Representing Textual Requirements as Graphical Natural Language for UML Diagram Generation

Ilieva, M.G.; Boley, Harold

National Research Council Canada
Conseil national de recherches Canada

Institute for Information Technology
Institut de technologie de l'information

NRC · CNRC

*Representing Textual Requirements As Graphical Natural Language For UML Diagram **

Ilieva, M.G., Boley, H.
July 2008

* published in the Proceedings of the 20th International Conference on Software Engineering and Knowledge Engineering (SEKE'08). Redwood City, California, USA. July 1-3, 2008. NRC 50380.

Canada

REPRESENTING TEXTUAL REQUIREMENTS AS GRAPHICAL NATURAL LANGUAGE FOR UML DIAGRAM GENERATION

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

Harold Boley
Institute for Information Technology
National Research Council Canada, Fredericton, NB
harold.boley AT nrc.gc.ca

ABSTRACT
Since the establishment of the Unified Modeling Language (UML) as a standard graphical notation for representing knowledge, new ideas have emerged about tools that can automatically extract knowledge from text and represent it with UML diagrams. As the targeted representation of knowledge is in a graphical notation, we propose to also represent Natural Language (NL) and the knowledge it carries in a common graphical form, and then translate this Graphical NL (GNL) into another graphical form (UML).

KEY WORDS
Knowledge representation, Knowledge reformulation, NLP, Semantic Networks, UML, SE modelling

1. INTRODUCTION

Knowledge can be represented in variety of forms. In Software Engineering (SE), for example, perhaps the most common way of representing knowledge is with diagrams. This way of representation fits the understanding of a wide range of users. Graphical representation is done with self-explanatory shapes, it is semi-formal, and is suitable for subsequent formal processing into program code. This type of knowledge representation is easy to understand and widespread in information technology.

Quite often knowledge is extracted from text. Texts are written in Natural Language (NL), which is the universal method for representing knowledge. As the targeted model of knowledge extracted from text employs a graphical language, UML for example [12], why not also represent the source text itself graphically? We can then match the two graph models – UML and Graphical NL (GNL) [14] – and discover analogies as well as simplify translation. This article is organized as follows. After a review of related work, we explain the main principles of Graphical NL. Then, the use of GNL is demonstrated with a study case. In the fourth part analogies are presented between the two ways of graphical representation of knowledge – GNL and UML. These analogies are used for deriving rules for the automatic translation of textual user requirements into SE graph models. We conclude with an evaluation and comparison of the proposed GNL and other graphical representations for NL and knowledge.

2. RELATED WORK

The importance of automatic translation of software User Requirements (URs) from text to SE diagrams is evident from the continuing emergence of new theories and applications in this domain. In brief, the purpose of those applications is to automate user requirements analysis and to speed up the phase of software design.

Mainly, two types of expertise have to be united in order to develop technology for translating textual URs into SE diagrams: linguistic engineering and software engineering. Often, those two types of competence are applied in order to represent or reformulate knowledge during several stages. Reformulation consists of distinguishing and restructuring the initial natural language represented knowledge of humans in order to obtain formal language represented knowledge for computers. The first phase is Language Modeling (LM), which manages linguistic objects (texts, sentences, words, etc.) and their relations. The second phase, Knowledge Modeling (KM), defines concepts and relations, which are important for problem solving. Subsequent reformulation of knowledge, Intermediate Knowledge Modeling (IKM), is needed in order to obtain a form appropriate for mapping into a final SE model. Drawing those phases together (Fig1),we can use them as a frame for reviewing existing projects.

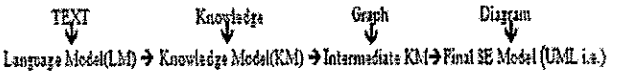


Fig1. Conceptual schema summarizing approaches

For example, in [7] LM is syntax patterns of restricted NL; KM consists of eight conceptual graphical patterns proposed for representing linguistic patterns extracted from text. Object Model (OM) and Behavior Model (BM), designed to capture the static and dynamic nature of requirements, serve as IKM (Intermediate Knowledge Model) from which the target OO diagram is derived. In a similar way, in [5], LM is restricted NL with particular syntax patterns; KM represents the types of data, operations over them and relations between them; for IKM a tree data structure is proposed, having three types of nodes: data, functionality and context. The authors in [6] consider KM as three types of graphs representing three types of knowledge for activity (emitted, absorbed and internal) extracted from restricted NL, namely, use case scenario specifications. Another example can be found in [4], where the few syntax constructs (LM) derived from the controlled language are grouped into relations (KM) that are subsequently represented as a conceptual lattice – an abstraction of a use case diagram. All of the above cited approaches obtain only one final graph model.

