.4 Integrity of an ontology

Integrity of an ontology is a state, when all constraints set on its members are maintained. Integrity is evaluated as truth of the following conditions for every Thing:

- A Thing has statements only with the Relations which have the Concepts of this Thing as their domain concepts (applying the rule of "Domain concepts inheritance").
- A Thing has statements only with objects, allowed by ranges of corresponding Relations (applying the rule of "Range concepts inheritance"). For the datatype values (Java objects), they must be assignable to the range Java class.
- A number of statements with a specific Relation satisfies to cardinality restriction for Relation, stated on the nearest Concept of this Thing (applying the rule of "Instances inheritance").
- Values of statements with a specific Relation satisfy to value restriction for corresponding Relation, stated on the nearest Concept of this Thing (applying the rule of "Instances inheritance").

2.5 Ontology members identifying

Each individual (Thing) within an ontology has a *unique identifier*. The identifiers are assigned at new objects creation and are used for retrieving and distinguishing the ontology members. Co-existing of two or more individuals with the same identifier value within the same ontology is not allowed and causes an exceptional situation (an error).

An ontology provides a single *namespace* for all its members. Following to the Semantic Web principles, a namespace value is a well-formed URI (Universal Resource Identifier) which can be treated as URL (Universal Resource Locator) for addressing and retrieving an ontology resource. The individual identifiers can be added to the ontology namespace value as URI fragments. This *qualified identifier* form is used to identify the individual ontology members outside the host ontology. Two or more individuals must be considered as equal if their qualified identifiers are coincided.

3 Implementation

3.1 SOFA Ontology Model API

The SOFA Ontology Model API is designed to reflect the Ontology Conceptual Model as a structure of Java classes. It forms an object-oriented model of an ontology, where every ontology member is represented as a Java object with a known class. The basic classes are defined by four interface definitions which are corresponded to four basic ontology member types (Things, Concepts, Relations and Ontologies). The datatype values are represented in an object model by their own classes.

3.1.1 `Thing` interface

This is a root interface defining a basic functionality of every ontology individual. The most important part of the Thing functionality is setting and getting the statements with relations. For those purposes, the interface defines a wide range of methods to for set, add or remove the single and multiple statements and retrieving of them (including inferencing capabilities). Another group of the `Thing` methods is related to the Concepts – the methods for adding and removing a given Thing to/from a Concept and retrieving references to the Concepts.

3.1.2 `Concept` interface

The subinterface of `Thing` which extends the basic Things functionality for Concept-specific tasks – management of subconcepts and instances, including their creation, retrieving and removing.

3.1.3 `Relation` interface

The `Relation` is a subinterface of Thing, defining a functionality of ontology Relation. It defines the methods for management of domain Concepts, ranges and subrelations.

3.1.4 `Ontology` interface

An `Ontology` object is a central point to manage all ontology individuals. The Ontology interface is a subinterface of Thing defining a variety of methods to create, remove and retrieve the ontology members of specific types (Things, Concepts and Relations).

3.1.5 Events and listeners

Changing of an ontology brings to arising of *an event*. A client application can track certain ontology events by setting the *event listeners* for an `Ontology` object, which will be notified about arising of events of specified type and execute certain tasks to handle these events.

The events mechanism is based on Java events framework (`java.util.EventObject` class and `java.util.EventListener` interface).

3.1.6 Exceptions

When the object model meets an illegal action, failure or in another situation, which may be considered as abnormal, it throws *an exception* . A client application can catch the exception to handle it in appropriate way. The exceptions mechanism is based on Java exceptions framework (`java.lang.Throwable` class hierarchy).

3.1.7 Ontology validation

The Ontology Model API includes a mechanism to check the ontology integrity. A client application is able to refer to the validation API to test upholding the constraints on separate statements, Things or among a whole ontology. The validation API report contains detailed information about integrity problems to client application could analyze and fix them.

3.2 API Reference Implementation

The SOFA API includes default implementation classes for Ontology Model interfaces. The set of those classes is known as the SOFA Reference Implementation. The design of API hides the implementation classes from client applications which interact with SOFA on an interface-level only. It allows to develop alternate SOFA interfaces implementations and change them in a transparent way. The SOFA Reference Implementation includes a singleton class which serves as a factory for creating new Ontology instances. The objects of ontology individuals are instantiated with the corresponded factory methods of the Ontology interface.

3.2.1 System metaontology

The SOFA Reference Implementation is built on the *system metaontology* which is an ontology of the SOFA Conceptual Ontology Model. The metaontology defines the concerns of the conceptual model and provides a metavocabulary for defining all custom SOFA ontologies. By another words, it describes the SOFA Ontology Model by terms of the same model and provides its representation as a special built-in SOFA Ontology instance. For example, every custom Thing is an instance of the system “Thing” concept, and every custom Concept or Relation are the instances of the system “Concept” or “Relation” concepts (which are the subconcepts of the “Thing”). The relationships in the conceptual model (e.g. “subconcept of”, “instance of”, “has

range" and so on) are specified by corresponding Relation members of the metaontology.

The system metaontology is immutable and it underlies the default implementations of the SOFA interfaces.

3.2.2 Interoperability of ontologies

Different Ontology instances may interoperate each other in terms of sharing the Concepts, Relations and making statements with the foreign Thing objects. Due to the singleton factory class for instantiating ontologies, all members of ontology instances created during the same session are visible for each other.

