Checking the Completeness of Ontologies: A Case Study from the Semantic Web*

Valentina Cordì and Viviana Mascardi

Dipartimento di Informatica e Scienze dell'Informazione – DISI,
Università di Genova, Via Dodecaneso 35, 16146, Genova, Italy.
1996s081@educ.disi.unige.it, mascardi@disi.unige.it

Abstract. The paper discusses a formal framework for proving correctness and completeness of ontologies during its life-cycle. We have adopted our framework for the development of a case study drawn from the Semantic Web. In particular we have developed an ontology for content-based retrieval of XML documents in Peer-to-peer networks.

1 Introduction

Peer-to-peer (P2P) systems [10] have emerged as a promising new paradigm for distributed computing, as witnessed by the experience with Napster and Gnutella and by the growing number of research events related to them. Current P2P systems focus strictly on handling semantic-free, large-granularity requests for objects by identifier (typical name), which both limits their usability and restricts the techniques that might be employed to access data. Intelligent agents that exploit ontologies to perform content-based information retrieval in P2P networks may represent a viable solution to overcome the limitations of current P2P networks [11,1].

A recent proposal for a semantic, policy-based system for the retrieval of XML documents in P2P networks comes from [9], where peers are organised into thematic groups coordinated by a "super-peer agent" that exploits a "group ontology" to set the concepts managed by the group. The focus of [9] is on the architecture of the system; the engineering stages that a developer must follow in order to design, build and evaluate the group ontology are not addressed at all.

Developing an ontology is akin to defining a set of data and their structure for other programs to use. Problem-solving methods, domain-independent applications, and software agents use ontologies and knowledge bases built from ontologies as data. The engineering stages that an ontology undergoes during its life-cycle include its evaluation with respect to general and domain-dependent requirements. In particular, proving the ontology completeness and consistency is a very important step to face in order to develop correct, re-usable and maintainable ontologies. From a logical point of view, completeness is a property associated with combining a procedure for constructing well-formed formulas, a definition of truth that relates to interpretations and models of logical systems, and a proof procedure that allows new well-formed formulas to be

* Parts of this document appear in [3].

derived from old ones. A logical system is logically complete if every true well-formed formula can be derived. The other side to logical completeness is consistency. If falsity can be derived, then any well-formed formula can be derived, so trivially all true well-formed formulas can be derived.

When talking about ontologies, completeness and consistency assume a different meaning, although the conceptual relation with their logical counterparts is usually respected. While the meaning of consistency w.r.t. ontologies is pretty simple – the ontology should not contain conflicting information – there are different definitions of ontological completeness.

According to Colomb and Weber [2], an information system has the *potential* of being "ontologically complete" if it matches the social reality of the organisation in which the system is embedded. The potential for completeness, which is analogous to logical and computational completeness, has been called "ontological adequacy" by Guarino [6]. Colomb and Weber propose a set of guidelines for checking the ontological completeness of information systems. Fox and Grüninger [4] define the "functional completeness" of an ontology as its ability to represent the information necessary for a function to perform its task. They also propose a set of theorems that state under which conditions an ontology is complete [5].

All the authors that deal with the problem of checking the completeness of an ontology w.r.t. its requirements, agree that this check should be designed in such a way to be easily automatised and computationally tractable. In this paper, we provide a notion of completeness based on [5] but simpler than that, and whose check can be partially automatised. Both the notion we propose and the framework for proving the completeness of ontology we have developed are based on computational logic.

We have adopted our framework for the development of a case study drawn from the Semantic Web, where proving the completeness of an ontology can be crucial for safety and security reasons.

The structure of the paper is the following: Section 2 introduces the case study based on [9]. Section 3 introduces some techniques from the literature and then explains our formal framework for proving completeness of ontologies. Section 4 shows the development of the case study emphasising the ontology evaluation by means of our framework. Conclusions follow.

