Integrating Natural Language Techniques in OO-Method

Isabel Díaz[1,2], Lidia Moreno[1], Inmaculada Fuentes[1], and Oscar Pastor[1]

[1] Universidad Politécnica de Valencia – Dpto. de Sistemas Informáticos y Computación, Camino de Vera s/n, 46022 Valencia, España,
[2] Universidad Central de Venezuela, Ciudad Universitaria, Edif. FACES, Piso 2, Caracas 1051, Venezuela
{idiaz, lmoreno, opastor}@dsic.upv.es, infuegil@inf.upv.es

Abstract. An approach that involves natural language analysis techniques for the treatment of software system functional requirements is described in this paper. This approach is used as the basis for a process developed to generate sequence diagrams automatically from the textual specification of use cases. This facility has been integrated in the Requirements Engineering Phase of OO-Method, an automatic production environment of software. For this purpose, a translator that is based on natural language parser is used. The translator provides grammatical information to each use case sentence and it identifies the corresponding interaction. The automatic transformation is conceived and specified following an orientation that is based on models and patterns. The results of the validation of the transformation patterns are presented.

1 Introduction

The OO-Method is an automatic production environment of object-oriented software that has been created at the Universidad Politécnica de Valencia [1]. It is supported by a tool whose industrial version was given the name OlivaNova Model Execution® (ONME). In the OO-Method, the construction of the *Conceptual Model* plays a leading role from which it is possible to generate the *Execution Model* automatically (Fig. 1). The *Conceptual Model* graphically describes the problem space from a structural, dynamic, and functional perspective, and from the point of view of the presentation. Each piece of the Conceptual Model's graphic information can be automatically transformed into an OASIS concept, an object-oriented formal specification language based on dynamic logic [2]. The OASIS specification is used to generate the Execution Model.

The construction of the Conceptual Model is supported by the models obtained during the OO-Method Requirements Engineering Phase [3]. This phase begins by defining the *Mission Statement* which describes its purpose and the main functionalities of the system. Taking into account the system's possible interactions with its environment, the *Functions Refinement Tree* (FRT) is obtained. The remaining nodes form a hierarchy of the system's functionalities at different abstraction levels. An FRT leaf node is an elementary function that can be activated directly by an actor or as a result of a temporal event. Each one of the ARF elementary functions is a use case in the *Use Case Model*. By applying an iterative strategy, this model is refined by identifying the actors that interact in these use cases and by describing them in natural language. A use case models the communication between an actor and the system for the exchange of information, as well as the actions that must be carried out internally by the system to respond to these requests for information [4].

The Use Case Model is the main input for the development of the *Sequence Diagram Model*. A sequence diagram is built by each use case scenario. The Sequence Diagram Model is used as a link between the Use Case Model (which specifies the interaction between the system and its environment) and the OO-Method Conceptual Model (whose purpose is to describe the system's internal components, relationships and restrictions).

Originally, the construction of the Sequence Diagram Model was formulated as a manual task, to be undertaken exclusively by the stakeholders. Nevertheless, traceability mechanisms have recently been established and make it possible to deduce the sequence diagrams automatically, based on the use case text. In order to do so, a linguistic approach has been used with the intention of establishing this fourth point of automatic translation in the OO-Method. This approach is supported by a framework that is based on patterns that are compliant with MDA (Model Driven Architecture) and with UML (Unified Modeling Language) [5,6].

The defined linguistic framework and its integration in the OO-Method is described in this paper. This article has seven sections. Section 1 is the introduction. Section 2 shows the phases of the translation process, its objectives, activities, inputs and outputs. Section 3 describes the transformation model based on patterns that support the translation. Section 4 explains the strategy of transformation pattern application. Section 5 describes the validation process of these patterns and the translator tool that has been developed. Sections 6 and 7 present our conclusions and references.

