A Novel Software Tool for Supporting and Automating the Requirements Engineering Process With the Use of Natural Language

Marinos G. Georgiades and Andreas S. Andreou

***Abstract*—This paper presents NALASS, a novel software tool that attempts to automate a large part of the Requirements Engineering (RE) process. The tool is based on a methodology that utilizes elements of natural language syntax and semantics to formalize activities of requirements discovery, analysis and specification. NALASS automates the creation of specific question sets for the elicitation stage, the organisation and classification of requirements for the analysis stage, with the use of predefined patterns, and the generation of diagrammatic notations, use case specifications and the SRS document.**

***Index Terms*—Automated requirements engineering, natural language.**

I. Introduction

Research and practice show that the least understood parts of systems' development are the stages of requirements discovery, analysis and specification [1]. In particular, it is observed that there is a significant gap between the clients' needs and the software engineers' understanding of the clients' needs [2]. Clients often speak with vague sentences and/or cannot express their functional needs or, even worse, they do not know what these needs really are. This problem is amplified further when the analyst does not provide the right questions, as he/she essentially does not know precisely what to ask.

Our standpoint is that if you know what to write, then you know what to ask. Therefore, if the analysts know, in advance, specifically what types of functions, data and constraints (Requirements Analysis - RA) they should search for and write down, then they will be able to ask specific questions (Requirements Discovery - RD) regarding that particular information. A second priority of engineering the requirements is to formalise the way the analysts write this information (Requirements Specification - RS) - that is, to organize it, apply correct syntax, use proper diagrammatical notation, etc. Similarly, the way the RD questions are written is part of this (second) priority. Conclusively, we claim that building the questions for RD, based on RA (mainly) and RS, is a reliable way to derive the right answers/requirements from the users. Such a methodology that provides specific steps in advance and, more importantly, a formalized and understandable way to engineer requirements, is proposed by [3]-[4], contrary to other approaches that try to elicit requirements from existing documents or by using a general template such as the IEEE SRS document template [5]. The NLSSRE (Natural Language Syntax and Semantics RE) methodology utilizes natural language (NL) syntactic and semantic elements, such as subject, verbs, nouns, genitive case, adjectives, and adverbs to: (i) identify and formalize adequately the various types of data and functions of an information system (IS), as well as their relations, because language, by its nature, is the most powerful medium of expression; (ii) provide a common terminology and eliminate redundancies in specifying names of functions, data and constraints; (iii) give requirements a NL-like description which is very understandable and useful as a communication medium between users, analysts and programmers of the software system.

To reduce the time required for the manual application of the NLSSRE methodology, and also to provide a friendly graphical environment for the Information Systems (IS) analyst, a software tool is required. Therefore, we introduce NALASS (Natural Language Syntax and Semantics), a supporting software tool that automates all the stages of the NLSSRE methodology, including RD, RA and RS. For the RD stage, specific sets of questions are automatically created based on the specific predefined types of data attributes and patterns of formalized sentences that are given in advance; for the RA stage, the requirements are automatically organised and classified based on the same types of data attributes and patterns; and for the RS stage, the tool can automatically generate Data Flow Diagrams (DFDs), Class Diagrams, Use case specifications and diagrams, and the Software Requirements Specification (SRS) Document.

The rest of this paper is organized as follows: Section II outlines related work on RE tools and describes how NALASS differs from similar propositions. Section III provides a general overview of the NLSSRE methodology and its application within the tool, while Section IV presents the tool in depth. Both sections provide examples of using NALASS in a real setting. Finally, Section V provides some conclusions and recommendations for future work.

II. Related Work

Current software tools both generally speaking and in the context of Natural Language Requirements Engineering (NLRE) are mainly limited to document parsers that can be used in various activities such as traceability, verification and prioritization of requirements, or even automated extraction

Manuscript received October 28, 2012; revised December 10, 2012.

The authors are with the Department of Electrical Engineering, Computer Engineering and Informatics, Cyprus University of Technology, Limassol, Cyprus. (e-mail: marinos.georgiades@cut.ac.cy, andreas.andreou@cut.ac.cy).

of requirements from NL requirements documents. Abstfinder [2] is based on the use of pattern matching techniques to extract abstractions (stakeholders, roles, tasks, domain objects, etc.). The frequency with which the abstractions occur within the text is taken as an indication of the abstractions' relevance. [6] proposed an automatic evaluation method called Quality Analyzer of Requirements Specification (QuARS) to evaluate quality in software requirements specification. This work developed a tool that parses sentential requirements written in Natural Language (NL) to detect potential sources of errors. COLOR-X [7] and Circe [8] parse a set of structured requirements in natural language to generate specific models (ER, DFD, OO design, etc.). The common characteristic of these tools is that they are mostly used and applied on pre-existing documents with disorganized text, redundancies and ambiguities. As a result, the retrieval approach is not particularly reliable, since requirements are often not written syntactically, grammatically and semantically correctly from scratch, and the rules applied to retrieve them cannot work well to produce reliable and complete results; additionally, there is a good possibility that the original texts do not cover all the requirements of the IS under development and also include redundancies and disorganized material. Other tools, such as [9] that are not parsers and offer the user the capability to enter the requirements from scratch do not provide any specific types of questions for requirements elicitation, which are linked to the identification of data and functions of an IS. In contrast, the NALASS tool proposed in this paper implements the NLSSRE methodology and creates automatically specific sets of questions, derived from predefined requirements patterns and predefined (standardized) types of data attributes, which correspond to IS elements (functions, data, functional conditions), provided by NLSSRE. The answers to these questions feed the analysis and specification stages. Hence, the way the requirements are elicited is clearly connected to the analysis and specification of requirements. In the current literature, this link does not exist, and this is exactly why the resulting requirements documents need to be re-organized, re-validated and re-adjusted.

