SOME APPROACHES TO THE GENERATION OF SENTENCES IN NATURAL LANGUAGE FROM UNL

Aram Avetisyan
Institute of Informatics and Automation Problems of
National Academy of Sciences of the Republic of Armenia
Yerevan, Armenia
AramAvetisyan@gmail.com

ABSTRACT

The purpose of this paper is to present some solutions used in the new UNL to Natural Language (NL) generation software, the results achieved in its development, and the basic advantages of the new Algorithmic system over the old one, giving the outline of the specificity of some key algorithmic solutions.

Keywords

Computer science, informatics, information, semantic, language, translation.

1. INTRODUCTION

1.1 UNL (Universal Networking Language) is a meta language for representation of semantic information. Its main purpose is to store "the meaning" of natural language texts in a language independent format. UNL data can be used for various types of processing such as, translation (NL1 to UNL to NL2) semantic search (language independent), data aggregation and navigation etc.

The UNL expresses information in form of semantic networks with hyper-nodes. In other words each sentence in UNL is a directed linked graph. Different from natural languages, UNL expressions are unambiguous. In the UNL semantic networks, nodes represent concepts, and arcs represent relations between concepts. Concepts can be annotated. These concepts are called "Universal words" (also referred to as UWs). The arcs connection UWs are called "relations". They specify role of each word in a sentence. The subjective meaning intended by the speaker can be expressed through "attributes". In addition, the "Knowledge Base (UNLKB)" is provided to define semantics of UWs. The UNLKB defines every possible relations between concepts including hierarchical relations and inference mechanism based on inclusion relations between concepts. Thus, the UNLKB provides semantic background of the UNL to make sure the meaning of the UNL expressions is unambiguous.

1.2 UNL to NL Generation - One of the key tools needed for the functioning of a UNL based system is a UNL to NL generation software. So far only one such tool was available. The tool is called UNL DeConverter, it was provided by UNL Center in Tokyo. Developed in the beginning of the UNL project in end 90s it was meant mainly for testing and development of linguistic resources. These resources are natural language word dictionary, natural language grammar generation rules and Co-occurrence Dictionary. Although the tool is very helpful for testing and development it wasn't meant for large scale application implying multithreading, high speed, and large documents to process. Also the technology used for its development has created several difficulties for the improvement.

1.3 UNL Dictionary stores word entries of the natural language with corresponding Universal Words which are language independent concepts and grammatical attributes of the entry in terms of the target natural language grammar. Dictionaries are being stored in text documents which are then being compiled into a native code using a special tool. These native code files are being fed to other UNL processing Software, such as DeConverter.

1.4 Generation Rules - Deconversion (or "generation" in general) rules describe the conditions for rule application: the way of rewriting the attributes of nodes that satisfy those conditions, as well as the way of composing a natural language sentence.

1.5 The algorithm of NL Generation - When DeConverter process starts, a sentence represented by a UNL expression (that is, a set of binary relations) is processes as a linked directed graph structure called Node-net. The root node (a starting point for algorithm) of a Node-net is called Entry Node and represents the main predicate of the sentence (Figure 1). This node is represented in the UNL expression with "@entry" attribute (Example 1). In the output DeConverter returns a word list with morphology containing NL words matched for the concepts from the UNL-NL Dictionary. The order of the words is being produced according to the generation rules. The algorithm basics are the following. On first step the entry node is being placed in to the output list, then on each next step the DeConverter searches for a pair of related nodes, such, that one of the nodes is already in the output list and the other is still in the graph. By finding one it looks up for rules to be matching attributes of both nodes respectively. By finding matching rule it is being applied adding the linked node to the output list in a position described by the rule.

Thus, the generation of the word (node) list in the target language basically depends on the generation rules and dictionary provided for that language. The main workflow of DeConverter is also described by the rules. They specify the direction the algorithm goes and the actions it makes. Thus the development of generation rules and dictionary is an essential part in generation process.

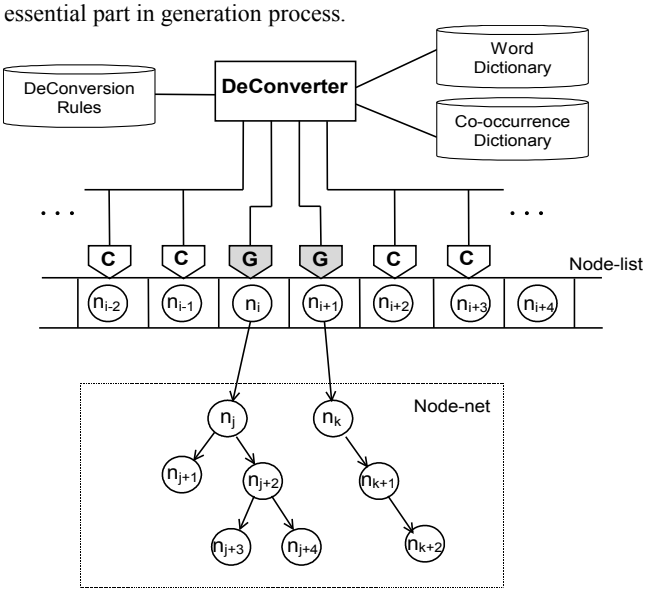

Figure 1

English: The boy reads.
UNL: agt(read(icl>do).@enrty.@present,boy(icl>person))

Example 1

2. Disadvantages of DeConverter

Below we are bringing some of the main disadvantages of the existing NL generation tool (DeConverter) that we have tried to improve in the new software:

1. The DeConverter is a single threaded application. It can run only one instance at a time when called from a same process. This brings up an obstacle for building web applications based on it.

