Declaring Local Contexts of Words with Extensible Dependency Grammar

Maxim Mozgovoy
University of Aizu
Aizu-Wakamatsu, Japan
mozgovoy@u-aizu.ac.jp

Extensible dependency grammar (XDG) is a modern formalism for declaring dependency relations between lexical entries, generally used to construct natural language parsers. This work shows how to use XDG to declare specific contexts of the words, thus turning XDG parser into a word sense disambiguation module or a context-sensitive bilingual dictionary. The capabilities of the proposed method are shown on the example of small English to Finnish dictionary, helpful for entry-level Finnish language learners.

Keywords—natural language processing; parsing; word sense disambiguation; computers in education.

I. Introduction

Most currently popular natural language parsers generate *phrase structures* that represent constituent parts of input sentences. However, in recent years there is a growing interest to *dependency-based* representations of natural language texts [10]. Dependency-oriented formalisms suggest that syntactic structure of a sentence consists of lexical elements linked by binary asymmetrical relations called *dependencies* [1]. Such a representation provides a number of advantages. For instance, dependency links are closer to semantic relationships interpreted on the next text processing stage [2]. Also, dependency-based parser only connects existing words (and does not create new nodes in the parse graph), which makes parsing more straightforward. Furthermore, as noted in [1], dependency parsing is often considered more adequate for languages with freer word order than in English.

As of today, Ralph Debusmann's XDK [3] is probably the only full-fledged parser maker's toolkit, aimed at producing dependency-based parsers. Basically, it consists of a universal parser that processes extensible dependency grammars (XDG) [4]. The input grammar defines both word-linking principles and operating parameters of the parsing algorithm. The grammar can be either handcrafted or automatically generated. While XDK (standing for "XDG Development Kit") comes with examples of handmade grammars, there was at least one attempt to generate a grammar automatically from dependency treebank data [5].

The syntax of extensible dependency grammars is close to the formalism, proposed in [6] for describing lexical entries of a context-sensitive multilingual dictionary. The basic idea of the dictionary is to declare words with their potential dependents in order to make translation less ambiguous. For example, a typical English verb can have numerous translations, often depending on its direct object. By declaring potential objects explicitly, one can provide a possible context for each meaning of the verb.

The dictionary heavily relies on the existence of words ontology, such as the one provided by EuroWordNet [7]. Instead of declaring specific dependent words, the formalism allows to specify a word's class (such as *animal*, *musical instrument*, *game*, etc.) in the given ontology.

The work [6] discusses the declarative side of multilingual dictionaries (inspired mostly by the approach of Tusov [11]), but does not talk about practical aspects of implementation. Providing a built-in parsing algorithm, XDK seems to be a suitable solution for implementing such a dictionary.

The most notable drawback of extensible dependency grammars in this task is the lack of hierarchical type system, which makes direct usage of EuroWordNet-styled ontologies impossible, thus requiring some technical tricks.

This paper shows how extensible dependency grammar can be used to declare a context-sensitive bilingual dictionary, and how a built-in XDK parser can process user queries. The current work is tailored for the needs of entry-level language learners due to two primary reasons. First, beginners need most help in word sense disambiguation. Second, even small-scale dictionaries, containing less than 1000 words can be still valuable for the students who have very limited vocabulary and word use scenarios.

In addition, it should be noted that ontological information can be very useful not only for the proposed dictionary project. We will see that ontology-based attributes can provide a parser with valuable semantic information, which can potentially increase the quality of parsing. Furthermore, this approach can be considered as a basis for general-purpose word sense disambiguation module.

II. XDG in a Nutshell

XDG is a dependency grammar formalism used to describe natural language structure for further processing by the XDK-supplied parser. There are several basic ideas behind XDG:

Lexicalization. An elementary grammar entry is a natural language word with its user-specified attributes. Each attribute has a name and a value. The most commonly used value types are "string" and "set of strings".