Other researchers focus on processing unrestricted NL. An example can be found in [1], where NL is modeled with a functional grammar, KM is presented as a Conceptual Prototyping Language, and two groups of graphs (IKM) are obtained – one for static knowledge and one for dynamic knowledge.

Another example can be found in [2, 3] where Case Grammar serves as LM while KM is presented as General Conceptual Model. Two IKMs are used for designing two target SE graph models: conceptual graphs - for obtaining activity diagrams and semantic networks - for supporting OO class diagrams.

A main point of correspondence between all theories is that they treat KM separately from LM. This separation limits the application of theories: they process specific types of knowledge applied to specific texts and receive one final graph model. Our approach differs here by offering a common graphical representation simultaneously of NL and the knowledge (both general and domain specific) included in it. After building the diagram of the text, we compare it with the diagram of the target SE graph model built by a human expert. Based on the discovered analogies between the two diagrams, we then define rules for translation of one graph into the other. This approach will make our methodology applicable to various texts, diverse knowledge and different target SE models. Our technology has fewer processing phases, which can increase its efficiency.

3. GRAPHICAL REPRESENTATION OF NL

Table structuring of an unrestricted NL: The graph representation of both language and knowledge in one unit is based on the graphical representation of relations between concepts. In order to represent text graphically we structure unrestricted NL into a table representation (TR). TR is described in [8,9], but for convenience we are going to discuss here one brief example from a case study: *"In a road traffic pricing system, drivers of authorized vehicles are charged at toll gates automatically. The gates are placed at special lanes called green lines. A driver has to install a device (a gizmo) in his/her vehicle."*

Structuring the text into a table is nothing but arranging it into three main columns – Su(bject), Pr(edicate) and Ob(ject), as also used in RDF. We obtain information for syntax structures and attached phrases when we process the text with one of the available POS taggers/chunkers [17]. Here is the outcome we got from the cited tagger:

1) In/IN ([a/DT road/NN traffic/NN]) pricing/VBG ([system/NN])./, ([drivers/NNS]) of/IN ([authorized/JJ vehicle/NN]) **<: are/VBP charged/VBN :>**at/IN ([tool/NN gates/NNS]) automatically/RB./.

2) ([The/DT gates/NNS])**<:are/VBP placed/VBN:>**at/IN ([special/JJ lanes/NNS])**<: called/VBD :>**([green/JJ lines/NNS])./.

3) ([A/DT driver/NN])**<: has/VBZ to/TO install/VB :>**([a/DT device/NN]) (/(a/FW gizmo/FW)/) in/IN his/PRP$ //CC ([her/PRP$ vehicle/NN])./.

Tags for the syntax category of words, attached phrases and sets of rules are used in order to arrange the text into TR (shown in Tab 1). Su and Ob columns are noun phrases, Pr is a verb phrase.

Looking at TR, we can outline the following advantages:

i) TR is convenient for automatic processing: a) representation in another semi-formal notation, for example XML, and then, e.g., in SVG; b) fast access for storing and retrieving information; c) unlimited, expandable space with new rows for storing extra text and new columns for storing diverse syntactic and semantic information required for automatic text processing. We thus use TR as a knowledge base supporting text analysis.

Se	subse	Preconj	Su	Pr		Ob	Postconj
				verb	adverb		
1	1					In a road traffic pricing system	,
	2		drivers of authorized vehicles	are charged	automatically	at toll gates	.
2	1		The gates	are placed		at special lanes	
	2			called		green lines	.
3	1		A driver	has to install		a device (a gizmo) in his/her vehicle	.

