Use Case Paths Model Revealing Through Natural Language Requirements Analysis

Magda G. Ilieva
Dept. of Computer Science and Software Engineering
Concordia University, Montreal, Canada
magda@cse.concordia.ca

Abstract - *The approaches for extraction and representation of knowledge from textual user requirements could be summarized in two groups: i) when we have in mind an end model and seek, mine and represent knowledge for it. In short, this approach can be defined as "knowledge driven from a target model"; ii) when we have in mind knowledge in a text, and seek an appropriate model to represent it. The intermediate model obtained in this manner can be processed further, to achieve another kind of end model. We define this approach as "model driven from source knowledge". This paper presents an approach of the second kind, for analyzing textual user requirements and extraction and representation of knowledge as a graphical requirement engineering (RE) model – Use Case Paths.*

Key words: Natural Language Processing, Knowledge extraction and representation, Requirement Engineering

1 Introduction

The systems which use the methods of artificial intelligence (AI), have the advantage of distinguishing as an important phase the processing of knowledge, as well as defining, revealing, and representing it. In this way they are a step ahead in the process of formalisation. By processing the knowledge, the AI systems in fact represent in an algorithmic way the experience of the expert and the intuition it generates. These algorithms create models in the early stage of the development of the systems. Later these models can be discussed, extended and modified. Their intermediary and distinguished position between the human expert of the domain knowledge and the human expert of the implementation of knowledge shortens the time and increases the quality of the developed systems.

Existing systems: The processing of knowledge often begins with the analysis of text which is the most popular and close to the human expert way of representation. The systems, which extract knowledge automatically from Natural Language (NL) description, can be most broadly classified in the following two categories:

- The first category of systems aims a defined target model and seeks specific knowledge in the text to build it. Most often this type of systems process limited texts appropriate for concrete end models. For instance, in [14] from linguistic patterns of text are extracted graphical relations appropriate for direct translation in an Object Oriented (OO) model. A second example can be found in [13], where knowledge about 3 types of activities is extracted from use case scenario specifications, in order to build an activity diagram. Another example can be found in [12], where a finite number of syntactic structures from the controlled language are divided and grouped into a relation, which is subsequently represented as a conceptual lattice.
- The systems of the second category do not have a definite end model in mind. They extract general and specific domain knowledge from uncontrolled texts. This knowledge builds the conceptual model, logical form or conceptual graph, which leads towards a specific target model. These systems start with uncontrolled NL and then go through a variety of specific knowledge with various ways of representing it, and finally generate a diagram. One example can be found in [9] where two models are shown – one for static knowledge and one for dynamic knowledge. Another example can be found in [10, 11], where two intermediate models are derived from the general conceptual knowledge – conceptual graph for reaching activity diagram and semantic network for OO class diagram.

The difference between the two approaches is shown graphically in Fig 1. In the second type of systems, the extracted knowledge is on a higher abstract level and can serve to obtain more than one target model.

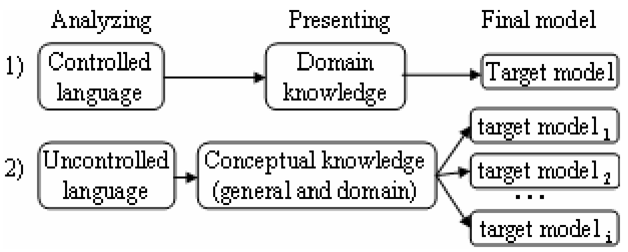


Fig. 1. Different approaches for building RE graphical models

Our approach is linked to the second class of systems. We analyze uncontrolled NL and in order to represent the knowledge we use two conceptual models of the source knowledge – table representation (TP) and semantic network (SN) [7, 8]. Through table representation we structure the text into four main groups: Subject (Su), Predicate (Pr), Object (Ob), and conjunctions. SN is obtained from the tabular representation by attaching to every concept (having the linguistic equivalent *noun*) all its attributes and activities in which it participates, whether as a Su or as an Ob. These models give us a different point of view for both linguistic knowledge and domain knowledge. The two models complement each other and in a different degree participate in the derivation different final RE models. For example, SN serves for revealing of graphical relations between the concepts and is appropriate for obtaining a domain model and OO diagram. Table representation can be considered as Knowledge Base (KB), and all models derived from textual user requirements are based on it.