Graph-level principles. By default, any word in the sentence can be linked to any other word. To obtain reasonable parse graphs, a grammar designer should specify a set or restrictions for the links. These restrictions are called *principles*. The simplest example is a *tree principle*: by declaring it, the designer says that the resulting graph should be a tree (technically XDK can build graphs of general form). Other useful principles are *valency principle* and *agreement principle*. We will consider them in more detail later in this section.

Multidimensionality. Each word attribute is declared in a certain user-defined namespace, known as *dimension*. Graph-level principles are also set for a user-specified dimension. Then the parser tries to build a parse graph for each dimension independently. This architecture allows constructing different "views" of the same sentence. For example, one dimension can be used to establish word-to-word dependence links, while another dimension can show linear precedence of words in the sentence.

A. *Valency principle*

Valency principle states that a link between words w_1 and w_2 can be established only if **out** attribute of w_1 matches with **in** attribute of w_2. Consider, for example, the following declaration:

```
defentry { dim lex { word: "eats"}
           dim syn {in: {} out: {subj obj adv*}}}
defentry { dim lex {word: "Peter"}
           dim syn {in: {subj? obj?}  out: {} } }
defentry { dim lex {word: "spaghetti"}
           dim syn {in: {subj? obj?} out: {} } }
defentry { dim lex {word: "today"}
           dim syn {in: {adv?} out: {} } }
```

The specification of **out** attribute of the verb "eats" says that this word can link one subject (`subj`), one object (`obj`) and an arbitrary number of adverbials (`adv*`). The **in**-attribute specification of "Peter" declares this word as a possible subject or object. The same specification is provided for "spaghetti". The word "today" is described as potential adverbial. So if we parse the sentence "Peter eats spaghetti today", the resulting tree will have "eats" as root, "Peter" and "spaghetti" will be linked to "eats" as subjects or objects, and "today" will be linked to "eats" as an adverbial. The *order principle* can be used to set the priority of the first word ("Peter") as a subject.

B. *Agreement principle*

Agreement principle requires the words to share some attribute values in order to be linked. For example, we can say that the verb "eats" should link a subject, having singular third-person word form only. At the same time, the verb "eat" should link other word forms of a subject:

```
defentry { dim lex { word: "eats" }
           dim syn { ...
                     agrs: {["3" sg]}
                     agree: {subj} } }
defentry { dim lex { word: "eat" }
           dim syn { ...
                     agrs: {["1" sg] ["2" sg]
                       ["1" pl] ["2" pl] ["3" pl]}
                     agree: {subj}}
```

Then the word "Peter" should be described as having singular, third-person form:

```
defentry { dim lex { word: "Peter" }
           dim syn { ...
                     agrs: {["3" sg]}
                     agree: {}} }
```

As it is seen from these examples, extensible dependency grammars do not directly support morphological variations in words. Each of word forms ("eat" and "eats") should have its own grammar entry. This limitation, however, is not too strict, since it is possible to generate grammar entries on the fly using external morphology analysis modules.

III. Basic Principles of a Context-Sensitive Dictionary

The work [6] points out the lack of contextualization in commonly used electronic dictionaries. While being adequate for experienced language users, these dictionaries do not provide enough help for beginners to choose the correct translation of a given word in a given context.

