A Method to Develop Description Logic Ontologies Iteratively Based on Competency Questions: an Implementation

Yuri Malheiros[1,2], **Fred Freitas**[1]

[1]Centro de Informática – Universidade Federal de Pernambuco (UFPE)
Recife – PE – Brazil

[2]Departamento de Ciências Exatas – Universidade Federal da Paraíba (UFPB)
Rio Tinto – PB – Brazil

yuri@dce.ufpb.br, fred@cin.ufpe.br

Abstract. *Many methodologies and tools are proposed to improve and make easy the process of develop ontologies. We are proposing a system to develop ontologies iteratively using competency questions. The system works as follows: a user asks a question to the system, and it tries to answer the question with the knowledge encoded in an ontology. If it cannot answer correctly, the system generates new questions to ask the user for more axioms. Then, the process restarts, until the system can answer all the generated questions, including the first one. Thus, we are creating a way to define requirements, evaluate, and to add new axioms to an ontology using natural language.*

1. Introduction

Since the 90's, ontology development was more like a craft or an arcane art form than engineering, because there are no patterns to guide engineers and each development team followed its own rules [Guarino et al. 2002] [Gómez-Pérez et al. 2004]. In a clear sign of progress, systematic methodologies have been proposed to support ontology development. These methodologies address the tasks of creating and maintaining an ontology; thus, they specify an ontology lifecycle, define how to describe the ontology scope and requirements (this latter consisting of the competency questions (CQs) [Gruninger and Fox 1995]), the ontology specification itself, and its evolution, etc.

Many methodologies have been proposed to date to build an ontology, for instance, Methontology [Fernandez-Lopez et al. 1997], On-To-Knowledge [Staab et al. 2001], and Ontology 101 [Noy and McGuinness 2008], to cite but a few. They define the steps that an ontology engineer should follow to create and maintain an ontology. A noteworthy fact is that these methodologies are slightly different, but share many common features. Two important ones consist of the iterative way of development, and the use of CQs to define requirements. There are also many tools to assist ontology development. Protégé [Gennari et al. 2003], OntoStudio[1], NeOn Toolkit [del Carmen Suárez-Figueroa et al. 2008], OntoEdit [Sure et al. 2002] and WebODE [Arpírez et al. 2001] are among the most employed tools that facilitate the process of creating an ontology.

In this paper, we present a system to build ontologies from scratch or evolve an existing ontology iteratively using CQs and their respective answers. The system uses

[1]http://www.semafora-systems.com/en/products/ontostudio/

CQs written in English to check OWL DL ontologies automatically using reasoning. This is different from many works that usually check the ontologies manually or at most use SPARQL queries. The user do not need to know DL syntax or other complex language to use the system, because all the interaction is made by natural language, thus there is not barriers to nonexperts users. Furthermore, the iterative nature of the system fits in many methodologies, then it can improve well-known development processes.

The basic methodology can be described as follows: the user asks a CQ to the system, and it tries to answer the question with the knowledge encoded in an ontology. If it cannot answer correctly, a system that implements the method asks the user for some more axioms, and generates other auxiliary CQs that the user can modify and answer; then the process restarts, until it can answer all the generated CQs, including the first. We also present here a first implementation of the method, which receives CQs in natural language of various types (which are described along the article) and convert them to OWL DL.

The remainder of this paper is organized as follows: Section 2 provides a background about description logic ontologies, ontology engineering and competency questions; Section 3 presents our proposal of the system to build ontologies iteratively using competency question; Section 4 details the implementation of the system; In section 5 we show the results of tests using the system; Section 6 discusses related work; and, Section 7 concludes the paper and presents some ideas for future works.

2. Background

To set the scene of the rest of this paper, the next three sections elucidate concepts related to description logic ontologies, ontology engineering and competency questions. These concepts serve as foundation of this work.

2.1. Description Logic Ontologies

Description Logics (DLs) are a family of knowledge representation formalisms that have been gaining growing interest in the last two decades, particularly after OWL (Ontology Web Language) [Patel-Schneider et al. 2004], was approved as the W3C standard for representing the most expressive layer of the Semantic Web.

