Formalizing the OPAL eBusiness ontology design patterns with OWL[1]

Fulvio D'Antonio, Michele Missikoff, Francesco Taglino

LEKS - Laboratory for Enterprise Knowledge and Systems
IASI - CNR, Rome, Italy

{missikoff, dantonio, taglino}@iasi.cnr.it

Abstract. Domain ontology building is one of the most critical activities required in Semantic Web applications. The task must be performed by domain experts, who do not (generally) have the background of a knowledge engineer. To ease this task, Ontology Management Systems (such as Kaon, Protégé, OntoEdit, Athos) are developing user friendly interfaces. However the problem is mainly of a cognitive nature. Difficulties comes from the fact that the existing ontology languages: (i) are hard to be used by unskilled people, (ii) have very basic constructs (e.g., class, property), (iii) are not domain specific, i.e., they are conceived for very diverse contexts (e.g., from medical sector to high energy physics). OPAL (Object, Process, Actor modelling Language) aims at supporting business experts who need to build an ontology by providing a limited number of high level conceptual templates.

Keywords: Ontology, Ontology representation languages, Axiomatic Semantics, Formal Languages, Design Patterns

[1] This work has been partially supported by the European FP6-IST Project Athena, contract number: 507849

1. Introduction

Ontology building is one of the most critical activities required in Semantic Web applications. The task must be performed by domain experts, who do not (generally) have the knowledge engineer expertise required to build an ontology. To ease this task, ontology management systems (such as Kaon, Protégé, OntoEdit, Athos) propose user friendly interfaces. However the problem is mainly of a cognitive nature and a GUI can only solve part of it. Difficulties come from the fact that the existing ontology languages: (i) are hard to be used by unskilled people, (ii) have very basic constructs (e.g., class, property), (iii) are not domain specific, i.e., they are conceived for very diverse contexts (e.g., from medical sector to high energy physics), showing hence little eBusiness specificity.

OPAL (Object, Process, Actor modelling Language) is an ontology modelling framework aimed at supporting business experts in building an ontology. To this end, OPAL provides a limited number of high level conceptual templates, conceived along the lines of software design patterns having in mind the business world. OPAL design patterns are formally defined by using OWL, allowing for a compatibility with this popular ontology language. OPAL is at the basis of the Athos OMS developed within the Athena Integrated Project[2]. It has been tested and validated in several national and international projects and applications, showing its effectiveness and high acceptance among business experts.

1.1 Learning from software design patterns

Software development is a challenging discipline that has significantly evolved in the last decades. To reduce cost and time of software development, the Software Engineering (SE) research has proposed several methods and techniques that cover the whole software lifecycle, from analysis and design to coding and testing. Ontology development is a hard discipline as well, showing a complexity that is comparable with software design. Therefore, we explored various SE solutions, to check their adaptability to the ontology domain.

In a previous paper [2] we addressed the adoption of a systematic methodology for ontology building, derived from the RUP, referred to as UPON (Unified Process for Ontology building). UPON includes the possibility of reusing existing resources (from lexicons and glossaries to thesauri and pre-existing ontologies) and a step-wise refinement of the developed ontology. In this paper we intend to address another important approach, based on design patterns.

In this paper we address the design patterns. Design patterns in software design have been proposed more than a decade ago by a group also known as the *Gang of Four* (GOF) [3] with the aim of supporting the design phase of object-oriented software development. A design pattern is a named, repeatable conceptual structure that can be used by a software designer to solve an algorithmic problem.

[2] http://www.athena-ip.org

It is independent from a specific programming language, since it addresses the conceptual intricacy of a problem rather than the coding difficulties, but assumes an object-oriented programming context. In this paper we propose a set of ontology design patterns, referred to as OPAL (Object, Process, Actor modelling Language.), to be used in modelling a business ontology.

1.2 Ontology Modelling and Domain Adequacy

In proposing a set of ontology design patterns, we introduce also the notion of *Domain Adequacy*. A language (a representation paradigm) is *adequate* for a given domain if it adopts specific jargon, constructs, and locutions that characterise the communication of experts in that domain. From physicians to computer engineers, but also in very different areas such as sailors or golf players, the members or such communities have developed their own specific language. The idea of having *domain specific languages* to develop software applications is an emerging research area.