3.2.3 Built-in Semantic Web client

The mechanism of ontology interoperability is connected to the Semantic Web client functionality. When the Reference Implementation meets an unsatisfied link to an external ontology resource, it automatically creates new ontology instance and attempts to load and parse ontology content from an external source. The path to a source is determined using the ontology namespace URI and predefined URI aliases which associate the ontology namespaces with real locations of the ontology resources. A specific parsing method is selected depending on a content type of a resource and may be extended with custom parsers for some specific content types. This functionality depends also on the SOFA Serialization mechanism (see below).

3.3 Storage API

The Ontology Storage Model API provides an abstract model of a storage utility for the SOFA Ontology Model. It represents an interface of an abstract storage engine which is independent from a specific way of storing the data with a particular physical storage back-end. This is a responsibility of a *storage model implementation*, which knows how to interact with a specific storage system.

The SOFA implementation refers to the storage model interface to store and retrieve the data (sets of statements) of the ontology internals. The client applications usually should not appeal directly to the storage API passing over the SOFA ontology model.

When create new Ontology instance, a client application may point SOFA to a storage model implementation which will be used to store that instance. If no storage model has been set, the default in-memory storage is used.

3.3.1 Default in-memory storage implementation

The default storage model implementation is the *in-memory storage* . This is a minimalistic storage utility based on the `java.util.Collection` and `java.util.Map` interfaces family. This is very fast and small-footprint storage implementation and it can use the ontology serialization mechanism for long-term storage.

3.3.2 Other storage implementations

RDBMS persistent storage The main productional storage model implementation is a *persistent storage*. It is built using the JDBC (Java DataBase Connector) framework to store and retrieve the ontology content with relational database management systems (RDBMS). This approach enables a lots of approved database software for storing the SOFA ontology models and brings necessary characteristics of an enterprise quality storage – scalability, transaction safety and security.

Kowari server persistent storage The Kowari Metastore[8] is an Open Source, massively scalable, transaction-safe, purpose-built database for the storage and retrieval of metadata. The Kowari project has its own implementation of the SOFA storage model to store the SOFA onologies with Kowari in a client-server manner.

Interoperability storage The special class of storage model implementations are adapters to existing applications and information systems. The adapters interpret the ontology internals into the structures of the external data model and vice versa. It provides a transparent way to interact the ontology applications with these systems and provides an ontological representation and way of manipulate of their data.

3.4 Serialization API

The SOFA ontology model is independent from specific ontology representation forms, but it can be interpreted in terms of some ontology languages having expressional capabilities to describe that model. As the SOFA model is compatible with semantics of W3C Ontology Web Language (positioned as an industry standard of ontology representation), the model can be entirely represented using this language syntax. Also it is rather true for DAML+OIL (the predecessor of OWL and still the most popular ontology definition language). Other languages may lose some details of the SOFA ontology model. The Ontology Serialization package includes the modules providing serialization of the ontology model with specific languages and restoring it from a serialized form. The primary of these modules are:

OWL (Ontology Web Language serialization module:
provides guaranteed reversible serialization of the SOFA ontology model without any losses.

DAML+OIL serialization module: provides reversible serialization with high reliability.

RDF + RDF-Schema serialization module: provides reversible serialization of the main aspects of the SOFA ontology model.

N-Triple serialization module: provides guaranteed reversible serialization of the SOFA ontology model as a set of {`subject, predicate, object`} triples described by common N-Triple syntax.

Every serialization module implements a common interface, so it is possible to develop new serializer implementations for specific ontology languages and easily integrate them with SOFA.

About the SOFA project

The SOFA API is developed as an open-source volunteer project in the SemWebCentral projects family (`http://www.semwebcentral.org`). The API is available in source code form under the terms of Lesser GNU Public License (LGPL).

Further information regarding to the SOFA project may be obtained from the following sources:

The SOFA web-site: `http://sofa.semanticweb.org`

The SOFA project homepage on SemWebCentral:
`http://projects.semwebcentral.org/projects/sofa`

References

[1] T. R. Gruber
Toward principles for the design of ontologies used for knowledge sharing.
Presented at the Padua workshop on Formal Ontology, March 1993, to appear in an edited collection by Nicola Guarino.
`http://ksl-web.stanford.edu/KSL_Abstracts/KSL-93-04.html`

[2] *W3C Semantic Web Activity*
`http://www.w3.org/2001/sw/`

[3] *Resource Description Framework (RDF): Concepts and Abstract Syntax*
Graham Klyne and Jeremy J. Carroll, Editors, W3C Recommendation, 10 February 2004
`http://www.w3.org/TR/rdf-concepts/`

[4] *RDF Vocabulary Description Language 1.0: RDF Schema*
Dan Brickley and R. V. Guha, Editors, W3C Recommendation, 10 February 2004
`http://www.w3.org/TR/rdf-schema/`

[5] *OWL Web Ontology Language Reference*
Mike Dean and Guus Schreiber, Editors, W3C Recommendation, 10 February 2004
`http://www.w3.org/TR/owl-ref/`

[6] *Jena Semantic Web Toolkit*
`http://www.hpl.hp.com/semweb/jena.html`

[7] *The Kazuki Project*
`http://kazuki.projects.semwebcentral.org`

[8] *Kowari Metastore*
`http://www.kowari.org`