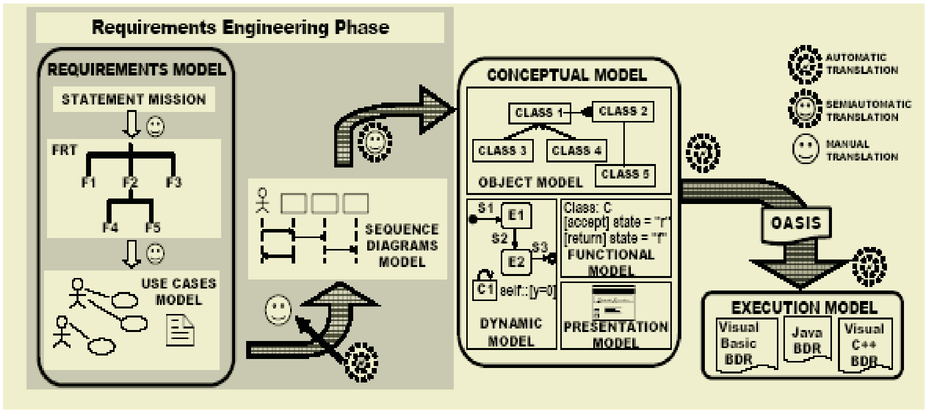

Fig. 1. The OO-Method Models

2 OO-Method Linguistic Approach

The automatic generation of the OO-Method Sequence Diagram Model is supported by linguistic information processing and control techniques [7]. This information is obtained through use case specification. It assumes that this specification is expressed as a document written in natural language that describes an elementary function of a software system [4,8]. The use case language is described by a previously defined grammar [9,10]. The translation of use cases into sequence diagrams is an iterative process developed through four sequential phases (Fig. 2). The result obtained in each phase is illustrated by an example in Figure 3 (based on the specification of a use case of a Sales Terminal System for Stores).

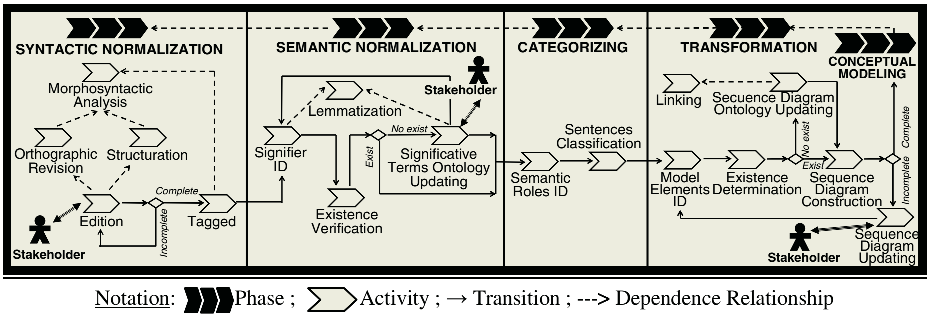

Fig. 2. Activities Diagram for the OO-Method Linguistic Approach

2.1 Syntactic Normalization

The goals of this phase are: (a) to generate a standard specification of the use case for the purpose of developing system documentation, and (b) prepare the use case text to be used to obtain information that enables the Sequence Diagram Model to be constructed automatically. In addition to improving the quality of requirements documentation, this phase acknowledges its lexical constituents in each sentence of the use case and provides them with useful morphological information so that it will be possible to identify the elements of a sequential diagram such as instances and parameters later on. As a result of this phase, a use case, which is structured according to the predefined grammar and is grammatically correct and enriched with morphosyntactic information, is obtained.

The Syntactic Normalization starts with the *editing* or transcription activity of the use case body and its descriptive information. The *spelling check* of the use case text includes the identification of words that do not exist in the language dictionaries. The *morphosyntactic analysis* enables possible morphological interpretations of each word in the use case body to be obtained and allows their respective grammatical features to be determined. It covers the disambiguation of those words that permit more than one morphological interpretation. Through *structuring*, the morphological constituents are grouped into syntactic categories of a superior level, i.e., in noun phrases. This activity makes it possible to determine whether or not the editing of the use case has respected the structure imposed by pre-established grammar and style rules. In addition to this, its function is to guarantee the grammatical agreement of the text sentences of a use case. Once the use case has been edited, each word is *tagged* with information related to its morphological features and the syntactic category to which it belongs (Fig. 3).

2.2 Semantic Normalization

The main objective of this phase is to guarantee the terminological consistency of the use case text. Upon completion, each word of the use case body will be given a sole meaning and useful information about its grammatical relationships (e.g. equivalence, antithesis, generalization and composition).