Today, OWL [4] appears to be the most popular language for ontology modelling. OWL is an ontology language rooted in specific *Description Logics* [5] that exhibits a compromise between expressive power and tractability of the associated reasoning procedures. OWL is a general purpose language that can be adopted for building ontologies in very different domains; however its generality does not match friendliness and usability. Therefore, when building an ontology, domain experts require a substantial help from ontology engineers. Our objective, in proposing the OPAL representation framework and design patterns, is to lay the basis for a methodology that can be directly used (when supported by a suitable tool) by business people when building an ontology.

In defining the OPAL design patterns, we analysed the emerging standards and methods in enterprise and business modelling identifying a few primary modelling notions, particularly suited to model a business scenario: *Business Object, Business Process, and Business Actor*. Then, we added a number of complementary modelling notions, such as *Business Event, Business Message, and Business Goal*, to enrich and complete a business ontology (please note that in the sequel, for sake of conciseness, we will drop the term "Business".). In our approach, we propose *ontology design patterns* (ODP) that correspond to the above modelling notions. Then an ODP is implemented by a template and ontology building essentially consists in filling a number of OPAL templates. Athos is the OMS, developed within the Athena European Integrated Project, that supports ontology building based on OPAL ontology design patterns. Athos has been already experimented in several business applications with encouraging results.

The rest of the paper is organised as follows. In the next section we briefly present some of the main ontology design pattern approaches proposed in the literature aimed at supporting ontology building. In Section 3 the OPAL templates are illustrated, with the help of a running example in the eProcurement area. Section 4 provides a formal account of OPAL, by presenting its axiomatic semantics based on OWL. Finally, in Section 5, conclusions and future research lines are reported.

2. Related work

Ontology design patterns represent a promising research line with a number of interesting proposals. Here we also consider, more in general, ontology representation framework, that propose modeling notions that can be used for defining ODP. However none of the illustrated proposals are specifically aimed at business ontologies.

The Semantic Web Best Practices and Deployment Working Group (SWBPDWD) has addressed the issue of ontology building by providing engineering guidelines, ontology/vocabulary repositories and patterns to solve specific problems (such as representing n-ary relationships or "part of" representation). Such patterns are of a general nature and are not specifically conceived for the eBusiness sector.

The General Use Cases (GUC) proposed in [6] have a formal counterpart called Conceptual Ontology Design Patterns CODeP. A CODeP is considered as a fragment of either a "foundational" or a "core" ontology (that is called its reference ontology); in [6] are also presented some design patterns drawn from the DOLCE foundational ontology.

The Resource Event Agent ontology framework (REA) can be considered a set of ontology design patterns derived from McCarthy's work [7] with strong roots in accounting and economics.

In [8] is addressed the problem of creating (or extracting) ontology design patterns from corpora of ontologies in the business domain. This work has different objectives with respect to OPAL. As the author states, enterprise ontologies are so far quite scarce and often sparsely documented, and therefore most of the work in the paper is devoted to the extraction of design patterns from enterprise databases.

Concerning different application domains, the Gene Ontology Next Generation (GONG)[3] project aims to make use of ODP for easing the migration of biological ontologies to formal languages, like OWL, and for the maintenance of large biological ontologies. They propose Ontology Design Patterns (ODPs) that, as Software Design Patterns (SDPs), are formalized, documented, efficient solutions to frequent modeling problems.

3. The OPAL ontology design patterns

In this section, we present the OPAL templates conceived to implement the ODP corresponding to the business concept categories.

An ODP is structured following the traditional Frame-Slot-Facet modeling paradigm. In particular, there is a design pattern (template) for each concept category (referred to as *kind*, in OPAL.)

OPAL templates are divided into: *primary* and *complementary*. The former represent the core of a business ontology, the latter are used to refine the former.

[3] http://www.gong.manchester.ac.uk/htlatex/design.html

3.1 The OPAL templates

The primary templates: *Object*, *Process* and *Actor*, that have their initials included in the acronym, are at the basis of the OPAL modeling framework and provide the backbone of an OPAL ontology.