One of the most used DL languages is $\mathcal{ALC}$, due to its good trade-off between expressivity and reasoning costs. We will describe in the following since this is the language used throughout the paper. An ontology or knowledge base in $\mathcal{ALC}$ is a set of axioms a_i defined over the triple (N_C, N_R, N_O) [Baader et al. 2003], where N_C is the set of concept names or atomic concepts (unary predicate symbols), N_R is the set of role or property names (binary predicate symbols); N_O the set of individual names (constants), instances of N_C and N_R: N_{CO} is the set of classes' instances and N_{RO} the set or role instances, with $N_{CO} \cup N_{RO} = N_O$. N_C contains concepts (like Bird, Animal, etc) as well as other concept definitions as follows. If r is a role $(r \in N_R)$ and C and D are concepts $(C,\ D \in N_C)$ then the following definitions belong to the set of $\mathcal{ALC}$ concepts: (i) $C \sqcap D$ (intersection of two concepts); (ii) $C \sqcup D$ (union of two concepts); (iii) $\neg C$ (complement of a concept); (iv) $\forall r.C$ (universal restriction of a concept by a role); (v) $\exists r.C$ (existential restriction of a concept by a role); (vi) $\top$, the universal concept that subsumes all concepts, and (vii) $\bot$, the bottom concept that is subsumed by all concepts. Note that, in the definitions above, C and D can be inductively replaced by other complex concept expressions.

There are two axiom types allowed in $\mathcal{ALC}$: (i) Assertional axioms, which are concept assertions $C(a)$, or role assertions $r(a, b)$, where $C \in N_C$, $r \in N_R$, $a, b \in N_O$ and (ii) Terminological axioms, composed of any finite set of GCIs (general concept inclusion) in one of the forms $C \sqsubseteq D$ or $C \equiv D$, the latter meaning $C \sqsubseteq D$ and $D \sqsubseteq C$, C and D being concepts. An ontology or knowledge base (KB) is referred to as a pair $(\mathcal{T}, \mathcal{A})$, where $\mathcal{T}$ is the terminological box (or TBox) which stores terminological axioms, and $\mathcal{A}$ is the assertional box (ABox) which stores assertional axioms. $\mathcal{T}$ may contain cycles, in case at least in an axiom of the form $C \sqsubseteq D$, D can be expanded to an expression that contains C.

$\mathcal{ALC}$ semantics is formally defined in terms of interpretations, model, fixpoints, interpretation functions, etc, over a domain or discourse universe Δ [Baader et al. 2003].

2.2. Ontology engineering

According to Gómez-Perez and colleagues, ontology engineering refers to the activities related to the process, lifecycle, methods, methodologies, tools, and languages to support the ontology development [Gómez-Pérez et al. 2004]. Devedzic defines that ontology engineering covers the set of activities done during the conceptualization, design, implementation, and deployment [Devedzić 2002].

In some ways, the methodologies to develop ontologies are similar to the ones for software engineering. They provide guidance to developers and are divided in phases, for example, specification, execution, and evaluation. Besides, the process is usually iterative, and the ontology can evolve during its lifetime in a very similar way of a software, in the sense that it requires maintenance, versioning, etc. Since the early 90's, several methodologies to build ontologies have been defined, with activities like requirements definition, implementation, and evaluation.

2.3. Competency questions

Competency questions [Gruninger and Fox 1995] are a set of questions that the ontology must be capable to answer using its axioms. The questions can be used to specify the problems an ontology or a set of ontologies must solve. Thus, they work as requirements' specification of one or more ontologies. With a set of CQs at hand, it is possible to know whether an ontology was created correctly, if it contains all the necessary and sufficient axioms that correctly answer the CQs.

Many works propose the use of CQs for ontology engineering, but they usually used them to check ontologies manually, or, at most, express them as SPARQL queries. In the case of answers that arise from more complex DL reasoning, in which the answers are not present in the ontology but can be entailed by it, no other option is yet offered, but to check CQs manually, what constitutes a slow and expensive process that could be impracticable with very large ontologies or when the quantity of CQs is huge.

