Semantic Mapping between UNL Relations and Software Components to the Execution of Natural Language Requisitions

Flávia Linhalis, Dilvan de Abreu Moreira
Institute of Mathematics and Science Computing (ICMC)
University of São Paulo (USP), São Carlos, Brazil
{flavia,dilvan}@icmc.usp.br

Abstract

This paper describes a new approach to the execution of imperative natural language (NL) requisitions. Like some other approaches, found in the scientific literature, our work uses restricted natural language to describe user intentions and software components to execute them. The advantage of our approach is that natural language requisitions are first converted to an interlingua, UNL (Universal Networking Language), before the suitable component, method and arguments are retrieved to execute each requisition. The interlingua allows the use of different human languages in the requisition (other systems are restricted to English). The NL-UNL conversion is preformed by the HERMETO system. In this paper, we also describe the SeMaComp (Semantic Mapping between UNL relations and Components), a module that extracts semantic relevant information from UNL sentences and uses this information to retrieve the appropriated software components.

1. Introduction

Natural Language has been the most convenient and usual way for communication between human beings. From the user point of view, the possibility to access computer functions with the same language used to interact with other people is very interesting, even in a restricted context. That is why several systems, developed throughout the last twenty-five years, have pursued the goal of describing user intentions in restricted natural language and have them executed by computers [1,2,3,4].

Our goal is to be able to execute user requisitions described in restricted natural language. In this paper, requisitions refer to user intentions described in a high level of semantic abstraction and related to a specific domain. For example, considering the domain of web course management, valid requisitions could be:

(a) "Add student John Smith to the Hypermedia course."
(b) "Send an e-mail to the students of the Operating Systems course saying that the test will be on December 14th."

This work goal is to retrieve and activate software components to execute user requisitions (made in restricted natural language). In order to do this, user requisitions are converted into UNL (Universal Networking Language) to extract relevant semantic information that will be used to retrieve and execute software components. These requisitions will be executed through the activation of specific component methods.

The main advantage and innovation of our approach is the use of UNL as an interlingua. In this way, natural language requisitions, expressed in different human languages, can be translated into the same UNL representation before being executed. To convert natural language into UNL, the HERMETO [9] system was used.

To achieve our goal, a new system, the SeMaComp (Semantic Mapping between UNL relations and Components) system is being developed. It uses an ontology and a mapping structure to identify what components, methods and arguments will be necessary to execute requisitions expressed in UNL.

This paper is organized in the following way: Section 2 discusses related works about execution of natural language requisitions. Section 3 describes the UNL project and the HERMETO system. Section 4 introduces some concepts related to ontologies. Section 5 presents an imperative natural language requisition system using the SeMaComp. Section 6 describes an application in the web course management domain. Section 7 concludes the paper with some remarks on future work.

2. Related Work

The first efforts to execute user requisitions expressed in natural language began in the later 70s. The NLC (Natural Language Processing) system [1] was designed to process data stored in matrices or tables. It enables a computer user to type English commands into a display terminal and watch them executed on the screen. A more recent example of the same idea is NaturalJava system [2]. It accepts English sentences as input and generates the Java source code to execute the sentences. Both systems are very limited because input must be in a restricted algorithmic fashion. Higher semantic level sentences, are not allowed.

Some approaches, such as the OAA (Open Agent Architecture) [3,5] and SOTA [4], have worked with software components and agents to get a higher level of abstraction. OAA is a framework for constructing agent-based systems that makes it possible for software services to be provided through the cooperative efforts of distributed collections of agents. OAA accepts English sentences as input that are converted to ICL (Interagent Communication Language), a Prolog-based language. ICL is used, by the agents, to communicate with each other and to register their capabilities with a facilitator agent. The facilitator is responsible for matching ICL requests to choose the most suitable agents to execute these requests.

SOTA is an office task automation framework that uses web services, ontology, and software agents to create an integrated service platform that provides user-centric support for automating intranet office tasks. SOTA can take plain English text sentences as input and serve users with a single and integrate user-interface form to access web services, thus avoiding the need to access each distributed service manually. SOTA performs its tasks in three phases: first it parses user input sentences to identify possible web services using an ontology, next it prepares most of the input data fields the services requires, and, finally, it combines related services to define a single task flow.