Our methodology has the advantage of considering uncontrolled NL. The models which it uses for representation of linguistic and general knowledge keep the text in full (represented as entities and relations between them) in its original form. In this way the tracking and juxtaposition of *source knowledge* ➔ *target model*, is accessible and easy. From these two knowledge models (TP, SN) we have for now extracted the following final RE diagrams: domain model, OO model, activity diagram, use case paths, use cases.

This paper describes our methodology for analysis of natural language description of software requirements, which is applied for building a Use Case Paths (UCP) model. This is a dynamic model, which traces the consequent actions executed by the different participants in the work of the system. This is a model which is derived from the text, and preserves it very clearly - the text is literally read in the model. It can serve as a basis for obtaining more detailed and precise UML models. The graphic representation also has the advantage of revealing relations, which can be easily interpreted from a different point of view. For example, the path of the actions in one case can be used to obtain a data flow diagram, and in another application – an activity diagram. Section 3 describes what aims the UCP model can serve. Description of the algorithm for building a UCP diagram follows.

2 Automatic building of UCP model

We demonstrate the algorithm of building an UCP model in a case study. This example is taken from a student's "analyzing and modeling" assignment. It is not previously processed in order to fit into a given linguistic theory, and thus it is interesting for us. The full and unchanged text of the example can be seen in table 1, which is the result of the first phase of processing. Algorithm consists of 5 steps.

Step 1. Algorithm for table representation of the text

- Define the grammar category of the words in the sentence;
- Mark the verbs in the sentence;
- Mark connections between sub sentences (conjunctions, relative pronouns);
- Use the following rules to divide one sentence into sub-sentences:
  - Each verb forms a distinct proposition, and has to be on a separate row of the table. Compound verbs and modal verbs are not separate categories.
  - The beginning of a new sub-sentence is marked by: i) an infinitive verb; ii) present participle of the verb; iii) relative pronouns; iv) a conjunction if there is a verb in the phrase between it and the following conjunction
  - If there is a doubt about the point of separation between two verbs we take into account: the prepositions and their links to the nearest words; the sentences can be expressed through Su, Pr, Ob.
- For each sub-sentence: put verb in the column Pr; put the phrase from the left side of the verb in the column Su, the phrase from the right side of the verb in the column Ob;
- Repeat until reach the last sentence.

In order to arrange the table we use Memory-Based Shallow Parser [4]: tagger output, which gives us the grammatical category of the words, and chunker output, which links the related parts (for example the parts of a compound verb) into one. For instance, after passing through the parser, the second sentence in our case study would look like the following:

```
When/WRB [a/DT customer/NN] arrives/VBZ {at/IN the/DT checkout//NN} {with/IN her/his//his grocery/NN items/NNS} {in/IN the/DT cart/NN} ,/, [the/DT checkout//NN operator/NN] pases//VBZ [each/DT item/NN] {over/IN the/DT bar/NN code/NN reader/NN} [to/TO get/VB] [the/DT bar/NN code/NN] [so/IN that/IN] [its/PRP$ price/NN] [can/MD be/VB retrieved/VBN] {from/IN the/DT price/NN list/NN} stored/VBN {in/IN a/DT computer/NN file/NN} ./.
```

The meaning of tags can be seen in [15]. The brackets surround connected words in a noun phrase or verb phrase, while the braces surround the prepositional phrases.

This is not the only way, however. All accessible linguistic tools can be used to facilitate and improve the work of the algorithm. In [2] can be found different tools, which could be tried online, and the ones most appropriate for the concrete application can thus be chosen. For instance, a Link Parser [3] would assign a constituent representation of the sentence, labelling noun phrases, verb phrases, clauses, subordinate clauses, etc. If we use a Link Parser representation formula for the sentence, its division into sub-sentence would become easy, as each sub-sentence would begin with a tag "S". In the last sub-sentence of the second sentence in our example however, Link Parser notes "stored" as a verb, but does not separate it into a sub-sentence. This is why we prefer to divide the sentence into sub-sentences according to the rules described above.

