A METHODOLOGY FOR DEVELOPING ONTOLOGIES USING THE ONTOLOGY WEB LANGUAGE (OWL)

Magdi N. Kamel, Ann Y. Lee and Edward C. Powers
Department of Information Sciences, Naval Postgraduate School, 589 Dyer Rd, Monterey, California, 93943, USA

Keywords: Ontologies, Ontology Development Methodologies, Ontology Web Language (OWL), Semantic Web.

Abstract: It is generally agreed upon that ontologies are the knowledge representation component of the Semantic Web. There is a growing need for developing ontologies in different disciplines as means for sharing a common understanding of the structure of information in a domain among both people and machines. This paper describes a seven-step methodology for developing ontologies using the Ontology Web Language (OWL) based on related approaches for software and ontology development. As with contemporary software development methodologies, the steps of the proposed approach are applied iteratively and in a cyclical fashion in order to accurately capture the domain knowledge.

1 INTRODUCTION

It is generally agreed upon that ontologies are the knowledge representation component of the Semantic Web (Berners-Lee 2001). Although the realization of the Semantic Web is still a distant goal, there is a growing demand for ontologies to be incorporated into current technologies. Many disciplines are seeing the immense value of ontologies as a way to codify a common set of information or knowledge to be shared across multiple applications. Ontologies provide users with a consistent and agreed-upon knowledge base that both humans and machines can process (Musen 1992; Gruber 1993).

While no ontology can model all the nuances of any domain area, it is possible and valuable to abstract the major concepts and how they relate to one another. Having a valid knowledge representation system that is widely shared saves tremendous amount of effort for those who do not have access to subject matter experts (SMEs). Likewise, SMEs are motivated to provide users and applications with basic domain knowledge through the development of ontologies, thus providing users with consistent sets of information that they can maintain and manage.

Unfortunately, there is no clear understanding on how to build ontologies, and good methodologies for developing ontologies are greatly needed. A number of suggestions for such methodologies have emerged as people reflect on their experience of building ontologies. Such suggestions include the experiences in the development of TOVE (Toronto Virtual Enterprise) (Grüninger and Fox 1995), the Enterprise Ontology (Uschold 1996), Methontology (Gomez-Perez et al 1996), and KBSI IDEF5 (KBSI 1994). While these approaches share some common elements, they differ in their emphasis on different aspects of ontology development.

This paper describes a methodology for developing ontologies based on related approaches for software and ontology development. It differs from previous approaches in that it is specifically designed for developing ontologies in OWL DL, an Ontology Web Language based on description logic. The process of the methodology involves modeling the real world concepts and their relationships into OWL classes, properties and instances.

Although, building ontologies requires a robust grasp of the language used to build it, there are many ontology development environment that provide graphical user interfaces that hide the complexity of the language syntax from the ontology developer. While we believe that it is important to understand the OWL constructs for building a valid OWL ontology, the purpose of this paper is to understand the process and methodology of building an ontology rather than the syntax of the language.

The paper is organized as follows. Section 2 describes ontologies, why they are important as a

N. Kamel M., Y. Lee A. and C. Powers E. (2007).
A METHODOLOGY FOR DEVELOPING ONTOLOGIES USING THE ONTOLOGY WEB LANGUAGE (OWL).
In *Proceedings of the Ninth International Conference on Enterprise Information Systems - SAIC*, pages 261-268
DOI: 10.5220/0002352302610268

knowledge representation system, and presents a brief overview of the Web Ontology Language (OWL). Section 3 details a proposed seven-step development methodology using Geography as the domain of interest. Finally Section 4 concludes the paper with a summary and directions for future research.

2 ONTOLOGIES AND THE WEB ONTOLOGY LANGUAGE (OWL)

Ontologies are used to capture knowledge and semantics about a domain of interest. They describe the concepts in the domain, their properties, and the relationships that exist between those concepts. Ontologies derive their value from their ability to share knowledge across information systems. An ontology can take on various forms. It may be as basic as a simple catalog, a finite list of terminology, and as semantically sophisticated as logical abstraction for disjointed and inverse relationships.

