

à gogo: Automatic Generation of Ontology APIs

F. Silva Parreiras¹, C. Saathoff¹, T. Walter¹, T. Franz¹, and S. Staab¹

ISWeb, University of Koblenz-Landau, Universitätsst. 1, Koblenz 56070, Germany
{parreiras, saathoff, walter, franz, staab}@uni-koblenz.de

Abstract. When programming APIs of upper level or core ontologies including many instances of Ontology design patterns, developers of semantic web applications usually have to handle complex mappings between Ontology design patterns and object oriented representations. We propose a domain specific language to tackle these mappings in a platform independent way - *agogo*.

1 Introduction

Upper level ontologies and core ontologies are usually complex since they comprise many occurrences of a variety of Ontology Design Patterns (ODPs). They usually require dedicated Application Program Interfaces (APIs) instead of generic solutions like RDF or OWL APIs.

When developing such dedicated APIs, developers of semantic web applications commonly face the challenge of mapping descriptions of complex relations or entities (like an event decomposition or a conversation among several participants) to object oriented representations thereof. As these complex structures are not represented by a single instance of a class but by ODPs involving a number of connected (linked) instances — e.g. decomposition criteria, descriptions of parts and wholes and more — the task of implementing correct data manipulation functionality becomes complex, too.

In current approaches like So(m)mer or Elmo, these mappings are handled by annotations stored as plain text on API source code, which leads to problems in reuse and maintenance.

We introduce a domain specific language (DSL), *agogo*¹, that provides an environment for API developers to handle complex structures defined in ontologies, allowing for defining and reusing complex ODPs. Moreover, *agogo* automates the development of Ontology APIs by generating code.

2 *agogo*

agogo is a model driven approach for automatically generating Ontology APIs on demand. It comprises a DSL and model transformations into code. The DSL captures domain concepts necessary to map ontologies onto OO representations and the model transformations allows for code generation.

The main features of *agogo* are high abstraction level, portability, reusability, maintainability and testability, described next.

¹ <http://isweb.uni-koblenz.de/Research/agogo>

High Abstraction Level. The definition of domain concepts in a metamodel allows API developers to work exclusively with constructs relevant to the domain. The following is a list of *agogo* key domains concepts:

Mappings. Associations between OO classes and Ontology classes are defined by the construct `map`. The construct `map` may be also used to link OO classes to patterns.

Patterns. When an OO class does not correspond directly to a single Ontology class but to an occurrence of an ODP, patterns may be described once using the `pattern` construct and reuse on every occurrence of such pattern. It is possible to define patterns for classes, properties and operations.

Operations. CRUD operations (Create, Read, Update and Delete) are defined in Ontology APIs to allow systems to manipulate Ontology classes. With *agogo*, operations and patterns are defined in a platform independent way by using SPARQL-like syntax.

Library. Patterns for classes, properties and operations may be group into libraries and made available for other packages or API specifications.

Portability. Ontology APIs are described once with platform independent models (PIM). These models are input into model transformations to generate code of APIs for different platforms. Figure 1 illustrates this model driven approach. PIM models are transformed into annotated Java code. The annotations describe SPARQL queries which implement the methods and use current solutions like SO(M)MER (Semantic Object Metadata Mapper) and Sesame.

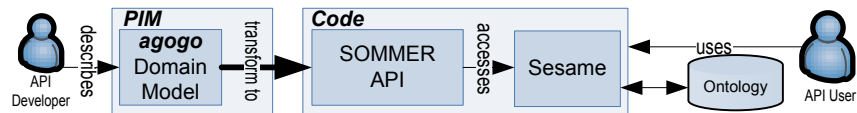


Fig. 1. Architecture of *agogo* approach.

Reusability Patterns are reused by many mappings. Moreover, libraries are reused to generate derived APIs. For example, a API developer may want to have different Ontology APIs of the same Ontology according to its complexity (lite and full).

Maintainability. The *agogo* metamodel allows for defining concepts in a structured way, improving maintainability. For example, elements of the Ontology API model are maintained as single units instead of being stored in annotations.

Testability. Definition of constraints on concepts in the *agogo* metamodel provides compile-time checking, i.e., it enables API developers to validate API specifications against these constraints and minimizing errors at runtime. Moreover, the syntax checker prevent syntax error from occurring at runtime as by existing approaches.