Table 1. Outcome of step 1. Structure representation of user's textual requirements

S	Conj. (pre)	Subject	Predicate	Object	Conj. (post)
1		Customers	go	to a supermarket for their weekly grocery shopping	.
2	When	a customer	arrives	at the checkout with her/his grocery items in the cart	,
		the checkout operator	passes	each item over the bar code reader	
			to get	the bar code	so that
		its price	can be retrieved	from the price list	
			stored	in a computer file	.
3	Then	the operator	informs	the customer of the total cost	.
4	then	The customer	can pay	in one of two ways: cash or debit card	.
5	If	payment	is	by cash	,
		a receipt	is generated		and
			given	to the customer	.
6	If	Payment	is	by debit card	,
		the customer	is asked		
			to enter	the account type and PIN number	
			to initiate	the fund transfer from his/her bank account	.
7	If	the bank	declines	the debit cart payment request	,
		the customer	is informed		that
		the payment	is refused		;
	otherwise,	the transfer	is accepted		and
		a receipt	is generated		and
			given	to the customer	.

Step 2. Algorithm for transforming the passive voice into active. We need this processing for obtaining all positions of a single action in a straight row: Subject, Predicate, Object. A description of the algorithm which we developed after researching numerous examples of compound sentences follows.

- Fill in the empty position of the subject (Su) in the passive sentences in the tabular representation of the text:
  - the conjunction 'and' between two passive sub-sentences introduces repetitiveness which leads us to assume that the Su from the first sentence is repeated in the second;
  - passive voice sub-sentence (be+ past participle: *is asked, is informed*) takes the Su from the previous sub-sentence;
  - the past participle (*given, stored*) is attached to the nearest neighbour that it supports: this is Ob; if there is no Ob, the Su is taken from the previous sentence.
- Transform all passive sentences into active:
  - concept from the Su position becomes direct object;
  - indirect object (in Ob column) preserves its position;
  - the verb transforms from passive into active voice.
- Fill in all empty positions of Su:
  - With an agent, if introduced in the previous sub-sentence. We provide possible agents in the glossary;
  - With 'SYSTEM' by default.

Applying the algorithm from step 2 we obtain the result in table 2. On every step of the algorithm we visualize the result and give the operator a possibility for proofreading and correction.

Step 3. We define the working nodes in the system. The working nodes in the system are identified with the actors - these are not the places where the actions are executed. The actors are abstraction of the components of the system and they are identical with subjects in the sentences. We consider Su of sentences which have verbs other than the verb "to be". The verb "to be" ties the subject to an attribute, and in the case of the UCP model, which represents a consequence of actions, attributive sentences are not considered. An exception are the IF sentences, because the verb "to be" defines a condition for two possible outcomes along the path. We apply the following algorithm for passing through the table2, for obtaining all Su (working nodes).

```
Extract all different Su from table and put them into list
LoA=List_of_Actors ➔ empty in the beginning;
Do until last row of Table2
  Read row;   take Su and Pr;
     If ( (Pr  not 'to be')  and   (Su not member of LoA) )
     Then  add  Su into LoA;
  Next row;
End do;
LoA=(Customer, System, Operator, Bank)
```

Table 2. Outcome of step 2. Active sentences with subject

S	**Conj.** (pre)	**Subject**	**Predicate**	**Object**	**Conj.** (post)
1		Customers	go	to a supermarket for their weekly grocery shopping	**.**
2	When	a customer	arrives	at the checkout with her/his grocery items in the cart	**,**
3		the checkout operator	passes	each item over the bar code reader	
4		OPERATOR	to get	the bar code	so that
5		SYSTEM	can retrieve	its price from the price list	
6		SYSTEM	store	price list in a computer file	**.**
7	Then	the operator	informs	the customer of the total cost	**.**
8	then	The customer	can pay	in one of two ways: cash or debit card	**.**
9	If	payment	is	by cash	,
10		SYSTEM	generate	a receipt	and
11		SYSTEM	give	receipt to the customer	**.**
12	If	payment	is	by debit card	**,**
13		SYSTEM	ask	the customer	
14		CUSTOMER	to enter	the account type and PIN number	
15		CUSTOMER	to initiate	the fund transfer from his/her bank account	**.**
16	If	the bank	declines	the debit cart payment request	**,**
17		SYSTEM	inform	the customer	that
18		SYSTEM	refuse	the payment	**;**
19	otherwise,	SYSTEM	accepte	the transfer	and
20		SYSTEM	generate	a receipt	and
21		SYSTEM	give	receipt to the customer	**.**

Step 4. We inscribe the actions of the actors in their working nodes in the order they appear in the text. Adverbs for time can change the order of actions, but for now they are not the object of our research. The working nodes are stable and must not confuse us in the following cases: verbs, which represent directed actions (go, arrive, send, …); the action which is executed by the actor is inscribed in its own working node, regardless of which point of the system it is executed (in the printer, in the post office, etc). In the following example, "a customer arrives at the checkout", the action "*arrives at the checkout*" will be inscribed in the working node of "*customer*".