Such as OAA and SOTA, our work uses restricted natural language to describe user requisitions and software components to execute these requisitions. One of the major differentials of our approach is that the natural language requisitions are first converted to an interlingua (UNL [6]), and then the requisitions are analyzed and the appropriated component methods are called (to process the requisitions). References to systems that convert user requisitions to an interlingua and use that interlingua semantic information to choose the appropriated components have not been found in the literature. The advantage of using an interlingua is that several languages (English, Spanish, Portuguese, French, etc.) can be used to describe the user requisitions. UNL is an interlingua and its goal is to break down, or at least to drastically lower, the language barrier for the Internet users. This seems to be the major obstacle standing in the way of international and interpersonal communication in the information society [7].

3. The UNL Project

The UNL project started in 1996 and currently embraces several universities and research institutions in the world. The project proposed an interlingua, entitled Universal Networking Language (UNL), which has sufficient expressive power to represent relevant information conveyed by natural languages. For each natural language, two systems should be developed: a "Deconverter" capable of translating texts from UNL to this natural language, and an "Enconverter" which has to convert natural language texts into UNL.

The procedure of producing a UNL text is not fully automatic. It is an interactive process with the labor divided between a computer and a human expert in UNL. A Deconverter and Enconverter for each language form a Language Server residing in the Internet. All language servers will be connected in the UNL network. They will allow any Internet user to deconvert an UNL document found on the web into his/her native language, as well as to produce UNL representations of the texts he/she wishes to make available to the whole Internet community [7].

UNL represents sentences using three elements [8]:

- Universal Words (UWs): Each UW relates to a concept and is represented as an English word that can be optionally supplied with semantic specifications to restrict its meaning. The following are examples of UWs: book, book(icl>publication), book(icl>reserve). In the two last examples, the meaning of book is restricted by other UWs ("publication" and "reserve"). The restrictions allow representing UWs as disambiguated English words.
- Relation Labels (RLs): RLs express semantic relations between UWs. There are today 44 RLs defined. The RLs are represented as a pair relation_label(UW1, UW2). For example:
  - agt (run, car): agt relation defines a thing that initiates an action. In our example "car" initiates the action "run". It means that "car runs".
  - obj (move, table): This relation defines a thing in focus that is directly affected by an event or state. In our example, it means the "table moved".
- *Atribute Labels* (ALs): ALs express additional information about UWs, such as verb tense, intention, emphasis, etc. ALs are represented as UW.@atrib$_1$.@atrib$_2$...@atrib$_n$. For example: obj(eat.@past, apple.@pl). The AL "@past" indicates past and "@pl" indicates plural.

We do not intend to describe the UNL language here in details. A full specification of UNL can be found at http://www.undl.org.

3.1 HERMETO

HERMETO is a computational environment for fully automatic syntactic and semantic natural language analysis. HERMETO was developed at the Interinstitutional Center for Computational Linguistics (NILC), in São Carlos, Brazil. It can be used to convert any natural language into the Universal Networking Language (UNL). HERMETO receives as input a

dictionary and a grammar that should be parameterized for each language, in a way very similar to the one required by the UNL Center Enconverter program. However, HERMETO brings together three special distinctive features: 1) it takes rather high-level syntactic and semantic grammars; 2) its dictionaries support attribute-value pair assignments; and 3) its user-friendly interface comprises debug, compiling and editing facilities. In this sense, HERMETO is said to provide a better environment for the automatic production of UNL expressions [9].

HERMETO takes a NL-UNL dictionary, whose entries, one per line, must be presented in the following format:

[NLE] {id} NLL "UW" (FEATURE LIST) <LG,F,P>;