2 The case study: describing and retrieving XML documents

To show how our formal framework for proving the completeness of ontologies works, we consider a scenario simpler than that for which we need to develop the "real" ontology, namely the P2P network described in [9]. There, peers are organised into thematic groups, each one coordinated by a "super-peer agent". The super-peer agent provides an ontology ("group ontology") that sets the concepts dealt with by the group and establishes the relationships among them. Each peer can dynamically enter and leave any group inside the P2P network. When the peer joins a group for the first time, it is requested to provide to the super-peer agent as much information as possible about the concepts that are dealt with by the documents it is willing to share. This allows the super-peer agent to know which peers are more likely to deal with which concepts.

When a query is submitted to a peer, the peer forwards it to the super-peer which can understand the meaning of the terms appearing in the query by exploiting the group ontology. Since the super-peer knows which peers deal with which concepts, it identifies the peers in the group that can contain an answer for the query and forwards the query only to them, in order to minimise the number of messages exchanged inside the group.

The simplified scenario that we consider involves the development of a system able to support the automatic classification of XML documents retrieved from the network (just a binary classification: "is the XML document talking about a given topic or not?"). In particular the case study faces the development of an ontology for structuring the knowledge about XML documents talking about movies. In this simplified scenario, formally demonstrating the completeness of the ontology is not a very critical issue and we perform this demonstration mainly for illustrative purposes. Nevertheless, there are many real situations drawn from the Semantic Web domain where this formal proof *must* be carried out for safety and security reasons.

The knowledge about XML documents talking about movies is based on:

1. the semantics of tags that appear in the XML document, and
2. the XML document structure.

For example, both documents in Table 1 describe a movie, even if they are characterised by different structure and different tags.

```
<movie>
  <title>Title1<title>
  <actors><actor>Act1</actor>
          <actor>Act2</actor>
  </actors>
  <directed_by><name>Name</name>
         <surname>Surn</surname>
  </directed_by>
</movie>
```

```
<film>
  <title>Title2<title>
  <actors>Act3, Act4</actors>
  <director>Dir</director>
</film>
```

Table 1. Two XML documents dealing with movies

As far as tags are concerned, the first document uses `<movie>` to refer to a movie, while the second document uses `<film>`. In this context, the semantics of "movie" and "film" is the same. The director is identified by the tag `<directed_by>` in the first document, and by the tag `<director>` in the second one. Again, despite to their syntactic difference, these two tags have the same semantics.

As far as the structure is concerned, the tag `<actors>` is structured into a list of `<actor>` and the tag `<directed_by>` is composed by `<name>` and `<surname>` in the first document, while the corresponding tags in the second document contain strings.

The prototypical ontology must contain all the information needed to classify XML documents talking about movies. In particular, it must contain the information that:

– the tags <film> and <movie>, and <director> and <directed_by> represent the same concepts in the movie context;
– <actors> can contain a string or a list of <actor> tags;
– the director, be it identified by <directed_by> or by <director>, may contain a string or a structure including <name> and <surname>.

In order to build the ontology, we retrieved a set of existing XML documents dealing with movies from the web and we manually analysed each of them in order to identify the structural and the semantic rules exemplified above. Some documents we used for our purposes are http://catcode.com/cit041x/assignment4a.html, http://www-db.stanford.edu/pub/movies/mains218.xml, and http://www.flixml.org/flixml/detour.xml.

The purpose was to build an ontology which could tell that a document starting with the tag <film> or <movie> (and others, that we do not discuss here), and containing somewhere a tag <director> or <directed_by>, possibly with different content, is likely to talk about movies. Given a new XML document, the ontology should allow to answer "yes, it talks about movies because it matches the semantic and structural rules" or "no, it does not talk about movies".

3 A formal framework for proving completeness of ontologies

Our formal framework is based on the work on TOVE by Grüninger and Fox [5]. TOVE is a methodology based on experiences in the development of TOVE (Toronto Virtual Enterprise).

The TOVE approach to ontology development starts with the definition of the motivating scenarios that arise in the applications. Such scenarios may be presented by industrial partners as problems which they encounter in their enterprises. The motivating scenarios often have the form of story problems or examples which are not adequately addressed by existing ontologies.

Given the motivating scenarios, a set of queries will arise which place demands on an underlying ontology. These queries can be considered as the requirements that are in the form of questions that an ontology must be able to answer. These are called the *informal competency questions*, since they are not yet expressed in the formal language of the ontology.