There are many goals for developing ontologies, the most important of which is the ability to share a common understanding of the structure of information in a domain among both people and machines (Noy and McGuinness 2001). By using an ontology that creates a common language amongst disparate systems, it becomes possible to share the same set of terms and concepts. This also allows software agents to aggregate and extract information from other systems and use them appropriately to answer queries. Other goals for developing ontologies include the ability to reuse domain knowledge, making domain assumption explicit, separating domain knowledge from the operational knowledge, and analyzing domain knowledge.

As ontologies move from simple taxonomies to a structured knowledge base with properties and restrictions, their need for expressiveness grows (McGuiness 2002). At this end of the spectrum, a semantically rich language becomes imperative to represent the concepts and relationships of the domain. Furthermore, an inference engine (reasoner) can be used to verify consistency and completion. It checks for consistency and makes inference where it deems the relationships to be incomplete. These are crucial elements of a meaningful ontology because applications and systems rely on valid knowledge representation.

Different ontology languages provide different facilities. The most recent development in ontology languages is the Web Ontology Language (OWL) from the World Wide Web Consortium (W3C). Derived from DARPA's DAML+OIL, OWL is an extension of Resource Description Framework (RDF), a language for metadata interoperability (Brickley and Guha 1999). OWL extends RDF in order to facilitate richer inferences than RDF Schemas. OWL provides a vocabulary to create hierarchy of classes and use of class inheritance. OWL's extensions include semantics for cardinality, class and instance equality, relationship between classes, and property characteristics. Using the variety of constructs provided by OWL, users can build complex and useful ontologies.

There are three flavors of OWL, each with different degrees of expressiveness, namely OWL Lite, OWL DL (Description Logic), and OWL Full. OWL Lite is used mainly for simple classification hierarchy and constraints. OWL DL supports maximum expressiveness while maintaining decidability and computational completeness. OWL DL uses all of the OWL constructs with certain restrictions such as type separation (where a class, property and individuals share all of the same features). OWL Full is the most expressive version which guarantees syntactic freedom of RDF without computational completeness.

The example below, from the Geography Ontology, defines the `Continent` class using OWL. This simple definition states that the class called `Continent` belongs to a parent class called `Body_of_Land`. As specified by the namespaces, OWL uses RDF Schema and RDF constructs to point to the resource identifiers.

```
<owl:Class rdf:ID="Continent">
<rdfs:subClassOf
  rdf:resource="#Body_of_Land"/>
</owl:Class>
```

In addition to the taxonomic hierarchy of classes, OWL provides a rich set of semantics to describe the relationship between classes and between individuals.

3 A METHODOLOGY FOR DEVELOPING ONTOLOGIES USING OWL

While there are numerous papers on ontologies, there is little guidance on how to go about their development, particularly for OWL ontologies. In this section we present a methodology that adapts

existing approaches for software and ontology development to develop OWL ontologies.

The proposed development methodology consists of seven steps. Similar to the life cycle model for software development, these steps are applied iteratively and in a cyclical fashion. The steps of the proposed methodology are:

1. *Determine the scope and application of the ontology*
2. *List relevant concepts of the domain*
3. *Create the class hierarchy*
4. *Define properties*
5. *Describe classes using property restrictions and complex definitions*
6. *Classify ontology with a reasoning tool*
7. *Create individuals and fill property values*

Each of these steps will be discussed in some detail in the sections that follow. A Geography Ontology will be used to illustrate the steps of the methodology. The choice of the Geography domain is based on the fact it is a commonly understood domain and thus will help the reader understand the process of building an ontology.

3.1 Determine the Scope and Application of the Ontology

This crucial first step requires a clear understanding of the purpose and scope of the ontology to be developed. Often, the purpose of an ontology is two-folds. If an ontology is to represent the knowledge base of a particular domain or segment of a domain, it will potentially function to "answer" all general questions related to that domain. A second reason for developing an ontology is its use as knowledge representations in specific applications. For a given ontology, the requirements for these two goals may be conflicting. Therefore, the developer must compromise the demand for specificity and generality of scope in order to create a useful ontology. He must carefully manage the scope and depth to develop a realistic and coherent ontology that serves the purpose of the application.