In order to study the vocabulary of a use case, the words that describe domain significant information are distinguished from those that lack it. The words that have a semantic content are called significant terms [11]. The main components, properties and restrictions of the significant terms of the use cases form an ontology of the domain. The purpose of this ontology is to define the *common vocabulary* that enables *information* about the *requirements* of the software system under development to be shared [12,13]. The *updating* of this ontology is the central activity of the Semantic Normalization phase. The *signifier identification* or symbol that represents each significant term recognized in the use case text is undertaken for this purpose. The canonical form of the signifier is obtained through *lemmatization.* In order to avoid information redundancy, it is necessary *to check* that the significant term has not been previously defined in the ontology.

2.3 Categorization

In this phase, the significant terms and the use case sentences are classified according to their role in the Use Case Model. The *identification of the semantic roles* of each significant term consists of determining the function of the elements involved in the communication modeled by the use case. Thus, "issuer" and "action" are semantic role examples. Syntactic patterns are used to identify them. Hence, semantic roles are intermediaries between syntactic patterns and abstractions of the use case, making them independent of the way they are expressed in natural language. This enables each semantic role to be related to equivalent syntactic patterns in different languages (Fig. 3).

Lastly, the Categorization *classifies each sentence* of the use case as follows: *actor-system interface* (the actor is the issuer of the communication), *system-actor interface* (the issuer of the communication is the system and the recipient is the actor), and process (the sentence describes a certain behaviour that is able to change the state of the system). The predefined grammar and semantic roles identified by the sentence are used for this classification.

2.4 Transformation

The main purpose of this phase is to build the sequence diagram that corresponds to a scenario. The first activity is the *identification of the sequence diagram elements.* It is

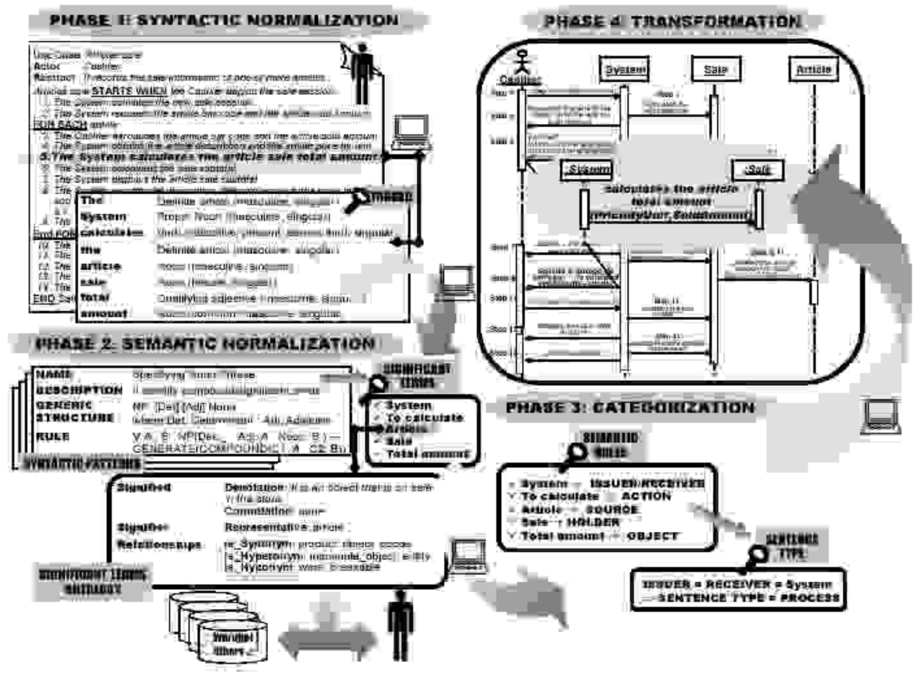

Fig. 3. An example

all a matter of recognizing instances, messages and parameters in each use case sentence. To do this, the information related to the semantic roles of the significant terms identified with the syntactic patterns is used. In order to guarantee that an element is described only once, its *pre-existence* is *determined*. Therefore, *updating* the analysis ontology involves defining a new Sequential Diagram Model element and *linking* it to the ontology of the domain significant terms. The description of the elements of the Sequence Diagram Model and their restrictions makes up an analysis ontology.

The *construction of the sequence diagram* is the graphic representation of these elements. This consists of acknowledging the interaction pattern associated with the syntactic pattern of each sentence type. An interaction pattern describes the interchange of messages between two or more objects. After the preliminary version of the sequence diagram that corresponds to a scenario has been automatically generated, the stakeholder can modify whatever he considers appropriate. The *diagram update* makes it possible to register the information concerned with these changes and verify their consistency.

