Using NLP techniques to create legal ontologies in a logic programming based web information retrieval system

José Saias and Paulo Quaresma
Departamento de Informática,
Universidade de Évora,
7000 Évora, Portugal
jsaias|pq@di.uevora.pt

ABSTRACT

Web legal information retrieval systems need the capability to reason with the knowledge modelled by legal ontologies. Using this knowledge it is possible to represent and to make inferences about the semantic content of legal documents.

In this paper a methodology for applying NLP techniques to automatically create a legal ontology is proposed. The ontology is defined in the OWL semantic web language and it is used in a logic programming framework, EVOLP+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. EVOLP is a dynamic logic programming framework allowing the definition of rules for actions and events.

An application of the proposed methodology to the legal web information retrieval system of the Portuguese Attorney General's Office is described.

1. INTRODUCTION

Modern web legal information retrieval systems need the capability to represent and to reason with the knowledge modelled by legal 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 represent semantic objects, to associate them with legal documents, and to make inferences about them.

OWL (Ontology Web Language) is a language proposed by the W3C consortium (http://www.w3.org) 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 - [13]) language and it is defined using RDF (Resource Description Framework - [8]).

In this paper a methodology to automatically create an OWL ontology from a set of legal 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. A preliminary version of this work aiming to create a daml+oil ontology was presented in [12].

This task has similarities with the the work of Boer et al. [5] in the context of the IST programme E-POWER and E-COURT. However, we do not intend to propose any kind of standard for legal ontologies; our aim is to define a methodology to automatically create a base ontology from a specific set of legal documents.

After the creation of the OWL legal ontology, documents are enriched with instances of legal classes and a logic programming based framework is used to support inferences over them. The logic programming framework is based on ISCO [1] and EVOLP [2].ISCO is a new declarative language implemented over GNU Prolog with object-oriented predicates, constraints and allowing simple connections with external databases. EVOLP is a dynamic logic programming language that is able to describe actions and events, allowing the system to make inferences about events, user intentions and beliefs and to be able to have cooperative interactions.

Section 2 describes the natural language processing techniques used to create the OWL ontology. Section 3 describes ISCO, the basic logic programming framework. Section 4 describes EVOLP, the dynamic logic programming framework defined over ISCO and Prolog. Section 5 describes the interaction manager and section 6 provides a simple example. Finally, in section 7 some conclusions and future work are pointed out.

2. OWL ONTOLOGY CREATION

The OWL ontology is created from 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 (http://visl.hum.sdu.dk/visl [4]). This parser is available for 21 different languages, namely for the Portuguese language.

- Actions extraction. From the parser output, actions were extracted with their associated agents and direct objects.

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 XML. A translation tool was developed in Prolog and it is available to the VISL users.

Having an XML version of the documents parse tree, several XSLT transformations were created allowing the extraction of specific chunks of the tree. Using this approach it was possible to obtain a set of verbs and their associated agents (subjects) and direct objects. This set of entities were transformed into OWL instances and associated with the documents.

As an example, suppose the following sentence:

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

This sentence has the VISL output:

```
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, our 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>
```

From this XML structure, using XSL transformations, it is quite easy to obtain the *subject*, *predicate* and *acc* (direct object) chunks.

The extracted chunks are used to automatically create an ontology of entities. Figure 1 show a graphical view of the top-level classes:

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

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

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

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

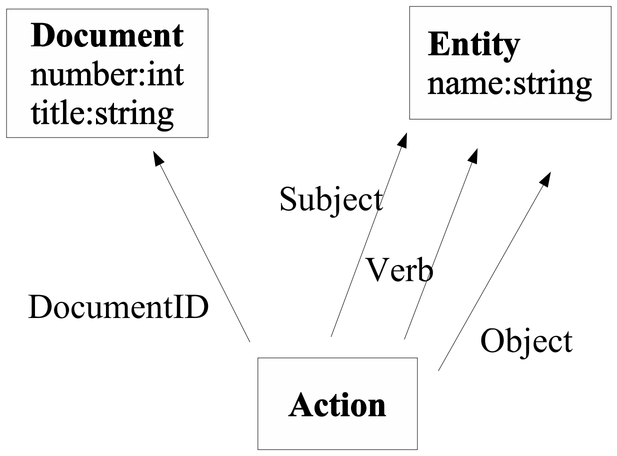

Figure 1: Top-level classes

This code defines classes *Action* and *Entity* (with property *name*).