The scope and purpose of the Geography Ontology is to define the basic physical and political geographies and represent the relationships between them for the purpose of using it with an Ontology Assisted Knowledge Discovery Application (OAKDA), which will access the ontology to provide meaningful context to tailor user web searches. The ontology represents the high-level understanding of geopolitics – the physical geographic characteristics existing within different types of political entities. We will use this example ontology in the sections that follow to demonstrate the development methodology of an OWL DL ontology.

3.2 List Relevant Concepts of the Domain

Once the scope is broadly defined, this step enumerates, in no particular order, the main concepts of the domain of interest. Although the final ontology may not necessarily include all the concepts defined during this phase, the developer should list as many relevant concepts as possible. At this point, one should not be concerned with overlapping concepts, the relationships between them, or their properties. The goal of this step is to create a comprehensive list of the concepts of the domain in preparation for the subsequent steps of development.

For the Geography example, the relevant concepts of the domain include the following: `ocean`, `lake`, `river`, `mountain`, `land`, `plains`, `valley`, `desert`, `tropics`, `climate`, `country`, `government`, `city`, `boundary`, `continent`, `language`, `ethnicity`, `latitude`, `longitude`, `archipelago`, `Mexico`, `South America`

While not every concept becomes a class, having a large pool of concepts relevant to the domain makes the hierarchy development easier. As in the requirements analysis for software development, the time and thought invested into the first two steps provide great benefits and rewards during the subsequent steps of the methodology.

3.3 Create the Class Hierarchy

This step creates a class hierarchy by specifying superclasses and subclasses. Superclasses and subclasses are related through an "is-a" relationship which indicates that a member of a subclass is also a member of the superclass.

Organizing the class hierarchy may be accomplished in several ways: top-down, bottom-up, or a combination approach (Noy and McGuiness 2001). The top-down approach starts with the most general set of concepts and works down to the subsequent levels of specialization. For example, `BodyOfLand` and `BodyOfWater` classes are identified as the highest level of the Geography hierarchy, and subsequent subclasses are added to these two classes. Thus, `Ocean`, `River`, and `Lake` are added as subclasses of `BodyOfWater`,

and Mountain, Desert, and Plains as subclasses of BodyOfLand.

The bottom-up approach starts with identifying the most specific classes, then grouping, and subsuming them under a parent class. For example, in the Geography Ontology, the developer may start with LandlockedCountry, IslandCountry, and BiCoastalCountry classes, which are then grouped as subclasses under the parent class of Country.

The combination approach develops the class hierarchy by defining the most salient terms of the ontology, adding successive classes at the different levels as appropriate. The advantage of the combination approach is that it allows the developer to start anywhere along the hierarchy and move up and down the hierarchy and add new classes as necessary. For example, in the Geography Ontology, BodyOfWater and BodyOfLand classes were initially defined as top level classes. In subsequent iterations, a PhysicalGeography class was defined and became a superclass for the BodyOfWater and BodyOfLand classes.

When grouping low-level concepts, developers should carefully differentiate between classes and their instances, known in OWL as individuals. The distinction between a class and an individual is not always clear and often depends on the purpose of the ontology. This means that a concept that is a class in a given ontology may be an individual in another. However, classes are generally "naturally occurring sets of things in a domain of discourse" while individuals correspond to real-world entities grouped under these classes (Noy and McGuiness 2001).

In the Geography ontology, PacificOcean is an instance of Ocean rather than its subclass since PacificOcean does not represent a group of entities but an actual entity itself. On the contrary, IslandCountry should be a subclass of Country rather than its instance since it represents a group of island countries, such as Ireland and Cuba.

The development of the class hierarchy of this step falls under the design phase of the software development cycle. As the ontology evolves, the developer will revisit this step and modifies the hierarchy as necessary. Additional requirements and knowledge acquired in the process refines the class taxonomy. In order to manage the constantly evolving ontology, detailed documentation and versioning is recommended.

It is important to note that in OWL DL, all classes are considered overlapping unless such separation or disjointness is made explicit. Specifying disjointness between classes requires an explicit specification using the OWL syntax owl:disjointWith. Only by defining a class as disjoint with others, the developer can assume class mutual exclusivity.