3 Transformation Model Based on Patterns

The automatic transformation from the Use Case Linguistic Model to the Sequence Diagram Model has been conceived and specified following an orientation based on models. Figure 4 shows the application of the transformation model using the MDA framework (Model Driven Architecture) [6]. The target and source models that take part in the transformation are platform-independent models (PIMs). These models don't display implementation details. The transformation model assumes that the Use Case Linguistic Model has been syntactically and semantically normalized.

The use of *patterns* is decisive in the transformation [14, 15]. These patterns allow us to identify generic conceptual structures and to describe how they can be reused whenever it is necessary to provide a solution to the same type of transformation. Each pattern is specified using a basic schema of five elements: *name* (identification that distinguishes a pattern from others), *source structure or context* (informal, formal or graphical representation that describes the situation in which the transformation can be applied), *target structure or context* (informal, formal or graphical representation of the transformation) and *transformation rules* (formal specification of a target structure from a source structure or context). Furthermore, a pattern can describe specific cases, contain application examples and attach observations. The patterns have been specified using the following metalanguages: (a) *EBNF* (Extended Backus Normal Form) for the specification of lexical component sequences [16] and (b) the combination of *OCL* (Object Constraint Language) and *UML* (Unified Modeling Language) to describe the participant models in the transformation [5,17].

The types of patterns used in OO-Method are described in the following sections. The patterns were designed for the Spanish language with the intention of also considering them in other languages. They were designed following the linguistic approach to transform the use case text into sequence diagrams.

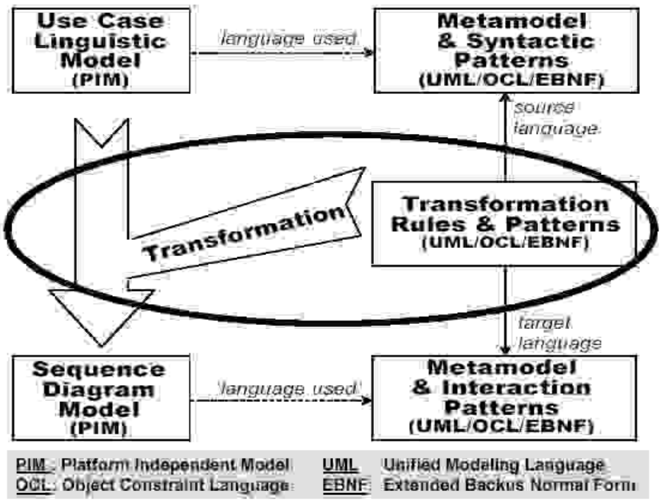
Use Case Linguistic Model (PIM)
language used
Metamodel & Syntactic Patterns (UML/OCL/EBNF)
source language
Transformation
Transformation Rules & Patterns (UML/OCL/EBNF)
target language
Sequence Diagram Model (PIM)
language used
Metamodel & Interaction Patterns (UML/OCL/EBNF)
PIM : Platform Independent Model
OCL : Object Constraint Language
UML Unified Modeling Language
EBNF Extended Backus Normal Form

Fig. 4. Linguistic Transformation Model

3.1 Syntactic Patterns

They allow the recognition of types of lexical component sequences from use case text [18]. They can be atomic or molecular. Each atomic syntactic pattern allows the deduction of a modeled element from generic lexical component sequences. The composition of two or more atomic syntactic patterns gives rise to a molecular syntactic pattern. Syntactic patterns of this type facilitate the acquisition of modeled elements and help to determine how these elements collaborate with each other. Figure 5 shows a molecular syntactic structure that is described by the "Properties Chain Pattern". This structure corresponds to a grammatical context of a particular type of use case process sentence.

3.2 Interaction Patterns

The interaction patterns specify generic types of sequence diagram fragments [5]. Figure 5 shows an interaction structure that is specified by the "Domino Effect Pattern". The structure is conformed by a border object and two or more domain objects[1]. The interaction initiates with a message that is sent by the border object to a domain object. This message induces the receiving instance to send another message to another domain instance and so on, until each instance sends a message with its respective answer.