2. The DeConverter is a win32 application which can be ran only on Windows platform, while most of the professional internet servers are running Linux/Unix OS. So, This is an essential point, because the UNL is an open project and it isn't supposed to be built only for a commercial OS.

3. At the time when the DeConverter was developed, the machine resources where strictly limited and the resource economy was of one of the highest priorities in software development, that is the reason why the DeConverter has limitations on the number of used attributes, and why it gets too slow when the processing large data.

4. Another problem that UNL developers meet during their work, is the limitation of the rule syntax. This limitations in many cases prevent the users to generalize some rules to cover more case, which brings to a bigger number of rules and slows down the performance.

5. The tools has some essential bugs and lacks the proper interface for tracing/debugging the resources developed.

6. The sources of the tool are not available for improvement.

3. The 2nd Generation

At the end of 2008 the development of 2nd generation of NL Generator has started. The new NL Generator has the code name "jDeCo". It is being developed in Java, which guarantees its platform-independence, both as web and standalone applications. The engine of the jDeCo is being developed separately from its interface, so the application can be available as a Java library and be implemented into any type of application, like web service, web application, standalone or even mobile application. This fact significantly expands the range of jDeCo's possible applications. Due to its new algorithmic solutions, there are no restrictions on the number of rules, dictionary entries, attributes or the UNL document, the only limit is the limit of the machine memory.
Below we will illustrate some examples of a UNL Sentence, and some algorithmic solutions used in JDeCo.

The *example 2* illustrates a UNL sentence node ([S:1]… [/S]) that includes the {unl}…{/unl} segment containing the UNL sentence relations between concept nodes called "Universal Words". So the UNL sentence is a number of Universal Words related to each other.

```
[S:1]
{unl}
aoj(outcome(icl>result):0W.@past.@def.@entry,
        description(icl>action):00.@topic)
mod(outcome(icl>result):0W.@past.@def.@entry,
        collaboration(icl>action):1B.@def)
obj(description(icl>action):00.@topic,  Egypt:0H)
agt(collaboration(icl>action):1B.@def, :01)
agt(accompany(agt>thing,obj>thing):3F.@past,   :01)
aoj(more(aoj>thing):1S,        :01)
aoj(prominent(aoj>thing):26, :01)
and:01(scientist(icl>scholar):2W.@entry.@pl,
        scholar(icl>person):2J.@pl)
bas(more(aoj>thing):1S,        150:22)
obj(accompany(agt>thing,obj>thing):3F.@past,
        Bonaparte(iof>person):3R)
gol(accompany(agt>thing,obj>thing):3F.@past,   Egypt:44)
tim(accompany(agt>thing,obj>thing):3F.@past,   1798:4D)
{/unl}
[/S]
```

Example 2

The first relation describes aoj type of relation between concepts "outcome(icl>result)" with id "0W" attributes ".@past, .@def, .@entry" and "description(icl>action)" with id "00" and attribute ".@topic".

NL Generator transforms the sentence represented by an UNL expression - that is, a set of binary relations - into the directed hyper graph structure, called Node-net. The root node of a Node-net is the Entry Node. It then applies generation rules to every node in the Node-net respectively, and generates the word list in the target language. In this process, the syntactic structure is determined by applying Syntactic Rules, while morphemes are generated by applying Morphological Rules.

3.1 Rule Applying

One of the key differences of jDeCo from its predecessor DeCo is the rule applying order. In DeCo the only ordering criteria is the priority of the Rule, while in jDeCo the type of the Rule also takes part in ordering. In jDeCo the Morphological Rules can be applied only after the Syntactic Rule applying process has been finished, making sure that all nodes from Node-net were found in dictionary and correctly inserted into the Node-list. The fact that in jDeCo all dictionary entries have nominative case (not like DeCo), significantly accelerates dictionary developing process. After the Syntactic Rule applying process is finished, the Node-list will consist of ordered UWs in nominative case, and the Morphological process may start. During that process, Morphological Rules are being applied to the nodes in Node-list changing their form, case, etc... Only after that, the generation process is finished.

3.2 Matching Process

Another new algorithmic solution is the method of attribute search, used for matching the appropriate rule and dictionary entry. In DeCo, after parsing all the input data, matrixes "attribute-rule" and "attribute-dictionary entry" are being created, for entry and rule matching, thus the quantity of elements are limited by the dimensions of matrixes. This problem was solved in jDeCo by using binary operations for the most of matching tasks.