3.4 Define the Properties

Following the creation of the class hierarchy, this step specifies the class properties. A property represents the relationship between two individuals, or between an individual and a literal string value. Properties are derived from the characteristics of the defined classes. Similar to the method of specifying classes and subclasses, properties are defined by listing the characteristics of the defined classes and then relating them to their classes. A defined property may apply to more than one class.

For example, the characteristics of the Country class include border, population, capital, language, climate, rivers, lakes, mountain, government, ethnic groups, and others. Most of these characteristics relate to other classes within the ontology. A verb is usually added as a prefix to property to specify the relationship between the classes or between a class and a data string. For example, Country has a property of hasCapital to denote the relationship that it has with the City class. For the class characteristics that do not relate to another class, the property depicts the class's relationship to a data string value. The property hasPopulation describes the connection between the individuals of class Country and their population value. In this case, population is a numeric value that represents the number of people in a particular country.

OWL DL ontologies allow the specification of different types of class properties. They include inverse, transitive, symmetric, functional and inverse functional properties. Each of these properties consists of its unique OWL DL constructs, and is described briefly in the sections below. It is important for the developer to correctly identify the type of property and specify it in the ontology.

3.4.1 Inverse Properties

Properties having an opposite relationship to one another are known as inverse properties. For example, if the property hasCountry has the corresponding inverse property of hasCity, and if individual A has a hasCountry property value of individual B, then individual B hasCity property value of individual A. Specifically, if the individual

Venice, an instance of the City class, has the hasCountry property value of Italy, then the instance of Country Italy will automatically have the hasCity property value of Venice. An inverse property is denoted using the OWL syntax, owl:inverseOf, a subproperty of owl:ObjectProperty, to indicate an opposite relationship to the specified inverse property

3.4.2 Transitive Properties

A transitive property is commonly used to represent "part-whole" relationships. If transitive property P_T links individuals X and Y as well as individuals Y and Z, then it is inferred, by the rules of transitivity, that P_T relates X to Z. For example, in the Geography Ontology, the locatedIn is a transitive property that is specified for the individuals VaticanCity, Rome, and Italy. If VaticanCity is locatedIn Rome and Rome is locatedIn Italy, then by the rule of transitivity, VaticanCity is locatedIn Italy. While this implication is not explicitly stated in OWL, the inferred relationship is made explicit when the ontology is used to make reasoning decisions. Inference engines, such as RacerPro, read the OWL syntax and make the implied link as defined by the transitive property.

3.4.3 Symmetric Properties

Symmetric properties allow individuals to have a reciprocal or a bi-directional relationship. For example, in the Geography Ontology, if adjacentTo is defined as a symmetric property, and individual A is adjacentTo individual B, then individual B is also adjacentTo individual A. Specifically, if individual Mexico is adjacentTo United States, then it is inferred that United States is adjacentTo Mexico.

3.4.4 Functional and Inverse Functional Properties

A functional property indicates that, for a given individual, there can be at most one value associated with that individual along the property path. For a functional property P_F, individual X is associated with at most one property value of individual Y. However, if P_F links X with another value, say individual Z, then individual Y and individual Z are one and the same. For example, in the Geography Ontology, the property hasCapital is defined as a functional property. The individual UnitedStates is associated with two different hasCapital values, namely DistrictOfColumbia and WashingtonDC. However, given that definition of functional property, it must be inferred that DistrictOfColumbia and WashingtonDC are the same individual.

Similar to inverse property, inverse functional property denotes that inverse property is functional. Since functional property is restricted to one property value, the same is applied to the inverse functional property. More formally, if individual X relates to individual Z via inverse functional property P_{IF} and individual Y relates to Z via the same property, then it is assumed that individual X is the same as individual Y. For example, in the Geography ontology, if both DistrictOfColumbia and WashingtonDC have an inverse functional property isCapitalOf with United States, this implies that the inverse property hasCapital is functional and therefore DistrictOfColumbia and WashingtonDC are the same individual.

3.5 Describe Classes Using Property Restrictions and Complex Definitions

Properties are also used to restrict and define classes. In order to associate a property with a class, it must be used as part of a class description.