Once informal competency questions have been defined, they should be restated using some formal language suitable for expressing the ontology terminology. This activity is carried out manually. The ontology terminology must be able to correctly and easily represent the objects in the domain of discourse as constants and variables in the language. Attributes of objects may be defined by unary predicates; relations among objects may be defined using n-ary predicates. The two languages that Grüninger and Fox suggest for expressing both the ontology terminology and the formal competency questions are first-order logic and KIF [13].

In [4], the concepts of competency and completeness of an ontology are informally stated:

Given a properly instantiated model of an enterprise and an accompanying theorem prover (perhaps Prolog or a deductive database), the competence of an ontology is the set of queries that it can answer. [...]
The Functional Completeness of an ontology is determined by its competency, i.e., the set of queries it can answer with a properly instantiated model. Given a particular function (application), its enterprise modelling needs can be specified as a set of queries. If these queries can be "reduced to"[1] *the set of competency questions specified for the chosen ontology, then the ontology is sufficient to meet the modelling needs of the application.*

This informal statement corresponds to the formal definition provided by the completeness theorems discussed in [5]. These theorems have one of the following forms, where $T_{ontology}$ is the set of axioms in the ontology, T_{ground} is a set of ground literals (instances), Q is a first-order sentence specifying the query in the competency question, and Φ is a set of first-order sentences defining the set of conditions under which the solutions to the problem are complete:

- $T_{ontology} \cup T_{ground} \models \Phi$ if and only if $T_{ontology} \cup T_{ground} \models Q$.
- $T_{ontology} \cup T_{ground} \models \Phi$ if and only if $T_{ontology} \cup T_{ground} \cup Q$ is consistent.
- $T_{ontology} \cup T_{ground} \cup \Phi \models Q$ or $T_{ontology} \cup T_{ground} \cup \Phi \models \neg Q$.
- All models of $T_{ontology} \cup T_{ground}$ agree on the extension of some predicate P.

Completeness theorems can also provide a means of determining the extendibility of an ontology, by making explicit the role that each axiom plays in proving the theorem. Any extension to the ontology must be able to preserve the completeness theorems.

Starting from Grüninger and Fox's definitions, and integrating suggestions coming from other methodologies such as EXPLODE [7], "A Guide to Creating your First Ontology" [12] (in the following identified by *OD101* for readability), and Uschold's "Unified Method" [14] (in the following identified by *UniMeth*), we define our guidelines for developing a complete ontology. UniMeth embraces TOVE and the Enterprise methodology [15] in a unique framework. For this reason, in the following we will refer to UniMeth instead of specifically referring to TOVE.

The engineering stages that an ontology developer should follow according to our integrated approach, fully discussed in [3], are:

- **Domain analysis.** This development stage can be faced by answering the questions that EXPLODE, OD101 and UniMeth suggest, such as which are the expected users of the methodology and which are the ontology domain and extended purpose. UniMeth also suggests to identify fairly general scenarios and use them to help clarify specific uses of the ontology.
- **Requirement definition.** EXPLODE, OD101 and UniMeth all suggest to identify the competency questions. Besides competency questions, UniMeth allows the developer to use other techniques for the extraction of the ontology requirements such as defining the detailed motivating scenarios, brainstorming and trimming.

[1] By reducible, Fox and Grüninger mean that the questions can be re-written using the objects provided by the chosen ontology.

EXPLODE also suggests to clearly identify the specific constraints from the hardware/software system that come from other modules in the system that interact with the ontology.