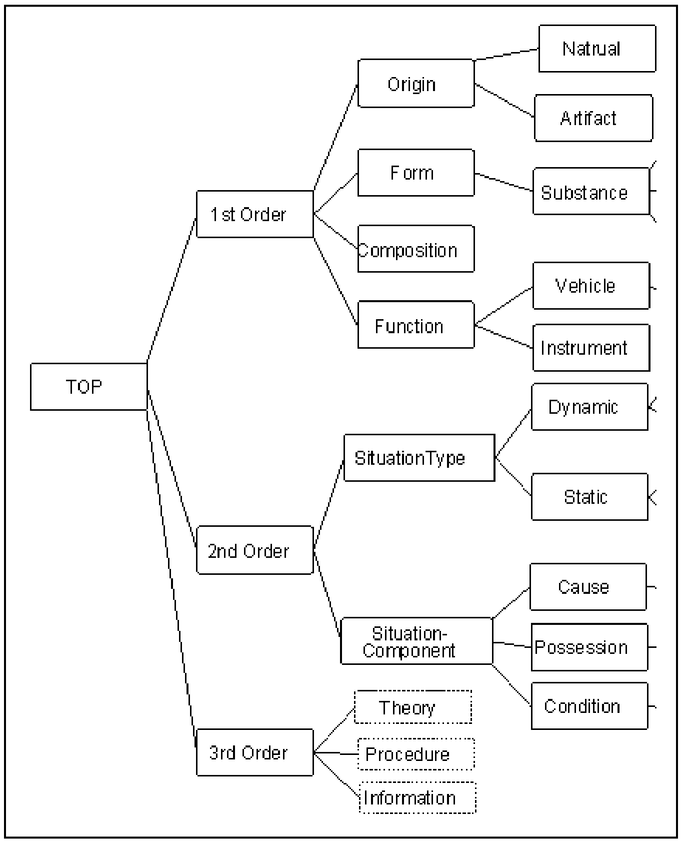

Figure 1. Part of the EuroWordNet Concept hierarchy

For example, the verb *to play* can be translated into Finnish language with, at least, three different words: *pelata* (to play a game), *soittaa* (to play a musical instrument) and *leikkiä* (to play as kids do, e.g. to play police, to play hospital, etc.) The correct word choice depends on the verb's direct object.

This problem is especially clear when dealing with English prepositions. For example, English-Russian LingvoUniversal dictionary lists 18 different definitions for the preposition *by* (and four more definitions for *by* as an adverb). Furthermore, the prepositions do not always have corresponding words in the target language. For example, preposition *to* in the expression "to travel *to* Helsinki" has no direct translation into Finnish. Instead, it affects the form of a verb's direct object: "matkustaa Helsinki*in*". It can be noted here that the reverse translation (from Finnish to English) is simply impossible in this case without a specific context. One can translate "matkustaa" and "Helsinki", but it is harder to find in a conventional dictionary that *-in* ending corresponds to English *to*.

The work [6] suggests to address this problem by declaring a set of dependent words for a given word explicitly. Each word is considered as belonging to a certain class in a chosen ontology (see Figure 1). Polysemic words can have more than one entry: a *table* is found both in *furniture* (kitchen table) and in *mathematical objects* (Excel table).

Dictionary entries are declared (using a grammar-like syntax) with their classes and the classes of their potential dependents as follows:

```
verb PLAY act(any, game))           pelata
verb PLAY act(any, NOT(instrument))

verb PLAY act(any, instrument)      soittaa
verb PLAY act(any, phone)

verb PLAY act(any)                  leikkiä
verb PLAY act(any, child-game)

noun FOOTBALL game()
noun VIOLIN   instrument()
noun PHONE    phone()
noun POLICE   institution()
noun POLICE   child-game()
```

(Here *any*, *act*, *game*, *instrument*, *phone*, *institution*, and *child-game* are classes found in the Concept hierarchy.) For example, the declarations say that the verb *to play*, translated as *soittaa*, has one subject of class *any* and one object of class *instrument* or *phone*.

In the following subsections, we will also use "extended" class names that include the names of the base classes. For example, if class A is derived from B, and B is derived from C, we can address A as C/B/A to avoid additional explanations of relations between these classes.

The proposed formalism includes also means for dealing with grammar cases. However, since morphology and grammar cases are not relevant for English-to-Finnish dictionary, this topic will not be covered here.

In addition to context-based word sense disambiguation, this approach allows declaring multiword expressions. For example, the phrase *bald eagle* should be translated into Finnish as a single word *valkopäämerikotka*. This fact can be declared in the dictionary as follows:

```
noun EAGLE  bird()                kotka
noun EAGLE  bird(property)        kotka
noun EAGLE  bird(property/bald)  valkopäämerikotka
adj  BALD   property/bald()       kalju
```

So if the user enters *eagle* with a dependent word found in the concept class *property/bald* (containing the only word "bald"), the dictionary should suggest *valkopäämerikotka* as a translation of this expression.