BusinessActor	It is aimed at modelling any relevant entity of the domain that is able to activate or perform a process (e.g. *Buyer*, *Supplier*). An Actor can be a human being like an *Employee*, but also any other kind of entity like an *Enterprise* or a *Computer System*.
BusinessObject	It is aimed at modelling a passive entity, on which a process operates, typically to modify its state. It is seen as an information object, representative of a material or an immaterial entity in the real world (e.g. Product)
BusinessProcess	It is aimed at modeling an activity that is performed by an Actor to achieve one or more given goals. (e.g. *Issuing Purchase Order*, *Sending Request for Quotation*)

Table 1: OPAL Primary templates

OPAL includes other conceptual categories typical of the eBusiness domain, such as, *Message, Business Object Document, Business State, Goal, Event, Rule* and *Decision*. These categories have been identified having considered several languages and methodologies for business modeling. For instance, messages are defined along the line of FIPA[4] and the speech act theory. For sake of space, we are not going to elaborate on complementary templates, except attributes.

In OPAL, attributes are distinguished as:

- Complex Attribute (CA) structured attributes, such as *Address*
- Atomic Attribute (AA), such as *Street Name.*

Essentially, a *Complex Attribute* is defined as an aggregation of lower level CAs and/or AAs.

3.2 OPAL templates organization

An OPAL templates is characterized by three sections, as summarized in the following table.

Identification Section	contains traditional metadata, such as the concept label, description, and other information (such as creation date, author) with an administrative flavor;
Structural Section	that contains the slots corresponding to the attributes (simple or complex) to be instantiated in the corresponding object; it contains also the semantic relations (such as ISA, decomposition, etc.) with other concepts;
Specific Section	that contains information and references to other entities that play a specific role (domain specific) in the correct definition of the concept (e.g,, executed processes in the Actor template).

Table 2: OPAL templates' sections

[4] www.fipa.org

The first two sections are identical for the three primary templates, while the third section is designed specifically for each template, with the aim to represent domain specific links and dependencies.

The Identification Section reports typical metadata, such as: concept identifier, description, synonyms, author (who introduced the concept in the ontology.) Then, the Structural Section provides the conceptual structure of an entity.

Structural Section

This section allows the modeler to express all the constructs of OWL, but in a user friendly manner, simply by form filling. In addition, there is a "part of" construct. The modelling notions offered by OPAL are the following:

- ISA (generalization) relation among concepts. E.g., *Invoice* ***ISA*** *Accounting Document*;
- Predication: relating attributes to a concept. E.g., *Invoice* ***Pred*** *Date*, *Amount*, *Recipient*. Additional constraints can be specified to the Predication:
    - *Cardinality constraints*: allow to specify the cardinality of attributes and relations concerning the concept;
    - *Explicit constraints*: axioms expressed by using the OCL [9] language[5];
    - *Property characteristics*: Functional, InverseFunctional, Symmetric, Transitive;
- *Decomposition*: part-of relationship among concepts. E.g., a *Department* **PartOf** *Enterprise*;
- *Relatedness*: domain specific relationships (*named or unnamed*) among concepts. E.g., *Customer* ***buys*** *Product* (named); *Invoice* ***RelTo*** *Customer* (unnamed).

Further details concerning *attributes*, *relations* and *hierarchies* will be given in the next sections.

An OPAL template is completed by the *Specific Section*, which is different for each concept category. The *Specific Section* gathers a number of domain-oriented axioms, aimed at capturing additional semantics. In the following, the Specific Section for the Object and the Actor kind are described.

Object Specific Section

An Object has a *Specific Section* that reports the following information:

- *GeneratedBy, UpdatedBy, UsedBy, ArchivedBy*: indicating the Processes that can create, manipulate, archive the Object;
- *ObjectOwner:* the Actors that have the responsibility of the whole

[5] Please note that in the first implementation of Athos, OCL constraints are not yet deployed. Hence, they will not be elaborated here.

lifecycle of the object;

- *States*, labelled boolean expressions over the Object attributes or those of related concepts;
- *Invariants:* specific constraints that must be always satisfied by the Object instances.

The last two items are currently treated informally. Their formal representation needs full adoption of OCL.

The Object Specific Section, instantiated for a Purchase Order example, is:

PurchaseOrder – Object Specific Section	
GeneratedBy	IssuingPO
UpdatedBy	RequestingOrderStatus
UsedBy	ProcessingPurchaseOrder
ArchivedBy	ClosingPO
ObjectOwner	BuyerPurchasingFU
States	{Issued, Confirmed, Open, Modified, Hold, Blocked, Cancelled, Fulfilled, Paid, Archived}
Invariants	the PO date follows the corresponding Quotation date and precedes the Invoice date.