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

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

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

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">
<owl:domain rdf:resource="#Action"/>
<owl:range rdf:resource="#Entity"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="object">
<owl:domain rdf:resource="#Action"/>
<owl:range rdf:resource="#Entity"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="verb">
<owl:domain rdf:resource="#Action"/>
<owl:range rdf:resource="#Entity"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="documentID">
<owl:domain rdf:resource="#Action"/>
<owl:range rdf:resource="#Document"/>
</owl:ObjectProperty>
```

This code gives examples of entities and one document.

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

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

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

<pgr:Document rdf:ID="d2">
  <pgr:number>555</pgr:number>
  <pgr:title>Pension for relevant services</pgr:title>
</pgr:Document>
```

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

- For each extracted verb, an instance of the correspondent action with its subject and direct object was created. For example, in document 555 the verb *to save* and the entities *fireman* and *child* are related by the following links:

```
<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>
```

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.

3. 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.

The proposed system uses ISCO's capability to establish connections from Prolog to relational databases in an efficient and simple way. For example, the following SQL table:

```
CREATE TABLE "document" (
 "number" int4 NOT NULL,
 "title" text,
 Constraint "number_pkey"
      Primary Key ("number")
);
```

Maps into the following ISCO class definition (and vice-versa):

```
external(pgr,document) class document.
  number: int. key.
  title: text.
```

Taking this ISCO feature into account, a translator from OWL into ISCO class definitions was developed. This translator was applied to every OWL class described in the previous section and, as a consequence, correspondent SQL tables and ISCO classes definitions were obtained. Moreover, each OWL class instance was transformed into an SQL table row and an ISCO logic programming fact. 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').
```

For each defined class a set of Prolog predicates implementing the four basic operations are created: query, insert, update and delete.

Variables occurring in queries are mapped to SQL and may carry CLP(FD) constraints, which will be expressed in SQL, whenever possible. For example, suppose variable `X` is an FD variable whose domain is `(1..1000)`, the query

```
document(number = X, title = Y)
```

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.

4. EVOLP

As it was described in the previous section, ISCO allows a declarative representation of ontologies and object instances. However, there is also a need to represent actions and to model the evolution of the knowledge.

In [3] it was introduced a declarative, high-level language for knowledge updates called *LUPS* ("Language of UPdateS") that describes transitions between consecutive knowledge states. Recently, a new language, EVOLP [2], was proposed having a simpler and more general formulation of logic program updates. In this section a brief description of the EVOLP language will be given. The interested reader should refer to the cited article for a detailed description of the language and of its formalization.

EVOLP allows the specification of a program's evolution, through the existence of rules which indicate assertions to the program. EVOLP programs are sets of generalized logic program rules defined over an extended propositional language L_{assert}, defined over any propositional language L in the following way [2]:

- All propositional atoms in L are propositional atoms in L_{assert}
- If each of $L_0, \ldots, L_n$ is a literal in L_{assert}, then $L_0 \leftarrow L_1, \ldots, L_n$ is a generalized logic program rule over L_{assert}.
- If R is a rule over L_{assert} then $assert(R)$ is a propositional atom of L_{assert}.
- Nothing else is a propositional atom in L_{assert}.

The formal definition of the semantics of EVOLP is presented at the referred article, but the general idea is the following: whenever the atom $assert(R)$ belongs to an interpretation, i.e. belongs to a model according to the stable model semantics of the current program, then R must belong to the program in the next state. For instance, the following rule form:

$$assert(b \leftarrow a) \leftarrow c \tag{1}$$

means that if c is true in a state, then the next state must have rule $b \leftarrow a$.

EVOLP has also the notion of external events, i.e. assertions that do not persist by inertia. This notion is fundamental to model interaction between agents and to represent actions. For instance, it is important to be able to represent actions and its effects and pre-conditions:

$$assert(Effect) \leftarrow Action, PreConditions \tag{2}$$

If, in a specific state, there is the event *Action* and if *PreConditions* hold, then the next state will have *Effect*.

5. INTERACTION MANAGEMENT

The interaction manager is built on the ISCO+EVOLP 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 interaction manager is composed by the following main tasks:

- Query management
- Interaction management

5.1 Query management

The analysis of a natural language query is split in three subprocesses: Syntax, Semantics, and Pragmatics.

5.1.1 Syntax

As syntactic analyser we are using the analyzer developed by E. Bick and referred previously [4]. The VISL output is translated into Prolog facts by the same translator referred in section 2. This translation can be handled by the same translator because there is a direct relation between the XML structure and the Prolog term structure.

As an example, the following query:

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

Has the following syntactical structure:

```
sentence(
  sc(pron_det('qual','M/F','P','<interr>'),'Quais'),
  subj(np),
    n(art('o','M','P'),'os'),
    h(n('documento','M','P'),'documentos'),
  advl(prp('em'),'em'),
  acc(fcl,
    sub(conj_s('que'),'que'),
    subj(n('bombeiro','M','P'),'bombeiros'),
    p(v_fin('salvar','PS/MQP','3P','IND'),'salvaram'),
    acc(n('crianca','F','P'),'criancas', '?')))).
```

5.1.2 Semantics

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 [7]). The 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. For instance, the semantic representation of the sentence above is the following expression:

document(A), fireman(B), save(C), child(D), action(E),
rel(E, B, C, D),
rel(A, E).

and the following discourse referents list:
[ref(A),ref(B),ref(C),ref(D),ref(E)]

These structures represent instances of firemen B, instances of children D and instances of the *save* action C, which are related by action instances E in documents D.

Note that, at present, we are not able to deal with general unrestricted queries and to translate them from a syntactical into a semantic structure. In fact this a quite complex NLP problem and we have decided to deal only with specific subsets of the Portuguese language, namely, with interrogatives about specific domains.

5.1.3 Pragmatic Interpretation

The pragmatic module receives the semantic query representation and tries to interpret it in the context of the database information, which was constructed from the translation of the OWL instances into ISCO facts (as described previously in section 3).

In order to achieve this behavior the system tries to find the best explanations for the sentence logic form to be true in the knowledge base. This strategy for interpretation is known as “interpretation as abduction” [6]. This process was described in detail by Quintano et al. in [11].

From the description of the OWL (and ISCO) classes it is possible to obtain the correspondent ISCO query:

document(id=A),
fireman(id=B),
save(id=C),
child(id=D),
action(id=I, subject=B, verb=C, object=D, documentID=A).

This query was obtained using additional logic programming rules for the *rel* predicate, such as the following:

```
rel(A, B, C, D) <-
  action(A),
  entity(B),
  verb(C),
  entity(D),
  abduct(action(A,B,C,D,_)).
```

Note that the ontology hierarchy is used to infer that *fireman* and *children* are entities and *to save* is an action.

The interpretation of the ISCO predicates is done by accessing the database in order to collect (and constraint) all entities identifiers:

- $I =_{\#}$ (104..109 : 156..157) – I constrained to all actions with the desired properties
- $A =_{\#}$ (123 : 145) – A is constrained to the documents that have instances of the correspondent actions

The above expression contains the possible interpretations of the query in the context of the database.

5.2 Interaction Management

The interaction manager has to represent the actions associated with the queries ($informs$ or $request$), and to model the user attitudes (intentions and beliefs).

This task is also achieved through the use of the EVOLP language (see [10, 9] for a more detailed description of these rules). For instance, the rules which describe the effect of an inform, and a request speech act are:

$$assert(bel(A, bel(B, P))) \leftarrow inform(B, A, P). \quad (3)$$
$$assert(bel(A, int(B, Action))) \leftarrow request(B, A, Action)(4)$$

These rules mean that if an agent A is informed of a property P, then it will start to believe that the other agent believes in P; additionally, if B requests A to perform an action *Action*, then A starts to believe that B intends *Action* to be performed.

In order to represent collaborative behavior it is also necessary to model the transference of information between the agents:

$$assert(bel(A, P)) \leftarrow bel(A, bel(B, P)). \quad (5)$$
$$assert(int(A, Action)) \leftarrow bel(A, int(B, Action)). \quad (6)$$

These two rules means that if an agent A believes another agent believes in P, then it will start to believe in P (it is a cooperative, credulous agent); moreover, it will also adopt the intentions of the other agents.

There is also the need for a rule linking the system intentions and the accesses to the databases:

$$assert(inf(A, B, P)) \leftarrow int(A, inf(A, B, P)), isco(P). (7)$$
$$assert(not\ int(A, B, inf(A, B, P))) \leftarrow inf(A, B, P). \quad (8)$$

The first rule defines that, if the system intends to inform the user about some property, then it will access the ISCO database and it will perform an inform action. The second rule means that the inform action will end the intention to perform the inform action!

6. EXAMPLE

Considering the already presented query:

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

The interaction manager receives the query pragmatic interpretation:

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

After having the sentence rewritten into its semantic representation form, the speech act is recognized:

