An Ontology Specification Language Based on a Controlled Natural Language

Josiane M. P. Ferreira, Sérgio Roberto da Silva and Edgard M. de Oliveira

Departamento de Informática
Universidade Estadual de Maringá
Av. Colombo, 5790, zona 07 – Maringá-PR Brazil – 87020-900
jmpinhei@din.uem.br, srsilva@din.uem.br, edgardmota@gmail.com

Abstract. The specification of ontologies has always been a complex problem, especially due to the difficulty of acquiring knowledge from domain experts. This paper proposes an ontology specification language based on a Controlled Natural language to be used by domain experts in the specification of their own ontologies. The language was made in order to tolerate the use of linguistic mechanism like anaphors and ellipses, allowing users to specify concept references by means of pronouns. To solve this anaphors the DRT approach is used. A mapping from this ontology specification language to a great part of the OWL-DL formalism was also developed to allow reasoning with its produced ontologies.

1. Introduction

A significant disadvantage of the current knowledge acquisition processes is that the transmission of knowledge from domain experts to the knowledge base (KB) is done in an indirect way. Because experts do not know how to use a knowledge representation technique (KRT) they have to transmit their knowledge to a knowledge engineer (KE), who then introduces it in the KB. This indirect process might generate different interpretations from experts and from engineers regarding the knowledge of the domain. This happens because, as semiotics shows [19] [6], when two people communicate, it is common for each of them to have different interpretations of the same sentence. When it comes to KB building the misinterpretation occurs frequently as experts are not aware of the task of representing knowledge by means of a KRT, and knowledge engineers, in turn, does not necessarily know the application domain. Consequently, the result of this indirect transmission of knowledge might be a KB of poor quality, which does not meet the user's expectations.

However, if we enable domain experts to build the KB themselves, we could solve not only the miscommunication between them and knowledge engineers, but also the high cost of knowledge engineers' services. To achieve this, we have mainly two alternatives: 1) we might have domain experts learn to codify their knowledge using a KRT, or 2) we may allow them to express their knowledge by means of a representation language which they can use more easily and, then, we map this language to a KRT automatically.

The first alternative is a difficult task because experts will have to learn a completely new language to express themselves — taking into account the fact that they are usually lay when it comes to representing knowledge, i.e., they may have no knowledge whatsoever of formal languages. The second alternative might be a viable solution as long as three factors are taken into account: 1) the language used to represent knowledge is similar to a language experts are familiar with as, for example, a natural language; 2) this language has enough expressivity to represent the domain at hand; 3) that language should be unambiguous and 4) it is possible to get an efficient mapping from this language to a KRT.

In this paper, we present a summarized version of our research into the study of the viability of the second alternative presented above. We propose to make available an *ontology specification* language which is based in a *controlled natural* language (CNL) [2], so that experts can express their knowledge, and also to make available the automatic mapping from this language to a KRT — in our case the OWL (Ontology Web Language). Besides that, we propose the use of an ontology editing environment guided by an ontology construction process which helps experts to express their knowledge. With the development of this CNL, its mapping to the used KRT, and its integration with an ontology editing environment, we aim at making part of the KB building task more efficient and less onerous.

In section 2, we explain the type of knowledge we are interested in allowing the experts to express and describe our language for the specification of ontologies. Section 3 shows the mapping of our proposed language to OWL. In section 4, we present an editor which used the language we propose for the creation of ontologies. Finally, in section 5, we discuss the results of this research.

2. The Proposed Language for Ontology Building

The word Ontology comes from the Greeks and means ‘the study of being’. In philosophy, Ontology is an area of metaphysics which ‘studies the being or the existence, as well as the basic existent categories, trying to find out which entities and types of entities there is’. In computer science, the sense of ontology is a little different and several authors tried to describe its meaning [9] [10]. To our research, the word *ontology* is “a conceptual model about a particular domain in which we can represent concepts and the relationships among them” [14], besides axioms which rule these relationships.