3. Proposal: Method to Develop Ontologies Iteratively Based on CQs

We developed a method and a system implementation to build ontologies iteratively using CQs and their respective answers. It is based on the idea of Uschold [Uschold 96], which was never tried in the Semantic Web context. Yet, all the questions and answers are written in English. The method's algorithm is given by the Figure 1:

CQOntoBuilder

Inputs: 1. CQ_{NL} (competency question in natural language) or CQ_{DL} (competency question DL)

2. $\propto \mid O \models \propto (CQ_{DL})$, i.e., the answer to CQ_{DL}

3. O = set of axioms a_i defined over the triple (N_C, N_R, N_O) [1], where

- N_C is the set of *concept names* ,
- N_R is the set of *role or property names*,
- N_I the set of *instances* of N_C and N_R:

Output: $O' \mid O' \models \propto (CQ_{DL})$

1. CQ_{DL} := conversion (CQ_{NL});
2. **If** $O \models \propto (CQ_{DL})$ **return** O;
3. **Else**
   - **a.** **{Adding new knowledge, typed by the user}**
   - **b.** $N_C' := \emptyset$;
   - **c.** **Repeat**
     - **i.** C= new Concept(read(C));
     - **ii.** $N_C' := N_C' \cup C$;
   - d. **Until user breaks;**
   - **e.** $N_R' := \emptyset$;
   - **f.** **Repeat**
     - **i.** r= new Role(read(r));
     - **ii.** $N_R' := N_R' \cup r$;
   - g. **Until user breaks;**
   - **h.** $N_I' := \emptyset$;
   - **i.** **Repeat**
     - **i.** C= new Instance(read(i));
     - **ii.** $N_I' := N_I' \cup i$;
   - j. **Until user breaks;**
   - **k.** $O' := O' \cup N_C' \cup N_R' \cup N_I'$;
   - **l.** **{Generation of a new CQ}**
   - **m.** CQ_{DL}' := generate(CQ_{DL}, O);
   - n. read ($\propto' \mid O' \models \propto' (CQ_{DL}')$);
   - **o.** **return** CQOntoBuilder ($CQ_{DL}', \propto', O'$) ;
4. **Endif**

Figure 1. Algorithm

This algorithm is recursive and receives a CQ in natural language or DL, converts it to DL when needed, and, in case it is not satisfied yet, asks for more knowledge, generates a new CQ that should help the ontology O to satisfy the original CQ and restarts this process all over again. Note that the algorithm assumes that the oracle function generate is available. For our current implementation, we assume that the user will do this job.

Example 1. An ontology with the following axioms is loaded:

$Herbivorous \equiv Animal \sqcap \forall eats.\neg meat$

$Cow \equiv Animal \sqcap \forall eats.grass$

Then, a CQ states "Are cows herbivorous?", where the expected answer is "true".

A system implementing the method tries to answer the question, but fails, because the ontology lacks the necessary axioms to infer that $Cows \sqsubseteq Herbivorous$. Next, the system generates a new CQ for the user, for instance, "are grass and meat disjoint?". If the user answers "yes" the system includes in the ontology an axiom stating that the classes Grass and Meat are disjoint ($Grass \sqsubseteq \neg Meat$). Now, the ontology has the necessary axioms to answer the initial question correctly. □

Using this iterative process, a user can evaluate if an ontology has the necessary axioms to answer questions, and can add new knowledge, "teaching" it through the answers to the CQ made by the method/system. In the current version, our system can answer many types of questions using natural language and can add new axioms to an ontology according to the answers to questions. The question generation by the system is

still being studied since it indeed represents a new DL problem, which requires additional specific research to determine for which DL languages the problem is decidable, and in case they are, the problem's computability. Currently, we are assuming that an oracle for that problem exists indeed, in this implementation, the user provides the questions.

In the next sections, we describe a first implementation of the method with its two components: the natural language query component and the ontology builder.

4. Implementation

The system includes three core components:

- Natural language query: in the system, the user can write CQs in natural language. This component parses the query, uses the knowledge specified in an ontology, and returns an answer;
- Question generator: when the system cannot answer a question, because the ontology does not have the necessary knowledge, it generates questions for the user, to gather more knowledge to answer the initial question;
- Ontology builder: all the new knowledge learned through the questions generated by the previous component are added to the ontology. This component is responsible for transforming the information of the previous component to an ontology specification language.

4.1. Natural language query

After loading an ontology, the next step of the process to build or evolve an ontology with the proposed system is to write a CQ in natural language. We choose this approach to compose a CQ, because it is easier to use natural language than description logics.