Tab.1. Structuring text into a table

ii) The roles of phrases in sentences and relations between them are easy to explore. At the top-level of text structures we have a sequence of predicative relations.

iii) The relations in the next structural level are clearly distinguished – in each of the three components (Su, Pr, Ob). These relations can be summarized as: prepositional, noun-noun(s) modifier, adjective-noun modifier, verb-adverb.

iv) TR can be used as verification for the correctness of the tagging..

Basic building blocks of GNL: Table structuring helps us to reveal that NL can be represented graphically as ordered triplets (**concept1 relation concept2**). In order to define such a triplet we have to define its members:

Concepts are noun phrases which can be simple (consist of one noun) or complex (main noun with modifiers – adjective(s) or noun(s)). For example, *sensor* is a simple concept and *toll gate sensor* a complex one. Complex concepts consist of more than one noun, connected with a relation (implicit *has_a*). The interpretation of *"toll gate sensor"* is: *toll **has a** gate which **has a** sensor.*

Relation can be: predicative, prepositional, is_a, has_a.

. Predicative relation is defined as two concepts connected with a verb, for example: *A driver **installs** a device*;

. Prepositional relation is defined as two concepts connected with preposition. For example, *gizmo **in** vehicle*;

. Attributive ("noun *is* adjective" or "adjective noun"). For example, '*lane **is** green*' or '*green lane*';

. Compositional relation could, in turn, fall into one of the following types:

- Noun-noun modifiers (*toll gate sensor*);
- Key-word/Enumerative structure (*types of tool gate: single, entry, exit*); (*services: deposit, withdraw, transfer, get balance*).
- Possessive (*bank's client*);

In summary, all relations can be represented with a triplet, i.e. through Su, R, Ob. In a predicative relation R is a verb; in a prepositional relation R is a preposition; in an attributive relation R is equal to *is a*; in a compositional relation R is one of the following: *has a*, colon (:), key-words (*types of, kind of, consist of, include, ...*).

Besides members, a relation has a direction.

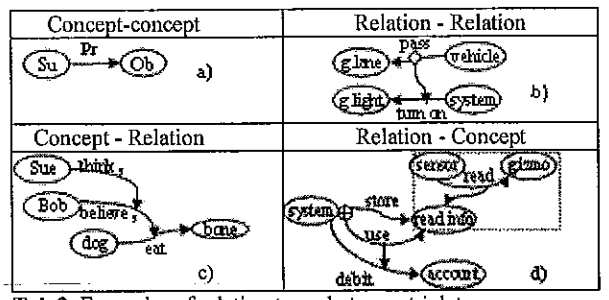
Direction signifies where the relation points. Predicative relations can have two directions: straight – from Su to Ob, which is represented through active voice, and reverse – from Ob to Su, represented through passive voice. In GNL

we represent a predicative relation through its straight direction, i.e. when we turn passive voice into active. The direction of a prepositional relation, too, does not match the order in which it is encountered in the prepositional phrase, which is from left to right, word after word. For example, "A to B" and "A from B" are two different directions. Or, the phrase "from A to B via C", does not mean to place them in order A,B,C but A,C,B. The attributive relations also have direction – for example, "green lanes" and "lanes are green" are the same relation represented with reversed directions. Opposite direction does not change the meaning of such relations, but it can change the importance of members when changing their positions. Normally, the most important member comes in the first position and becomes the head of the relation. This fact is used in some heuristics to discover empty positions in triplets (discussed in [14]).

During the process of restructuring the text into triplets some of the positions within a triplet may stay empty. Empty positions mean that their content is implicitly known from the context, or it is not important at this moment, or the sentence is not syntactically correct. For example, *the gate is placed at a special lane*, after changing the verb into active voice verb (straight direction), means that (Someone) (place) (the gate at a special lane). The position of Su (*Someone*) is left empty. It can remain empty until the analyst fills it or until we apply heuristics for discovering and filling it.

Graphical glue among triplets: In the previous section we discussed the decomposition of text into basic triplets. Their detailed graphical representation can be seen in [14]. In brief, a concept (noun) is represented as a solid oval; an attribute (adjective), as a dashed oval; a predicate, as a directed solid arc which connects related concepts; and a preposition, as a directed dashed arc which connects related concepts.

