Extensible Dependency Grammar for Education: Ideas and Experiments

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 discusses how XDG can be used to build computer-assisted language learning instruments, such as a context-sensitive dictionary, a cases and prepositions learning tool, and a grammar checking module. The paper outlines several ideas, currently being researched by the author and his colleagues, and some of the current experiments.

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 subsequent text processing stages [2]. Since dependency-based parser only connects existing words (and does not create new nodes in the parse graph), the results are easier to analyze and interpret. 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].

Our current experiments with XDG are primarily concentrated on using handcrafted XDG-style grammars for the purposes other than plain text parsing (we are mostly interested in computer-assisted language learning). In the future, we also plan to experiment with automatic grammar generation by elaborating ideas proposed in [5].

In contrast to physics and chemistry, where educational software has developed significantly from interactive electronic books to complex virtual labs, computer-assisted language learning still relies on relatively simple educational instruments that provide few means to experiment with the language. We are planning to supply the students with the following tools:

1) *A context-sensitive bilingual 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, which can be handy for the beginning language learners.

2) *A system for learning cases and prepositions*. Case system and prepositions pose a particular problem in learning foreign languages. Schmied [12] mentions that traditional grammar books and dictionaries present no detail information on prepositions. However, prepositions and cases have specific syntactic and semantic requirements and the choice of the correct preposition/case is often dependent on the meaning of the syntactic element that determines it. Context-sensitive grammar definitions of cases and prepositions should help the learners to improve their skills in these language features.

3) *A basic grammar checking system*. Most available grammar-checking systems are not specifically tailored for the needs of language learners. As Krishnamurthy notes, "As a result of my testing, I am convinced that this feature [*MS Word's spelling and grammar check*] works well for good writers and not for bad ones. Good writers follow most of the rules and this feature can help them on the margins. If you are a bad writer with a poor understanding of the rules, this feature will not help you at all. This is, clearly, a problem. The feature does not help those who can most benefit from it." [13] Simpler systems with more capabilities for language learners would be more appropriate in educational environment (and there are attempts to build such a tool [14, 15]).

This paper outlines the experiments we used to design the prototypes of above mentioned instruments. The experiments were performed in order to form a full-scale project proposal for the EU-funded Seventh Framework Programme [9]. The intended project is specifically dedicated to the use of NLP technologies in educational environments.

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". Typical attributes include grammatical categories, such as gender, case, number, and specifications for admissible dependent words (e.g., a noun can link a dependent adjective).

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 agrees 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 all 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. DECLARING LOCAL CONTEXT OF WORDS IN XDG

Extensible dependency grammars do not provide explicit instruments to declare context-sensitive properties of words. Such a possibility is handy, e.g., for electronic dictionaries. If the user provides the word to be translated along with its dependents, the dictionary would be able to find a more appropriate translation.

The works [6] and [16] explain how word contexts can be used in bilingual electronic dictionaries, and how to declare them using XDG statements. The basic idea is to declare a corresponding *concept class* for every word, listed in the grammar. These classes can be taken from any existing system, such as EuroWordNet's [7] top ontology (see Figure 1). The word's class is listed as a user-defined attribute in the corresponding XDG statement. Next, the XDG agreement principle is extended to include word classes in addition to grammatical categories. This idea can be illustrated with the following simplified syntax:

```
// "to play" as in "to play football":
// class act, any subject, object of class game
PLAY_1 act(subj: any, obj: game))

// "to play" as in "to play piano":
// class act, any subject,
// object of class instrument
PLAY_2 act(subj: any, obj: instrument)

FOOTBALL game()
VIOLIN   instrument()
```

(Here *any*, *act*, *game*, *instrument* are classes found in the ontology). We should distinguish "to play football" from "to play piano", for example, in the case of electronic English to Finnish dictionary, since in Finnish the corresponding verbs are different (*pelata* for playing a game, and *soittaa* for playing a musical instrument).

Unfortunately, current XDG syntax does not directly support hierarchical types, so each word entry should be duplicated with

all its possible superclasses. For example, the entry *worker* of class *any/people/profession* will get three records in the grammar, corresponding to the following dictionary declarations:

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

We consider this peculiarity as a minor disadvantage, since in our experiments extensible dependency grammars are normally auto-generated on the fly from higher-level descriptions.