The aim of this work is to provide means for domain experts to express their knowledge directly in the computer, so that they are able to build an initial ontology of the domain, which is adequate for knowledge engineers to refine later. There are numerous KRTs employed for this task, among which we can mention [1]: LOOM, Ontolíngua, OML, RDF and RDFS, OIL, and OWL. Unfortunately, most of them are designed aiming at the best way to represent knowledge and maintaining its computational efficiency, without worrying about its learning process or its communication aspects with people [7] [11]. Therefore, the users of these KRTs have to learn a way to express their knowledge by means of a language which is totally different from the one they use on a daily basis.

Another way to make experts represent their knowledge more naturally is to supply a language that is similar to the one they already know as a way to express their knowledge. From this point of view, the most adequate language would be the natural language (NL), which all of us know and has sufficiently appropriate communicative aspects. However, the NL has inherent ambiguities and the processing of its unrestricted form is not viable yet, despite the enormous recent advances. We can, nevertheless, make use of a subset of NL only, so that this subset does not have ambiguities, neither for experts nor for the parser that will be processing it, keeping its processing viable and maintaining its communicative aspects.

The use of *controlled natural* language has been explored with relative success in tasks such as the specification of requirements [7] and the acquisition and representation of knowledge [12] [23] [20], among others. Using a CNL means that all the sentences of the CNL are correct in the NL but not all the sentences of the NL are allowed in the CNL, because this is a subset of that one. Thus, experts will only have to learn which types of sentences they can and which ones they cannot use to express their knowledge of the domain, which it is simpler than learning to express their knowledge in a totally new way (by means of a KRT). In general, a CNL allows experts to use communicative mechanisms (such as quantifiers, qualifiers and figures of speech) to express their knowledge in a more natural way. However, it is extremely important that the communicative mechanisms allowed in the CNL keeps the computational processing of the language within a range of reasonable efficiency, so that its implementation is viable.

2.1. The Foundation of the Controlled Natural Language

The controlled natural language we propose has its foundation in the work of [4]. In that work, the author aimed at, amongst other things, getting a type-language in the form of a CNL for end-user programming. He emphasizes that the difference between languages normally used by end-users, for the specification of entities and processes, and formal languages (programming languages, KRTs, among others) is mainly in the mechanism used to refer to objects in the text code. The language normally used by these users presents a natural way to work with structured objects, using common sense knowledge of the domain and linguistic mechanisms such as quantifiers (of the type *all*, *every* and *each* and plurals), qualifiers, selectors and figures of speech (anaphors and ellipsis) to facilitate the reference to these objects. Such mechanisms help people to refer to the characteristics of complex objects in a direct and simple way, as they hide many details about their inner structure. Moreover, the nouns, which describe objects in the domain, are generally substituted by pronouns in the subsequent references (after their introduction in the text), a case of anaphor. Also the use of ellipsis is common, limited to the basic forms, to omit objects and verbs in the sentences. In this case, the omitted names always make reference to objects that have been previously introduced in the context formed by the previous sentences, making their omission possible without interfering with its interpretation.

These kind of communicative mechanisms are very important because they make the process of describing something by means of a language much more natural for their users. Moreover, they prevent the necessity of writing and, therefore, interpreting very long sentences, which is particularly difficult for people. Also, due to their characteristics, these references are simple to solve, which facilitate its implementation. In this way, the computational cost needed to solve these mechanisms is low, when compared with the profits gained in the communication with their usage. The language proposed was divided into three sublanguages:

- The **reference** sublanguage (to objects), which contain the communicative mechanisms mentioned before, acting in an orthogonal way to the other sublanguages, aiming at enhancing the textual cohesion and, therefore, facilitating the interpretation by experts. It accepts references like: “all wine”, “the colors of the wines”, and “each message”;
- The **metalanguage** sublanguage, which allows for the creation and extension of an ontology domain by means of a definition of declarative knowledge. It accepts references constructors like: “A car is a kind of automobile” and “It has a color and a mark”;
- The **control** sublanguage, which allows for the creation and definition of the procedural knowledge and/or of the inferential control (in some cases). It accepts references constructors like: “Send the <message> to all the <e-mail_addresses>”.