3.3 Transformation Patterns

The transformation patterns describe how the grammatical contexts (recognized by the syntactic patterns from use case text) are turned into sequence diagram fragments (in accordance with the interaction patterns). The next section explains how the OO-Method transformation patterns are applied.

4 Applying Transformation Patterns

The transformation patterns act within the scope of each use case step. As a result of the syntactic and semantic normalization and the categorization, each use case step contains the following information: (a) an identification that indicates its position in the use case with respect to the other steps, establishing a partial order among them; (b) a part-of-speech tag according to the predefined grammar for each word in a step; and (c) the type of the step, depending on the sentence type that it contains: interface or process sentences (see Section 2.3).

This information allows the transformation pattern to recognize a certain grammatical context and to deduce the interaction structure that corresponds to it. The recognition of the grammatical context that underlies a step also implies the recognition of the transformation pattern that must be applied. Thus, according to the guidelines established in the transformation pattern, the information relative to the participants of the interaction is extracted from the grammatical context.

The transformation matches an interaction with each use case step. The interaction can be compounded by one or more messages and by one or more instances that fulfil the roles of senders and receiver of these messages. In order to deduce complementary information on the interaction, it is necessary to make a later analysis

[1] Hereafter, we will use the terms "instance" and "object" interchangeably. The definitions of "border class object" and "entity class object" correspond to the ones given in [8].

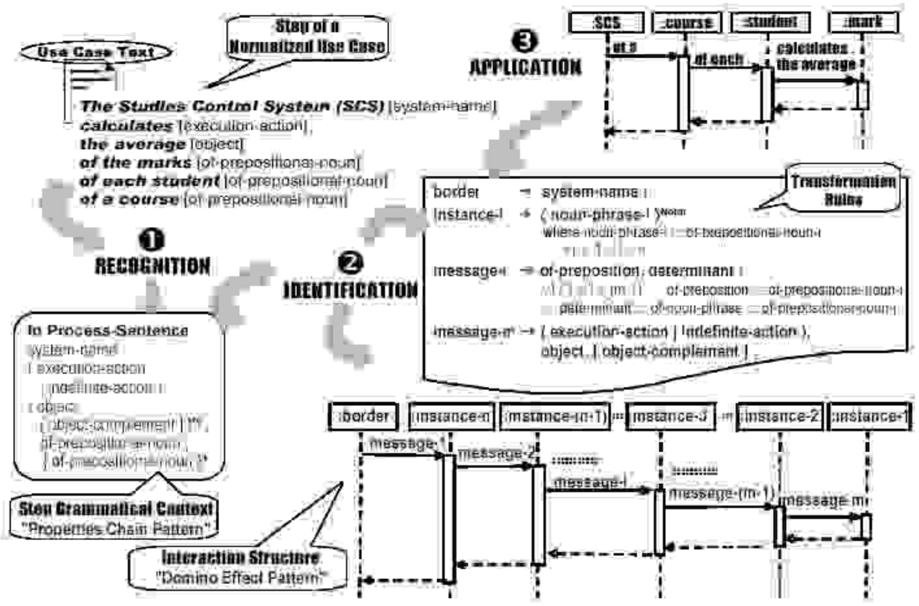
Use Case Text
Step of a Normalized Use Case
The Studies Control System (SCS) [system-name]
calculates [execution-action]
the average [object]
of the marks [of-prepositional-noun]
of each student [of-prepositional-noun]
of a course [of-prepositional-noun]
RECOGNITION
IDENTIFICATION
APPLICATION
In Process-Sentence
Step Grammatical Context
"Properties Chain Pattern"
Interaction Structure
"Domino Effect Pattern"
Transformation Rules
border → system-name
instance-i → (noun-phrase-i)
message-i → of-preposition, determinant
message-m → (execution-action | indefinite-action), object [object-complement]
SCS
course
student
mark
of each
calculates the average
border
instance-n
instance-(n-1)
instance-3
instance-2
instance-1
message-1
message-2
message-i
message-(m-1)
message-m

Fig. 5. Simple Communication Transformation Pattern from Different Sources

considering groups of two or more steps. This analysis allows us to recognize the parameters of a message or to determine if it is synchronous or asynchronous. This analysis is also done to incorporate information relative to conditionals and iterations (deduced from special sentences).

A sequence diagram is obtained by combining the interactions deduced from the use case steps. One or more sequence diagrams are matched to each use case. One of these corresponds to the basic path of the use case. There is also a sequence diagram for each alternative path. The process of application of a transformation pattern is described in Figure 5. Some details have been omitted for reasons of brevity.