- **Informal specification of the ontology.** This step can be faced by identifying the most important terms of the ontology and using an ontology-editing environment to graphically represent the ontology concepts and the relations among them.
- **Formal specification of the ontology.** Following UniMeth, in this step we suggest to use definite Horn Clauses as the formal language for defining the ontology. We refer to the set of Horn Clauses specifying the ontology as $Progr_{ontology}$.
- **Testing, validation, verification.** The primary validation technique that all the methodologies support consists of informally checking the ontology against the competency questions. By performing this check, it may be realised for example that some motivating scenarios were not correctly addressed. Previous choices can then be adjusted and corrected.
- **Completeness check.** According to UniMeth and to our suggestion for formally specifying the ontology, the developer should manually restate the informal competency questions as goals (negative Horn clauses) and should demonstrate that for each competency question restated as a goal, Q_{Goal}, there exists a refutation for $Progr_{ontology} \cup Q_{Goal}$ [8]. Obviously, this can be automatised by using any Prolog interpreter or compiler to demonstrate that the goal Q_{Goal} succeeds if called within the Prolog program $Progr_{ontology}$. In this sense, the approach to completeness check that we propose is "partially automatised": once the ontology and the informal competency questions have been manually restated as Prolog programs and goals, the completeness check can be performed in a completely automatic way.
- **Other engineering steps.** Before the goal of developing an ontology can be considered achieved, other engineering steps must be faced besides the demonstration of its completeness. EXPLODE, OD101 and UniMeth suggest to face:
  - the development of intermediate prototypes;
  - the iterative refinement of previous choices, according to the outcomes of the completeness check and of the prototype execution;
  - the implementation of a machine-readable ontology;
  - the meetings with clients to perform an iterative check; and
  - the production of documentation on the ontology and on its development process.

  These steps are not a central issue in this paper, so we will not face them. The reader can refer to [3] for details.

4 Developing a complete ontology for the retrieval of XML documents on movies

In this section we discuss the stages – from the domain analysis to the check of the ontology completeness – that we followed to develop the ontology introduced in Section 2.

– **Domain analysis.**
Since the ontology under development is just a toy-example, the answers to the questions suggested by OD101 and UniMeth for analysing the domain are not very meaningful: there are no expected users of the methodology, the domain is the one described in Section 2, and the intended purpose of the ontology is to provide a test-bed for evaluating our approach to completeness check. UniMeth suggests to identify fairly general scenarios and use them to help clarify specific uses of the ontology. For example, being able to recognise both documents in Table 1 as documents dealing with movies is a general motivating scenario for our ontology.
– **Requirement definition.**
We have identified the following competency questions for our ontology.
1. **Competency Question:** Should the ontology be able to separate the concepts related to the document structure from those related to the document semantics?
Expected answer: Yes, it should. This separation is very important because it will allow re-using the ontology to classify documents in domains different from movies, only requiring an extension to the ontology concepts related with the document semantics.
2. **Competency Question:** What are the syntactic equivalent representations of the tag "title" in the context of tag "heading"?
Expected answer: The representations of "title" in the context of "heading" are 't', 'Title', 'title', 'TITLE', 'movieTitle', 'titleMovie'.
3. **Competency Question:** What is the meaning of the tag "title" in the context of the tag "heading"?
Expected answer: The meaning is the "title" element (which is different from the "title" tag).
4. **Competency Question:** Can the tag "actors" contain either a string or a list of "actor" tags?
Expected answer: yes, it can.
5. **Competency Question:** Can the information about the title of the film be an attribute of the tag "movie"?
Expected answer: yes, it can.

Besides using the competency questions, UniMeth suggests to define the detailed motivating scenarios that include possible solutions to the problem addressed by the ontology. A motivating scenario for our ontology is that it must be able to classify the documents whose fragments are shown in Tables 2 and 3, as well as other documents that we downloaded from `http://www-db.stanford.edu/pub/movies/` and `http://catcode.com/cit041x/assignment4a.html`, as movie documents.
EXPLODE suggests to clearly identify the specific constraints from the hardware/software system that come from other modules in the system that interact with the ontology. Our ontology will be used by intelligent agents that help the peers in a P2P network in deciding which of the XML documents they are willing to share deal with movies, and which do not. The modules that the ontology will interact with are those described in [9]. Assuming that all the documents shared by peers in a P2P network have a common structure is not realistic: peers share documents

```
<title role="main"> Detour </title>
<releaseyear role="initial"> 1945 </releaseyear>
<language> English </language>
<studio> PRC (Producers Releasing Corporation) </studio>
<cast> <leadcast>
     <male id="TN"> T. Neal <role> Al </role> </male>
     <female id="AS"> A. Savage <role> Vera </role> </female>
   </leadcast>
   <othercast> <male> .... </male> ....
   </othercast> </cast>
<crew><director>Edgar G. Ulmer</director> ....
```

