# A Method to Develop Description Logic Ontologies Iteratively with Automatic Requirement Traceability

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

[1] Universidade Federal de Pernambuco (UFPE), Recife - PE, Brazil
{yamb,fred}@cin.ufpe.br,
[2] Universidade Federal da Paraíba (UFPB), Rio Tinto - PB, Brazil

**Abstract.** Competency Questions (CQs) play an important role in the ontology development life-cycle, as they represent the ontology requirements. Although the main methodologies use them, there is room for improvement in the current practice of ontology engineering. One of the main problems is the lack of tools to check if CQs defined in OWL, are being fulfilled by an ontology, preferably in an automated way. Moreover, requirement (or CQ) traceability is rarely explored. Recently, there has been a trend on checking CQs against ontologies using RDF and SPARQL. Naturally, this language, being created for Semantic Networks, is inadequate to check the fulfillment of OWL CQs. In this paper, we introduce a semi-automatic, full-fledged method to develop ontologies iteratively, using CQs as requirements. It presents many novelties: a tracker to monitor the relations among CQs and the OWL code; an NLP component that translates CQs in natural language into OWL queries; a logic checker that confirms whether CQs are satisfied or not; a generator of new CQs, which comes into play when the CQ being dealt is not satisfied yet; and an axiom generator that suggests to the user new axioms to be included in the ontology.

**Keywords:** ontology engineering, competency questions, requirement traceability

## 1 Introduction

For years, ontology development was more like a craft or arcane art form than engineering, because there are no patterns or methodologies to guide the engineers. Thus, each development team followed its own rules [13] [10].

In a clear sign of progress, systematic methodologies and tools have been proposed to support ontology development. These methodologies address the tasks of creating and maintaining an ontology; thus, they specify an ontology life-cycle, define how to describe the ontology scope and requirements (this latter consisting of the competency questions (CQs) [12]), how to create the ontology specification, how to conduct its evolution, etc. Some well-known methodologies

to ontology development are: Methontology [8], On-To-Knowledge [21], Ontology 101 [17], and NeOn [4].

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 [9], OntoStudio[3], NeOn Toolkit [4], OntoEdit [22] and WebODE [2] are among the most employed tools that facilitate the process of creating an ontology.

Despite the advances in ontology engineering, there are many areas to improve. The CQs are widely used to define requirements, however most of works use the CQs to check manually the ontology. This way could create a bottleneck, because it is slow to check a huge quantity of questions, and error-prone.

Traceability is other area that could be explored further. DL ontologies have a formal logic foundation that offers possibilities to trace changes in an automatic way that is very hard to replicate in software engineering. For example, it could be possible to check which requirements are associated with which axioms, and vice-versa. It could be possible to check if all requirements are satisfied or not. And, it could be possible to check if some changes in the ontology code break some requirement. Thus, we are missing an opportunity to build powerful traceability tools to improve the ontology development process.

In this paper, we propose a method and introduce a system to improve some areas of ontology engineering. Using the system an engineer can build or evolve ontologies iteratively using CQs and their respective answers. The CQs are used as requirements and, with the system, every relation among them and axioms are traceable without extra effort from the engineer.

The system uses 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 does 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 are not barriers to people without DL expertise. Furthermore, an ontology engineer can retrieve the axioms for every requirement or, for a given axiom, the system can show the requirement. Finally, the iterative nature of the system fits in many methodologies, then it can improve well-known development processes.

The basic usage can be described as follows: the user asks a CQ for the system, and it tries to answer the question with the knowledge encoded in an ontology. If it cannot answer correctly, the system asks for the user for more axioms, and generates other auxiliary CQs that the user can answer; then the process restarts, until it can answer all the generated CQs, including the first.

The remainder of this paper is organized as follows: Section 2 provides a background about description logic ontologies, ontology engineering, competency questions, and requirement traceability; Section 3 presents our proposal of the system to build ontologies iteratively using competency question with automatic tracing; Section 4 details the implementation of the system; In section 5 we

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

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) [18], 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 language is used in the paper.

An ontology or knowledge base in $\mathcal{ALC}$ is a set of axioms $a_i$ defined over the triple $(N_C, N_R, N_O)$ [3], 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 (for example, 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 $\Delta$ [3].