Table 3: *Object*'s Specific section filled for "PurchaseOrder"

Actor Specific Section

The relations specified in the *Specific Section* of the Actor template are:

- *Goals:* objectives that must be accomplished by the Actor, in the form of an OCL expression (e.g., *salaries should be less that 60% of department budget*);
- *Skills:* indicating the actions that the Actor is able to perform or monitor (i.e., list of processes and/or operations);
- *Responsibilities:* the processes in which the Actor is involved, in achieving a Goal (as above), with his/her/its respective role (i.e., performer, controller, stakeholder, supporter), and the Objects he/she/it can manage;
- *Collaborations:* the other actors involved in the performed activities.

PurchasingFunctionalUnit – Actor Specific Section	
Goals	BetterPerformanceGoodsPurchasing
Skills	IssuingRFQ, EvaluatingRFQ, Issuing PO
Responsabilities	(SendingRFQ,{Enabler, Performer}, {Request For Quotation-Doc}), (IssuingPO ,{Enabler,Performer}), {Quotation-Doc, PurchaseOrder-Doc})
Cooperation	BAdministrativeFU, BGenericFU, BReceivingFU

Table 4: *Actor*'s specific section for the concept "PurchasingFunctionalUnit"

4. OPAL axiomatic semantics

In this section we describe the formal semantics of OPAL ontology design patterns. This could be done following two approaches:

- model theoretic semantics.
- axiomatic semantics

"A model-theoretic semantics for a language assumes that the language refers to a 'world', and describes the minimal conditions that a world must satisfy in order to assign an appropriate meaning for every expression in the language"[10].
An axiomatic semantics for a language, instead, specifies "a mapping of a set of descriptions [of that] language into a logical theory expressed in first-order predicate calculus." The basic idea is that "the logical theory produced by the mapping of a set of such descriptions is logically equivalent to the intended meaning of that set of descriptions" [11]. Axiomatic semantics have been given for the Resource Description Framework (RDF) [12], RDF Schema (RDF-S) [13], and DAML+OIL [14].

We prefer to rely on the second option and we selected OWL as the target of our mapping, to benefit of its strong formal foundation. However, there are three possible sorts of OWL, OWL-Lite/DL/Full, with increasing expressing power. We will analyse this issue to choose which of them is the best suited to our purpose.

4.1 Abstraction levels

The first issue to be addressed concerns the fact that a design pattern can be considered a meta-concept. In fact a concept (e.g., Researcher) is obtained by instantiating a design pattern of the kind Actor. The first option is to follow this vision but this requires the availability of classes and meta-classes, as offered by OWL-Full. A second option is to consider a concept as a specialization of a template. For instance saying that *Researcher ISA Actor* In this case we fall into the OWL-DL expressivity. The two options correspond to two different OWL statement set, as reported below.

We will present OWL expressions using N3 syntax [15] assuming the usual prefixes: *rdf*, *owl*, *xsd*, plus *opal* and *ex* for the specific example.

The first option can be exemplified by the following set of OWL statements:

- **opal:Actor a owl:Class**
- **ex:Reasercher a opal:Actor**
- **ex:MicheleMissikoff a ex:Researcher**

Here we define *opal:Actor* to be an *owl:Class* (so we instantiate *owl:Class*), the we define *Researcher* to be an *opal:Actor* (second instantiation) and, at last, we reach the level of ground instance by declaring *MicheleMissikoff* to be an instance of *Researcher* (third instantiation).

The second option is realized through the following statements:

- **opal:Actor a owl:Class**
- **ex:Reasearcher rdfs:subClassOf opal:Actor**
- **ex:MicheleMissikoff a ex:Researcher**

In this formulation, the second statement says that *Researcher* is a subclass of *opal:Actor*. In essence, we rely on *rdfs:subClassOf* for indicating both *opal:ISA* (the actual specialization of concepts) and *opal:Kind* (the fact that a concept is of a given *kind*, e.g., Actor.

Although the first type of formalization is the most faithful to the OPAL original conception, we decided to use the second to take the advantage of DL reasoners to perform inference services on the result of the translation. Therefore in the rest of this formalization we will perform our mapping onto OWL-DL language.

The OPAL axiomatic semantics will be given therefore by means of a set of OWL-DL statements (except for two specific cases described below).