We apply an algorithm for passing through the table2 for obtaining all actions (Pr) for each Su. The rules we follow are: i) The actions have the number of the sentence they are in; ii) Identical sentences (with matching corresponding Su, Pr, Ob), receive identical numbers and are inscribed only ones. In our example these are 10 with 20, and 11 with 21.; iii) If after IF there is an attributive relations *(is a)*, we inscribe it as an action (e.g. 9, 12).

As a result we obtain the following list of structures:

L=[**Customer**(go-1, arrive-2, pay-8, enter-14, initiate-15), **System**(canRetrieve-5, store-6, isByCach-9, generate-10, give-11, isByDebitCard-12, ask-13, inform-17, refuse-18, accept-19), **Operator**(passes-3, get-4, inform-7), **Bank** (decline-16, notDecline-19)].

Conditional actions (IF-THEN-ELSE) are represented with a diamond situated in the corresponding working node of the subjects, and we attach two branches to it: i) the action after IF followed by action after THEN; ii) a negation of the action after IF followed by action after ELSE. For each sub-sentence after IF we look for a corresponding else IF by matching Su, Pr, Ob. IF we find such, we connect the "IF actions" of both sentences with a diamond. For example, sub-sentences 9 and 12: *IF payment is by cash* ⇔ *IF payment is by debit card*, we present as shown in figure 3, second row. If we don't find such a correlation, we present the IF-THEN sentences (e.g. 2nd sentence) as a consequence of actions.

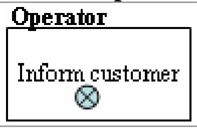
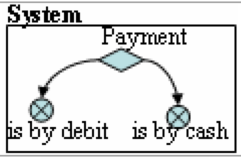
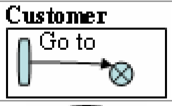
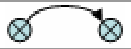
Step 5. We design graph elements and make paths between consecutive action, by applying the algorithm on list L from step 4. We use graph elements shown in table 3.

```
Design elements
Do until end of the members in L
  Take member from L;
  Design a working node as rectangular;
  Label the working node with the head of the member;
  Design in the rectangular as many "junctions"
            as there are activities;
  Label it with the name of actions;
  Next member;
End do
```

```
Connect elements
i = 1;
Do until end of
    activities;
  Connect activity i
    with activity i+1;
  Next i;
End do
```

The back-end actions, which clarify or sustain the work of the system, should not be included in the use case paths. This is illustrated in the last sub-sentence of the second sentence: "*System stores price list in the computer file*". Through clear presentation and visualisation of the knowledge, we allow the operator to define which actions are back-end, and they could be "disconnected" from the path or working node.

Table 3. Use Case Paths Elements

Notation	Example
Working knot *Operator* with action *inform customer*	
Conditional action	
Trigger	
Connection path	

By applying the methodology for building of UCP, step1 – step5, we obtain the diagram in Fig. 3.