Table 2. A fragment of `http://www.flixml.org/flixml/detour.xml`

```
<fid> SMg10 </fid>
<t> Bridget Jones's Diary </t>
<year> 2001 </year>
<dirs> <dir> <dirk> R </dirk><dirn> NancyMeyer </dirn> </dir>
```

Table 3. A fragment of `http://www-db.stanford.edu/pub/movies/mains218.xml`

characterised by very different structures and very different tags. An ontology that tries to conciliate these differences can prove extremely useful in this context. In order to be used in a real P2P application, our ontology should be extended to deal with other subjects besides movies (hence, the requirement that the syntactic and the semantic aspects are clearly separated in the ontology).

– **Informal specification of the ontology.**
  Following OD101, we identified the most important terms of the ontology. For example, the terms "tag", "attribute", "movie", "title", "year" must be represented. Afterwards, we used an ontology-editing environment to graphically represent the ontology concepts and the relations among them.
  Figure 1 represents the hierarchy of concepts that belong to our ontology. The ontology was edited using Protégé 2.0, a drawing tool developed by the Stanford University.
  Note that semantic aspects are separated from syntactic ones. The former are collected under the general concept "Element", while the latter are collected under the "Tag" concept. The relationship between tags and elements is that a tag has a context, which may be the root of the document or another tag, and a meaning, which is an element.
  The hierarchy of concepts alone is not enough informative. In order to make the ontology useful and complete with respect to its requirements, we had to describe the internal structure of concepts. For example, Figure 2 shows the attributes of the concept Movie.
– **Formal specification of the ontology.**
  The ontology graphically represented in Figures 1 and 2 can be also represented using definite Horn clauses.

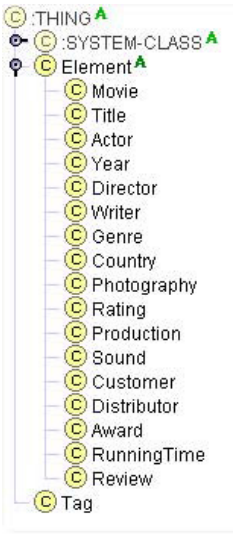

Fig. 1. Concept hierarchy

Movie (type=:STANDARD-CLASS)

Name: Movie

Documentation

Constraints

Role: Concrete

Template Slots

Name	Type	Cardinality	Other Facets
title	Class	required single	parents={Title}
actor	Class	multiple	parents={Actor}
year	Class	required single	parents={Year}
director	Class	required single	parents={Director}
writer	Class	required single	parents={Writer}
genre	Class	single	parents={Genre}
country	Class	single	parents={Country}
photography	Class	single	parents={Photography}
rate	Class	single	parents={Rating}
production	Class	required single	parents={Production}
sound	Class	single	parents={Sound}
customer	Class	single	parents={Customer}
distributor	Class	required single	parents={Distributor}
award	Class	multiple	parents={Award}
runningTime	Class	single	parents={RunningTime}
review	Class	multiple	parents={Review}

Fig. 2. The attributes of the concept “Movie”

We can represent the hierarchy of concepts in a standard way by means of the `isA` relation as shown in Table 4. We adopt a Prolog-like syntax for Horn clauses; text preceded by one or more "%" is a comment.

```
%%% THING CONCEPT %%%
isA(tag, thing).
isA(element, thing).

%%% ELEMENT CONCEPT %%%%
isA(movie, element).
isA(title, element).
isA(actor, element).
isA(year, element).
.........
```

Table 4. Formal specification of the ontology: isA relation

The instances of a concept can be defined by means of an `instanceOf` relation which has an instance of a concept and the concept to which the instance belongs as its arguments. Instances are represented by the functor `instance` plus a set of arguments which represents the attributes of the instance. For example, tags are identified by an atom (the tag identifier), another atom (the identifier of the tag context[2]), an element (the tag meaning) and a list of strings (all the possible syntactic representations of the tag inside the XML document). As shown by the first clause of Table 5, an instance of the `title` tag may have an `heading` context (2nd argument, this argument must be an instance of a tag), the `title` meaning (3rd argument; this argument must be an element), and the list of `['t', 'Title', 'title', 'TITLE', 'movieTitle', 'titleMovie']` syntactic representations (4th argument).