Since the classes form a hierarchy, dependent words can be addressed using broader or narrower classes. In the example above, *property/bald* is a subclass of *property*. Therefore, the word *kotka* is used as a translation of *eagle* in a broad context (applicable to any kind of eagle), while *valkopäämerikotka* applies to *bald eagle* only.

IV. Processing User Queries

The XDG-powered bilingual dictionary operates according to the following scenario:

1. The user enters a query (e.g. "we travel to Helsinki").

2. An automated morphology analysis module tags each word with a number of attributes, including base word form, part of speech, person, number, etc. We used the module, developed by Alexey Sokirko [8]. A word can have several different morphological descriptions, since word context is not analyzed at this stage.

3. On the basis of existing dictionary entries and obtained morphological information, the preprocessing module generates the XDG description that includes word entries for the given sentence.

4. The built-in XDK parser processes the user query by means of the generated grammar. As a result, a set of the parse trees for the given sentence is constructed. Each tree node contains, in particular, the translation of the corresponding word.

5. A list of possible translations for each word in the input sentence is returned to the user.

In the following subsections, we will discuss these stages in more detail.

A. Morphology Analysis Stage

Sokirko's English morphology analyzer prints a list of possible entries for each given word in the input sentence. For the example user query, the analyzer provides the system with the following information:

```
> we
WE PN Plural Nominative
      PersonalPronoun FirstPerson
> travel
TRAVEL NOUN Singular Narrative
TRAVEL VERB Infinitive
> to
TO ADV
TO PARTICLE
TO PREP
> helsinki
HELSINKI NOUN Proper
```

For the next stage, our system extracts the matching elements from the dictionary (the second line of each entry is a target Finnish translation):

```
pn   WE       any/people()
     me

verb TRAVEL   act(any, preposition/to/place)
```

```
     matkustaa + [illatiivi]

verb TRAVEL   act(any, preposition/by/using)
     matkustaa + [adessiivi]

verb TRAVEL   act(any, preposition/by/across)
     matkustaa + [partitiivi]

prep TO       preposition/to/place(any/place)
     _

noun HELSINKI any/place()
     Helsinki
```

The two versions of the verb *to travel* with the preposition *by* describe cases such as “to travel by car” and “to travel by the riverside”.

B. *XDG Generation*

Since the basic aim of the project is to provide a simple word-translation tool for the beginners, we decided to extend a simplified English grammar `nut.ul`, provided with XDK. This grammar defines the necessary “skeleton” by declaring tree, agreement, and valency principles. Also it specifies the output format of the resulting parse graphs.

The subsystem of XDG generation extends this ready-made grammar by converting selected dictionary entries into XDG elements. The resulting entries include: (a) source word forms (taken from the input phrase); (b) morphological attributes; (c) dictionary-supplied declarations of the dependent words.

Since extensible dependency grammars do not support hierarchical types, the conversion routine duplicates the same entry for all its superclasses. For example, the entry *worker* of a class *any/people/profession* will get three records in the grammar, corresponding to the following dictionary declarations:

```
noun WORKER any/people/profession()
noun WORKER any/people()
noun WORKER any()
```