In the system, there are predefined types of questions that it understands. The types are defined by rules, and each rule is defined using grammatical tags (nouns, adjectives, verbs, etc.) and regular expression operators (*, +, ?, and |). Each word of a question is labeled using the NLP Stanford POS Tagger [Toutanova et al. 2003]. The labels are the grammatical category of the word. Then, the component verifies if the words and its POS tags match with some question rule. If it satisfies a rule, the component will perform the operations to retrieve information of the ontology according to the question type. Otherwise, the system returns that it does not understand what the user asks.

The component can find names defined in the ontology even though they are written in the question in plural, or separated by spaces, or with different capitalizations. For example, "red wine" in a question can be matched with a class "RedWine" in the ontology, or the word "cows" in a question can be matched with a class "Cow". The component tests many variations of names in the question to find the correct match in the ontology. Thus, the user can make questions in a very natural way regardless the specific notation used to specify the ontology.

This component uses OWL API [Horridge and Bechhofer 2011] and HermiT OWL reasoner [Shearer et al. 2008] to search for answers. Thus, it can infer information that is not explicitly defined in an ontology to give the correct answer.

The following are the types of questions supported. There are three simple types of CQs to check different characteristics of an ontology. We present a general explanation of each rule, usage examples, the regular expression rules and the type of answers.

4.1.1. Is-a question

The first type of question verifies if a class is subclass of another class.

Example: Is red wine a wine?
Rule: is (Noun|Adjective|Number)+ (a|an) (Noun|Adjective|Number)+
Answers: Yes, no, true or false.

Both (Noun|Adjective|Number)+ in the rule refer to classes names in the ontology. Then, this question type supports class names composed by nouns and adjectives. For example, "red wine" is a valid class name, because red is an adjective and wine a noun. The order is unimportant, thus a class name can start with a noun or an adjective. The quantity of nouns and adjectives does not matter too. The class name must have at least one word, but all combinations of nouns and adjectives with any number of words (greater than one) are possible.

4.1.2. Property value question

This type of question verifies if a property of an instance has a specified value. The system will answer "yes" or "true" if the property of the instance indeed has the specified value, and "no" or "false" in the opposite case. Property value questions have two distinct rules.

Examples: Does bancroft chardonnay have color white?
Do birds eat animals?
Rules:
(does|do) (Noun|Adjective|Number)+ have Noun (Noun|Adjective|Number)+
(does|do) (Noun|Adjective|Number)+ Verb (Noun|Adjective|Number)+
Answers: Yes, no, true or false.

In both rules the instance name is defined by the first (Noun|Adjective|Number) and the second (Noun|Adjective|Number)+ defines the value of the property. The first rule verifies only properties names starting with "has" followed by a noun. Properties like "hasColor", "hasPart", etc., are common in ontologies; therefore we created a special rule for such cases. In the second rule, the property name is a verb.

4.1.3. Existence question

The existence questions have two rules too. This type of question verifies which subclasses of a class exist. These questions support DL existential and universal restrictions. The system will answer the list of the subclasses found.

Examples: Which wines exist?
Which wines have sugar dry?
Rules:
which (Noun|Adjective|Number)+ exist
which (Noun|Adjective|Number)+ have (Noun|Verb) (some|only)? (Noun|Adjective|Number)+
Answers: A list of classes separated by commas or the word "and". For example, "red wine, white wine".

In the rules, the first (Noun|Adjective|Number)+ defines the class name. The second rule expects extra information: a property name starting with "has" followed by a noun or a verb. In the end of the second rule, there is another (Noun|Adjective|Number)+, which defines the property's value. The user can write the words "only" or "some" optionally to specify existential and universal restrictions respectively.

4.2. Ontology builder

The goal of the ontology builder component is to add new knowledge to an ontology. Answering questions generated by the system, the user acts as a teacher to the system, that stores what it learns in the ontology. The system has predefined types of questions it can generate. These questions are called system's questions (SQ), a competency question generated by the system. In this case, there are not rules for each SQ, because the system knows exactly the format of the question it will generate. The user only needs to answer the question properly.

The following are the types of SQs supported. We present a general explanation of each SQ, usage examples, axioms generated, and answers' types they expect.

4.2.1. Is-a system's question

The system uses this type of SQ when it needs to know about the subclass relation of two classes, if it is true or false.

Example: Is red wine a wine?
Answers: Yes, no, true or false.
Axiom: $RedWine \sqsubseteq Wine$

4.2.2. Property value system's question

In this type of SQ, the system looks for knowledge about the value of some property. When answering this question positively, the user specifies that a class has a certain property and that this property has a certain range of values. It is also possible to make SQs using universal and existential restrictions.