Now, we will explain how to graphically synthesize the diagram of an entire text from these basic triplets. In order to form a text representation, triplets are joined upon the relations between them. Relations are categorized upon the reason/result relationship between concepts/triplets. Tab 2 summarizes and gives examples of the different categories:



Tab 2. Examples of relation types between triplets

The graph of a simple predicative relation, i.e. the ordered triplet Su, Pr, Ob, is represented as in Tab 2a). In Tab. 2b) a complex implicative relation between two relations is shown, representing the following text: *If a vehicle passes through a green line, the system turns on a green light.* Two simple predicative relations are connected into one complex, implicative relation via a directed arc connecting the predicative arcs of the simple sentences. At the start of the connecting arc there is a small diamond, which indicates the condition of an implication. Tab. 2c) shows an example (taken from [11]) of a simple (*eats*) relation at the end of a complex epistemic (*believes*) relation, which itself is at the end of another complex epistemic (*thinks*) relation: *Sue thinks that Bob believes that the dog eats a bone.* Three different relations are aggregated with a relative pronoun (*that*), which defines the direction and connections between them: *Sue thinks* → *Bob believes* → *the dog eats a bone.* Tab. 2d) shows an example of a resultant relation (framed as a box): *Sensor reads gizmo.* ***Read info*** *is stored by the system and used to debit account.* Both sentences have to be aggregated because the concept "*read info*" in the second sentence is the result of the activity "read" from the previous sentence. In the second sentence we have three simple predicative relations (*store, use* and *debit*) which form a complex sentence. We represent them as connected relations.

This was a summary of the principles which stand at the basis of graphic representation of text. The most important part of our methodology is to restructure the text in the form of basic triplets – relations, which would be subsequently represented in a unified graphic manner. In order to structure the text as basic triples we use technologies such as POS taggers, parsers, and chunkers. We write the basic building blocks (triplets: Su, Pr, Ob) into a table representation (TR), which helps us in further automated processing: i) turning the passive voice into active; ii) defining the heuristics and algorithms for filling out the empty positions of the triplets; iii) making it easier to resolve an anaphora and ellipsis. In [8,9] we described the stages of text analysis for the tabular representation of text. The Graphical Natural Language with which the text is made into a Semantic Network (SN) is described in [14]. Different aspects and applications of these TRs and SNs are described in [15].

4. CASE STUDY ON MODEL DISCOVERY

The objective of graphical NL is to represent concepts in a compact object-centered manner, i.e. to attach to each concept all relations in which it participates. This way we obtain a structured diagram of an entire text which shows the exact place and role of each concept, group of concepts, and connection between them. Fig. 2 illustrates a graphical representation of short text taken from [13]. Let us examine part of the diagram in order to explain how to read graphical symbols. We focus our attention on 'vehicle'.

Vehicle is a concept (noun) and as such it is represented inside an oval. *Vehicle* has two attributes – *authorized* and *non-authorized* (each attribute is represented inside a dashed oval). *Vehicle* participates in two predicative relations (drawn as solid lines) and three prepositional relations (dashed lines). The predicative relation that starts from *non-authorized vehicle* is labeled *pass*. It directs activity towards *green lane* and this activity is conditional (inside a diamond). If the condition is met, the implicative arrow that goes out of the diamond leads to one complex relation consisting of two simple relations connected conjunctively

(double circle): *System turns on yellow light and camera takes a photo. Photo* has a 'pin' with a number inside, which is compressed information about the photo (listed in a legend) and explains what *photo*'s role and features are. The graph which is obtained after processing the text has a lot of similarities with UML models. In order to show these similarities, we observe a part of the graphically represented text.