Despite the fact that the corpus used in the analysis in [4] is in the English language, his findings are valid for others languages as well (given some tuning) as the elements which were studied belong to the *universal linguistic* mechanisms [15] as, for example, the phrasal structure of the different ways to refer to objects in NLs.

The language proposed in this paper makes use of the reference and the metalanguage sublanguages only. Examples of the elements in the reference sublanguage language are shown in Table 1.

Table 1. Types of valid references in the reference sublanguage.

Reference Type	Example
noun	Wine
[a/an] noun	a wine
noun(s)	Wines
cardinal noun(s)	three grapes
the/this noun	the wine
the noun of [the] noun	the sender of the message
pronouns	it has …
all [the] noun(s)	all [the] wines
each/every noun	each message
reference and reference	a color and a body

The reference sublanguage is what makes the language we propose different in relation to the language mentioned in Pulman’s works [20]. There, he emphasizes that the sentences he calls *notation in macro style* (which are similar to the ones proposed in the metalanguage sublanguage) do not have flexibility. He uses examples such as: *A <subtype> is_a <type>*, where the elements between <...> can only be objects. The metalanguage proposed here is formed by similar sentences, but we allow for users to work with complex objects in a natural way, including the use of ellipses and anaphors, which gives us a profit in the usability of the language. Thus, our language will be closer to the one users is accustomed to use to express themselves. For example, if an expert introduced the element *book* in a previous sentence as in *A book is a publication*, s/he will be able to reference it in other future sentences, by using the pronoun "*it*", as for example, *It has chapters and sections*.

The metalanguage sublanguage we propose makes extensive use of the reference sublanguage and is used to define the domain's ontology itself. Each sentence allowed in the metalanguage has a purpose in the definition of the domain's ontology. Below we present the types of metalanguage sentences and we also show the purpose of two sentences, and one example of the usage and the syntax of the metalanguage.

Sentences of the metalanguage sublanguage for the definition of classes[1]:

1a) Ref1 **'is a kind of'** Ref2* **'.'**
1b) Ref1 **'is a'** Ref2* **'.'**
2a) Ref1 **'is a kind of'** Ref2* **'that has'** Ref3 **'.'**
2b) Ref1 **'is a'** Ref2* **'that has'** Ref3 **'.'**
3) Ref1 (**'is part of'** | **'are parts of'**) Ref2* **'.'**
4) Ref1 (**'has'**|**'have'**) **'at least one'** Ref2 **'that must be'** Ref3* '.'
5a) Ref1 Ref2 **'must be'** Ref3* **'.'**
5b) Ref2 **'of'** Ref1 **'are'** Ref3* **'.'**
6) Ref1 **'of'** Ref2 **'must be'** Ref3 **'.'**
7) Ref1 **'must have exactly'** Ref2 **'.'**
8) Ref1 **'must have at least'** Ref2 **'.'**
9) Ref1 **'must have at most'** Ref2 **'.'**
10) Ref1 **'can only be'** Ref2 **'.'**
11) Ref1 **'is also known as'** Ref2^{*+} **'.'**

Sentence for the definition of properties and values to the properties: therefore

12) Ref1* (**'has'** | **'have'**) Ref2 **'.'**
13) Ref1$^{\&}$ (**'is'** | **'are'**) Ref2$^{\#}$ **'.'**
14) Ref1$^{\&}$ **'has possible values'** Ref2 **'.'**
15) Ref1* **'has'** Ref2 **', with possible'** (**'value'** | **'values'**) Ref3 **'.'**
16) Ref1* **'has'** Ref2 **', which is'** Ref3$^{\#}$ **'.'**
17) Ref1 **'has default value'** Ref2 **'.'**
18) Ref1^{+} **'of'** Ref2 **'is'** Ref3 **'.'**

[1] Here, "ref" indicates the presence of a reference — sees Table 1 —, which can be complex. Underlined elements are entities which are being introduced in the ontology and, so, are not part of the lexicon. Elements in **bold** and 'single inverted commas' are fixed elements in the language. References marked: with ***** might be any of the classes previously defined in the ontology; with **+** might be any of the properties previously defined in the ontology for the class which is being specified in the sentence; with **&** might be any of the properties previously defined in the ontology, but which have not had their domain specified; and with **#** might be, besides any of the classes in the ontology, any of the datatypes allowed by the employed KRT.