For the example phrase “we travel to Helsinki”, the following grammar entries will be generated[1]:

```
defentry { dim lex { word: "we" tran: "me" }
           dim syn { in: {anypeople} out: {}
                     agrs: {["1" pl]} }
}
defentry { dim lex { word: "we" tran: "me" }
           dim syn { in: {any} out: {}
                     agrs: {["1" pl]} }
}
defentry {
    dim lex {  word: "travel"
               tran: "matkustaa + [illatiivi]" }
    dim syn {  in: {act}
               out: {any prepositiontoplace }
               agrs: {["1" sg] ["1" pl] ["2" sg]
                      ["2" pl] ["3" pl] }
               agree: {any} }
}
defentry {
    dim lex {  word: "travel"
               tran: "matkustaa + [adessiivi]" }
    dim syn {  in: {act}
               out: {any prepositionbyusing}
               agrs: {["1" sg] ["1" pl] ["2" sg]
                      ["2" pl] ["3" pl] }
               agree: {any} }
}
defentry {
    dim lex {  word: "travel"
               tran: "matkustaa + [partitiivi]" }
    dim syn {  in: {act}
               out: {any prepositionbyacross}
               agrs: {["1" sg] ["1" pl] ["2" sg]
                      ["2" pl] ["3" pl] }
               agree: {any} }
}
defentry { dim lex { word: "to" tran: "_" }
           dim syn { in: {prepositiontoplace}
                     out: {anyplace} }
}
defentry { dim lex { word: "to" tran: "_" }
           dim syn { in: {prepositionto}
                     out: {anyplace} }
}
defentry { dim lex { word: "to" tran: "_" }
           dim syn { in: {preposition}
                     out: {anyplace} }
}
defentry { dim lex { word: "Helsinki"
                     tran: "Helsinki" }
           dim syn { in: {anyplace} out: {}
                     agrs: {["3" sg]}   }
}
defentry { dim lex { word: "Helsinki"
                     tran: "Helsinki" }
           dim syn { in: {any} out: {}
                     agrs: {["3" sg]}   }
}
```

Each grammar entry has the following attributes:

1. the corresponding word, taken from the input sentence (as *word*);
2. its translation as *tran*;
3. word class name as *in*;
4. the classes of the dependent words as *out*;
5. number and person as *agrs* (for nouns; for verbs, the attributes of a potential subject are used);
6. (for verbs) the class of the potential subject as *agree*.

There are also two technical points in this process. First, it is necessary to list all used class names as admissible graph labels in the grammar:

```
deftype "syn.label" { any anypeople ... }
```

Second, a special end-of-sentence word should be added to the grammar (we use dot):

```
defentry {
  dim lex {word: "." tran: ""}
  dim syn {in: {}
           out: {act}
           agrs: top  }
}
```

The dot agrees with the verbs (of class *act*) and represents a root of a parse tree (*top*). The dot marker should be also added to the end of the user query.

C. *Processing User Query*

At this stage, the XDK parser is invoked. The parser generates a set of possible parse trees, thus giving a possibility

[1] The slash symbol is not allowed in XDG as a part of a type name, so the conversion routine has to remove it.

to collect potential translations for any given word in the input sentence. The output can be requested both in a visual and in a machine-readable form. The parse tree for the example sentence “we travel to Helsinki” is shown in Figure 2.

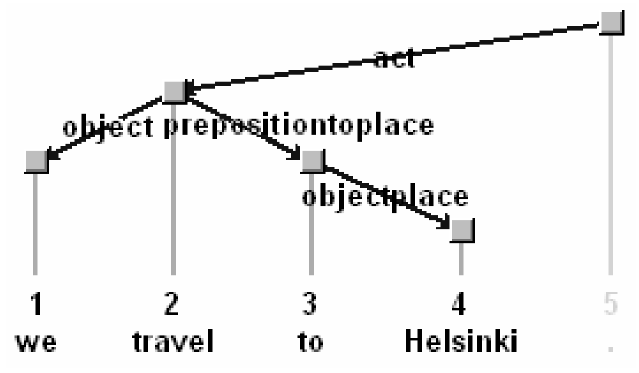

Figure 2. Parse tree of the phrase “we travel to Helsinki”

The system suggests the following translations:

```
we:          me
travel:      matkustaa + [illatiivi]
to:          _
Helsinki:    Helsinki
```

V. Discussion

In some subjects, such as physics and chemistry, educational software has developed significantly from interactive electronic books to complex virtual labs. In contrast, software solutions used in computer-assisted language learning are still relatively simple and provide few means to experiment with the language. To give more possibilities to the learners, such software should make more use of natural language processing technologies.