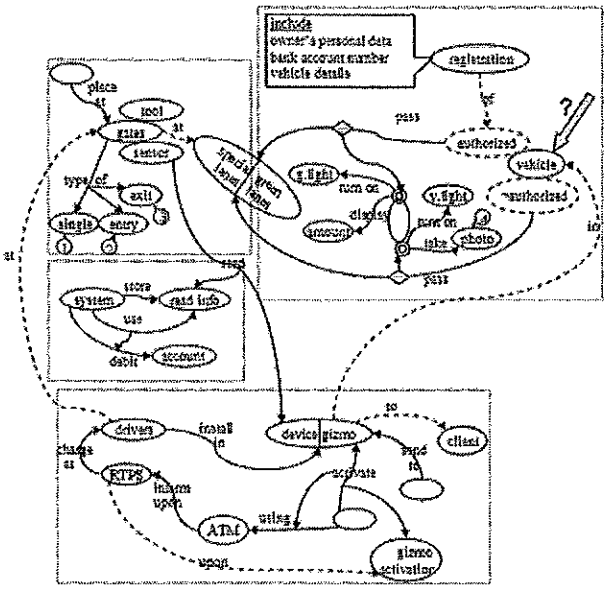


Fig.2. Graphical representation of text – Semantic Network

Domain model discovery: By a slight rearrangement of the shapes in a Semantic Network (SN) and ignoring the predicative relations, we notice that we can directly obtain a domain model (DM) from our GNL, as shown in Fig.3.

By analysis of the linguistic structure and DM structures, we come to defining the following rules for translation of SN into DM. Since DM is a static model and represents a hierarchical structuring of concepts, the following language structures are important for its generation: noun-noun attachment; adjective-noun attachment; prepositional attachment; key-word attachment. We use the term 'attachment' (rather than a phrase), to express the analogical relations that exist in NL and DM. We are interested in the static prepositions within prepositional attachments – the ones expressing place and possession. Key-word attachments are important for their representation of structural relations. For example: *consist of, involve, type of, part of, has a,* etc.

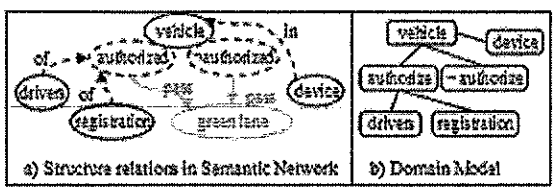


Fig.3. Mapping Semantic Network to Domain Model

Having defined which linguistic structures we have to translate into DM, we still have the knowledge engineering effort of the translation: extracting certain linguistic structures from the text, representing them in the nested format (see formula 4.1), defining operations over nested structures, simplifying, regrouping, and visualizing. The technology is described in detail in [16].

Object oriented (OO) model discovery: The concepts in the target model have properties and behavior. The first is a static characteristic while the second is dynamic. By analyzing our SN we notice that, apart from the structural relations (static), the nodes also have communicative relations (they 'send' and 'accept' predicative arcs). According to the number and type of predicative relations in which the different nodes of SN enter, they can be characterized as active and passive. The active nodes are candidates for objects in the OO model. By comparing in this way the peculiarities of the two graphic models – SN and OO diagrams, shown in Fig.4, we arrive at defining heuristics and rules for the translation of SN into OO diagrams. In general: **(i)** The domain model can serve as a structural basis for organizing the OO model. **(ii)** The nodes that are distinguished with attribute(s) / adjective(s) are candidates for parent nodes with instances. For example, instances of *vehicle* are *authorized vehicle* and *non-authorized vehicle*.

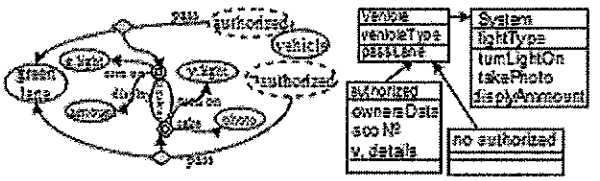


Fig.4. Mapping Semantic Network to Object Oriented Model

(iii) All predicative arcs which come out of "object nodes" are represented as methods. For example, *display* and *turn on* come from *system* and are represented as system methods; **(iv)** Terminal nodes – those that do not send predicative arcs – are regrouped as part of methods or data types. For example, *amount, photo, green light, yellow light,* are attached to the methods and represented as: displyAmount, takePhoto, turnLightOn. **(v)** Simplifying and regrouping, conceived for DM [16], can be applied to OOM. For example, two methods 'turn on *green* light' and 'turn on *yellow* light can be represented as one method with an argument thus:

turnOnLight (lightType (green, yellow)). (4.1)

We regroup the two adjectives of light into one abstract group, namely lightType; **(v)** the common methods of a node's instances are lifted to the parent node. For example, *passLane* is a method of *authorized* as well as of *non-authorized vehicle* and that's why we lift this method from instances to the parent node *Vehicle*. The same lifting technique is applicable for properties. The OO model is described in detail in [8].