NLE stands for "NL entry", which can be a word, a subword or a multiword expression, depending on the user choice. NLL stands for "NL lemma". It is an optional field that can be used to clarify the string intended as NLE. The feature list consists of a list of attribute-value pairs, separated by comma. LG stands for a two-character language flag, according to the ISO 639. F and P indicate frequency and priority and are used for analysis and generation, respectively. Finally, any entry can be glossed and exemplified after the semi-colon [9]. Examples of dictionary entries are presented in Figure 1.

```
[a.m.] {} a.m. "a.m.(icl>ante meridiem)" (pos:abr) <EN,1,1>;
[AM] {} a.m. "a.m.(icl>ante meridiem)" (pos:abr) <EN,1,1>;
[a] {} a "a" (pos:art,typ:ndf) <EN,1,1>;
[access] {} access "access" (pos:ver) <EN,1,1>;
[add] {} add "add(icl>do)" (pos:ver) <EN,1,1>;
[admin] {} administrator "administrator" (pos:nou) <EN,1,1>;
[after] {} after "after(icl>how)" (pos:adv,typ:tme) <EN,1,1>;
[after] {} after "after(icl>time)" (pos:pre) <EN,1,1>;
```

Figure 1. Examples of dictionary entries

HERMETO's grammar is a phrase-structure grammar defined by the 6-uple <N,T,P,I,W,S>, where N stands for the set of non-terminal symbols; T is the set of terminal symbols; P is the set of production rules; I is the set of interpretation rules; W is the weight (priority) of rules; and S stands for the start symbol. It is a context-sensitive grammar, written in a plain text file, to be automatically compiled by the machine. The set of terminal symbols to be used as variables should be defined in the top of the grammar file, and the mapping between this set and the dictionary attribute values should be stated at the end of the document.

The rules should follow the formalism: p –> i, where $p \in P$, and $i \in I$. P, which is the syntactic component, can be expanded as a[w] := b, where $a \in N$, $b \in N \cup T$, and $w \in W$. I, the semantic component, is expanded as a list of attributes and relations in the following format: **att_1, att_2, ..., att_n, rel_1, rel_2, ..., rel_n** where att stands for attributive rules, and rel stands for relational rules, both comprised in the UNL Specification.

The grammar also takes a given set of primitive operators (such as '[ ]', for optional; '{ }', for exclusive; '< >' for lemma; '+' for blank space; '#' for word delimiter, etc.) in order to extend the expressive power of the formalism and reduce the necessary number of rules. The '@entry' marker should be stated in every level, and the entry word is to be considered the head of each phrase. Examples of HERMETO's rules are presented in Figure 2.

```
; 2. PHRASE LEVEL
; 2.1. IMPERATIVE VERB PHRASE (IVP)

IVP[1] := VER.@entry + NOU + NOU -> nam(:02,:03),
obj(:01,:03)
IVP[2] := VER.@entry + NOU + NOU + PRE + ART + NOU
+ NOU -> nam(:02,:03), nam(:06,:07), obj(:01,:02),
gol(:01,:07)
IVP[2] := VER.@entry + NOU + NOU + PRE + NOU + NOU
-> nam(:02,:03), nam(:05,:06), obj(:01,:02), gol(:01,:06)

; 3. WORD LEVEL
; 3.1. VERB

VER[1] := ver.@entry

; 3.2. NOUN

NOU[1] := nou.@entry - 'es' -> :01.@pl
NOU[1] := nou.@entry - 's' -> :01.@pl
NOU[1] := nou.@entry
NOU[2] := ppn.@entry
```

Figure 2. Grammar rules examples

4. Ontologies

Gruber [10] defines ontology as the specification of a conceptualization. The goal of ontologies is to organize concepts in a hierarchical fashion, keeping on its structure the relations and dependencies between concepts.

An ontology (or concept definitions) is necessary to establish a common terminology between applications. It specifies the terms each party must understand and use during communication. Creating an ontology involves explicitly defining every concept to be represented in a domain [11].

There is a good list of ontology development tools currently available – Protégé, OntoEdit, OilEd, LexiCon Explorer, and so on. Most of them are generic tools although others are domain-specific (e.g. the LexiCon). There are several formats used to represent ontologies including XML, DAML+OIL and OWL. Interfaces are graphic-based and most common functionalities are

offered by most of the tools, such as browsing, definition of synonyms, multi-lingual feature, and so on. Java is the language used by most tools and Web-related features are found in almost all of them [12].

Protégé [13] is a highly recommended tool based on many reasons: OWL-compatibility, freeware tool, it is extensible through the addition of plug-ins and it has a good base of developers supporting it around the world [12]. Based on these advantages, Protégé was chosen do edit our ontology. OWL (Web Ontology Language) [14] was chosen as the format to represent our ontology because it has been consolidating as the Semantic Web ontology language and its acceptance has grown in the scientific community.