4.1 Phase 1: Grammatical Context Recognition

The part-of-speech of a use case step allows us to recognize the grammatical context. A grammatical context is described by a syntactic pattern. A transformation pattern is specified from this syntactic pattern. Thus, the recognition of the grammatical structure helps determine which transformation pattern must be applied. In the example, the "Different Origin Simple Communication Transformation Pattern" was applied because its grammatical context corresponded to the part-of-speech of the step (Figure 5).

4.2 Phase 2: Participants and Interaction Type Identification

A transformation pattern always ties a grammatical context (described in a syntactic pattern) to a generic type of interaction (specified in an interaction pattern). The transformation pattern also describes how to deduce information from the part-of-speech to obtain the elements that participate in the interaction. In the example, the transformation pattern establishes that the border instance name is the system name that is being modeled (Figure 5). The names of the other instances are obtained from the noun phrases contained in each one of the "OF prepositional phrase" in the step.

4.3 Phase 3: Transformation Pattern Application

The transformation pattern uses the grammatical information that is recognized in the step, the type of interaction, and the participant elements identified. By rewriting, the pattern infers the interaction contained in this step. This way, a specific fragment of the sequence diagram of the use case is obtained. The combination of these fragments (one fragment per step) allows us to complete this diagram.

In the example, the obtained interaction contains three domain instances: "course", "student" and "mark" (Figure 5). These objects were recognized from noun phrase content in each "OF prepositional phrase". The canonical form of these noun phrases was taken into account. Therefore, the label "marks" was substituted by label "mark" by means of the word normalization function: ⟨noun-phrase-i⟩Norm. Furthermore, the name given to the border instance was the same as the name given in the system been developed (SCS: Studies Control System). The three synchronous messages that were sent and received by these instances were identified. The order of the messages was inverse to the order of "OF prepositional phrases" in the step. The second and first messages allowed the referencing of their respective receiving instances ("course" and "student"). The third message was responsible for activating the execution of the "calculates the average" operation in the "mark" instance.

5 The Experience

In principle, the transformation patterns defined were designed through the direct observation of a sample of sequence diagrams obtained from the use cases of some academic and commercial information systems. A strategy was devised to validate these patterns. The strategy permitted us to establish the limitations of the transformation patterns designed initially and then improve and enrich them. The automatic validation strategy was supported by a translator developed by way of a prototype. The following sections make a description of the validation process.

5.1 The RETO-UPV Translator

The characteristics of RETO-UPV (Requirements Engineering TOol of the Universidad Politécnica de Valencia) were taken into account in the translator implementation and design [3][19]. This tool supports all the activities of the Requirements Engineering Phase of the OO-Method (see Section 1). Figure 6 shows the translator architecture and its interaction with the RETO-UPV components.

The stakeholder must use RETO-UPV to elicit and specify the use cases. This implies defining the Statement Mission, constructing the FRT and developing the Use Case Model (see Figure 1). Every use case text is normalized and then every step is considered as the input of the translator. The first action of the RETO-UPV Translator is to obtain the tag of each word indicating its part-of-speech. To do this, we used the MS-Analyze tool which was developed at the Research Centre on Language and Speech Technologies and its Applications (TALP) at Universitat Politècnica de Catalunya (Spain) [20]. This tool is responsible for splitting use case text into tokens. Each token is tagged with its part-of-speech.