Another instance of the `title` tag may have the same arguments as the previous one except for the context, which may be `movie` (second clause of Table 5). This means that two XML documents where the tag `title` appears as a sub-element or an attribute of either the `heading` tag or the `movie` tag can be both considered documents that represent movies. Other examples of instances are included in Table 5.

Definite Horn clauses can be also used to express consistency constraints on the structure of the ontology. For example, the clause shown in Table 6 is an axiom stating that the context of a tag must be a tag and that the semantics of a tag must be an element.

– **Testing, validation, verification.**
The primary validation technique that all the methodologies support consists of checking the ontology against the informal competency questions. One of the com-

[2] The tag A is in the context of the tag B if either A is an attribute of B or if it is a sub-element of B.

```
%%% TAG TITLE %%%
instanceOf(instance(title, heading, title,
[’t’, ’Title’, ’title’, ’TITLE’,
’movieTitle’, ’titleMovie’]), tag).

instanceOf(instance(title, movie, title,
[’t’, ’Title’, ’title’, ’TITLE’,
’movieTitle’, ’titleMovie’]), tag).

%%% TAG ACTOR %%%
instanceOf(instance(actor, actors, actor,
[’actor’, ’ACTOR’, ’Actor’]), tag).

%%% TAG ACTORS %%%
instanceOf(instance(actors, credits, actor,
[’actors’, ’ACTOR’, ’Actor’, ’cast’, ’Cast’, ’CAST’]), tag).

.......
```

Table 5. Formal specification of the ontology: `instanceOf` relation

```
instanceOf(instance
              (TagId, TagContext,
               TagSemantics, TagSyntax),
           tag) :-
instanceOf(instance(TagContext, _, _, _), tag),
isA(TagSemantics, element).
```

Table 6. Formal specification of the ontology: consistency axioms

petency questions we identified for the ontology is: "*What are the syntactic representations of the tag* `title` *in a document dealing with movies?*" The expected answer, based on the set of real XML documents we used as our training set, is: "*The syntactic representations of the tag* `title` *are:* `movieTitle`, `titleMovie`, `t`, `Title`, `TITLE`." Figure 3 shows that all these representations are considered by the ontology. By checking the ontology, we realised that some motivating sce-

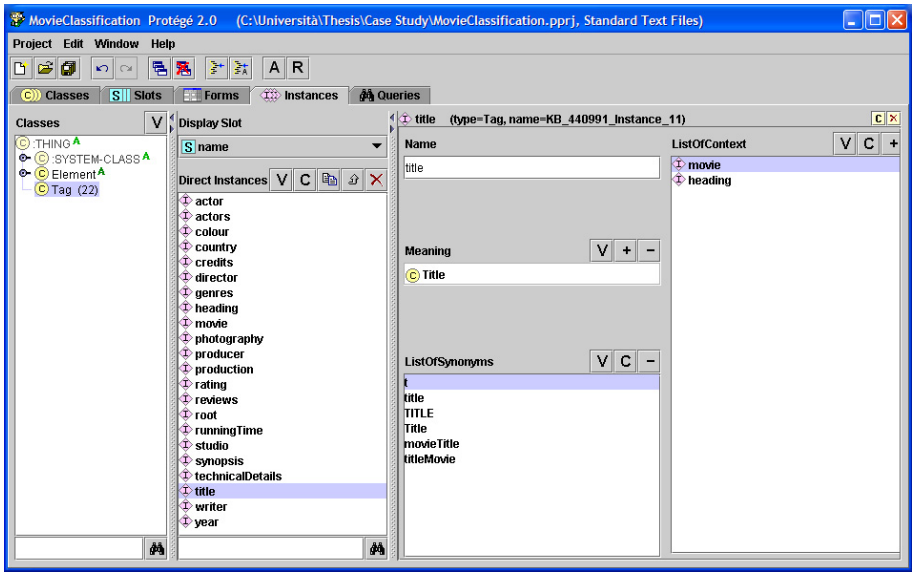

Fig. 3. One instance of the ontology

narios were not correctly addressed. In particular, the ontology did not include the information that `dirn` may be used as an alternative syntactic representation of the concept of `director`, which is indeed necessary to correctly classify the document in Table 3. Thanks to this testing, verification and validation stage we got feedback useful to refine the definition of the ontology.