Use Case Path (UCP) model discovery: Another type of model, which is important for the representation of the dynamics of a system, is derived from tracking different activities. The SN gives us a basis to arrange groups of concepts, as working nodes in which different actions are being executed. In our example from Fig.2 such structures of concepts (after their spatial arrangement guided by the prepositions for place with which they are connected) are as

shown in Fig.5: *vehicle* has *driver* and *gizmo*; *green lane* in which *toll gate* and *sensor* are placed; *RTP System*. If we write down the executable activities in the so-defined nodes, and we connect them with directed arcs in the order in which we read them in the text, we will obtain the diagram in Fig.5. In order to succeed in building this diagram, we change the point of view by considering triplets of the form 'actor-action-result'. We accept the following basic rule: the result of an activity is transferred, only if the working node is being changed, no matter if there is a recipient of the activity like it is in a UML sequence message chart. Based on this rule, no signal will go to *gizmo* after *install gizmo* or *activate gizmo*. *Driver* does not communicate with the other nodes. The type of *vehicle* is important for *System* to switch to *green or yellow light* and therefore verification of *vehicle type* is performed in the *System* node. The connection between *vehicle* and *system* is clear from the SN, while the connection between *vehicle* and *sensor* (depicted with a dashed line) has to be determined by the analyst.

The UCP model has no precise analog among the UML diagrams, but it is natural and stays close to the NL description of activities, hence can be used as an intermediary between NL and other UML diagrams (further explained in the article). The algorithm and a detailed description for processing this kind of diagram can be found in [15].

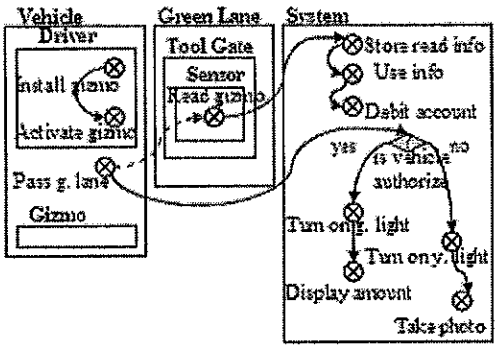


Fig.5. Use Case Path model

Hybrid Activity Diagram (HAD): This diagram can be obtained from UCP if we rearrange the working nodes as 'swimming lanes'. We inscribe the activities that are being executed in a swimming lane/working node in the same sequence in which we read them in the text. A message arrow connects swimming lanes in places where the result of the activity is being transferred. Following this logic we obtain the graph in Fig. 6a. Since our activity diagrams combine characteristics from both sequence message charts (swimming lanes and messages between them) and activity diagrams (conditional diamonds and activities), we call them Hybrid Activity Diagrams. From an **HAD** we can obtain sequence message charts by unrolling every path separately, as shown in Fig.6b).

Use Case (UC) model: In order to build this type of diagram, we are guided by the UML understanding of use cases as interactions only between the user and the system. The relations that we need from the text for this type of diagram are: i) only those in which the *user* is a Su, and the