### 2.2 Ontology engineering

According to Gómez-Perez and colleagues, ontology engineering refers to the activities related to the process, life-cycle, methods, methodologies, tools, and languages to support the ontology development [10]. Devedzic defines that ontology engineering covers the set of activities done during the conceptualization, design, implementation, and deployment [5]. Mizoguchi and Ikeda define the ontology engineering purpose regarding provide a basis of building models of all things in which computer science is interested, and that it should cover subjects as philosophy, knowledge representation, ontology design, standardization, reuse and sharing of knowledge, media integration, etc [16].

In some ways, the methodologies to develop ontologies are analogous 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.

### 2.3 Competency questions

Competency questions [12] 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. For 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.

### 2.4 Requirement traceability

Requirement traceability is a problem studied by many software engineer researches. But, for ontology engineers this area is almost absent. Requirement traceability links the requirement specification and the code. It brings to the engineer the capability of verify the completeness of an implementation with respect to stated requirements [1].

There are two fundamental types of requirements traceability [11]:

- Pre-requirements specification traceability: it is concerned with those aspects of a requirement's life prior to its inclusion in the requirement specification;

– Post-requirements specification traceability: it is concerned with those aspects of a requirement's life that result from its inclusion in the requirement specification.

In our approach, we focus on post-requirements specification traceability.

## 3 Proposal

We developed a method and a system implementation to build ontologies iteratively using CQs and their respective answers with an automatic requirement tracker. It is based on the idea of Uschold [24], which was never tried in the Semantic Web context. Yet, all the questions and answers are written in English.

Considering an ontology with the following axioms: $Herbivorous \equiv Animal \sqcap \forall eats.\neg meat$ and $Cow \equiv Animal \sqcap \forall eats.grass$. 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, can add new axioms to an ontology according to the answers to questions, and it traces all changes. 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 an oracle for that problem exists, in this implementation, the user provides the questions.

In the next section, we describe an implementation of the method with its four components: the natural language checker, the question generator, the ontology builder, and the tracker.

## 4 Implementation

The current implementation includes four components:

– Natural language checker: this component parses questions in English, uses the knowledge specified in an ontology, and returns an answer;
– Question generator: when the system cannot answer a question correctly using the natural language checker, 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 into an ontology specification language.
– Tracker: all changes using the system are traceable. This component creates relations among all axioms added to an ontology and their correspondent competency question.

Next, each component is detailed, except the question generator, because in the current implementation the user needs to create the question manually.

### 4.1 Natural language checker

The first 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 for the engineer 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 [23]. 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 some 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 asked.

There are three main question types: is-a, property value, and existence. They are detailed by table 1. Each table shows a usage example, the regular expression rules and the type of answers the question support. In the tables, Noun means all POS tags related do nouns, Adj means all POS tags related to adjectives, and Num means all POS tags related to numbers.

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” 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 [14] and HermiT OWL reasoner [20] to search for answers. Thus, it can infer information that is not explicitly defined in an ontology to give the correct answer.

### 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

**Table 1.** Types of questions

| | |
|---|---|
| Is-a question type | |
| Example | Is red wine a wine? |
| Rules | is (Noun\|Adj\|Num)+ (a\|an) (Noun\|Adj\|Num)+ |
| Answers | Yes, no, true or false |
| Property value question type | |
| Examples | Does bancroft chardonnay have color white? |
| | Do birds eat animals? |
| Rules | (does\|do) (Noun\|Adj\|Num)+ have Noun (Noun\|Adj\|Num) |
| | (does\|do) (Noun\|Adj\|Num)+ Verb (Noun\|Adj\|Num)+ |
| Answer | Yes, no, true or false |
| Existence question type | |
| Examples | Which wines exist? |
| | Which wines have sugar dry? |
| Rules | which (Noun\|Adj\|Num)+ |
| | which (Noun\|Adj\|Num)+ have (Noun\|Verb) (some\|only)? (Noun\|Adj\|Num)+ |
| Answer | Classes separated by commas or the word "and" |

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), in other words, 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. To add knowledge to an ontology, the user only needs to answer the question properly.

There are three main system's question types: is-a, property value, and existence. They are detailed by table 2. Each table shows an example, the type of answers, and examples of axioms created.