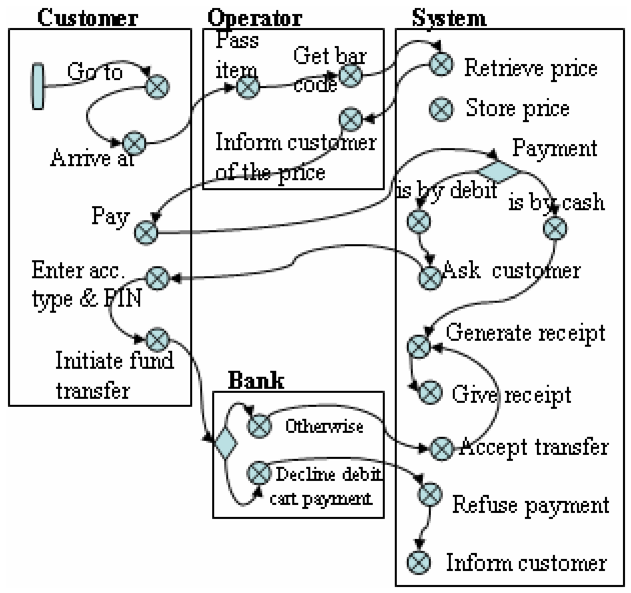


Fig. 3. Use Case Paths model

Difference between connection path and message path

In our Use Case Paths model we use a term "connection path", which is different from the term "message" used in the UML sequence diagrams [5]. "Message" in UML means when one actor initiates the action of another actor. The initiation is expressed through verbs, but in order to define which verb is a message, and which is action, we need knowledge and intuition. Automatic analysis lacks the intuition of the human analyst and in order to make our work simpler we consider the following logic: all actions are executed in the working nodes (identified with the actors). By connecting the actions we are in fact submitting the *result* of the actions between the working nodes. In this way we avoid the inclusion of complicated processes for distinguishing between message and action: the type of verbs – is it directed or not, towards which object is it directed, is this object an actor or not, etc.

For example, in the sentence *"Customers **go** to a supermarket for their weekly grocery shopping"* how shall we represent the verb *"go"*? As a message? Maybe not, because there is no one to send it to as in this case "*supermarket"* is not an actor. Another example is the sentence *"Customers enter the account type and PIN number"*. In this sentence there is no indirect object, and if we accept that the verb expresses a message, there is again no one to send it to. In a similar example from [1] (*The registrar inputs the name, address, and phone number of the applicant*), the verb *inputs* is represented as a message towards the system. This is a solution of the human analyst, who uses knowledge, experience and intuition. But very often such knowledge and experience are not available to a novice, for example, or the problem field is unfamiliar. In this case it is more appropriate to apply a singe algorithm for the processing of verbs. In this way the verbs are inscribed as actions attached towards the working node, and their result is submitted towards the next action to the corresponding working node.

Fig. 4 presents an example from [1], and we have compared the two solutions: of the human analyst for sequence diagram and automatic solution for UCP.

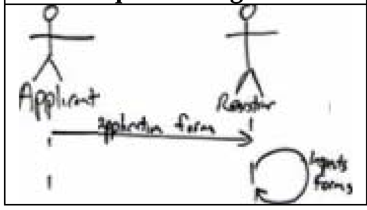
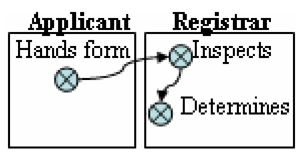
Example text: The applicant hands a filled out copy of form "*UI13 University Application Form"* to the registrar. The registrar visually inspects the forms. The registrar determines that the forms have been filled out properly.	
Sequence diagram	**Use Case Paths diagram**
 Applicant Registrar application form Inspects forms	 Applicant Registrar Hands form Inspects Determines

Fig. 4. Comparison between diagrams

3 What can be modeled using UCP

UCP diagram can be used for preliminary model, extracted directly from an unrestricted NL description of the user requirements. This model consist of two representation – table and graphical. We list some of its advantages.

Table representation can be considered as a summarized **use case scenario.** Te text is processed automatically, as each compound sentence is divided into sub-sentences, as each contains one active verb. The sentences are complete - they are represented with the triplet Subject, Predicate, Object. Each sentence is in fact a small portion of the work of the system. The text processed in this manner presents one visible and identifiable way of system behaviour associated with one or more actors. We consider the tabular representation as a **knowledge base.** The sentences are ordered into a table, which has all the advantages for a

processing which possesses a relational model for databases. The table can be expanded - it can include different columns for different knowledge extracted from the text, appropriate for different applications.
Table representation is a type of a **logical form of the knowledge extracted from text**. This form can find various applications in projects using knowledge extracted from text. Since the tabular representation of text is structured and formal, it can easily be automatically processed further. It can be represented through XML, and from this form it can be translated into various other graphic forms.
Graphical representation of knowledge is very important, because it graphically distinguishes the different elements. In this way visually, quickly and clearly we grasp the aim of the knowledge. The **UCP** graph is very close to the representation of knowledge through text. The working nodes represent the actors (Subject in the sentences). The actions (Predicates in the sentences) are inscribed in the working node which is corresponding for the actor. The objects are attached to the actions, and can be graphically expressed in the graph as labels of the actions. Since the text is inscribed in a table, which serves us as knowledge base (KB), we can skip the details in the description of the graphic elements and to consult the KB when necessary.
The UCP model can be translated into other graphic models used in the RE practice, for instance **activity diagram**. In [6] is promoted a hybrid activity diagram (HAD), which unites in a single diagram the message sequence chart and the activity diagram. By applying the relation: working node➔swimming lane; connection path➔message; action and decision, our UCP from figure 3 can be represented as a HAD shown in figure 5.