5. The Proposed System

As stated in section 1, the SeMaComp system executes user requisitions described in restricted natural language (UNL). Figure 3 shows our approach.

Currently, the input requisitions must be imperative sentences (the system will be extended in the future). The input requisitions also have to obey grammar rules and words defined in a dictionary, except for proper names. Both, grammar and dictionary have to be defined according to the intended application domain. HERMETO will use them to convert natural language sentences into UNL.

The UNL sentence is the input for the SeMaComp (Semantic Mapping between UNL relations and Components) module. The SeMaComp goal is to identify what components, methods and arguments will be required to execute the UNL requisition. To achieve its goal, the SeMaComp module uses the Ontology of Components and the Mapping Structure (described at 5.1 and 5.2 respectively). The application domain software components have to be already installed and ready to use.

5.1 The Ontology of Components

Figure 4 presents the Ontology of Components. It shows the ontology classes end relationships between them. As stated in section 4, the ontology was developed using Protégé and it is represented in OWL.

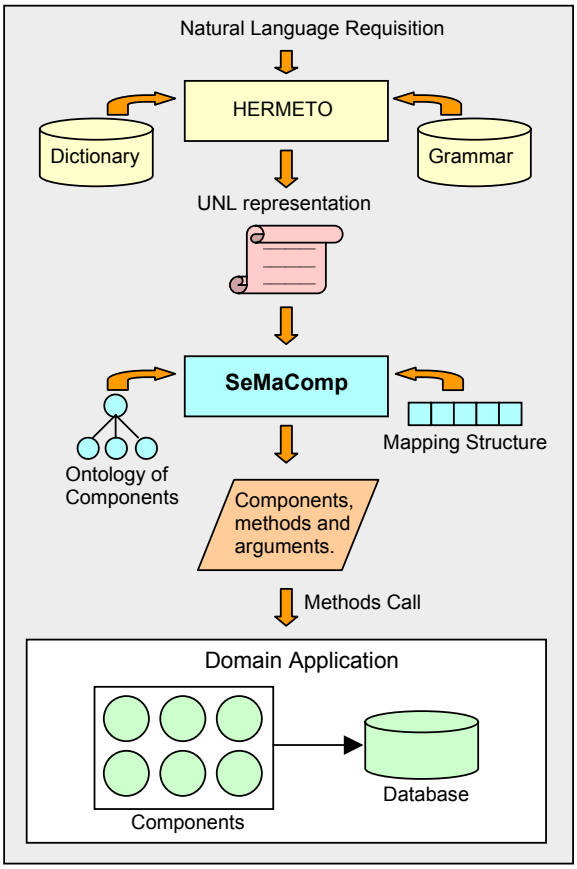

Figure 3. The Proposed System

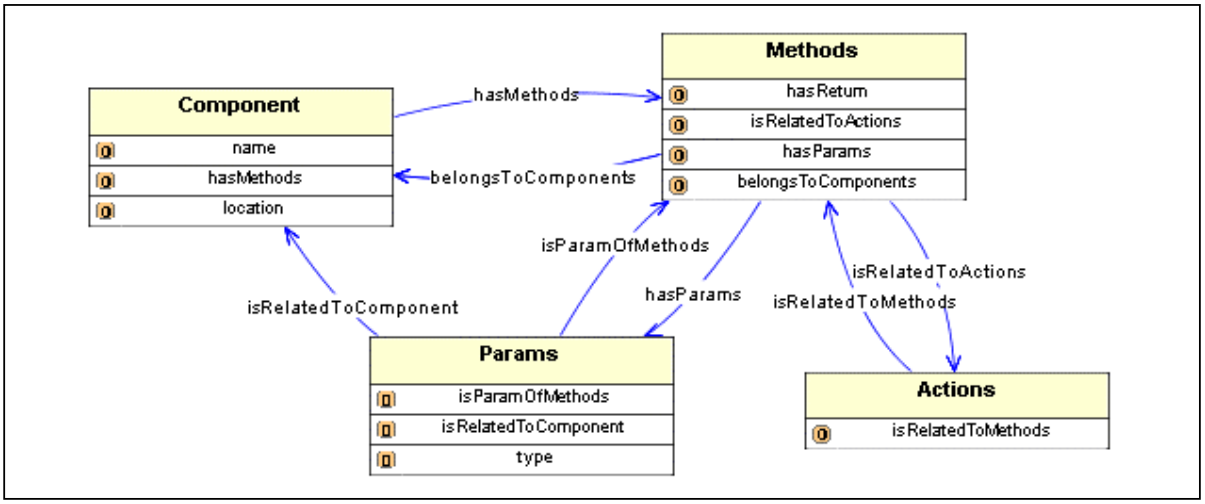

Figure 4. Ontology of Components

The Ontology of Components has to be instantiated in accordance with the application domain software components characteristics.

The instances of class **Component** correspond to concepts of the application domain. For example, considering the educational course management domain, "Student", "Teacher" and "Course" could be instances of the class Component. Each instance can have one or more software components related to it. For example, instance Teacher could be associated to Component "TeacherComponent", which would be responsible for the execution of actions related to Teacher.

The instances of class **Method** correspond to the methods of each software component of the application domain. The instances of class **Params** correspond to the arguments of each method. Finally, the instances of class **Actions** correspond to actions (imperatives verbs) related to the application domain. Each action is related to one or more methods, and each method is related to one action. For example, if we have the action "Create", then possible methods related to it could be "createCourse", "addSudent", and so on.

5.2 The Mapping Structure

As stated in section 3, UNL relations (Relation Labels) express semantic relations between UWs and are represented as a pair relation_label(UW1, UW2).