Examples of the Usage of the Sentences

1) _Sentences of type 4_: Ref1 (‘**has**’|‘**have**’) ‘**at least one**’ Ref2 ‘**that must be**’ Ref3* ‘**.**’
   Purpose: to define a class by means of a value restriction for a property. The restriction determines that all elements of a class (Ref1) that have at least one element of another class (Ref3) as a value for a property (Ref2) belong to this class. Thus, this restriction requires that at least one value for the property (Ref2) belong to a given class (Ref3) so that it belongs to the class that is being defined (Ref1). Other values can exist for the property that are not elements of the given class (Ref3).

   • A _wine_ **has at least one** _maker_ **that must be** a winery.
   • All _wines_ **have at least one** _maker_ **that must be** a winery.

2) _Sentences of type 13_: _Ref1_$^{\&}$ _(**‘is’** | **‘are’**) Ref2_$^{\#}$ **‘.’**

   Purpose: sentences of this type can be used for two purposes: a) to define a type (Ref2) to a property (Ref1) that has already been defined; and b) to define an instance (Ref1) of a class.

   a) The maker **is a** winery.
   _b) W1_ **is a** wine.

   — The differentiation between the two intentions will be done by using the structure of the reference Ref1. In type **a** sentences the defining article “the” will be used as a anaphoric determiner [21], that is, with the purpose of making reference to an entity of the discourse that has already been defined in the ontology. Thus, every time Ref1 is a defined reference, we are defining a type for a property that already exists in the ontology; otherwise, we will be introducing a new instance of a class in the ontology. Sentences of the type **a** are normally used after a definition of a property, using the sentences of type 2, 4, 5, 6, 7, 8, 9 or 12.

3. The Automatic Mapping of the Proposed Language to the OWL

The mapping of the CNL proposed here for a KRT begins by the implementation of a parser for this language. One of the main points in this task is the process of anaphor resolution. To solve the anaphors correctly, we have to consider the set of sentences that composes our codified description as a discourse in which one sentence can reference entities that have been introduced in previous sentences. We adopt as an intermediate discourse representation schema the *Discourse Representation Theory* (DRT) [13], a different representation for the first order logic that deal with discourses and offers methods for anaphors and presuppositions resolution in a sufficiently attractive way. For the parser implementation, we used a parser (in Prolog language) for a wide subset of the English grammar developed by Blackburn e Bos

[3]. That parser maps NL to Discourse Representation Structures (DRSs), solve the anaphors and ellipsis and deals with presuppositions naturally.

We decide to use the OWL as the formal language for ontology building mainly because it has become a standard in Semantic Web. Moreover, there are good reasoners available to it like RACER[2] and PELLET[3]. The OWL language [16] has ways to represent semantics about a domain and also to make this representation available to be processed by means of software applications. OWL entails three sublanguages which combine a balance between expressiveness and efficient reasoning — OWL Full, OWL DL and OWL Lite. In our research, we chose OWL DL as the KRT to which the language we propose is mapped.

Mapping the sentences first to DRSs *results in a good advantage* because from there we can map them for different KRTs, only mapping a formal language (the DRS) to another formal language (a KRT). This is simpler than working with CNL. Thus, after the text in CNL is processed by the parser, the resulting DRSs are processed by an algorithm that maps them to OWL. Although we choose OWL-DL, from the three sublanguages of OWL, our proposed language does not deal with: some characteristics of RDF Schema (rdfs:subPropertyOf, rdfs:domain, DifferentFrom, AllDifferent); some characteristics of properties (InverseOf, TransitiveProperty, SymmetricProperty, FunctionalProperty, InverseFunctionalProperty); some axioms of class (disjointWith), and some boolean combinations for the description class (unionOf, complementOf, intersectionOf). This happens because we are trying to find simple and natural sentences that can represent these OWL constructors in CNL. Other researches with similar aims have had the same problems [24]. However, some constructors like boolean combinations should be deal with in the next version of the proposed CNL.

