A methodology to create ontology-based information retrieval systems

José Saias and Paulo Quaresma

Departamento de Informática,
Universidade de Évora,
7000 Évora, Portugal
jsaias|pq@di.uevora.pt

Abstract. Modern information retrieval systems need the capability to reason about the knowledge conveyed by text bases.
In this paper a methodology to automatically create ontologies and class instances from documents is proposed. The ontology is defined in the OWL semantic web language and it is used by a logic programming framework, ISCO, to allow users to query the semantic content of the documents. ISCO allows an easy and efficient integration of declarative, object-oriented and constraint-based programming techniques with the capability to create connections with external databases.

1 Introduction

Modern information retrieval systems need the capability to represent and to reason with the knowledge conveyed by text bases. This knowledge can be represented through the use of ontologies. In fact, ontologies allow the definition of class hierarchies, object properties, and relation rules, such as, transitivity or functionality. Using this knowledge it is possible to define instances of classes, to associate them with documents, and to make inferences about them.

OWL (Ontology Web Language) is a language proposed by the W3C consortium (http://www.w3.org [9]) to be used in the "semantic-web" environment for the representation of ontologies. This language is based in the previous DAML+OIL (Darpa Agent Markup Language - [10]) language and it is defined using RDF (Resource Description Framework - [5]).

In this paper a methodology to automatically create an OWL ontology and OWL class instances from a set of documents is proposed. The methodology is based on natural language processing techniques, namely, a syntactical parser and a semantic analyzer able to obtain a partial interpretation of the documents. Similar approaches aiming to create a daml+oil/owl ontology were presented in [7, 8]. These approaches presented preliminary work in this area but they did not propose a general methodology for the creation of OWL ontologies and the enrichment of documents with OWL instances.

After the creation of the OWL ontology, documents are enriched with instances of classes and a logic programming based framework is used to support inferences over them. The logic programming framework is based on ISCO [1]. ISCO is a new declarative language implemented over GNU Prolog with object-oriented predicates, constraints and allowing simple connections with external databases.

Section 2 describes the natural language processing techniques used to create the OWL ontology. 3 describes the NLP techniques used to create the OWL instances associated with each document. Section 4 describes ISCO, the logic programming framework. Section 5 provides an example of interaction. Finally, in section 6 some conclusions and future work are pointed out.

2 OWL ontology creation

In order to be able to deal with documents from different domains, a methodology to automatically create basic ontologies of concepts is proposed. This methodology allows the definition of a base ontology with the relevant concepts but having few hierarchical relations. After having defined this ontology, it may be necessary to develop manual work by human experts in order to fully organize the set of extracted concepts.

The methodology to automatically obtain an ontology of concepts is based on the output of natural language processing tools:

– Text syntactical parsing. The documents are analysed by the parser developed by E. Bick in the domain of the VISL project[1] [2]. This parser is available for 21 different languages, namely for the Portuguese language.
– Partial semantic analysis.
– Entities extraction. From the semantic analysis output, entities are extracted and represented by ontology classes.

2.1 Syntactical analysis

The syntactical parser developed by E. Bick in the domain of the VISL project is based in the Constraint Grammars formalism and it is able to cover a large percentage of the Portuguese language. However, its output is in a non-standard format and it was necessary to transform it into a structured form, like XML and Prolog terms. A translation tool from the VISL output into XML and Prolog terms was developed and it is available to the VISL users (a detailed description of this tool was presented in [3]).

As an example, suppose the following sentence:

O bombeiro salvou a criança. *The fireman saved the child.*

This sentence has the VISL output:

[1] http://visl.hum.sdu.dk/visl

```
STA:fcl
SUBJ:np
=>N:art('o' M S)        O
=H:n('bombeiro' M S)    bombeiro
P:v-fin('salvar' PS 3S IND)     salvou
ACC:np
=>N:art('a' F S)        a
=H:n('crianca' F S)     crianca
```

As it can be seen, the subject, predicate and direct object were correctly parsed. From this output, the XML translator produces three files:

1. The first file links each word with a *word* tag with a specific *id*.

```
<!DOCTYPE words SYSTEM "words.dtd">
<words>
<word id="word_1">O</word>
<word id="word_2">bombeiro</word>
<word id="word_3">salvou</word>
<word id="word_4">a</word>
<word id="word_5">crianca</word>
<word id="word_6">.</word>
</words>
```