– **Completeness check.**

The last ontology development stage that we take under consideration in this paper is the completeness check. In order to face this stage, we restate all the informal competency questions as negative Horn clauses. Since we are going to use a Prolog interpreter to check whether or not these goals can be demonstrated starting from the Prolog program $Progr_{ontology}$ partly shown in Tables 4, 5, and 6, we take advantage of standard Prolog predicates to express conditions such as *X is not a variable* (`nonvar(X)`) and *X cannot be unified with Y* (`X \= Y`). An underscore (`_`) is used for unnamed variables for which no binding is required.

Below, we show how each competency question introduced in the "Requirement definition" stage can be expressed as a negative Horn clause Q_{Goal}, and which answer is computed by the Sicstus Prolog interpreter for $Progr_{ontology} \cup Q_{Goal}$. It is easy to see that the answers we got are consistent with the answers we expected, thus demonstrating the completeness of our ontology with respect to its requirements. By issuing a "`;`" command to the Sicstus Prolog interpreter after it returns one unification for the goal variables we force the interpreter to look for more answers. A `no` means that no more answers were found.

1. **Competency Question:** Should the ontology be able to separate the concepts related to the document structure from those related to the document semantics?
   Corresponding negative Horn clause:
   ```
   :- isA(tag, thing), isA(element, thing).
   ```
   Answer provided by the Sicstus Prolog interpreter:
   ```
   yes
   ```
2. **Competency Question:** What are the syntactic representations of tag "title" in the context of tag "heading"?
   Corresponding negative Horn clause:
   ```
   :- instanceOf(instance(title, _, _, SyntRepr), tag).
   ```
   Answer provided by the Sicstus Prolog interpreter:
   ```
   SyntRepr = [t, 'Title', title, 'TITLE' , movieTitle,
   titleMovie] ? ;
   SyntRepr = [t, 'Title', title, 'TITLE', movieTitle,
   titleMovie] ? ;
   no
   ```
   Here two answers are provided: one for the case the tag "title" is in the context of the tag "heading", and one for the case it is in the context of the tag "movie".
3. **Competency Question:** What is the meaning of the tag "title" in the context of the tag "heading"?
   Corresponding negative Horn clause:
   ```
   :- instanceOf(instance(title, heading, Meaning, _),
   tag).
   ```
   Answer provided by the Sicstus Prolog interpreter:
   ```
   Meaning = title ? ;
   no
   ```
4. **Competency Question:** Can the tag "actors" contain either a string or a list of "actor" tags?
   Corresponding negative Horn clause:
   ```
   :- instanceOf(instance(actor, actors, actor, _), tag),
   instanceOf(instance(actors, _, actor, _), tag).
   ```
   Answer provided by the Sicstus Prolog interpreter:
   ```
   yes
   ```
5. **Competency Question:** Can the information about the title of the film be an attribute of the tag "movie"?
   Corresponding negative Horn clause:
   ```
   :- instanceOf(instance(title, movie, _, _), tag).
   ```

Answer provided by the Sicstus Prolog interpreter:

```
yes
```

In this way we have demonstrated that our ontology is able to answer all the competency questions, moreover these answers are consistent with the XML documents retrieved from the web and used as motivating scenario during the development of the ontology.