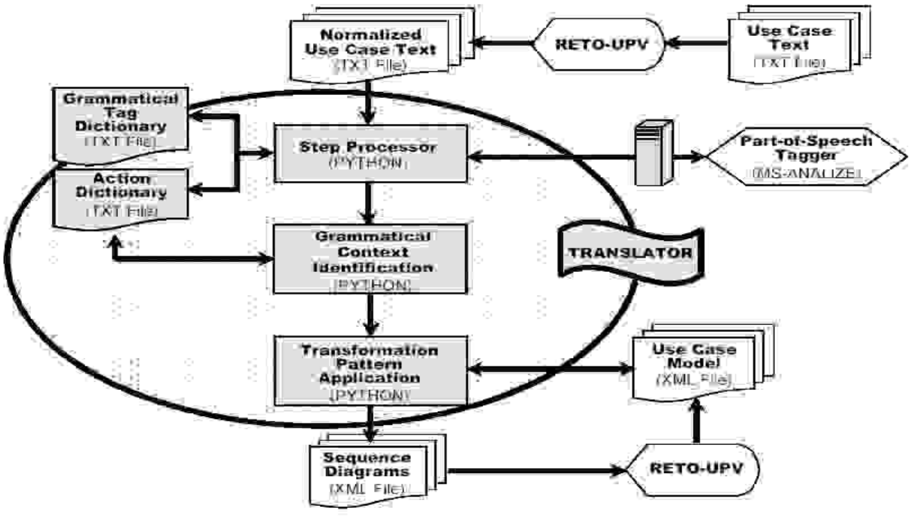
Normalized Use Case Text (TXT File)
RETO-UPV
Use Case Text (TXT File)
Grammatical Tag Dictionary (TXT File)
Action Dictionary (TXT File)
Step Processor (PYTHON)
Part-of-Speech Tagger (MS-ANALIZE)
Grammatical Context Identification (PYTHON)
TRANSLATOR
Transformation Pattern Application (PYTHON)
Use Case Model (XML File)
Sequence Diagrams (XML File)
RETO-UPV

Fig. 6. The RETO-UPV Translator Architecture

With the help of dictionaries, a set of grammar rules and the tagged step, the translator recognizes the structures that correspond to the grammar symbols. This permits the translator to identify the grammatical context of the step, including the type (interface or process sentence). The grammatical context determines the transformation pattern that the translator must apply. The translator must provide itself with additional information about the Use Case Model so that the transformation process that indicates the pattern can be made. This process generates the interaction fragment specification that corresponds to analyzed step as output. Finally, the RETO-UPV Translator combines the fragments of every use case step until the sequence diagram specification is obtained. This information is held in XML File. Then, the RETO-UPV Sequence Diagram Editor displays the graph representation of this specification.

5.2 Validation

A manual validation of the transformation patterns proposed at the beginning of this study was made. To do this, the Use Case Model of the Car Rental System (CRS) following the OO-Method guidelines was developed. After normalization of the use cases, the Sequence Diagram Model was constructed. Both models were exhaustively revised by stakeholders in order to reach a consensus on the results obtained manually. The sequence diagrams were then compared with the sequence diagrams generated using the RETO-UPV Translator in order to determine differences and similarities. Forty-one use cases were analyzed. This included a total of 574 steps of which only 14% were special (conditionals, iterations, etc.).

The interactions manually obtained were compared with the interactions generated automatically for each step of the CRS use cases. The comparison had to establish whether automatically generated interactions were the one expected by stakeholders. This implied determining if both interactions were equal, equivalent or different. We considered them equal when they were compounded by the same instances and the messages that these instances exchanged. We considered two interactions equivalents if both represented the same interaction goal even though the instances and messages weren't the same.[1] If the interactions were neither equal nor equivalent, we considered them to be different. Using these criteria, 66% of the transformation patterns, 23% were equivalent and only 11% were categorized as different.[2] This experience allowed us to establish which of the transformation patterns had to be improved or rejected. It was also possible to identify new transformation patterns of the grammatical contexts that were not considered by the designed ones initially.

6 Conclusions

In this paper, a linguistic approach for the automatic deduction of sequence diagrams from the use case textual specification has been presented. The deduction process has been defined following a software development approach that is based on the Use Case Model transformation in a Sequence Diagram Model. The transformation assumes the semantic and syntactic normalization of the use cases. This linguistic approach has been integrated into the OO-Method, a software automatic production environment. To do this, a translator was developed that was incorporated into the Requirements Engineering tool of OO-Method. The translator uses a natural language tool to provide each use case sentence with the necessary information to recognize its grammatical context. This context determines the type of transformation pattern that the translator must apply to obtain the interaction that corresponds to each sentence. The sequence diagram is obtained by the ordered combination of all the interactions of the use cases. An experiment was designed and executed that allowed us to validate the transformation patterns used. Actually, we are working on the definition of new transformation patterns and the design of an evolution strategy of sequence diagrams to guarantee the bidirectional traceability between sequence diagrams and their corresponding use cases to improve and to enrich the transformation process defined.