4.2 OPAL Axiomatic Semantics

For every meta-concept *k* in *OPALKinds = {opal:Actor, opal:Object, ...}* we introduce a declaration of the type ***k a owl:Class*** such as:
opal:Actor a owl:Class
opal:Object a owl:Class
…(etc.)

Every relation kind in OPAL is mapped onto a declaration of an OWL ObjectProperty:

opal:RELATEDNESS a owl:ObjectProperty
opal:DECOMPOSITION a owl:ObjectProperty
opal:PREDICATION a owl:ObjectProperty

OPAL predications relate OPAL Concepts with Attributes; from this perspective this would suggest the use of owl:Datatype properties. However there are in OPAL two kinds of attributes: *Atomic* and *Complex*, and the second kind will be translated into an owl:Class; so for sake of symmetry we transform also Atomic Attributes into classes.

To represent *opal:ISA* (that doesn't behave exactly as rdfs:subClassOf being *opal:ISA* cycles forbidden) we could declare the statement *opal:ISA rfds:subPropertyOf rdfs:subClassOf*; however this kind of declaration is only allowed in OWL-Full. We will use the *rdfs:subClassOf* property to approximate the semantics of *opal:ISA*. However we need to further restrain its interpretation; this will be realized by means of additional axioms.

Finally, *opal:kind* instead will be realized using *rdfs:subClassOf* or *rdf:type* depending wether we are creating ground instances or instantiating a metaconcept.

4.3 OPAL inherent constraints

In this section we present the formalization of OPAL inherent constraints by means of OWL-DL expressions and (as mentioned above), in two cases, of FOL axioms. OPAL inherent constraints ensure a guideline for the ontology developer, improving the verifiable quality of an ontology.

C1) Concept categories (*kind*) are disjoint:
These can be realized in OWL by means of the **owl:disjointWith** construct; being it a binary construct we have to express this fact by means of a list of OWL disjointness statements;

- *opal:Actor owl:disjointWith opal:Object*
- *opal:Actor owl:disjointWith opal:Process*
- ...

C2) $(c_i,c_h) \in$ ISA $\Rightarrow$ kind(c_i) = kind(c_h)

This constraint can be checked by DL-reasoners exploiting the transitivity of rdfs:subClassOf (assuming that both of **opal:ISA** and **opal:Kind** are realized in **OWL-DL** by means of **rdfs:subClassOf);** it is not possible for c_i to have two **different kinds** because the OPAL meta-concepts corresponding OWL Classes are **pairwise disjoint** (Constraint *C1*).

C3) The ISA graph is acyclic. i.e there are no sequences $c_1..c_n$ such that (c_1,c_2), …,(c_{n-1},c_n) $\in$ISA and $c_1=c_n$.

Realizable in OWL-DL plus the following FOL Rule:

c_i rdfs:subClassOf C_h $\wedge$ C_h rdfs:subClassOf C_i $\Rightarrow$
owl:Thing rdfs:subClassOf owl:nil

(or putting in the right side of the rule any other possibile contraddiction)

C4) $(c_i,c_h) \in$ Pr $\wedge$ kind(c_i) $\in$ {O, A, P} $\wedge$ kind(c_h) $\in$ {CA, AA}
C5) $(c_i,c_h) \in$ D and kind(c_i) $\in$ {A, O} $\Rightarrow$kind(c_i) = kind(c_h)
C6) $(c_i,c_h) \in$ D and kind(c_i) = CA $\Rightarrow$ kind(ch) $\in${AA, CA}
C7) $(c_i,c_h) \in$ D and kind(c_i) = P $\Rightarrow$ kind(ch) $\in$ {P, Op}
C8) $(c_i,c_h) \in$ Re and kind(c_i) = {A, O, P} $\Rightarrow$ kind(c_h) $\in${Ar, O, P}
C9) $(c_i,c_h) \in$ Re and kind(c_i) = {CA, AA} $\Rightarrow$ kind(c_i) = kind(c_h)

These constraints are obtained by creating *owl:Restriction*s on the property for the appropriate domains for example *C4* is realized by:

```
Actor rdfs:subClassOf [
    a owl:Restriction;
    owl:onProperty Predication;
    owl:allValuesFrom [owl:UnionOf (CA,AA) ] ]
```

and the others can be realized in a similar way.