5 Conclusions and future directions

In this paper we have outlined a methodology for developing ontologies which takes inspiration from three existing methodologies, namely OD101, UniMeth and EXPLODE. In particular, we have concentrated our efforts in the stage of checking the ontology completeness. Consistently with the existing literature on the topic [5,4], we suggest that the ontology developer performs the completeness check by formally defining the ontology as a set of definite Horn clauses (a Prolog program) and by stating the competency questions as negative Horn clauses (Prolog goals). The developer should then check that, for each competency question restated as a negative Horn clause, a refutation exists for the defined ontology and the competency question. From a practical point of view, this check can be carried out by means of any Prolog interpreter. We have used an example taken from the Semantic Web domain to illustrate our approach.

The main future direction of our work consists of the extension of the ontology for representing and retrieving XML documents in order to cope with other domains besides to "movie" one. The integration of such extended ontology into a prototypical peer-to-peer network implementing the ideas of [9] will demonstrate the suitability of our approach in a real scenario.

References

1. A. Castano, S. Ferrara, S. Montanelli, E. Pagani, and G. P. Rossi. Ontology-addressable contents in P2P networks. In *Proc. of the 1st SemPGRID Workshop*, 2003.
2. R. M. Colomb and R. Weber. Completeness and quality of an ontology for an information system. In N. Guarino, editor, *Proc. of FOIS'98*, pages 207–217. IOS-Press, 1998. `http://www.itee.uq.edu.au/~colomb/Papers/Ontology.html`.
3. V. Cordì, V. Mascardi, M. Martelli, and L. Sterling. Developing an ontology for the retrieval of xml documents: A comparative evaluation of existing methodologies. In *Proc. of AOIS'04*, 2004. `http://www.disi.unige.it/person/MascardiV/Download/CMMS04.pdf.gz`.
4. M. S. Fox and M. Gruninger. On ontologies and enterprise modelling. In *International Conference on Enterprise Integration Modelling Technology 97*. Springer-Verlag, 1997.
5. M. Gruninger and M. S. Fox. Methodology for the design and evaluation of ontologies. In *Proc. of the Workshop on Basic Ontological Issues in Knowledge Sharing*, 1995. `http://www.eil.utoronto.ca/enterprise-modelling/papers/gruninger-ijcai95.pdf`.
6. N. Guarino. Formal ontology, conceptual analysis and knowledge representation. *International Journal of Human Computer Studies*, 5/6(43):625–640, 1995.

7. M. Hristozova and L. Sterling. Experiences with ontology development for value-added publishing. In S. Cranefield, T. Finin, V. Tamma, and S. Willmott, editors, *Proc. of the OAS03 Workshop*, 2003. http://oas.otago.ac.nz/OAS2003/papers/OAS03-hristozova.pdf.
8. J. W. Lloyd. *Foundations of Logic Programming (2nd edition)*. Springer-Verlag, 1993.
9. M. Mesiti, G. Guerrini, and V. Mascardi. SAPORE P2P: A semantic, policy-based system for the retrieval of XML documents in P2P networks. Tech. rep., DISI-TR-04-05, Computer Science Department of Genova University. http://www.disi.unige.it/person/MascardiV/Download/MGM04.pdf.gz, 2004.
10. D. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-peer computing. Tech. rep., HPL-2002-57, HP Labs Palo Alto., 2002.
11. W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M. Schlosser, I. Brunkhorst, and A. Löser. Super-peer-based routing and clustering strategies for RDF-based peer-to-peer networks. In *Proc. of WWW'03*, pages 536–543, 2003.
12. N. F. Noy and D. L. McGuinness. Ontology development 101: A guide to creating your first ontology. Tech. rep., KSL-01-05, Stanford Knowledge Systems Laboratory., 2001.
13. The Knowledge Interchange Format Home Page. http://www-ksl.stanford.edu/knowledge-sharing/kif/.
14. M. Uschold. Building ontologies: Towards a unified methodology. In *Proc. of ES'96*, 1996. Also published as technical report, AIAI-TR-197. http://www.aiai.ed.ac.uk/project/ftp/documents/1996/96-es96-unified-method.ps.
15. M. Uschold and M. King. Towards a methodology for building ontologies. In *Proc. of the Workshop on Basic Ontological Issues in Knowledge Sharing*, 1995. Also published as technical report, AIAI-TR-183. http://www.aiai.ed.ac.uk/project/ftp/documents/1995/95-ont-ijcai95-ont-method.ps.