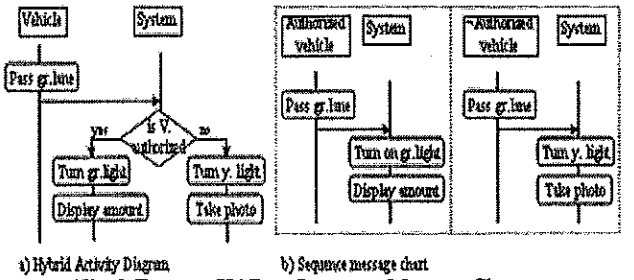


Fig.6. From an HAD to Sequence Message Charts

system is an Ob; ii) the *system* is a Su and continues an action initiated by the *user*. These types of relations, extracted from the graph of the text in Fig.2 are represented in Fig.7. The actions from case i) are connected with *user*, while those from case ii) are in the backend, and are represented as <extend> or <include>, depending on whether they are executed under specific condition, or not. For example, we observe in Fig.2., that *turn on green/yellow light* are activities placed after the diamond shape, i.e. they are conditional. In this case, activities will be included in the UC Diagram (UCD) as <extend> of the activity '*pass green lane*'. The activities of the system, with which a response is given to 'Sensor read gizmo', are not included into an UCD diagram, because there is no user participation, and thus they are not a part of the Use Case.

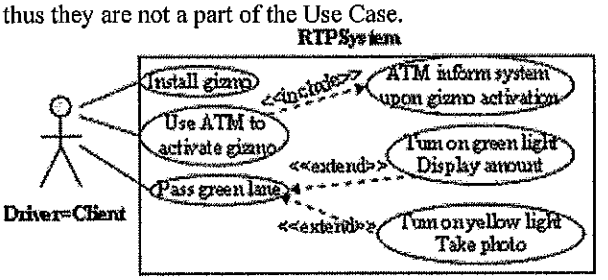


Fig.7. Use Case diagram

5. FINAL REMARKS

Summary: The idea of representing NL graphically is not new. Diverse graphical models keep appearing from both the fields of computer linguistics and SE modeling. While linguists tend to concentrate their efforts on the graphic representation of natural languages and aim to create more complete and precise models of languages, engineers are more interested in the domain knowledge, its extraction and representation. In order to automatically extract knowledge from text, we need a common model, which would represent both the text and the knowledge it contains. In order to make the model of the language (text) more universal and applicable to a wide range of problems, it has to represent both general and domain knowledge. While linguists offer models which mostly represent the general knowledge, engineers often prefer to create their own models of language, where they implicitly include specific domain knowledge. For example, the eight graphic templates proposed in [7] aim to summarize those characteristics of the NL model that are appropriate for its automatic mapping into an OO model. These templates are not likely to be appropriate for texts in which we cannot find these special language constructs, or for other target SE models. The

model in [5] is also obtained after the processing of special texts where the focus is on special linguistic templates, representing data structures and various processes applied to them.

The right balance between linguistic, general and problem domain knowledge in a single common representation has still not been discovered.

The current paper suggests one possible solution. We propose a unified model of natural language and the knowledge it carries. Working upward from the definition of natural language building blocks as relations between concepts, which are also building blocks of the knowledge represented by UML diagrams, we achieved a correlation between the two graphic representations. The graphic representation of text through Semantic Networks has served us in discovering patterns, analogical to the UML representation. This analogy helps us to reveal heuristics and rules with which the automatic generation of UML diagrams can be considered as a process of translating one graphic language into another.

Advantages of graphical formalisms: GNL was designed for SE purposes, namely, for creating executable models of knowledge described in natural language. GNL tries to capture unrestricted NL and to represent language and knowledge in one common model. That is what differentiates our methodology from other ones that separate the two models. The disadvantage of this separation is that if language patterns do not correspond to knowledge patterns, the theory loses validity.

In Fig.8 we present the same example in the two notations - Conceptual Graph formalism [10] and GNL. This brief visual comparison leads us to the following observations:

1) GNL is more compact, uses less space, and allows presenting larger volumes of information for visual inspection.

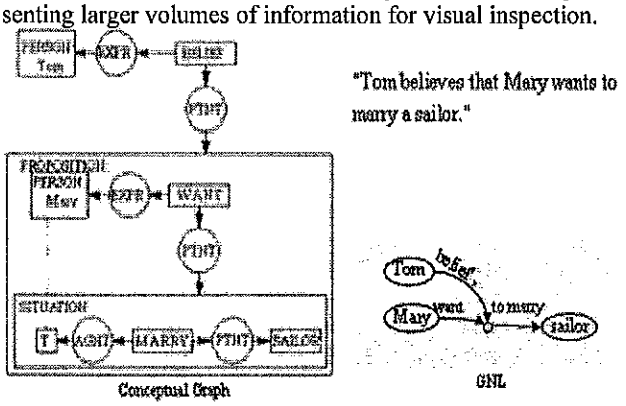


Fig.8. Two representations of the same example - comparison

2) In GNL, concepts and relations which form one simple sentence are free to participate in other relations too. This makes the concepts dynamic, and one concept can participate in many relations. **3)** As a consequence of the dynamism of the concepts in GNL, we can build a diagram of an entire text. **4)** The unambiguousness of the relations in GNL is supported by their strict indication with labels and with the use of different graphic symbols according to their semantic interpretation.