C10) Decomposition is irreflexive, antisymmetric, transitive. To state the transitivity we use the following expression:
opal:Decomposition a owl:TransitiveProperty

The other two properties use two FOL rules:

- Anti-reflection: c_i *opal:Decomposition* $c_i \Rightarrow \perp$
- Anti-symmetry: c_i *opal:Decomposition ch* $\wedge$ c_h *opal:Decomposition* $c_i \Rightarrow \perp$

5 Conclusions and future research lines

In this paper we presented a set of modeling constructs that represent the OPAL design patterns, conceived to support the building of business ontologies. There is a design pattern for each of the relevant concept categories (referred to as *kinds* in OPAL) useful to model a business scenario. An *ontology design pattern* (ODP) is an abstract structure. In order to make it usable in a tool for ontology management, we designed a number of templates, i.e., concrete structures that can be represented as forms on a screen and proposed to the end-user. With this approach, ontology building consists in filling a number of predefined screen forms.

Considering the emergence of the W3C ontology language OWL, we conceived the semantics of OPAL in order to keep it aligned with the former. To this end, in the paper we presented the axiomatic semantics of OPAL that proves the compliance of this ontology representation method with OWL-DL (except two specific situations that have been separately dealt with.)

OPAL is the foundation of the ontology management system Athos, developed within the Athena European Integrate Project. The use of the proposed axiomatic semantics has been also useful for the implementation of the import/export functions of Athos.

Another important feature of OPAL is the axiomatization of the implicit constraints that are used in Athos to provide guidance to the user and to guarantee a better quality for the produced ontology.

The research activities in OPAL continue on the one hand with experimentation on new business applications, on the other with a refinement of the design patterns and their axiomatization. Finally, we intend to explore the possibility of seamlessly integrate the expressive power of OCL in the method.

References

[1] Rational Software, *Rational Unified Process.* version 2002.05.00
[2] A. De Nicola, M. Missikoff, R. Navigli. *A Proposal for a Unified Process for Ontology Building: UPON.* In Proceedings of the 16th International Conference on Database and Expert Systems Applications (DEXA 2005), pages 655-664. Springer

[3] E. Gamma, R. Helm, et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Published by Adison Wesley Professional. Series: Addison-Wesley Professional Computing Series.
[4] D. L. McGuinness, F. van Harmelen. *OWL Web Ontology Language Overview*. W3C Recommendation 10 February 2004. http://www.w3.org/TR/2004/REC-owl-features-20040210/.
[5] D. Calvanese, D. McGuinness, et al. *The Description Logic Handbook - Theory, Implementation and Applications*. Edited by Franz Baader. January 2003.
[6] A. Gangemi. *Ontology design patterns for semantic web content*. Lecture notes in computer science. ISSN0302-9743.
[7] W.E. McCarthy. *The REA Accounting Model: A Generalized Framework for Accounting Systems in a Shared Data Environment*. The Accounting Review 57(3), 554-578.
[8] E. Blomqvist, *Patterns in Ontology Engineering: Classification of Ontology Patterns*, In Proc of the ICEIS 2005.
[9] Mark Richters , Martin Gogolla, OCL: Syntax, Semantics, and Tools, Object Modeling with the OCL, The Rationale behind the Object Constraint Language, p.42-68, January 2002
[10] W3C RDF Model Theory W3C Working Draft Editor, P Hayes 14 February 2002, http://www.w3.org/TR/rdf-mt/
[11] R.Fikes and D. McGuinness. An Axiomatic Semantics for RDF, RDF Schema, and DAML+OIL, KSL Technical Report KSL-01-01, 2001. http://www.ksl.Stanford.EDU/people/dlm/daml-semantics/abstract-axiomatic-semantics.html
[12] G. Klyne, J. J. Carroll. *Resource Description Framework (RDF): Concepts and Abstract Syntax*. W3C Recommendation 10 February 2004. http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/
[13] D. Brickley, R.V. Guha. *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Recommendation 10 February 2004. http://www.w3.org/TR/rdf-schema/
[14] D. Connolly, F. van Harmelen, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. *DAML+OIL (March 2001) Reference Description*. December. 2001.
[15] T. Berners-Lee. *Notation 3*. Technical report, World Wide Web Consortium (W3C), 1998. Design Note, http://www.w3.org/DesignIssues/Notation3.