```
request(user, system, inform(user, system, Q))
```

Using the *request* and the transference of intentions rules the following property is supported:

```
int(system,inform(system, user, Q))
```

Now, using the rules presented in the previous section, the system accesses the ISCO databases and it is able to obtain the final constraints to the discourse referent variables:

- $I =_{\#}$ (104..109 : 156..157) – I constrained to all actions with the desired properties
- $A =_{\#}$ (123 : 145) – A is constrained to the documents that have instances of the correspondent actions

Using the inferred constraints it is possible to obtain the set of solutions to the user query and to answer him:

"Documento P123 e documento P145"
Document P123 and document P145.

7. CONCLUSIONS AND FUTURE WORK

A methodology to automatically create a legal ontology was proposed. The methodology used a syntactical analyser to obtain sentence parse trees and XSL transformations 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 a logic programming based interaction manager was developed. The interaction manager uses many important features from its base LP framework: objects, constraints, inheritance.

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.

8. REFERENCES

[1] Salvador Abreu. Isco: A practical language for heterogeneous information system construction. In *Proceedings of INAP'01*, Tokyo, Japan, October 2001. INAP.

[2] J. Alferes, A. Brogi, J. Leite, and L. Pereira. Evolving logic programs. In S. Flesca, S. Greco, N. Leone, and G. Ianni, editors, *JELIA'02 – Proceedings of the 8th European Conference on Logics and Artificial Intelligence*, pages 50–61. Springer-Verlag LNCS 2424, 2002.

[3] J. J. Alferes, L. M. Pereira, H. Przymusinska, T. C. Przymusinski, and P. Quaresma. Preliminary exploration on actions as updates. In M. C. Meo and M. Vilares-Ferro, editors, *Procs. of the 1999 Joint Conference on Declarative Programming (AGP'99)*, pages 259–271, L'Aquila, Italy, September 1999.

[4] Eckhard Bick. *The Parsing System "Palavras". Automatic Grammatical Analysis of Portuguese in a Constraint Grammar Framework.* Aarhus University Press, 2000.

[5] A. Boer, R. Hoekstra, R. Winkels, T. van Engers, and F. Willaert. Proposal for a dutch legal xml standard. In *EGOV2002 – Proceedings of the First International Conference on Electronic Government*, 2002.

[6] Jerry Hobbs, Mark Stickel, Douglas Appelt, and Paul Martin. Interpretation as abduction. Technical Report SRI Technical Note 499, 333 Ravenswood Ave., Menlo Park, CA 94025, 1990.

[7] H. Kamp and U. Reyle. *From Discourse to Logic.* Kluwer, Dordrecht, 1993.

[8] O. Lassila and R. Swick. *Resource Description Framework (RDF) - Model and Syntax Specification.* W3C, 1999.

[9] P. Quaresma and J. G. Lopes. Unified logic programming approach to the abduction of plans and intentions in information-seeking dialogues. *Journal of Logic Programming*, 54, 1995.

[10] Paulo Quaresma and Irene Rodrigues. Using logic programming to model multi-agent web legal systems – an application report. In *Proceedings of the ICAIL'01 - International Conference on Artificial Intelligence and Law*, St. Louis, USA, May 2001. ACM.

[11] Luis Quintano, Irene Rodrigues, and Salvador Abreu. Relational information retrieval through natural lanaguage analysis. In *Proceedings of INAP'01*, Tokyo, Japan, October 2001. INAP.

[12] José Saias and Paulo Quaresma. Semantic enrichment of a web legal information retrieval system. In T. Bench-Capon, editor, *JURIX'2002 - Fifteenth Annual International Conference on Legal Knowledge and Information Systems*, London, UK, Dezember 2002. IOS Press.

[13] www.daml.org. *DAML+OIL – DARPA Agent Markup Language*, 2000.