Now, we present, as examples, the mapping of two metalanguage sentences to the resulting OWL code. We chose to use example sentences instead of generic sentences to facilitate the understanding. The underlined elements are elements that are being introduced in the ontology and, therefore, do not belong to the lexicon yet.

Example 1: Sentences of **type 4** used for the definition of a class by means of a property restriction.

- *A wine* ***has at least one*** *maker* ***that must be*** *a winery.*

Resulting OWL mapping:

```
<owl:Class rdf:ID="Wine">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasMaker"/>
      <owl:someValuesFrom rdf:resource="#Winery"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:ObjectProperty rdf:ID="hasMaker">
<rdfs:domain rdf:resource="#Wine">
```

[2] http://www.racer-systems.com/

[3] http://www.mindswap.org/2003/pellet/

```
<rdfs:range rdf:resource="#Winery">
</owl:ObjectProperty>
```

Example 2: Sentences of **type 13** used for the definition of a type to a property.

a) *The maker* ***is a*** *winery.*

Resulting OWL mapping:

```
<owl:ObjectProperty rdf:ID="hasMaker">
  <rdfs:range rdf:resource="#Winery"4>
</owl:ObjectProperty>
```

b) *W1* ***is a*** *wine.*

Resulting OWL mapping:

```
<Wine rdf:ID="W1" />
<owl:AllDifferent>
<owl:distinctMembers rdf:parseType="Collection">
       <Class rdf:about="#Instance"/>◊
    </owl:distinctMembers>
</owl:AllDifferent>
```

4. The Ontology Editing Environment

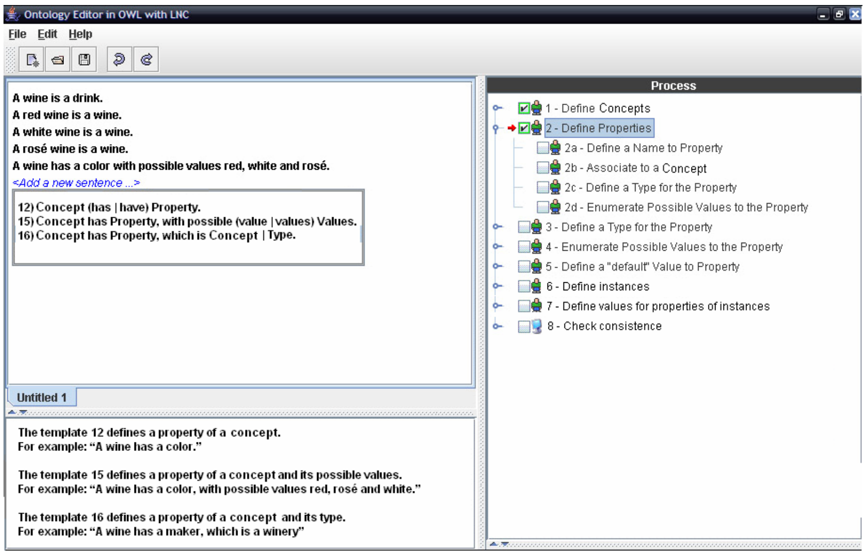
As we mentioned before, we believe domain experts will have less difficulties to express their knowledge by means of a controlled natural language than if they have to do it using a KRT. In order to assist the interaction between experts and the knowledge base even further, we propose the use of a process-driven editor, which suggests a sequence of actions to the building of ontologies and, at the same time, filters the sentences, from the metalanguage sublanguage, that are valid in the current situation and the sentence elements which must be used in each step of the process. This editor is an adaptation of Godoi [8] and Oliveira's [17] work.

The editor we propose is formed by an editing area — in which the text which represents the ontology is written; an area which represents the ontology building process (that is implemented as a hierarchical state machine) and indicates the phase of the process the user is in; and an explanation area which can, for example, show users the meaning of the sentence which is being written for the ontology. Fig. 1 shows the editor's interface.