Examples: Does bancroft chardonnay have color white?
Does bird eat some grass?
Answers: Yes, no, true or false.
Axioms:
$BancrofChardonnay \sqsubseteq \forall hasColor.White$
$Bird \sqsubseteq \exists eat.Grass$

4.2.3. Existence system's question

The last type of SQ is similar to the first, but it is concerned with the multiple relation of classes and a superclass. When answering this SQ, the user is specifying that multiple classes are subclasses of one class.

Table 1. Natural language query tests with wine ontology

Competency Question	Answer
Is red wine a wine?	true
Does bancroft chardonnay have color white?	true
Which red wines exist?	Beaujolais, CabernetFranc, CabernetSauvignon, Chianti, CotesDOr, DryRedWine, Margaux, Medoc, Meritage, Merlot, Pauillac, PetiteSyrah, PinotNoir, Port, RedBordeaux, RedBurgundy, RedTableWine, StEmilion, and Zinfandel
Which wines have sugar sweet?	IceWine, LateHarvest, Port, Sauternes, and SweetRiesling

Table 2. Natural language query tests with pizza ontology

Competency Question	Answer
Is napoletana a cheesey pizza?	true
Which spicy pizzas exist?	AmericanHot, Cajun, CheeseyVegetableTopping, IceCream, PolloAdAstra, and SloppyGiuseppe
Which meaty pizza has topping some ham topping?	Capricciosa, CheeseyVegetableTopping, IceCream, LaReine, Parmense, and Siciliana

Example: Which wines exist?
Answers: A list of classes separated by commas or the word "and". For example, "red wine, white wine".
Axioms:

$$RedWine \sqsubseteq Wine$$
$$WhiteWine \sqsubseteq Wine$$

5. Results

We have some preliminary results using the components detailed in the previous section. For the natural language query component we performed three rounds of tests, each one with a different ontology. The used ontologies were the wine ontology, the pizza ontology, and the travel ontology, all available in the Protégé website[2]. For each round, we used at least one CQ for each type, except in for the pizza ontology, because it lacks individuals with properties. For the ontology builder, we test each type of SQ using the wine ontology.

5.1. Natural language query tests

In the tests of the natural language query component, we used three different ontologies. For each ontology, we show the CQs used, and the answers for them. First, we tested the wine ontology, the Table 1 displays the results using it. Further, the Table 2 has the results for the pizza ontology. Last, the Table 3 displays the results for the travel ontology.

5.2. Ontology builder

For present the results of the SQs, we created a new class in the wine ontology called TestWine. Nothing was specified about this class, only that it exists. Then, we make SQs and answer them as follow:

1. Is test wine a wine? True.
2. Does test wine have color white? True.

[2]http://protege.stanford.edu/download/ontologies.html

Table 3. Natural language query tests with travel ontology

Competency Question	Answer
Is yoga an activity?	true
Does four seasons have rating three star rating?	true
Which adventures exist?	BunjeeJumping, and Safari
Which accommodations has rating one star rating?	Campground, and Safari

3. Which test wines exist? Red wine.

For the first question, the system wrote OWL code in the ontology to define that TestWine is subclass of Wine. The code written was:

```
Class: vin:TestWine

    SubClassOf:
        vin:Wine
```

Answering the question two positively the component wrote the definition that TestWine class has the property hasColor with value White. After, the code for the Test-Wine class was:

```
Class: vin:TestWine
    SubClassOf:
        vin:Wine
        vin:hasColor only vin:White
```

For the last question, the answer specified that RedWine is subclass of TestWine. Then, the class code changed to:

```
Class: vin:RedWine

    EquivalentTo:
        vin:Wine
         and (vin:hasColor value vin:Red)

    SubClassOf:
        vin:TestWine
```

5.3. Discussion of the results

In the first CQ tested in the wine ontology, we can already see the importance of the reasoner. In the ontology, there is no code defining directly that the class RedWine is subclass of Wine. However, the system answers true. It seems correct, because RedWine is indeed a type of Wine, but if the ontology does not specify this, the answer must be false. What happened was that the system ran the HermiT reasoner before search for an answer. The reasoner infers that RedWine is subclass of Wine, then the answer is really correct. In the test of the pizza ontology, we got some strange results. For example, in the CQ "Which meaty pizza has topping some ham topping?", one of the classes listed in the answer was IceCream. It seems a wrong answer, but the ontology was created in a way that the reasoner infers this awkward relation between IceCream and MeatyPizza.