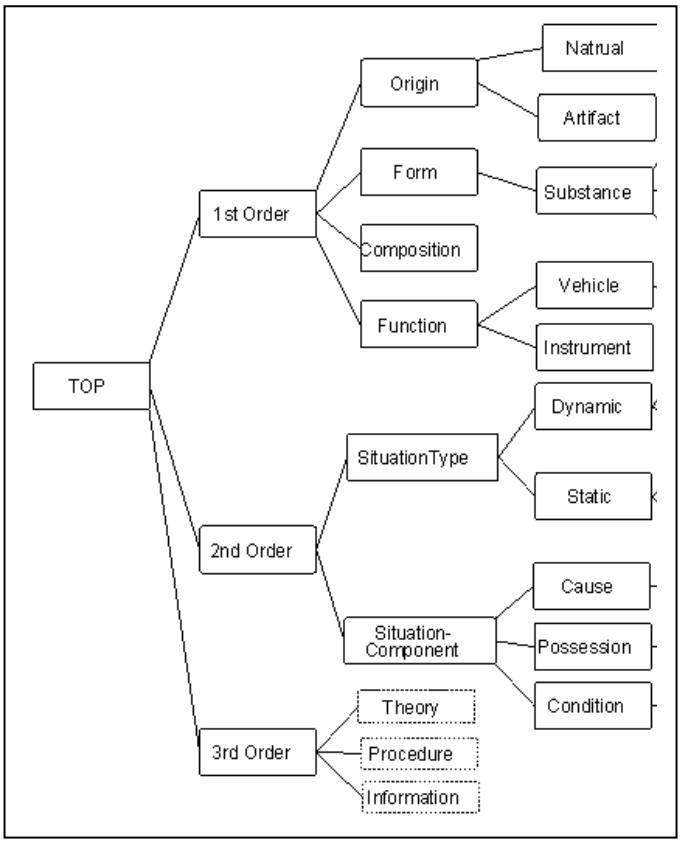

Figure 1. Part of the EuroWordNet Concept hierarchy

IV. Using Word Dependencies in a Context-Sensitive Dictionary and a Cases/Prepositions Learning Tool

In the previous section, we already mentioned the problem of ambiguity of words to be translated (such as the verb *to play*). This ambiguity becomes especially challenging for the language learners in case of English prepositions. For example, English-Russian LingvoUniversal dictionary lists 18 different definitions of the preposition *by* (and 4 more definitions of *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*.

A. Processing User Queries

The prototype of the XDG-powered English-to-Finnish 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 (see Table 1).

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 (specified as a user-defined attribute of the word).

A list of possible translations for each word in the input sentence is returned to the user. The simplified definitions of the entries needed to translate the phrase "we travel to Helsinki" are provided below (see [16] for more details).

```
WE        any/people()
          { finnish: "me" }

TRAVEL_1  act(any, preposition/to/place)
          { finnish: "matkustaa + [illatiivi]" }

TRAVEL_2  act(any, preposition/by/using)
          { finnish: "matkustaa + [adessiivi]" }

TRAVEL_3  act(any, preposition/by/across)
          { finnish: "matkustaa + [partitiivi]" }

TO     preposition/to/place(any/place)
HELSINKI any/place()
```

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:

```
EAGLE  bird()           { finnish: "kotka" }
EAGLE  bird(property)   { finnish: "kotka" }

EAGLE  bird(property/bald)
       { finnish: "valkopäämerikotka" }


BALD   property/bald()  { finnish: "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.

```
   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]}   }
}
```

Table 1. XDG statements for the phrase "we travel to Helsinki".

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.

B. Learning Prepositions and Cases

By limiting word contexts with preposition + verb + noun chains, we can obtain a tool for learning prepositions and case system of the language. As already mentioned, this topic (verb government) is challenging for most students, so a separate educational instrument would be handy.

In the simplest scenario, the student can just enter the phrase, and let the system check it. If the phrase is correct, the built-in XDK parser will build a complete parse tree (see Figure 2).

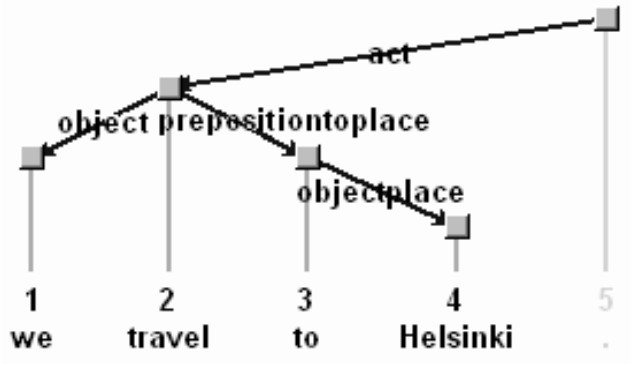

Figure 2. Parse tree of the phrase "we travel to Helsinki"

Other relatively easily implementable scenarios include:

1) Helping the user to complete the phrase. For example, the user enters "we travel", and the system shows which are the options (with explanations of the meaning of each alternative).

2) Asking the user to select the correct preposition and/or verb form for a given subject — verb — object phrase. For example:

```
(system)> me matkustamme Helsinkiin
          (we, travel, Helsinki) ?
(user)> we travel to Helsinki
(system)> correct!
```

For the second scenario, an additional module that generates user-specified morphological forms is needed. For Finnish, we used freely available Omorfi project [18]. For the simplest grammatical constructions, the phrases to be translated by the student can be even auto-generated, since the system has sufficient knowledge about the words and the word-linking rules.

C. Grammar Checking

Naturally, a high-quality automatic grammar checker is a great support for any language learner. However, most grammar checkers are intended to catch occasional mistakes, more typical for native speakers than to language learners. Attempts to build such a student-oriented grammar checker are rare [14, 15], while general-purpose checkers cannot detect most of the mistakes, made by the students [17].

XDK comes with several example grammars (of a limited size) that can be used as grammar checkers: if the XDK parser can build a complete tree for the given phrase, it is considered grammatically correct. While we haven't performed reliable experiments yet, we believe that extensible dependency grammars can serve as high-quality grammar checkers for the needs of the novice language learners.

Today, a typical grammar checker works as follows. First, a sentence is analyzed with a part of speech tagger and a syntactic parser in order to get its structural representation. Next, the

obtained structures are checked against a set of "mal-rules" that represent typical user-made errors [11]. In most cases the parser is robust, i.e. it tries to build a parse tree even if the sentence is incorrect (this behavior is entailed by the probabilistic nature of most popular parsing algorithms that try to build the most probable parse tree regardless of the phrase's correctness).

In case of novice language learners with very limited active sets of words and grammar rules exact rule-based XDG parsing can be more beneficial, since:

(a) grammatically incorrect sentences are simply non-parsable;

(b) for each grammatical error it is possible to point out its exact reason (why a certain construction is incorrect and which other options/rules are available in this case).

Here we should again emphasize that the beginner's vocabulary and grammar are very limited, so the manual creation of the corresponding XDG system is feasible.

At the same time, there is an (already mentioned) alternative method of creating an extensible dependency grammar: the grammar rules can be extracted automatically from available treebank data [5]. However, this technique is still not mature enough to be used in real-life projects.

One should note, though, that the current version of XDK is not well-tailored for grammar checking. It lacks error-reporting capabilities (only partial information on parse errors is available), and the parser output needs significant post-processing in order to be used in a grammar checking module.

V. Conclusion

The creation of NLP-powered language learning tools is challenging, mainly due to the following reasons:

- the current state of the art in NLP is not adequate for many educational tasks;

- 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. The ideas and experiments described in this paper are among attempts to do it.

The core supporting technology for our research is XDK framework, developed by Ralph Debusmann. As we discovered, extensible dependency grammars are powerful enough to describe local word contexts, necessary for a context-sensitive bilingual dictionary and a preposition/cases learning tool that we have presented in this work. Certain minor XDG disadvantages (such as lack of hierarchical types) can be overcome with simple technical tricks.

We also believe that XDG can be a powerful back-end for a student-oriented grammar checking software, able to catch common mistakes in simple natural language phrases.

On the other hand, XDG itself is just a language for writing grammars. The key idea of our approach is to introduce hierarchical object types into grammar statements. This technique was never applied in the original XDK collection of example grammars.

Another important technology for our project is automatic morphology analyzer and word form generator, i.e. a module that can discover morphological attributes of a given word form or generate a word form, given a base form and a set of attributes. We used AOT [8] for English and Russian, and Omorfi [18] for Finnish.

We plan to develop our tools further to full-fledged educational instruments. 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] A. Arppe, "Developing a grammar checker for Swedish," *The 12th Nordic Conference of Computational Linguistics*, 2000, pp. 13-27.

[12] J. Schmied, "Learning English prepositions in the Chemnitz Internet Grammar," *Language and Computers*, vol. 48(1), 2003, pp. 231-247.

[13] S. Krishnamurthy, "A Demonstration of the Futility of Using Microsoft Word's Spelling and Grammar Check," *http://faculty.washington.edu/sandeep/check/*

[14] O. Knutsson, T. Pargman, K. Eklundh, "Transforming Grammar Checking Technology into a Learning Environment for Second Language Writing," *Proc. of the HLT-NAACL 03 Workshop*, 2003, pp. 38-45.

[15] E. Bender, D. Flickinger, S. Oepen, A. Walsh, T. Baldwin, "Arboretum: Using a precision grammar for grammar checking in CALL," *Proc. of InSTIL/ICALL Symposium*, 2004.

[16] M. Mozgovoy, "Declaring Local Contexts of Words with Extensible Dependency Grammar," *Proc. of the 3rd Int'l Conference on Human-centric Computing*, 2010, pp. 1-5.

[17] J. Johannessen, K. Hagen, P. Lane, "The performance of a grammar checker with deviant language input," *Proc. of the 19th International Conference on Computational linguistics*, 2002, pp. 1-8.

[18] "Open Morphology for Finnish Language", *http://gna.org/projects/omorfi*