4 Benefits of using UCP diagrams

We consider the advantages of the UCP model from the standpoint of its possible applications. A list of basic application follows:
Analyzing tool: The UCP model serves us for extraction of knowledge. This model is obtained as a result of automatic analysis of text. The text is uncontrolled. Its processing includes structuring and clear division of the compound sentences into “Su, Ob, Pr” triplets. This is the basic building block of our models.
Precision of knowledge: By automatically analyzing the text we exclude the intuition (typical for the human analyst) from intervention into the analysis. The search for rules of automated analysis helps us process the analysis in depth. The automatic processing is also independent from the general and problem knowledge of the expert. In this way the quality of the user requirements can be revised in an unbiased manner.

Collaboration tool: The automatically-generated model can serve as a basis for communication of the essence of the system – functionality, completeness, consecutiveness - between the stakeholder and the implementation team. The UCP diagram keeps the text unchanged, and represents all included concepts and the relations between them. This model can be used as a control for the user description; after its correction, the text can be put through a new automatic analysis for building a new diagram. The methodology we apply for analysis of NL and representation of the extracted knowledge (tabular or semantic network) can be applied for the construction of other UML diagrams – domain model, activity diagram, use cases, OO diagram. Through the representation of the extracted knowledge from the same text, but in different manners, we achieve detailed analysis and wider view of the system.

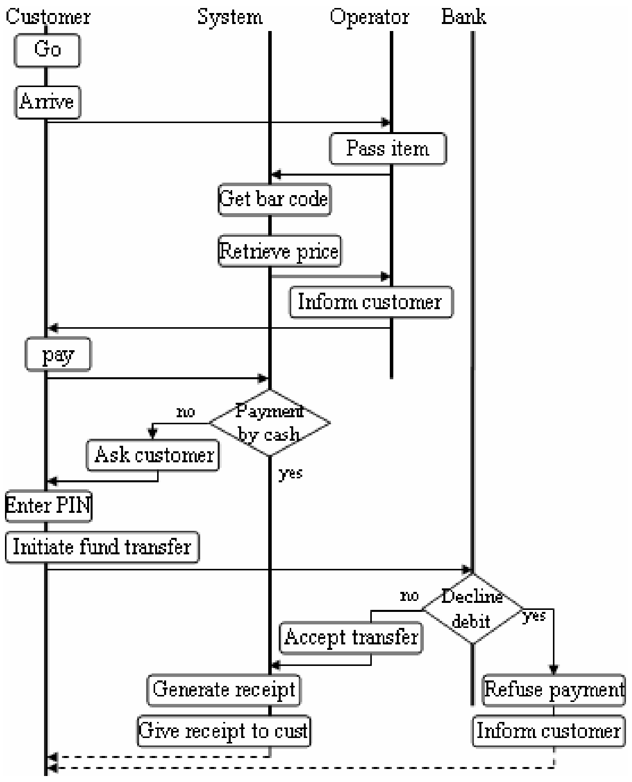


Fig. 5. Hybrid activity diagram

Teaching tool: Since the UCP diagram is derived directly from textual user requirements, it is very appropriate for teaching and training purposes. Through this model we can achieve knowledge in three aspects: a) without requiring special knowledge for problem domain, a graphic model for the actions in the system could be built. This preliminary model can serve a good role for acquiring knowledge from the developing team regarding the problem field, through a discussion with the user; b) the UCP diagram is a type of formal representation of the knowledge extracted from textual requirements. As such it fills the gap **between** the knowledge of the client, who is comfortable with the textual description of the problem domain knowledge **and** the implementation team, who prefer the formal specialized representations (with elements of implementation); c) Since the UCP model graphically represents the text, it is easier to go from such a type of representation towards a more specialized and detailed representation. Thus it can serve to teach and train modeling of software specifications.