There are three types of class descriptions in OWL DL, namely *enumeration*, *property restriction*, and *complex class definition*. *Enumeration* describes a class by exhaustively listing all of its members or instances. Using the construct owl:oneOf, the class description consists of every individual that belongs to the class. *Property restrictions* describe the constraints on relationships that the individuals participate in for a given property. There are three types of property restrictions, *quantifier*, *hasValue*, and *cardinality restrictions*. A *quantifier restrictions* constrain the range value of the property when applied to the class definition. It can be either an existential restriction or a universal restriction. A *hasValue restriction* describes an anonymous class of individuals that are related to another specific individual along a specified property. *Cardinality restrictions* constrain the number of property values the class instance is allowed. *Complex class descriptions* are defined using simple class descriptions that are combined together using logical operators of intersection (AND), union (OR) and

complement (NOT). They represent advanced class logic of OWL DL.

Due to space limitation we will restrict our discussion to quantifier restrictions as well as a discussion of the related topics of open world assumption, and primitive and defined classes. For a comprehensive discussion of property restrictions and complex definitions, the reader is referred to the article by Horridge et al. (Horridge 2004).

3.5.1 Universal and Existential Restrictions

The existential restriction, denoted by ∃, states that individuals of the class being defined must have *at lease one* property relationship with a specified set of individuals of a class. In other words, if a property restriction for `ClassX` is $\exists\ Property_E\ ClassY$, then individuals of `ClassX` have at least one $Property_E$ relationship with the individuals of `ClassY`. With existential quantifiers, it is possible for individuals of `ClassX` to have $Property_E$ relationship with individuals of other classes as long as it satisfies the "at least one" requirement.

On the other hand, universal restriction, denoted by ∀, states that individuals of the class being defined must have *all* of their property relationships with a specified set of individuals of a class. If `ClassX` has a property restriction of $\forall\ Property_U\ ClassY$, then individuals of `ClassX` have a $Property_U$ relationship only with individuals of `ClassY`. However, it is possible for individuals of `ClassX` not to have any $Property_U$ values at all. Unlike the existential restriction, universal restriction does not require the individuals to have a property relationship with the defined set of objects.

For example, if the class `Country` has the existential restriction, ∃`containsFeature` `BodyOfLand`, then each individual of the `Country` class must have at least one a `containsFeature` property value from the individuals of `BodyOfLand`. Individuals of the `Country` class may have `containsFeature` property value from individuals from other classes, as shown in Figure 1.

On the other hand if the class `Country` has the universal restriction, ∀`containsFeature` `BodyOfLand`, then if an individual of `Country` has a `containsFeature` property value, it must be an individual of `BodyOfLand`. This restriction does not require all of `Country` individuals to have a `containsFeature` property value, as shown in Figure 2. Unlike the existential restriction, individuals may not be associated with any `containsFeature` relationships.

3.5.2 Open World vs. Closed World

While databases, logic programming and frame languages are based on the "closed world assumption," which assumes that when something is not specified, it is false, description logic based languages, such as OWL DL, are based on "open world assumption" which associates negation with "unsatisfiability." That is, falsehood can only be asserted if it is made explicit.

For example, by using an existential quantifier in the Geography ontology, we assert that an `IslandCountry` has land type that is kind of `Island` and has border that is kind of `Ocean`. Because of the open world assumption, until we explicitly say that an `IslandCountry` has only these kinds of land types and borders, it will be assumed by a reasoner that an `IslandCountry`

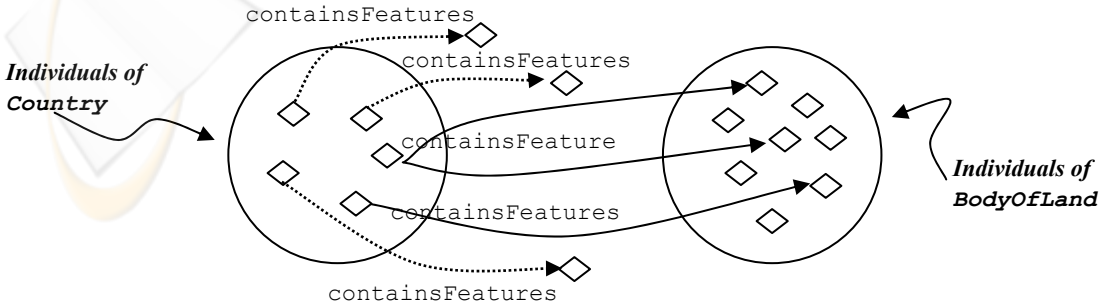

Figure 1: Existential restriction example.