In the text area, users can use the right button of the mouse and may choose from a list of templates, which represent the sentences which are currently possible to be used to edit the ontology. The main advantage of using templates is that users do not

[4] If this name was a *datatype*, we would define a *DatatypeProperty*. But as we are considering that it is the name of a previously defined ontology class, we define an *ObjectProperty*.

◊ This clause will be repeated for each instance defined in the ontology.



A wine is a drink.
A red wine is a wine.
A white wine is a wine.
A rosé wine is a wine.
A wine has a color with possible values red, white and rosé.
<Add a new sentence ...>

12) Concept (has | have) Property.
15) Concept has Property, with possible (value | values) Values.
16) Concept has Property, which is Concept | Type.

The template 12 defines a property of a concept.
For example: "A wine has a color."

The template 15 defines a property of a concept and its possible values.
For example: "A wine has a color, with possible values red, rosé and white."

The template 16 defines a property of a concept and its type.
For example: "A wine has a maker, which is a winery"

- 1 - Define Concepts
- 2 - Define Properties
  - 2a - Define a Name to Property
  - 2b - Associate to a Concept
  - 2c - Define a Type for the Property
  - 2d - Enumerate Possible Values to the Property
- 3 - Define a Type for the Property
- 4 - Enumerate Possible Values to the Property
- 5 - Define a "default" Value to Property
- 6 - Define instances
- 7 - Define values for properties of instances
- 8 - Check consistence

Fig. 1. Ontology editor interface.

need to learn all the details of the specification language. The editor makes the templates of permitted sentences in the language available and the users choose the one they want. From this point onwards, the references of this sentence can be filled in so that it is actually processed by the parser. At this stage, the editor has an intense interaction with the knowledge base, because according to the sentence which is chosen by the user, the base needs to be consulted and the elements which can be used by the user to fulfill the references must be made available by means of pop-up menus. This approach prevents users from making common mistakes as, for example, using the name of a non-existent class, or using the name of a class instead of a property. Besides that, we can pre-process the DRSs which correspond to each of the templates, lowering the processing which is necessary to map the sentences to OWL. Thus, the DRSs would only be used to solve anaphors and presuppositions.

The templates which are available in this editor correspond to the sentences which are part of the proposed metalanguage sublanguage. However, instead of using the elements "Ref", as a name for the references, we use the ontology elements which are expected in each of them (e.g. concept, property, types, values, etc.), so that users can understand them more easily.

5. Results and Conclusions

The ontology editors we know of are mostly aimed at being used by knowledge engineers, or experts with some experience in the ontology building area. As shown by Pérez [18], they usually use the KRT in which the ontology will be codified, besides graphic resources which are made available by the editor as a means of communication between the tool and its user. This type of communication is not efficient for our purposes, as our idea is to allow domain experts to use the tool in order to place their knowledge in the system directly and they frequently have no knowledge of a KRT.

Other researches have the same or a similar aim as ours. The Attempto project [7] presents an approach which is similar to ours. It also makes use of a CNL, called ACE, as an interface to knowledge representation. It is an incremental project that was originally defined to create software requirement specifications and has evolved to be applied to represent general knowledge as well. The Schwitter's [22] approach, which is derived from the Attempto project, also makes use of a CNL as an interface between the user and the KRT. In this approach, users begin by entering the text and the editor suggests the syntactic structures which should come next, keeping the controlled aspect of the language. If a sentence is not part of the lexicon, users have the option of extending it, by means of a lexical editor. Ambiguous sentences will be reported to users so that they can decide on the meaning of each of them. Users can also add a word or modify the content of the lexicon using a specific interface which requires minimal linguistic knowledge.

Although the CNL proposed in this paper presents limitations if compared with Attempto [7], we may easily broaden its expressiveness in future works. Comparing to Schitter's [22] work, which builds sentences in a similar way to ours, we may say that the expressiveness of the CNL which he proposed is similar to ours. Although he covers some OWL-DL elements which we do not, for example, sentences which allow for the representation of the characteristics of a property (e.g., a property being transitive, inverse or symmetrical), he does not cover other elements which we do as, for example, the representations of the existential quantificator (some ValuesFrom), of the equivalence of instances, of the enumeration of members of a class, and of the cardinality restrictions. What's more, some sentences which are proposed by Schwitter are not very natural for a lay user as, for example, the sentence *'ObjectProperty' has the range 'class'* which can be instantiated to *A maker has the range winery.* In our language, the same purpose would be achieved by means of the sentence *The maker is a winery.* As the property *maker* should have been defined before the definition of its value range, we know that *maker* is a property and *winery* is its range of possible values.