5 Conclusion

Summary: This article described automatic construction of Use Case Paths, a RE model, which is realized through the following scheme: NL ➔ POS tagging / chunking / parsing ➔ Table representation (visualization / verification) ➔ Translation into graphical model. The model graphically represents knowledge for the actions executed in the work of a system. The knowledge is extracted automatically from an uncontrolled text, through an algorithm described in section 2. UCP model can be transformed into other RE diagrams: for example the consequence of actions in one case can serve to obtain a data flow diagram; in another application only the action of the user can be considered, and in this way we can obtain a use case diagram; or if the actions are situated in the corresponding swimming lanes of the actors and the connection paths are represented as messages – we can obtain a sequence message chart.
Further work: Extension of the UCP model can go in two directions:

• Extraction of knowledge from a text: i) the consequence of actions managed by adverb of time, preposition of time, and conjunction of time; ii) extraction of knowledge from the text for parallel operations.

• Manipulation of the graphic elements: i) reduction of the complexity of models through representation of parts of the diagram as distinct modules; ii) development of operations that merge, split, add, cut graphics; iii) developing a tool for UCP model building.

References

[1] UML 2 Sequence Diagrams: www.agilemodeling .com/artifacts/sequenceDiagram.htm

[2] Interactive online CL Demos:www.ifi.unizh.ch/ CL/InteractiveCLtools/index.php

[3] Daniel Sleator and Davy Temperley. 1993. Parsing English with a Link Grammar. Third International Workshop on Parsing Technologies.

[4] Walter Daelemans, Sabine Buchholz, Jorn Veenstra. Memory-Based Shallow Parsing. In Proceedings of CoNLL-99, Bergen, Norway, June 12, 1999.

This research is sponsored by SIRIUS Services Conseils - www.siriusconseils.com

[5] Unified Modelling Language (UML 2.0). http://www.uml.org/

[6] Ormandjieva O., Ilieva M.: Automatic Comprehension of Textual User Requirements and their Static and Dynamic Modeling. In Proceedings of Software Engineering Research and Practice 2006: 266-273

[7] Ilieva M., Ormandjieva O.: Automatic Transition of natural Language Software Requirements Specification into Formal Presentation, NLDB 2005.

[8] M. G. Ilieva, O. Ormandjieva: Models Derived from Automatically Analyzed Textual User Requirements. In Proc. of Fourth International Conference on Software Engineering Research, Management and Applications (SERA'06), 2006: pp. 13-21

[9] Burg, J.F.M. and van de Riet, R.P., "Analyzing Informal Requirements Specifications: A First Step towards Conceptual Modeling", Proceedings of the 2 nd International Workshop on Applications of Natural Language to Information Systems, Amsterdam, The Netherlands. IOS Press, 1996.

[10] Fliedl, G.; Kop, Ch.; Mayerthaler, W.; Mayr, H.C.; Winkler Ch.: The NIBA workflow: From textual requirements specifications to UML-schemata In: ICSSEA '2002 - International Conference "Software & Systems Engineering and their Applications", Paris, December 2002.

[11] Kop, Ch.; Mayr, H.C.: Mapping Functional Requirements: From Natural Language to Conceptual Schemata, In Proceeding of the 6th International Conference Software Engineering and Applications (SEA 2002), Cambridge, USA, November 4-6, 2002.

[12] D. Richards, K. Böttger, O. Aguilera: A Controlled Language to Assist Conversion of Use Case Descriptions into Concept Lattices. In 15th Australian Joint Conference on AI, 2002

[13] Mencl, V.: Deriving Behavior Specifications from Textual Use Cases. In Proceedings of Workshop on Intelligent Technologies for Software Engineering (WITSE04, Sep 21, 2004, part of ASE 2004), Lenz, Austria, ISBN 3-85403-180-7, pp. 331-341, Oesterreichische Computer Gesellschaft.

[14] Subramaniam K., Liu D., Far H. B. and Eberlein A.: UCDA: Use case driven Development Assistant Tool for Class Model Generation, In SEKE 2004, 324-329

[15] 15.http://ilk.uvt.nl/cgi-bin/tstchunk/demo.