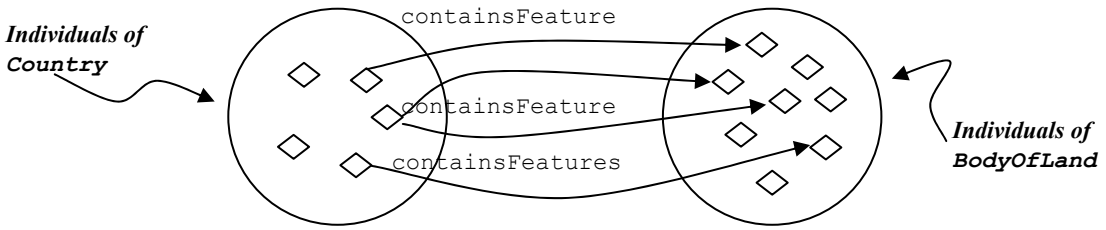

Figure 2: Universal restriction example.

could have other land types and borders. By using closure axioms that consist of a universal restrictions along both the `hasLandType` and `hasBorder` properties, we explicitly specify that an `IslandCountry` has only land type of kind of `Island` and only border that is kind of `Ocean`.

3.5.3 Primitive and Defined Classes

Unlike other languages, OWL differentiates between "primitive" and "defined" classes. Primitive classes, also referred to as "partial," are those defined by *necessary* conditions only. Defined, or "complete" classes, have at least one *necessary and sufficient* condition. The difference is the level of completeness associated with the class definition. Reasoning tools can base their classification inferences only on defined or complete classes; no definitive conclusions can be made on primitive classes.

For example, `CoastalCountry` is a defined class because it has necessary and sufficient conditions as part of the class specification. The necessary and sufficient *conditions* imply that *any* class that is country and has a land type of `Coastline`, amongst other things, is a `CoastalCountry`. If this class definition was primitive, with necessary conditions only, such unambiguous inference cannot be made. It is crucial for developers to understand that unless classes are complete, using necessary and sufficient conditions, the classifier will not infer class subsumption.

3.6 Classify Ontology with a Reasoning Tool

One of the main advantages for developing an OWL DL ontology is that it can be processed by a reasoner. Based on necessary and sufficient conditions of the classes, a reasoner will find those classes that should be subsumed under more than one class. Another service of the reasoner is consistency checking which checks whether or not it is possible for a class to have any instances. These services are of a tremendous value, especially with a large and complex ontology, because it helps developers keep their ontologies modular and thus more manageable, in addition to verifying the consistency of the class descriptions as the ontologies are being developed.

For the Geography ontology, the `CityState` class was defined as a necessary and sufficient condition of the intersection of two classes, `City`∩`Country`. As a result, the reasoner reclassified the `CityState` class as subclass of both the `City` *and* `Country` classes.

3.7 Create Individuals and Fill Property Values

The last step for developing an OWL ontology is creating individuals and filling their property values. Individuals represent the actual entities of the domain of interest. Individuals are also used as part of class description and restrictions. There are specific OWL constructs used with individuals, such as `owl:hasValue`, `owl:sameAs`, and `owl:differentFrom`. Furthermore, individuals are used to define enumerated classes with the syntax `owl:oneOf`.

Many individuals are specified early in the development process, when the domain concepts are informally listed in Step 2 of the methodology. The concepts that were at the lowest level of specification, or can not be grouped as a class, become individuals. Unlike the other components of an ontology, such as classes and properties, individuals are the actualization of the descriptions. In the Geography ontology, some of the concepts appropriate as individuals are `Italy`, `France`, `Mexico`, `Rome`, `VaticanCity`, `PacificOcean`, `GangesRiver`, `MtVesuvius` and `LakeOntario`.