Acknowledgments. This work has been supported by the research projects CICYT TIC2003-07158-C04-03 and ICT EU-India (ALA/95/23/2003/077-054); it has been

[1] This decision was taken by stakeholders and who designed the transformation patterns.

[2] The interactions that did not come from grammar contexts recognized by a transformation pattern were also considered like different interactions. In these situations, the translator supposed that the interactions were formed by a single self-message on a border object.

financed also by *Consejo de Desarrollo Científico y Humanístico* of the *Universidad Central de Venezuela*.

References

1. Pastor O., Gómez J., Insfrán E., Pelechano V.: The OO-Method Approach for Information Systems Modeling: from Object-Oriented Conceptual Modeling to Automated Programming. Information Systems 26 (2001): 507-534.
2. Pastor O., Ramos I.: Oasis 2.1.1. A Class-Definition Language to Model Information Systems Using and Object-Oriented Approach. Departamento de Sistemas Informáticos y Computación. Universidad Politécnica de Valencia. España, 1995.
3. Insfrán E., Pastor O., Wieringa R.: Requirements Engineering-Based Conceptual Modeling. Requirements Engineering, 7(2), 61-72. Springer-Verlag. March 2002
4. Díaz I., Losavio F., Matteo A., Pastor O.: A Specification Pattern for Use Cases. Information & Management, Vol. 41/8 (2004). Pp. 961-975. Elsevier Science B.V.
5. Object Management Group: Unified Modeling Language Specification: Superstructure. Version 2.0. August 2003. http://www.omg.org/uml.
6. Object Management Group: MDA Guide. Version 1.01. Jun 03. http://www.omg.org/uml
7. Métais E.: Enhancing IS Management with Natural Language Processing Techniques. Data & Knowledge Engineering. 41(2002), 247-272. Elsevier Science B.V.
8. Jacobson I., Christerson M., Jonsson P., Övergaard G.: Object-Oriented Software Engineering. A Use Case Driven Approach. Addison-Wesley, 1992.
9. Díaz I., Pastor O., Moreno L., Matteo A.: Una Aproximación Lingüística de Ingeniería de Requisitos para OO-Method. Memorias del VII Workshop Iberoamericano de Ingeniería de Requisitos y Desarrollo de Ambientes de Software (IDEAS'04). Perú. Mayo, 2004.
10. Díaz I., Moreno L., Pastor O.: Traducción de Casos de Uso en Patrones de Interacción de Instancias: una Aproximación Lingüística. Memorias de las 3eras. Jornadas Iberoamericanas de Ingeniería de Software e Ingeniería del Conocimiento. Chile, 2003.
11. Rolland C., Ben-Achour C.: Guiding the Construction of Textual Use Case Specifications. Data & Knowledge Engineering 25(1998), 125-160. Elsevier Science.
12. Aussenac-Guilles N., Biébow B., Szulman S.: Revisiting Ontology Design: a Method based on Corpus Analysis. Proceedings of the 12th European Workshop on Knowledge Acquisition, Modeling and Management (EKAW 2000). Pp. 172-188. Springer-Verlag.
13. Velardi P., Fabriani P., Missikoff M.: Using Text Processing Techniques to Automatically Enrich a Domain Ontology. Proceedings of the International Conference on Formal Ontology in Information Systems (FOIS'2001). Pp. 270-284. ACM C. P.
14. Fowler M. Analysis Patterns: Reusable Object Models. Addison-Wesley, 1997.
15. Gamma E., Helm R., Johnson R., Vlissides J.: Design Patterns. Elements of Reusable Object-Oriented Software. In Professional Computing Series, Addison-Wesley, 1992.
16. International Standard ISO/IEC 14977. Extended Backus-Naur Form. 1996.
17. Object Management Group. OCL 2.0. October 2003. http://www.omg.org/uml.
18. Juristo N., Moreno A., López M. How to Use Linguistic Instruments for Object-Oriented Analysis. IEEE Software Vol. 17 Issue 3. May/June 2000. Pp. 80-89.
19. Requirements Engineering Tool (RETO-UPV). Universidad Politécnica de Valencia. DSIC. http://retoweb.europe.webmatrixhosting.net/home.aspx
20. Carreras X. Padró L. A Flexible Distributed Architecture for Natural Language Analyzers. TALP Research Center Departament de Llenguatges i Sistemes Informàtics Universitat Politècnica de Catalunya, Barcelona, España.