The ontology builder component is working correctly for the defined SQs. The OWL API allows the system write new axioms flawlessly. Then, we only need to extend this component to support the inclusion of more types of axioms in the future.

6. Related work

This work was inspired by the iterative way of develop ontologies proposed by many methodologies in literature, and by the use of CQs to evaluate and define requirements.

Methontology [Fernandez-Lopez et al. 1997] defines activities to perform during the ontology development and it defines the ontology life cycle too. During its life, an ontology moves through the following states: specification, conceptualization, formalization, integration, implementation, and maintenance. This life cycle seems analogous to the waterfall life cycle in software engineering [Royce 1987], however the authors make it clear that it is not an adequate path to develop an ontology. Then, it is proposed an evolving life cycle that allows the engineer to go back from any state to other if it is necessary. Thus, this life cycle permits inclusion, removal, or modification anytime during the development. The Ontology 101 methodology [Noy and McGuinness 2008] defines an iterative process. The engineer starts with a simple model and refines it during the development. The steps of this process are: determine the domain and scope, consider reusing ontologies, enumerate important terms, define the classes and the class hierarchy, define the properties, define the facets of the slots, and create instances. The authors suggest to use CQs in the first step to determine the scope of the ontology.

Other methodologies to build ontologies emerged since 1990. Lenat and Guha presented the steps of Cyc development in one of the first works in this area [Lenat and Guha 1989]. Uschold and Gruninger contribute in many papers to evolve the ontology engineering field, proposing and refining guidelines [Gruninger and Fox 1995] [Uschold and King 1995] [Uschold et al. 1996]. In 2001, the On-To-Knowledge methodology appeared, it was a result of the project with the same name [Staab et al. 2001].

During the emergence of the methodologies, many tools to support the ontology development process are proposed. The first was the OntolinguaServer in the beginning of 1990. It started only with a simple editor, and later other components were added, such an equation solver and an ontology merge tool [Farquhar et al. 1997]. The WebOnto tool was developed in 1997, its main innovation was the collaborative edition of ontologies [Domingue 1998]. Protégé, an ontology editor with extensible architecture, is one of the most popular ontology tools nowadays. This tool supports the creation of ontologies in multiple formats [Gennari et al. 2003]. In the first years of 2000, WebODE [Arpírez et al. 2001] and OntoEdit [Sure et al. 2002] appeared. The WebODE supports multiple formats of ontology, it has an editor, and components to evaluate and merge ontologies. Also, WebODE supports most of the activities and steps of Methontology. Last, the OntoEdit has similar characteristic of the previous tools, for example, extensible architecture, ontology editor, etc.

All these works presented tried to improve the way of develop ontologies. In this paper, we are not proposing a new full featured ontology editor, neither a new methodology to create ontologies, but we are creating a system to support some phases of iterative methodologies and it may be integrated in some existing tools.

7. Conclusion

In this paper, we presented our progress in developing a method and its respective system to support a novel process of DL ontology building. Two key components are already

operating, the natural language query and the ontology builder. The former is responsible to process a natural language CQs and tries to answer it using the knowledge specified in a DL ontology. The latter is concerned with incorporating new knowledge in an ontology; it uses answers stated by the users to CQs generated by the system. We also defined the process to build or evolve an ontology iteratively using the system.

There are still limitations in the work. Each component needs to evolve. The natural language query component must support more types of question and treat more intrinsic details of the written language. The ontology builder needs to support more SQs too. Finally, we need to study the problem and develop the automatic question generation when some knowledge is missing in the ontology and the system cannot answer a question correctly. For now, this process is made manually.

As future work, besides evolving the system's components, we intend to build a complete tool for ontology engineers based on the proposed method. Then, they can create or evolve an ontology using all the process defined in this paper. The tool may be integrated with popular ontology environments like Protégé and NeOn.

References

Arpírez, J. C., Corcho, O., Fernández-López, M., and Gómez-Pérez, A. (2001). Webode: a scalable workbench for ontological engineering. In *Proceedings of the 1st international conference on Knowledge capture*, K-CAP '01, pages 6–13, New York, NY, USA. ACM.

Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., and Patel-Schneider, P. F., editors (2003). *The description logic handbook: theory, implementation, and applications*. Cambridge University Press, New York, NY, USA.

del Carmen Suárez-Figueroa, M., de Cea, G. A., Buil, C., Dellschaft, K., Fernández-López, M., García, A., Gómez-Pérez, A., Herrero, G., Montiel-Ponsoda, E., Sabou, M., Villazon-Terrazas, B., and Yufei, Z. (2008). D5.4.1 neon methodology for building contextualized ontology networks.

Devedzić, V. (2002). Understanding ontological engineering. *Commun. ACM*, 45(4):136–144.

Domingue, J. (1998). Tadzebao and webonto: Discussing, browsing, and editing ontologies on the web. In *In Proceedings of the 11th Knowledge Acquisition for Knowledge-Based Systems Workshop*.

Farquhar, A., Fikes, R., and Rice, J. (1997). The ontolingua server: a tool for collaborative ontology construction. *Int. J. Hum.-Comput. Stud.*, 46(6):707–727.

Fernandez-Lopez, M., Gomez-Perez, A., and Juristo, N. (1997). Methontology: from ontological art towards ontological engineering. In *Proceedings of the AAAI97 Spring Symposium*, pages 33–40, Stanford, USA.

Gennari, J. H., Musen, M. A., Fergerson, R. W., Grosso, W. E., Crubézy, M., Eriksson, H., Noy, N. F., and Tu, S. W. (2003). The evolution of protege: an environment for knowledge-based systems development. *Int. J. Hum.-Comput. Stud.*, 58(1):89–123.

Gómez-Pérez, A., Fernández-López, M., and Corcho, O. (2004). *Ontological Engineering: With Examples from the Areas of Knowledge Management, E-Commerce and the Semantic Web*. Advanced Information and Knowledge Processing. Springer.

Gruninger, M. and Fox, M. S. (1995). Methodology for the design and evaluation of ontologies.

Guarino, N., Welty, C., and Common, E. (2002). Evaluating ontological decisions with ontoclean.

Horridge, M. and Bechhofer, S. (2011). The owl api: A java api for owl ontologies. *Semant. web*, 2(1):11–21.

Lenat, D. B. and Guha, R. V. (1989). *Building Large Knowledge-Based Systems; Representation and Inference in the Cyc Project*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition.

Noy, N. F. and McGuinness, D. L. (2008). Ontology development 101: A guide to creating your first ontology.

Patel-Schneider, P. F., Hayes, P., and Horrocks, I. (2004). OWL web ontology language semantics and abstract syntax. W3C recommendation, W3C. Published online on February 10th, 2004 at `http://www.w3.org/TR/2004/REC-owl-semantics-20040210/`.

Royce, W. W. (1987). Managing the development of large software systems: concepts and techniques. In *Proceedings of the 9th international conference on Software Engineering*, ICSE '87, pages 328–338, Los Alamitos, CA, USA. IEEE Computer Society Press.

Shearer, R., Motik, B., and Horrocks, I. (2008). HermiT: A Highly-Efficient OWL Reasoner. In Ruttenberg, A., Sattler, U., and Dolbear, C., editors, *Proc. of the 5th Int. Workshop on OWL: Experiences and Directions (OWLED 2008 EU)*, Karlsruhe, Germany.

Staab, S., Studer, R., Schnurr, H.-P., and Sure, Y. (2001). Knowledge processes and ontologies. *IEEE Intelligent Systems*, 16(1):26–34.

Sure, Y., Erdmann, M., Angele, J., Staab, S., Studer, R., and Wenke, D. (2002). Ontoedit: Collaborative ontology development for the semantic web. In *Proceedings of the First International Semantic Web Conference on The Semantic Web*, ISWC '02, pages 221–235, London, UK, UK. Springer-Verlag.

Toutanova, K., Klein, D., Manning, C. D., and Singer, Y. (2003). Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, pages 173–180, Stroudsburg, PA, USA. Association for Computational Linguistics.

Uschold, M., Gruninger, M., Uschold, M., and Gruninger, M. (1996). Ontologies: Principles, methods and applications. *Knowledge Engineering Review*, 11:93–136.

Uschold, M. and King, M. (1995). Towards a methodology for building ontologies. In *In Workshop on Basic Ontological Issues in Knowledge Sharing, held in conjunction with IJCAI-95*.