For each individual, there is an associated list of properties as specified in the class definition. Since properties denote relationships between individuals,

or between an individual and a datatype string, the developer needs to input those values at this stage of the developments. For example, the individuals of the `City` class have the `containsPhysicalGeography`, `adjacentTo`, `locatedIn`, and `hasPopulationCount` property values to be filled as part of the individual instantiation.

Although this step is the least difficult of development stages, it could be the most time consuming. Depending on the domain and scope of the ontology, the number of individuals can be very large. However, as long as the schema of the ontology is developed and valid, creating and managing individuals should not be much of a challenge.

4 CONCLUSION

There is a general agreement that ontologies are the knowledge representation component of the Semantic Web. This paper presented a methodology for developing semantically rich ontologies using the OWL DL language. The proposed seven steps methodology is based on related methodologies for software and ontology development. Step 1 defines the scope, purpose, and application of the ontology. Step 2, enumerates a preliminary list of domain concepts as the basis for defining the classes of the ontology. Step 3 organizes those concepts into a class hierarchy. Step 4 defines the properties of the domain of interest using the property constructs provided by OWL. Understanding the semantics of the different types of property and the kinds of relationship they imply are important for creating a rich ontology. Step 5 uses the defined properties and other constructs to further restrict and describe classes. Step 6 uses a reasoner to check the consistency of the classes and infer new superclass/subclass relationships. Finally, Step 7 creates class instances (individuals) and specifies their properties.

It is important to emphasize two aspects of ontology development that are crucial to its success. The first is that although the steps of the methodology are presented in a linear fashion, and as with contemporary software development methodologies, their application is highly iterative. The second is that there is no one correct way to model an ontology for a given domain. Similar to conceptual modeling, ontology development is to a great extent an art rather than a science that will vary from one developer to another.

Modeling real-world domain knowledge into abstract ontological models is challenging. However, armed with a thorough understanding of the ontology language semantics, and the detailed guidance of a development process, such as the one presented in this paper, accurate and useful ontologies can be successfully built.

REFERENCES

Berners-Lee, T., Hendler, J. & Lassila, O. 2001, "The semantic Web: a new form of Web content that is meaningful to computers will unleash a revolution of new possibilities", *Scientific American*, vol. 284, no. 5, pp. 34.

Brickley, D. and Guha, R. 1999, "Resource Description Framework (RDF) Schema Specification. Proposed Recommendation, World Wide Web Consortium: http://www.w3.org/TR/PR-rdf-schema

Gomez-Perez, A., Fernandez, M., and De Vicente, A.J. 1996, "Towards a Method to Conceptualize Domain Ontologies", *Proc. ECAI-96 Workshop on Ontological Engineering*, Budapest.

Gruber, T.R. 1993, "A translation approach to portable ontology specifications", Knowledge Acquisition, vol. 5, pp 199-220.

Grüninger, M., Fox, M.S. 1995, "Methodology for the Design and Evaluation of Ontologies", *Proc. IJCAI-95 Workshop on Basic Ontological Issues in Knowledge Sharing*, Montreal, August 19-20.

Horridge, M. 2004, "A Practical Guide to Building OWL Ontologies with the Protégé-OWL Plugin", Available at www.co-ode.org/resources/tutorials/ ProtegeOWLTutorial.pdf

KBSI, 1994, "The IDEF5 Ontology Description Capture Method Overview", KBSI Report, Texas.

McGuinness, D. 2002, "Ontologies come of age" in Spinning the Semantic Web; *Bringing the World Wide Web to Its Full Potential*, eds. J. Hendler, H. Lieberman & W. Wahlster, MIT Press.

McGuinness, D. and Van Harmelen, F. (eds) 2004, *OWL Web Ontology Language Overview"*, W3C Recommendation, Available at http://www.w3.org/TR/owl-features/

Musen, M.A. 1992, "Dimensions of knowledge sharing and reuse", *Biomedical Research*, vol. 25, pp. 435-467

Noy, N. and McGuinness, D. 2001, "Ontology Development 101: A Guide to Creating your First Ontology", Available at http://protege.stanford.edu/publications/ontology_development/ontology101.pdf

Uschold, M. 1996, "Building Ontologies: Towards A Unified Methodology", *Proc. Expert Systems 96*, Cambridge, December 16-1