The creation of NLP-powered language learning tools is challenging, mainly due to the following reasons: (a) the current state of the art in NLP is not adequate for many educational tasks; (b) NLP software is often language-dependent and not tailored for learners’ needs; (c) this work is very labor-intensive.

Nevertheless, as NLP technologies become more mature, it seems quite natural to try to use them in educational software. Recently, a project proposal entitled “NLP for Future Schools” was submitted for EU-funded Seventh Framework Programme [9]. The project is specifically aimed at creation of new-generation language learning software[2].

The context-sensitive dictionary described in this paper is one of the attempts to utilize NLP methodologies in educational software. Being a prototype, the dictionary has not been tested yet in a real school environment, but we believe it can be used, at least, as an auxiliary educational instrument.

Furthermore, the inclusion of concept classes hierarchy into extensible dependency grammar formalism opens new possibilities to express word-word relationships within XDG framework. It can be useful for designing precise parsers and word sense disambiguation modules.

[2] Source: personal communications with the person in charge at the University of Joensuu, Finland. As of today, the proposal is under review.

VI. Conclusion

We have implemented and tested a simple version of a context-sensitive bilingual dictionary, described in [6]. A well-known XDK framework was used as an underlying natural language processing engine. We have shown how dictionary constructions can be converted into extensible dependency grammar entries, making possible to utilize an XDG-based parser for phrase analysis. It was also demonstrated how to use hierarchical types in XDG by “flattening” subclasses into a collection of entries, having non-hierarchical types. The dictionary automatically disambiguates words in the sentence, using their local contexts, and provides an appropriate translation for each word. If the context gives not enough information, all alternative translations are returned to the user.

We plan to develop the dictionary further to a full-fledged educational instrument. We also hope to motivate more research projects aimed at adapting NLP methods for the use in educational software.

References

[1] J. Nivre, "Dependency grammar and dependency parsing. Technical Report 05133," *MSI report*, 2005.

[2] M. Covington, "A Fundamental Algorithm for Dependency Parsing," *Proc. of the 39th Annual ACM Southeast Conference*, 2001, pp. 95-102.

[3] R. Debusmann, D. Duchier and J. Niehren, "The XDG Grammar Development Kit," *Lecture Notes in Computer Science*, 2004, vol. 3389, pp. 190-201.

[4] R. Debusmann, D. Duchier and G. Kruijff, "Extensible Dependency Grammar: A New Methodology," *Proc. of the COLING 2004 Workshop on Recent Advances in Dependency Grammar*, 2004.

[5] O. Bojar, "Czech Syntactic Analysis Constraint-Based, XDG: One Possible Start," *Prague Bulletin of Mathematical Linguistics*, 2004, vol. 81, pp. 43-54.

[6] M. Mozgovoy and T. Kakkonen, "An Approach to Building a Multilingual Translation Dictionary that Contains Case, Prepositional and Ontological Information," *Proc. of the 12th International Conference on Humans and Computers*, 2009, pp. 135-139.

[7] P. Vossen, L. Bloksma, H. Rodriguez, S. Climent, N. Calzolari, A. Roventini, F. Bertagna, A. Alonge and W. Peters, "The EuroWordNet Base Concepts and Top Ontology," *Deliverable D017D034D036 EuroWordNet LE2-4003*, 1997.

[8] A. Sokirko, "Morphological modules on the website www.aot.ru (in Russian)," *Proc. of Dialog'04 International Conference*, 2004, pp. 559-564.

[9] European Commission, "Seventh Framework Programme," *http://cordis.europa.eu/fp7*, 2009.

[10] S. Kübler, R. McDonald, J. Nivre, "Dependency Parsing," *Morgan & Claypool Publishers*, 2009, 127 p.

[11] V. Tusov, "Computer Semantics of the Russian Language" (in Russian), St. Petersburg University Press, 2004, 400 p.