After parsing the input, each attribute gets a numerical ID (2^1, 2^2, 2^3, 2^4…) presented binary (0001, 0010, 0100, 1000...). Thus, because all dictionary entries and rules may have several attributes with their binary IDs, they also will have a new element, which is the summary of their attributes' binary IDs. For example if a rule has attributes with binary IDs 0010, 0001 and 1000, their summary will be 1011. This allows us to know exactly what attributes the rule has, by having the attributes' summary only.

Suppose that a Rule has attributes summary value 101101, which means that it demands the Dictionary Entry to have attributes A, C, D and F respectively. If the Entry, currently being checked, has attributes summary 011111(ABCDE), to check if the Entry matches the Rule, the program will make a simple binary "AND" (1 AND 1 = 1, 1 AND 0 = 0, 0 AND 0 = 0) operation with the attributes' summary values:

101**1**0**1** = F DC A **AND**
011**1**1**1** = EDCBA
001**1**0**1** = DC A

The result doesn't equal to the Rule attributes summary (101101 ≠ 001101), so the entry is not valid for the rule. Now let's suppose that another Entry has attributes summary 111101.

1**0**1101 = F DC A **AND**
1**1**1101 = FEDC A
1**0**1101 = F DC A

In this case the binary AND returns a result of 101101 which equals to the Rule's attributes summary, this means that it contains the attributes demanded by the rule. The same algorithm is used for matching the appropriate Rules for the Node in Condition Window.
Now let's see an example from jDeCo NL generation process

Description de l'Egypte was the outcome of the collaboration of more than 150 prominent scholars and scientists who accompanied Bonaparte to Egypt in 1798.
{unl}
aoj(outcome(icl>result):0W.@past.@def.@entry,
description(icl>action):00.@topic)
mod(outcome(icl>result):0W.@past.@def.@entry,
collaboration(icl>action):1B.@def)
…
…
…

{/unl}

The Node "outcome(icl>result)" is the Entry Node of the sentence, so this Node is inserted into Node-List on the first step of generation. In UNL sentence it has attributes "past" and "definite". In dictionary this Entry is "[outcome] {}"outcome(icl>result)"(N)<A,0,0>;", where it has attribute "N(Noun)". So generally the word "outcome" has attributes summary of "past", "definite" and "noun" 2203404206082 (in decimal numeration), and while process of matching a rule for this node, the rule must have no other attribute, but these ones. Suppose that in this case the rule that fits this Node is "I"::aoj"(::)P1;", it has no attributes, but as there were no other rules for "aoj" with the mentioned attributes, this Rule is being applied. Now let's see another case where the generation matches a Rule having the needed attributes for the Node.

Let's assume that the generation is on the stage of morphological analysis, where the output result is "<<< Egypt description was the outcome collaboration of more 150 prominent << scientists and scholars >> accompany Bonaparte Egypt 1798 >>>", and the Condition Window is on the Node "collaboration of " with attributes "N", "def", "gen" and attributes summary 1099797102594. Here the generation, using the binary AND operation, matches this Node with morphological rule "m(N,@def: -@def , the &)P210" (if the Node is a noun and is definite, remove the attribute @def, and add a string "the " to the beginning of the Node) with attributes "N" and "def", and att. summary 1099796840450

10000000000010001000001000000000000000010 *(1099797102594)*
AND
10000000000010001000000000000000000000010 *(1099796840450)*

10000000000010001000000000000000000000010 *(1099796840450)*

1099796840450 = attributes "N" and "def"

If there is no other matching rule with higher priority (>210) this rule will be applied to the Node, after that action the Node will be "***the*** *collaboration of* ".

4. CONCLUSION

jDeCo has completely solved the mentioned disadvantages of DeConverter.

jDeco is platform-independent, supports multiple threads, so it can run several instances at the same time. jDeCo is limitied only by the machine resources, so large data processing now will be much faster. Fewer limitations on the rule syntax, the rules now are more flexible, allowing the users to generalize them to cover more cases. This helps to develop more hierarchal and structured rules with minimized quantity and better performance.

Currently the jDeCo engine is in a beta version as a web application and standalone. Along with the 1st generation DeConverter it will be available in UNL Platform, a web framework that is being developed by the UNDL Foundation. The jDeCo is still in development process.

5. REFERENCES

[1] "The Universal Networking Language (UNL) Specifications", v.3 edition 2, *UNDL Foundation*, 2003

[2] Vahan Avetisyan, Robert Urutyan, Liana Hovsepyan, Susanna Tioyan, "Development of deconversion rules fro genegation of Armenian sentences from UNL", *Institute for Informatics and Automation Problems of National Academy of Sciences of the Republic of Armenia* 2005-8

[3] Uchida H., Zhu M., "The Universal Networking Language (UNL) specifications" version 7, *UNDL Foundation*, June 2005.