2. The second file associates each *word* with its part-of-speech information.

```
<!DOCTYPE words SYSTEM "wordsPOS.dtd">
<words>
<word id="word_1">
<art canon="o" gender="M" number="S"/>
</word>
<word id="word_2">
<n canon="bombeiro" gender="M" number="S"/>
</word>
<word id="word_3">
<v canon="salvar">
<fin tense="PS" person="3S" mode="IND"/>
</v>
</word>
<word id="word_4">
<art canon="a" gender="F" number="S"/>
</word>
<word id="word_5">
<n canon="crianca" gender="F" number="S"/>
</word>
</words>
```

3. The third file has the parsing structure.

```
<!DOCTYPE text SYSTEM "text_ext.dtd">
<text>
<paragraph id="paragraph_1">
<sentence id="sentence_1" span="word_1..word_6">
```

```
<chunk id="chunk_1" ext="subj" form="np"
                               span="word_1..word_2">
<chunk id="chunk_2" ext="n" form="art" span="word_1">
</chunk>
<chunk id="chunk_3" ext="h" form="n" span="word_2">
</chunk>
</chunk>
<chunk id="chunk_4" ext="p" form="v_fin" span="word_3">
</chunk>
<chunk id="chunk_5" ext="acc" form="np"
                              span="word_4..word_5">
<chunk id="chunk_6" ext="n" form="art" span="word_4">
</chunk>
<chunk id="chunk_7" ext="h" form="n" span="word_5">
</chunk>
</chunk>
</sentence>
</paragraph>
</text>
```

2.2 Semantic analysis

Each syntactical structure is translated into a First-Order Logic expression. The technique used for this analysis is based on DRS's (Discourse Representation Structures [4]). The partial semantic representation of a sentence is a DRS built with two lists, one with the rewritten sentence and the other with the sentence discourse referents.

At present, we are only dealing with a very restricted semantic analysis and we only try to represent predicates with their subjects and direct objects.

From the XML structure, using XSL transformations, it is possible to obtain the semantic representation of each sentence.

The semantic representation of the example presented in the previous subsection is:

sentence(doc1, [fireman(A), child(B), save(A,B)], [ref(A), ref(B)]).

This structure represents an instance of a fireman A and an instance of a child B which are related by the action *to save*.

A general tool able to obtain similar semantic partial representations for every sentence was developed and it was applied to the full set of legal documents of the Portuguese Attorney General's Office (7000 documents).

2.3 Entities extraction

From the sentence semantic representation, entities are extracted and they are the basis for the creation of an ontology of concepts. In fact, for each new concept, a new class, subclass of the *Entity* class, is created.

In the referred example it would be possible to extract the following entities:

– bombeiro *fireman*
– salvar *to save*
– criança *child*

These three entities would cause the creation of correspondent ontology classes.

Figure 1 shows a graphical view of the top-level hierarchy:

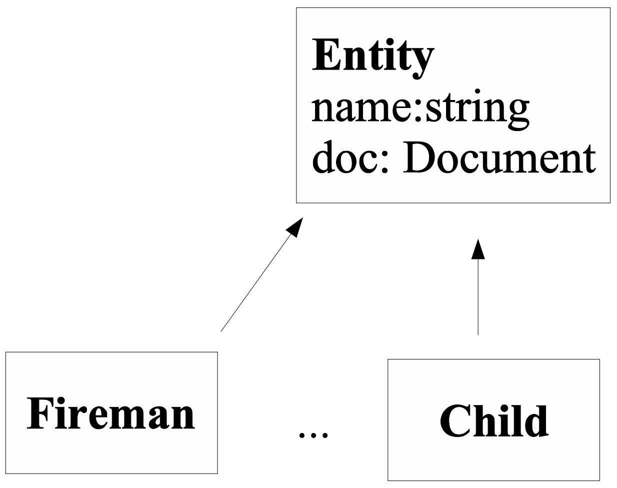

Fig. 1. Top-level entity hierarchy

As it can be seen from the figure, the proposed procedure for the automatic creation of ontologies is only able to obtain a simple two-level hierarchy of concepts. At present, manual intervention is needed to refine this ontology and to represent more complex structures.