The Mapping Structure relates UNL relations to information about components. It is a data structure that should contain all the UNL relations currently being used in the imperative sentences of the application domain (and hence the relations should also be present at HERMETO's grammar). The goal of the Mapping Structure is to indicate the mapping between a particular relations_label(UW1, UW2) and the components, methods, arguments and actions in the Ontology of Components. The Mapping Structure should state if a UW, related to a particular relation label, corresponds to a concept in the application domain (an instance of class Component at the Ontology of Components), an action, an argument type, an argument value or a return value. Figure 5 shows an example of the kind of information that should be present at the Mapping Structure.

As shown in Figure 5, the Mapping Structure indicates that UW1 is going to be related to an action that should be in the Ontology of Components. UW2 is going to be related to a concept (i.e. an instance of the class Component at the Ontology of Components – that should have one or more software components related to it).

The Mapping Structure depends on the application domain. Before defining the Mapping Structure it is necessary to observe what semantic information can be extracted from the UNL relations that are relevant to UWs-Components mapping.

```
Relation_label1 (UW1, UW2)
UW1 = action
UW2 = concept

Relation_label2 (UW3, UW4)
UW3 = concept
UW4 = argument

Relation_label3 (UW5, UW6)
UW5 = argument_type
UW6 = argument_value
...
```

Figure 5. Mapping Structure

6. Web Course Example

We can demonstrate our approach with a scenario that involves a web course management domain. Figure 6 shows some examples of natural language imperative sentences (requisitions) that can serve as input to the system.

- Create Operating System course.
- Delete Java course.
- Add student John Smith to the xxx03 class.
- List classes of Susan teacher.
- Delete administrator Mary from the Java course.
- Update Java course candidate name from Mary Smith to Maria Smith.

Figure 6 - Imperative Sentences Examples

The requisitions must obey the grammar rules and dictionary created for the domain. Figures 1 and 2 show a small part of the dictionary and rules created for this domain. The requisition will be converted into UNL using HERMETO (see Figure 3). For example, consider the following requisition:

"Delete administrator Mary from the Java course." (c)

HERMETO will generate the UNL representation showed in Figure 7 for the requisition on (c).

```
obj(delete,administrator)
gol(delete,course)
nam(administrator,Mary)
nam(course,Java)
```

Figure 7. UNL representation generated by HERMETO for the sentence on (c)

Table 2. UNL relations and their meaning for the educational course management domain