As for the lexical extension issue, the editor we propose presents advantages in relation to the proposals we have discussed above, because it allows for the *automatic* inclusion of nouns, since the declaration of a new entity in the ontology is automatically translated in the inclusion of a new noun in the lexicon. The inclusion of elements of other parts of speech such as verbs, adjectives, etc., has not been considered yet, but these elements can be treated in a similar way.

The process-driven ontology editor we propose partially limits the users' expression when it comes to specifying the knowledge base, making them follow a partial sequence in order to define the elements of the ontology. This approach of making users utilize an element which is in the pop-up menus has the following benefits: 1) preventing experts from making mistakes which are very common such as to refer to an element which does not yet exist in the ontology; 2) helping users to remember the elements which have already been defined in the ontology, thus reducing the cognitive burden on users and 3) preventing experts from making reference to a class instead of a property, which would imply in a mistake in the language mapping phase. Another interesting fact is that the use of templates allows for, as a secondary effect, the pre-processing of the sentences (i.e., their pre-parsing), making the use of an explicit parser almost unnecessary.

There are other forms to implement the ontology building process which reduce the demands of ordering and give users more freedom of speech as, for example, using late evaluation. However, the ideal balance between the potential of the user's freedom of speech, the cost of implementing the parser and the cognitive burden on users can only be identified after a detailed assessment of the editor's usability, enlightening which approach is best.

The proposal of an ontology building process which is associated to a specification language represents a step forward when compared to the Attempto project and to Schwitter's work, as they do not consider a process like this to guide experts in this task. Another ontology editor, called DOE (Differential Ontology Editor) [5], which is aimed at knowledge engineers, suggests some steps which should be followed in the ontology specification process. This editor requires that users define a complete taxonomy for the elements in the ontology so that they can later define the implied restrictions. In this sense, our process is more flexible. If users find it necessary to define a subclass, and soon after, want to define its properties, possible value range and its restrictions as well, they can do it.

Thus, our proposal was to make it possible for domain experts to build a knowledge base. This proposal entailed the definition of a controlled natural language, its mapping to a KRT and the proposal of a process-driven editor. To make sure that the CNL we presented here is easily used and learnt by experts, we need to carry out usability tests regarding both the language and the editor-language set, which is a future goal of this research. Moreover, in the future, we intend to make a comparison between our proposal and the main current ontology editors like Protegé[5], SWOOP[6], etc., to make it possible to combine textual and graphical representations. Also, if we consider the possibility of expressing procedural knowledge, we need to broaden the set of templates which are available to the users and define a mapping to a rule language, like SWRL. The same type of language may be used in the specification of business rules in the web semantic, making this type of task accessible to lay users.

References:

1. ALMEIDA, M. B. *Linguagens utilizadas na construção de ontologias*. Available in: <http://www.eci.ufmg.br/mba/p1_2_2.html>. On: 25 abr. 2005.
2. ALTWARG, R. (2000). *Controlled languages: an introduction. Macquarie University, Graduate Program in Speech and Language Processing*. Macquarie, Available in: <http://www.shlrc.mq.edu.au/masters/students/raltwarg/clwhatisa.htm>. On: 22 jun. 2004.
3. BLACKBURN, P.; BOS, J. (1999) *Representation and inference for natural language: a first course in computational semantics*. Universität dês Saarlandes. V.2 – Working with

[5] http://protege.stanford.edu

[6] http://www.mindswap.org/2004/SWOOP/

discourse representation structure. Available in: <http://www.cogsci.ed.ac.uk/~jbos/comsem/book2.html>. On: 21 jul. 2004.