As an example, we present the OWL code correspondent to these concepts:

```
<owl:Class rdf:ID="Entity">
</owl:Class>

        <owl:DataTypeProperty rdf:ID="name">
          <rdfs:domain rdf:resource="#Entity"/>
          <rdf:type rdf:resource="&owl;FunctionalProperty"/>
          <rdfs:range rdf:resource="&xsd;String"/>
        </owl:DataTypeProperty>

        <owl:ObjectProperty rdf:ID="entDoc">
          <rdf:type rdf:resource="&owl;FunctionalProperty" />
          <rdfs:domain rdf:resource="#Entity" />
          <rdfs:range rdf:resource="#Document" />
        </owl:ObjectProperty>
```

```
<owl:Class rdf:ID="child">
  <rdfs:subClassOf rdf:resource="&pgr;Entity" />
</owl:Class>

<owl:Class rdf:ID="fireman">
  <rdfs:subClassOf rdf:resource="&pgr;Entity" />
</owl:Class>
```

3 OWL instance

After having defined an ontology of classes, it is necessary to extract and to represent instances of those classes and to associate them with documents.

This association is presently done via a new class *Action*, which relates subjects, predicates, and direct objects with specific documents. The overall architecture is presented in figure 2.

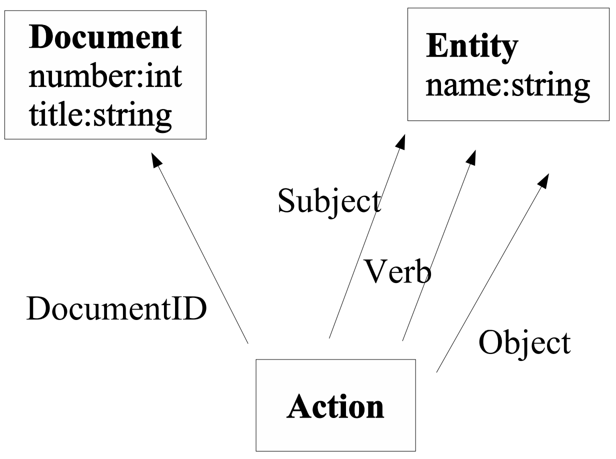

Fig. 2. Top-level classes

The following OWL code defines class *Document* with two properties: *number* and *title*.

```
<owl:Class rdf:ID="Document">
</owl:Class>

        <owl:DataTypeProperty rdf:ID="number">
          <rdfs:domain rdf:resource="#Document"/>
          <rdf:type rdf:resource="&owl;FunctionalProperty"/>
          <rdfs:range rdf:resource="&xsd;integer"/>
        </owl:DataTypeProperty>
        <owl:DataTypeProperty rdf:ID="title">
          <rdfs:domain rdf:resource="#Document"/>
          <rdf:type rdf:resource="&owl;FunctionalProperty"/>
          <rdfs:range rdf:resource="&xsd;string"/>
        </owl:DataTypeProperty>
```

This code defines class *Action*:

```
<owl:Class rdf:ID="Action">
</owl:Class>
```

This code relates *Action* and *Entity* through four different object properties: subject, verb, object, documentId. This means that each action is characterized by the document where it appears and by its subject, verb,and object entities.

```
        <owl:ObjectProperty rdf:ID="subject">
          <rdf:type rdf:resource="&owl;FunctionalProperty" />
          <rdfs:domain rdf:resource="#Action"/>
          <rdfs:range rdf:resource="#Entity"/>
        </owl:ObjectProperty>

        <owl:ObjectProperty rdf:ID="object">
          <rdf:type rdf:resource="&owl;FunctionalProperty" />
          <rdfs:domain rdf:resource="#Action"/>
          <rdfs:range rdf:resource="#Entity"/>
        </owl:ObjectProperty>

        <owl:ObjectProperty rdf:ID="verb">
          <rdf:type rdf:resource="&owl;FunctionalProperty" />
          <rdfs:domain rdf:resource="#Action"/>
          <rdfs:range rdf:resource="#Entity"/>
         </owl:ObjectProperty>

         <owl:ObjectProperty rdf:ID="documentId">
          <rdf:type rdf:resource="&owl;FunctionalProperty" />
          <rdfs:domain rdf:resource="#Action"/>
          <rdfs:range rdf:resource="#Document"/>
         </owl:ObjectProperty>
```

As it was referred, the next step was to add semantic information to each document.