UNL relation and Domain Example	Generic Meaning	Domain Meaning
obj (UW1, UW2) Example: obj (create, course)	Defines a thing in focus that is directly affected by an event or state. Typically, UW1 is an action (verb) and UW2 is the affected object.	The example means that a "course will be created". UW1 is always an action and should be an instance of the class Action in the Ontology of Components. UW1 provides a clue on which method should be called. UW2 is always an instance of class Component (at the Ontology of Components). If this instance has one (ore more) component associated to it, this component will probably be chosen to execute the requisition.
nam (UW1, UW2) Example: nam (course, Java)	Defines a name of a thing. UW2 is a name of UW1.	In the example, "Java" is the name of "course". UW1 will always be an instance of class Component and UW2 will always be the name of the instance. It can be asserted that UW2 is an argument value and UW1 is the argument type. In the given example, "Java" is the argument value and its type is "course".
mod (UW1, UW2) Example: mod (course, name)	Defines a thing that restricts or modifies a focused thing. UW2 indicates specifically what is going to be modified on UW1.	The example means that the "name" of the "course" will be modified. UW1 is always an instance of class Component. Hence, it can indicate a possible component to be called. UW2 relates to what is going to be modified at UW1. We can conclude that UW2 is an argument value that should be passed as a parameter to a method that should provide the appropriated alteration at the component represented by the concept at UW1.
gol (UW1, UW2) Example: gol (add, course)	Defines a final state of object or a thing finally associated with the object of an event. UW1 relates to an event (typically a verb), and UW2 relates to a final state.	In the example, "add" is an event and "course" is the final state. It means that something is going to be "added" at "course". UW1 is an action (at the Ontology of Components) and UW2 indicates the object whose final state will be altered by the action. We can conclude that UW2 is a feasible component to be called.

The UNL relations in Figure 7 will serve as input to the SeMaComp module (see Figure 3). It will use the Ontology of Components and the Mapping Structure to detect which components methods and arguments should be used to execute the requisition. The Ontology of Components should have been previously instantiated according to the domain software components characteristics.

As stated in section 5.2, before defining the Mapping Structure it is necessary to observe what semantic information can be extracted from the UNL relations, present on the application domain, that are relevant to the UWs-Components mapping. Table 2 shows this relations, describes each relation meaning, shows an example of each relation applied to the underling domain and describes their meaning according to the domain. It describes what semantic information can be extracted from the relations, according to the specified domain, that will help the SeMaComp module to find a suitable component, method and arguments to execute the requisition.

Using the information available in Table 2, it is easier to build the Mapping Structure for the educational course management domain, showed in Figure 8. The Mapping Structure indicates which information can be extracted from the UNL representation of the sentence to help finding the most suitable components, methods and arguments to execute the requisition.

```
obj (UW1, UW2)
UW1 = action
UW2 = concept

nam (UW3, UW4)
UW3 = argument_type
UW4 = argument_value

mod (UW5, UW6)
UW5 = concept
UW6 = argument_value

gol (UW7, UW8)
UW7 = action
UW8 = concept
```

Figure 8. Mapping Structure for the educational course management domain

The SeMaComp module was developed in Java and the Mapping Structure is currently represented as a Vector.

After defining the Mapping Structure and instantiating the Ontology of Components, the SeMaComp module is ready to receive an UNL sentence, as the one specified in Figure 7. SeMaComp will separate the tokens of the sentence and classify them using the semantic information declared in the Mapping Structure. For the UNL requisition of Figure 7,

SeMaComp will identify the following relevant information presented in Figure 9.

```
Action = delete
Main Component: administrator
Other Component: course
Argument: Mary
Argument type: administrator
Argument: Java
Argument type: course
Number of arguments: 2
Return type: none
```

Figure 9. Relevant information extracted from the UNL sentence of Figure 7

With the information presented in Figure 9, SeMaComp searches the Ontology of Components to discover which methods are related to action "delete" and belongs to one of the components associated to "administrator". This search returns the methods `deleteAdmin` and `deleteAdminCourse`. Still at the Ontology of Components, SeMaComp retrieves data about the number of arguments, arguments types and return type of each identified method. According to the information extracted from the UNL sentence (Figure 9), the method that satisfies the requisition received two arguments ("Java" and "Mary"), its arguments types are "administrator" and "course" and it has no return type. SeMaComp uses these information to compare the searched methods and to conclude which one is the most suitable to execute the requisition.

7. Conclusions and Future Work

This paper described a new approach to the execution of natural language requisitions. This approach proposes a semantic mapping between UNL relations and software components. UNL is an interlingua and for that reason several languages can be accepted to describe user requisitions.