### 4.3 Tracker

All changes made using the system are traceable. When a user creates a CQ the system store this CQ, and it stores, if necessary, the SQs associated with the CQ. When a user answers a SQ the system stores the answer, create a relation between the answer and the SQ, and it also stores the OWL code added (a patch) to the ontology specification. This way, every step is monitored for later analysis. The Figure 1 shows the tracker data structure. Where $CQ_i$ represents competency questions and $i$ is the CQ index, for example, $CQ_1$ is the first CQ, $CQ_2$ the second, etc. $SQ_{ij}$ are the system's questions, where $i$ is the CQ index and $j$ is the SQ index, for example, $SQ_{12}$ is the second system's question related to the first CQ, or $SQ_{31}$ is the first system's question related to the third CQ. Finally, $A_{ij}$ are the answers of the SQs, and $P_{ij}$ represents the code patch related to the answers.

**Table 2.** Types of system's questions

| | |
|---|---|
| Is-a system's question type | |
| Example | Is red wine a wine? |
| Answers | Yes, no, true or false |
| Axioms | $RedWine \sqsubseteq Wine$ |
| Property value system's question type | |
| Examples | Does bancroft chardonnay have color white? |
| | Do birds eat animals? |
| Answer | Yes, no, true or false |
| Axioms | $BancrofChardonnay \sqsubseteq \forall hasColor.White$ |
| | $Bird \sqsubseteq \exists eat.Grass$ |
| Existence system's question type | |
| Examples | Which wines exist? |
| | Which wines have sugar dry? |
| Answer | Classes separated by commas or the word "and" |
| Axioms | $RedWine \sqsubseteq Wine$ |
| | $WhiteWine \sqsubseteq \forall hasSugar.Dry$ |

Using the relations stored in the data structure, this component provides a series of useful operations:

- Log: this command shows every CQ and every SQ and their answers in a hierarchy. Thus, an engineer has a manner to see the SQs generated for each CQ. Using the log command the user has an overview of how the ontology is being built through the change history. To do this, the system only needs to iterate over the nodes of the data structure and presents the information to the user.
- Diff: using this command the user specifies a SQ using the indexes and the system shows the OWL code included or removed. It is a good way to know exactly the axioms added for each SQ. This is possible, because the system saves the code patch, in other words, only the code added and the information where it was added in the ontology file. Thus, the system has all the information to presents the diff to the user.
- Rollback: if some change is not desirable, the user can undo what he did. The rollback returns the system to a specified point according to a CQ and SQ in the change history provided by the log command. Again, this is possible because the code patches saved in the data structure. In this operation, the system unapply every patch since the current state of the ontology until, but no including, the patch of the choosen SQ.
- Code to question: this command is in some way the opposite of the diff. Using this command the user specifies an axiom in the OWL code, and the system returns, if there is one, a SQ. To do this, the system searches for the axiom selected in the added code.

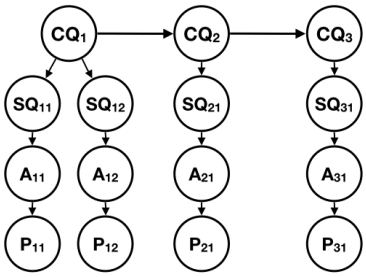


**Fig. 1.** Tracker data structure

## 5 Using the system

To test our implementation we perform some operations to check requirements, add code and use the tracker in the wine ontology. This ontology is available in the Protégé website[4].

First, we loaded the wine ontology and defined the CQ "is red wine a wine?" with the expected answer "true". Because, we expected that the ontology has the necessary axioms to answer this question with this answer. For this case, the system returns "ok", in other words, the requirement is satisfied by the ontology.

Further, we asked other CQ, but now about a type of wine that does not exist in the ontology. The CQ is "is foo wine a red wine?" and the expected answer is "true". Since the ontology did not have this knowledge the system returned an error, because the requirement was not satisfied.

Now, the system should create SQs to question the user. This process was made manually. Two SQs was creates, the first "is foo wine a wine?" and the second "does foo wine have color red?". Both question was answered with "yes", then the axioms $FooWine \sqsubseteq Wine$ and $FooWine \sqsubseteq \forall hasColor.Red$ were added to the ontology. After, if the CQ "is foo wine a red wine?" is asked again the system can answer "ok". It is interesting to notice that we did not explicitly create the subclass relation between FooWine and RedWine, but, the reasoner can infer it after the addition of the new axioms.