- For each sentence,
  - For each predicate belonging to the entity ontology and relating two concepts,
    - an instance of the correspondent action with its subject and direct object instances is created.

For example, suppose in document 555 the verb *to save* and the entities *fireman* and *child* are related by the already presented sentence:

sentence(doc1, [fireman(A), child(B), save(A,B)], [ref(A), ref(B)]).

Using our methodology, an instance of a new action will be created, relating new instances of concepts *fireman*, *child*, and *to save*.

```
<pgr:Action rdf:ID="a1">
  <pgr:subject rdf:resource="#e142"/>
  <pgr:object rdf:resource="#e21"/>
  <pgr:verb rdf:resource="#e32"/>
  <pgr:documentID rdf:resource="#d2"/>
</pgr:Action>

<pgr:Fireman rdf:ID="e142">
  <pgr:name>fireman</pgr:name>
</pgr:Fireman>

<pgr:Child rdf:ID="e21">
  <pgr:name>child</pgr:name>
</pgr:Child>

<pgr:ToSave rdf:ID="e32">
  <pgr:name>to save</pgr:name>
</pgr:Entity>

<pgr:Document rdf:ID="d2">
  <pgr:number>555</pgr:number>
</pgr:Document>
```

This code means that in document 555 there is an instance of an action with verb *to save* and having a *fireman* as subject and a *child* as direct object.

4 ISCO

After having represented documents through an ontology on concepts and instances of those concepts it is necessary to be able to support inferences about the represented knowledge.

As basic formalism to support knowledge representations and knowledge inferences we propose the use of ISCO. ISCO [1] is a logic based development language implemented over GNU Prolog that gives the developer several distinct possibilities:

– It supports Object-Oriented features: classes, hierarchies, inheritance.
– It supports Constraint Logic Programming. Specifically, it supports finite domain constraints in ISCO queries.

- it gives a simple access to external relational databases through ODBC. It has a back-end for PostgreSQL and Oracle.
- It allows the access to external relational databases as a part of a declarative/deductive object-oriented (with inheritance) database. Among other things, the system maps relational tables to classes – which may be used as Prolog predicates.
- It gives a simple database structure description language that can help in database schema analysis. Tools are available to create an ISCO database description from an existing relational database schema and also the opposite action.

Taking these ISCO features into account, a translator from OWL into ISCO class definitions and Prolog facts and rules was developed. This translator was applied to every OWL class described in the previous section and, as a consequence, correspondent ISCO classes definitions were obtained. Moreover, each OWL class instance was transformed into ISCO logic programming facts.

As an example, the *action* $a1$ presented previously is translated into the following fact:

```
action(ID=a1, subject='#e142', object='#e21',
       verb='#e32', documentID='#d2').

fireman(ID=e142).
child(ID=e21).
tosave(ID=e32).
document(ID=d2).
```

Variables occurring in queries may carry CLP(FD) constraints. For example, suppose variable `X` is an FD variable whose domain is `(1..1000)`, the query

$$\texttt{document(number = X, title = Y)} \tag{1}$$

will return all pairs `(X, Y)` where `X` is a document number and `Y` is the document's title. `X` is subject to the constraints that were valid upon execution of the query, ie. in the range 1 to 1000.

ISCO class declarations feature inheritance, simple domain integrity constraints and a global integrity constraints.

5 Interaction Example

The interaction is based on the ISCO logic programming framework.

As final goal, we aim to handle the following kind of questions:

- Documents where action A is performed
- Documents where action A is performed having subject S
- Documents where S is the subject of an action

Note that the inference engine needs to be able to deal with the ontology relations. For instance, the question ”documents where action A is performed having subject S” means ”documents where action A (or any of its sub-classes) is performed having subject S (or any of its sub-classes)”.

The translation of natural language queries into correspondent logic forms will not be discussed in this paper (see for instance [6]) and it will be assumed to be handled by some external module.