The HERMETO system converts natural language sentences into UNL and the SeMaComp module performs the semantic mapping between UNL relations and software components. The semantic mapping can be used in different application domains; it is just necessary to build the appropriate software components set, define the dictionary and grammar rules that will serve as input to HERMETO, create instances of the Ontology of Components and define the Mapping Structure.

The Ontology of Components and the Mapping Structure, presented in this paper, still need some improvements. For example, the semantic mapping between UNL relations and software components currently performed is limited to the information given by the user at the natural language requisition. As future work, we intend to extend the Ontology of Components and Mapping Structure to support context information.

As another future work, we intend to perform the semantic mapping between UNL relations and software components using not only imperative sentence structures, but also conditional and repetition sentence structures.

Acknowledgments

We would like to thank the Interinstitutional Center for Computational Linguistics (NILC) for making the HERMETO system available to our research. We are specially grateful to Ricardo Hasegawa and Ronaldo Martins for their invaluable explanations about HERMETO capabilities and usage.

References

[1] B. A. Ballard, and A. W. Bierman. "Programming in Natural Language: NLC as a Prototype". In: ACM Annual Computer Science Conference – ACM SCS'79. *Proceedings*. ACM Press, 1979. pp. 228-237.

[2] D. Price, E. Riloff, J. Zachary, and B. Harvey. "NaturalJava: A Natural Language Interface for Programming in Java". In: 5th ACM International Conference on Intelligent User Interfaces. *Proceedings*. ACM Press, New Orleans, 2000. pp. 207-211.

[3] A. Cheyer and D. Martin. " The Open Agent Architecture". *Journal of Autonomous Agents and Multi-Agent Systems*, v.4, n.1, 2001, pp.143-148.

[4] T. M. Tsai, H. K. Yu, H. T. Shih, P. Y. Liao, R. D. Yanh, and S. T. Chou. "Ontology-Mediated Integration of Intranet Web Services". *IEEE Computer*. v. 36, n. 10, 2003. pp. 63-71.

[5] D. L. Martin, A. J. Cheyer, and D. B. Moran. "The Open Agent Architecture: A framework for building distributed software systems". *Applied Artificial Intelligence*, v. 13, n. 01-02, 1999, pp. 91-128.

[6] H. Ushida, and M. Zhu. "The Universal Networking Language beyond Machine Translation". In: International Symposium on Language and Cyberspace, 2001, Seoul (South Korea).

[7] I. Boguslavsky et al. "Creating a Universal Networking Language Module within an Advanced NLP System". In: ACM International Conference On Computational Linguisitics. *Proceedings*. Saarbrücken, Germany. July- August, 2000, pp. 83-89.

[8] UNL Center. "The Universal Networking Language (UNL) Specifications. Version 3, edition 2. July, 2003. Available at: <http://www.undl.org/unlsys/unl/UNL%20Specifications.htm>.

[9] R. T. Martins, R. Hasegawa, and M. G. V. Nunes. "HERMETO: A NL Analysis Environment". In: 2nd

Workshop da Tecnologia da Informação e da Linguagem Humana– TIL'04. *Proceedings*. Salvador, Brazil, 2004, pp. 64-71.

[10] T. R. Gruber. "A Translation Approach to Portable Ontology Specifications". *Knowledge Acquisition*, v. 5 n. 2, pp. 199-220.

[11] H. S. Nwana, and D. T. Ndumu. "A Perspective on Software Agents Research". *The Knowledge Engineering Review*, v. 14, n. 2, 1999, pp. 125-142.

[12] C. F. da Silva, R. Vankeisbelck, and C. Lima. "European eConstrution Software Implementation Toolset". In: Workshop on eConstruction. May, 2004.

[13] N. F. Noy, M. Sintek, S. Decker, M. Crubezy, R. W. Fergerson, and M. A. Musen. "Creating Semantic Web Contents with Protégé-2000". *IEEE Intelligent Systems*. v.16 n.2, 2001, pp.60-71.

[14] D. L. McGuinness, and F. van Harmelen, "Web Ontology Language Overview: W3C Recommendation", February 2004. Available at: < http://www.w3.org/TR/owl-features/>.