4. DA SILVA, S. R. P. (2001) *Um modelo semiótico para programação por usuários finais*. Tese (Doutorado) - Departamento de Informática, PUC-Rio Janeiro, Rio de Janeiro.
5. DOE. *Differential ontology editor*. Available in: <http://opales.ina.fr/public/>. On: 21 jun. 2005.
6. ECO, U. (1976) *Theory of semiotics*. Bloomington: Indiana: University Press.
7. FUCHS, N. E.; SCHWERTEL, U.; SCHWITTER, R. (1999) Attempto Controlled English – not just another logic specification language. In: *LNCS 1559*. Springer pages 1–20.
8. GODOI, M. S. (2004) *Implementação de Uma Interface para o Ambiente de Extensões para Aplicações* Extensíveis. Relatório Técnico. Maringá: DIN, UEM Brasil. 22 p.
9. GRUBER, TOM. (1996) *What is an ontology?* Available in: <http://www.ksl.stanford.edu/kst/what-isan-ontology.html>. On: 15 set. 2004.
10. GUARINO, N. (1995) Formal ontology, conceptual analysis and knowledge representation. *International Journal of human and Computer Studies*, v. 43 n. 5/6.
11. HALL, A. (1990) Seven myths of formal methods. *IEEE Software*, v. 48, n. 1, pgs 67–79.
12. HOEFLER, S. (2004) *The syntax of Attempto Controlled English: an abstract grammar for ACE 4.0*. Technical report. Zürich: Department of informatics, University of Zurich, Switzerland.
13. KAMP, H. (1981) A theory of truth and semantic representation. In: GROENENDIJK, J. et al. (Ed.). *Formal methods in the study of languages Foris*, Amsterdam: Mathematisch Centrum.
14. KNUMBLAUCH, H. (2004) *Editing OWL ontologies with Protégé*. Stanford University. Available in: <http://protege.stanford.edu/plugins/owl/publications/2004-07-06-OWL-Tutorial.ppt> On: 24 fev. 2005.
15. LYONS, J. (1981) *Language and Linguistics: an introduction*. Cambridge, UK: Cambridge University Press.
16. MCGUINESS, D. L.; HARMELEN, F. (2003) *OWL web ontology language - overview*. Available in:<http:www.w3.org/TR/2003/PR-owl-features-20031215>, On: 10 jan. 2004.
17. OLIVEIRA, E. M. (2004) *Integração e Implementação de um Ambiente de Extensão para Aplicações Extensíveis*. 51 f. Monografia de Graduação – DIN, UEM, Maringá, jan. 2004.
18. PÉREZ, A. G. et al. (2002) *A survey on ontology tools. OntoWeb*. Available in: <http://ontoweb.aifb.uni-karlsruhe.de/About/Deliverables/D13_v1-0.zip>. On: 23 jun. 2005.
19. PEIRCE, C.S. Collected papers. Cambridge: Ma. Harvard University Press. (Excerpted in BUCHLER, J., ed., *Philosophical Writings of Peirce*. New York: Dover, 1955).
20. PULMAN, S.G. (1996) Controlled Language and Knowledge Representation. In: *Proceedings of International Workshop on Controlled Language Applications*, 1., Belgium. Belgium: Katholieke Universiteit Leuven. pages 233-242.
21. QUIRK, R. (1972) et al. *A grammar of contemporary English*. 1st ed. Essex, UK: Longman Group. ISBN 0 582 52444 X.
22. SCHWITTER R. (2005) *Controlled natural language as interface language to the semantic web*. Sydney, Australia: Centre for Language Technology Macquarie Universit.
23. SCHWITTER, R. (2004) Representing knowledge in controlled natural language: a case study. In: NEGOITA, M. G.; HOWLETT, R. J.; JAIN, L. C. (Ed.). Knowledge-based intelligent information and engineering systems. In: *Proceedings of international conference*, KES, 8. Wellington., New Zealand, pages 711-717, Sep. 2004, Part I, Springer LNAI 3213.
24. KAAREL KALJURAND and NORBERT E. FUCHS. Bidirectional mapping between OWL DL and Attempto Controlled English. In *Fourth Workshop on Principles and Practice of Semantic Web Reasoning*, Budva, Montenegro, 2006.