For the questions presented above, we would have:

- Documents where action A is performed
  - A(id=V), action(verb=V, documentID=ID).
- Documents where action A is performed having subject S
  - A(id=V), S(id=E), action(verb=V, subject=E, documentID=ID).
- Documents where S is the subject of an action
  - S(id=E), action(subject=E, documentID=ID).

5.1 Fireman example

Suppose the following query:

Quais os documentos em que bombeiros salvaram crianças?
“Which are the documents where firemen saved children?”

This query is transformed into its pragmatic interpretation:

Q = [document(id=A),fireman(id=B), tosave(id=C), child(id=D), action(subject=B, verb=C, object=D, documentID=A)].

Using this query, the ISCO inference engine is able to constraint variables A, B, C, and D to its possible values accordingly with the existent OWL instances.

$A =_{\#} (123 : 145)$ – A is constrained to the documents that have instances of the correspondent actions

6 Conclusions and Future Work

A methodology to automatically create ontologies and ontology instances from general documents was proposed. The methodology uses a syntactical analyser to obtain sentence parse trees and XSL transformations to obtain partial semantic analysis. From these semantic analysis it is possible to extract triples of subject-verb-objects. These triples are used to define and to create instances of entities and actions. The obtained ontology and the inferred instances are represented in the OWL language and are used to enrich the initial documents.

On the other hand, translators from OWL into ISCO/Prolog were developed and the ISCO/Prolog inference engine may be used to answer queries about the documents content.

At present, the system is in a prototype phase and it needs work in many areas:

- Ontology creation. The ontology was created automatically but it was not possible to create many hierarchical relations between the classes. In order to be able to define these relations we intend to have two approaches:
  - Create connections with existent ontologies
  - Manually define ontologies for specific sub domains
- Normalisation of concepts. The parsing process was not able to eliminate all entities duplicates and incorrections.
- OWL translation into ISCO/Prolog. A full translation of the OWL language needs to be implemented.
- Evaluation. The system needs to be evaluated and to be tested by users.

References

1. Salvador Abreu. Isco: A practical language for heterogeneous information system construction. In *Proceedings of INAP'01*, Tokyo, Japan, October 2001. INAP.
2. Eckhard Bick. *The Parsing System "Palavras". Automatic Grammatical Analysis of Portuguese in a Constraint Grammar Framework.* Aarhus University Press, 2000.
3. Caroline Gasperin, Renata Vieira, Rodrigo Goulart, and Paulo Quaresma. Extracting xml syntactic chunks from portuguese corpora. In *TALN'2003 - Workshop on Natural Language Processing of Minority Languages and Small Languages of the Conference on "Traitement Automatique des Langues Naturelles"*, Batz-sur-Mer, France, June 2003.
4. H. Kamp and U. Reyle. *From Discourse to Logic.* Kluwer, Dordrecht, 1993.
5. O. Lassila and R. Swick. *Resource Description Framework (RDF) - Model and Syntax Specification.* W3C, 1999.
6. Paulo Quaresma and Irene Pimenta Rodrigues. A natural language interface for information retrieval on semantic web documents. In E. Menasalvas, J. Segovia, and P. Szczepaniak, editors, *AWIC'2003 - Atlantic Web Intelligence Conference*, Lecture Notes in Artificial Intelligence LNCS/LNAI 2663, pages 142–154, Madrid, Spain, May 2003. Springer-Verlag.
7. José Saias and Paulo Quaresma. Semantic enrichment of a web legal information retrieval system. In T. Bench-Capon, A. Daskalopulu, and R. Winkels, editors, *JURIX'2002 - Fifteenth Annual International Conference on Legal Knowledge and Information Systems*, volume 89 of *Frontiers in AI and Applications*, pages 11–20, London, UK, Dezember 2002. IOS Press.
8. Jos Saias and Paulo Quaresma. Using nlp techniques to create legal ontologies in a logic programming based web information retrieval system. In *Workshop on Legal Ontologies and Web based legal information management of the 9th International Conference on Artificial Intelligence and Law*, Edinburgh, Scotland, June 2003.
9. Michael Smith, Chris Welty, and Deborah McGuinness. Owl web ontology language guide. Technical report, www.daml.org, 2003. http://www.w3.org/TR/owl-guide/.
10. www.daml.org. *DAML+OIL – DARPA Agent Markup Language*, 2000.