To check the change history, the tracker provides the log command. For now, we had:

1. is red wine a wine?
2. is foo wine a red wine?
   (a) is foo wine a wine? true.
   (b) does foo wine has color red? true.

With this information, it is possible to check the code added for each SQ using the diff command. For example, the code added for SQ 1.a was:

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

```
Class: FooWine

    SubClassOf:
        vin:Wine
```

And the code added for SQ 1.b was:

```
vin:hasColor only ({vin:Red})
```

Other useful command is the rollback. For example, if we do not want the change made by the SQ 1.b, we can rewind to the state of SQ 1.a. After the command, the code line added by the answer of SQ 1.b does not exist anymore.

We can continue asking CQs, generating SQs and answering them, and use the tracker commands to analyze all the process until all the requirements were satisfied. Further, if more requirements appear, the engineer can use the system again to ask for new questions and he can continue the process every time he needs.

## 6 Related Work

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

Methontology [8] 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 [19], 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. The Ontology 101 methodology [17] 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 (using CQs), reuse ontologies, enumerate important terms, define the classes and the class hierarchy, define the properties, define the facets of the slots, and create instances.

Other methodologies emerged since 1990. Lenat and Guha presented the steps of Cyc development in one of the first works in this area [15]. Uschold and Gruninger contribute in many papers to evolve the ontology engineering field, proposing and refining guidelines [12] [26] [25]. In 2001, the On-To-Knowledge methodology appeared, it was a result of the project with the same name [21].

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 [7]. The WebOnto tool was developed in 1997, its main innovation was the collaborative

edition of ontologies [6]. 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 [9]. In the first years of 2000, WebODE [2] and OntoEdit [22] 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, but we are presenting a method and creating a system to support some phases of iterative methodologies and to improve some characteristics in ontology development overlooked in previous works.

## 7 Conclusions and Future Work

In this paper, we presented a method, and an implementation of a system to support the process of building a DL ontology. Further, we presented a novel approach to trace requirements automatically, thus engineers can analyze the relations among requirements (CQs) and axioms. We also defined the process to build or evolve an ontology iteratively using the system implemented.

In the current implementation, we can already see some useful features that can be used during and ontology development. For example, the natural language checker to confirm if an ontology satisfies some requirements and the traceability tools provided by the tracker to analyze and navigate for all development history.

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. With the improvements of the previous components, the tracker will be even more powerful, providing a useful tool for ontology engineers.

Besides evolving the system's components, we can also integrate the whole system or some parts with popular ontology environments like Protégé and NeOn.

## References


1. Antoniol, G., Canfora, G., Casazza, G., De Lucia, A., Merlo, E.: Recovering traceability links between code and documentation. IEEE Trans. Softw. Eng. 28(10), 970–983 (Oct 2002), http://dx.doi.org/10.1109/TSE.2002.1041053
2. Arpírez, J.C., Corcho, O., Fernández-López, M., Gómez-Pérez, A.: Webode: a scalable workbench for ontological engineering. In: Proceedings of the 1st international conference on Knowledge capture. pp. 6–13. K-CAP '01, ACM, New York, NY, USA (2001), http://doi.acm.org/10.1145/500737.500743

3. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The description logic handbook: theory, implementation, and applications. Cambridge University Press, New York, NY, USA (2003)
4. 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., Yufei, Z.: D5.4.1 neon methodology for building contextualized ontology networks (Feb 2008)
5. Devedzić, V.: Understanding ontological engineering. Commun. ACM 45(4), 136–144 (Apr 2002), http://doi.acm.org/10.1145/505248.506002
6. Domingue, J.: Tadzebao and webonto: Discussing, browsing, and editing ontologies on the web. In: In Proceedings of the 11th Knowledge Acquisition for Knowledge-Based Systems Workshop (1998)
7. Farquhar, A., Fikes, R., Rice, J.: The ontolingua server: a tool for collaborative ontology construction. Int. J. Hum.-Comput. Stud. 46(6), 707–727 (Jun 1997), http://dx.doi.org/10.1006/ijhc.1996.0121
8. Fernandez-Lopez, M., Gomez-Perez, A., Juristo, N.: Methontology: from ontological art towards ontological engineering. In: Proceedings of the AAAI97 Spring Symposium. pp. 33–40. Stanford, USA (March 1997)
9. Gennari, J.H., Musen, M.A., Fergerson, R.W., Grosso, W.E., Crubézy, M., Eriksson, H., Noy, N.F., Tu, S.W.: The evolution of protege: an environment for knowledge-based systems development. Int. J. Hum.-Comput. Stud. 58(1), 89–123 (Jan 2003), http://dx.doi.org/10.1016/S1071-5819(02)00127-1
10. Gómez-Pérez, A., Fernández-López, M., Corcho, O.: Ontological Engineering: With Examples from the Areas of Knowledge Management, E-Commerce and the Semantic Web. Advanced Information and Knowledge Processing, Springer (2004), http://books.google.com.br/books?id=UjS0N1W7GSEC
11. Gotel, O.C.Z., Finkelstein, A.C.W.: An analysis of the requirements traceability problem. pp. 94–101 (1994)
12. Gruninger, M., Fox, M.S.: Methodology for the design and evaluation of ontologies (1995)
13. Guarino, N., Welty, C.: Evaluating ontological decisions with ontoclean (2002)
14. Horridge, M., Bechhofer, S.: The owl api: A java api for owl ontologies. Semant. web 2(1), 11–21 (Jan 2011), http://dl.acm.org/citation.cfm?id=2019470.2019471
15. Lenat, D.B., Guha, R.V.: Building Large Knowledge-Based Systems; Representation and Inference in the Cyc Project. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edn. (1989)
16. Mizoguchi, R.: A step towards ontological engineering. In: 12th National Conference on AI of JSAI. pp. 24–31 (1998)
17. Noy, N.F., McGuinness, D.L.: Ontology development 101: A guide to creating your first ontology (November 2008)
18. Patel-Schneider, P.F., Hayes, P., Horrocks, I.: OWL web ontology language semantics and abstract syntax. W3C recommendation, W3C (February 2004), http://www.w3.org/TR/2004/REC-owl-semantics-20040210/, published online on February 10th, 2004 at http://www.w3.org/TR/2004/REC-owl-semantics-20040210/
19. Royce, W.W.: Managing the development of large software systems: concepts and techniques. In: Proceedings of the 9th international conference on Software Engineering. pp. 328–338. ICSE '87, IEEE Computer Society Press, Los Alamitos, CA, USA (1987), http://dl.acm.org/citation.cfm?id=41765.41801

20. Shearer, R., Motik, B., Horrocks, I.: HermiT: A Highly-Efficient OWL Reasoner. In: Ruttenberg, A., Sattler, U., Dolbear, C. (eds.) Proc. of the 5th Int. Workshop on OWL: Experiences and Directions (OWLED 2008 EU). Karlsruhe, Germany (October 26–27 2008)
21. Staab, S., Studer, R., Schnurr, H.P., Sure, Y.: Knowledge processes and ontologies. IEEE Intelligent Systems 16(1), 26–34 (Jan 2001), http://dx.doi.org/10.1109/5254.912382
22. Sure, Y., Erdmann, M., Angele, J., Staab, S., Studer, R., Wenke, D.: Ontoedit: Collaborative ontology development for the semantic web. In: Proceedings of the First International Semantic Web Conference on The Semantic Web. pp. 221–235. ISWC '02, Springer-Verlag, London, UK, UK (2002), http://dl.acm.org/citation.cfm?id=646996.711413
23. Toutanova, K., Klein, D., Manning, C.D., Singer, Y.: 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. pp. 173–180. NAACL '03, Association for Computational Linguistics, Stroudsburg, PA, USA (2003), http://dx.doi.org/10.3115/1073445.1073478
24. Uschold, M.: Building ontologies: Towards a unified methodology. In: In 16th Annual Conf. of the British Computer Society Specialist Group on Expert Systems. pp. 16–18 (1996)
25. Uschold, M., Gruninger, M., Uschold, M., Gruninger, M.: Ontologies: Principles, methods and applications. Knowledge Engineering Review 11, 93–136 (1996)
26. Uschold, M., King, M.: Towards a methodology for building ontologies. In: In Workshop on Basic Ontological Issues in Knowledge Sharing, held in conjunction with IJCAI-95 (1995)