GNL is appropriate for the automatic drawing of text. An important supporting phase of its processing is the tabular representation (TR) of text. In order to construct a TR we use technologies of NL processing – POS taggers, parsers, chunkers. Then, for proceeding from TR to graphical and visual representations (e.g., SVG), it is possible to use scripting languages (e.g., PHP) and XML technologies.

Future work: We are going to develop GNL in two directions: **1)** Theoretical research which comprises the following: i) New extension to the knowledge base: examples, case studies, and comparison with examples from similar theories. ii) Add to, update and improve the collection of rules and heuristics. iii) Explore various methods and logical languages for the formal representation of SN. **2)** We will continue with the development of a software application which comprises the following projects: i) Architecture of an integrated environment for automatic analysis and formal representation of textual software requirements; ii) Structured representation of the text in a tabular format; iii) XML format of the TR; iv) Visualization.

REFERENCES

1. Burg, J.F.M. and van de Riet, R.P.: Analyzing Informal Requirements Specifications: A First Step towards Conceptual Modeling, Proc.of the 2nd Int. Workshop on Applications of Natural Language to Information Systems, Amsterdam, 1996.
2. Fliedl, G.; Kop, Ch.; Mayerthaler, W.; Mayr, H.C.; Winkler Ch.: The NIBA workflow: From textual requirements specifications to UML-schemata In: ICSSEA, Paris, 2002.
3. Kop, Ch.; Mayr, H.C.: Mapping Functional Requirements: From Natural Language to Conceptual Schemata, In Proc. of the 6th Int. Conf. SEA, Cambridge, USA, 2002.
4. 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
5. Lee, B.-S., Bryant, B.R.: Automated conversion from requirements documentation to an object-oriented formal specification language. In Proceedings of SAC(ACM), Madrid, Spain, 2002.
6. Mencl, V.: Deriving Behavior Specifications from Textual Use Cases. In Proc of 'Workshop Intelligent Technologies for Software Engineering (WITSE, part of ASE), Linz, Austria, 2004.
7. Moreno A.: Object-Oriented Analysis from Textual Specifications", In Proc. of 9th International Conference on Software Engineering and Knowledge Engineering (SEKE), 1997.
8. Ilieva M., Ormandjieva O.: Automatic Transition of Natural Language Software Requirements Specification into Formal Representation, NLDB 2005.
9. M. G. Ilieva, O. Ormandjieva: Models Derived from Automatically Analyzed Textual User Requirements. Proc. of SERA'06
10. John F. Sowa: Knowledge Representation: Logical, Philosophical, and Computational Foundations, Brooks Cole Publishing Co., Pacific Grove, CA, ©2000
11. John Sowa: SemNet http://www.jfsowa.com/pubs/semnet.htm
12. Unified Modeling Language (UML) 2.0 http://www.uml.org/
13. J. Araújo, A. Moreira, I. Brito, A. Rashid. Aspect-Oriented Requirements with UML. Workshop on "Aspect-oriented Modeling with UML", UML 2002, Dresden, Germany
14. Ilieva M.: Graphical Notation for Natural Language and Knowledge Representation. In Proc. of 19th SEKE, 2007.
15. Ilieva M.: Use Case Paths Model Revealing Through Natural Language Requirements Analysis, Proceedings of ICAI, 2007.
16. Ilieva M, Ormandjieva O.: NLP and FCA Technology for Automatic Building of DM, Proceedings of SEA, 2007
17. Infogistics' NLProcessor Interactive Demo: Tagging and Syntax Chunking http://www.infogistics.com/posdemo.htm