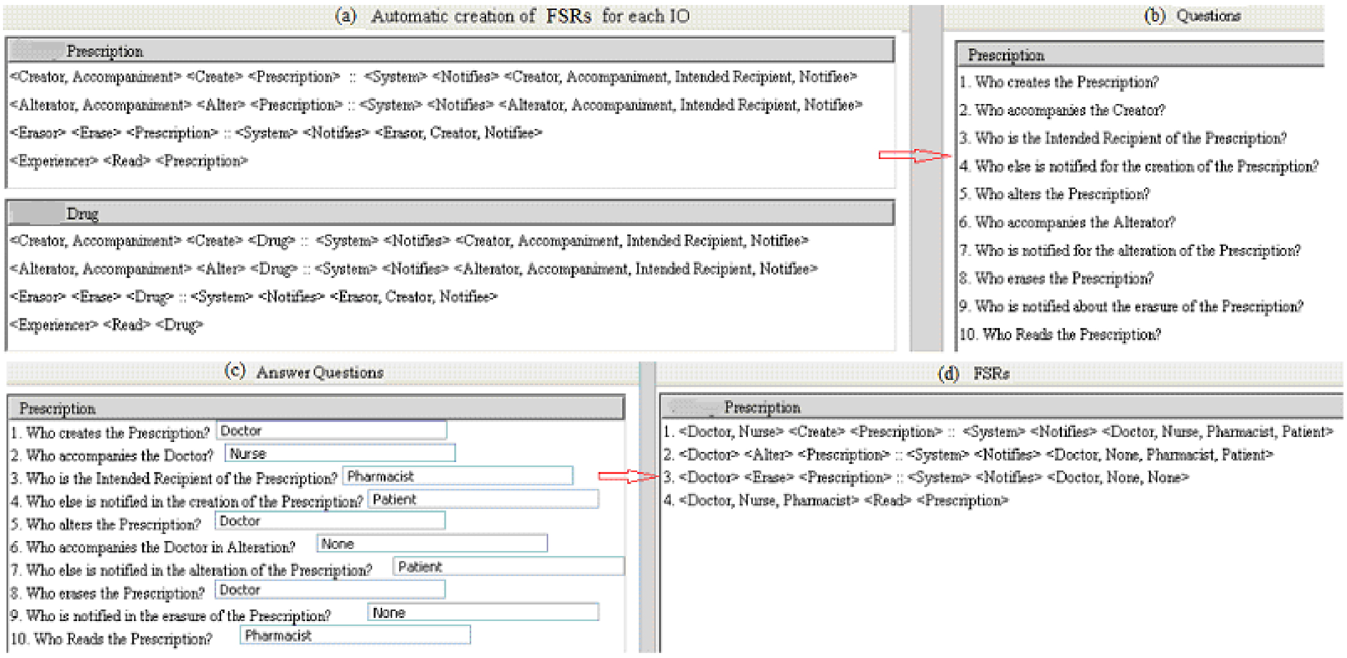
Additionally, NALASS may be conceived as a complete toolset that can generate DFDs, Class Diagrams, Use Case descriptions, scenarios and diagrams, as well as a well-structured NL SRS document that covers the essential parts of the IEEE SRS template. Regarding the latter, there is a lack of CASE tools that can produce textual descriptions of requirements and embed them automatically in a well-structured SRS document template. Tools such as Rational Rose [10] and MagicDraw [11] provide significant capabilities for drawing diagrams and generate code but not adequate facilities for the above – hence the analysts need to write their project's SRS using regular text editors and templates.

III. Methodology Overview

The NLSSRE methodology introduced by [3] and [4] provides formalization of the major activities of RE including Requirements Discovery, Analysis and Specification, so that the analyst will know in advance, through a step-by-step approach, what questions to ask, in what specific way to analyse the answers to the questions, and how to write them in a specific way. The application domain of the methodology is an IS (e.g. Hospital IS or Bookstore IS) that deals mainly with management of documents or other physical objects that can be conceived as electronic information which can be Created, Altered, Read and Erased.

The first step of the methodology is the identification of the *Information Objects (IOs)* of the system. An IO denotes a separate entity of information (attributes) that can stand on its own and can be created, altered, read and erased within the context of the IS. For example, for a Hospital IS, some of the IOs include *Prescription, Pharmacy, Patient,* and *Doctor.* As an additional, important step, for each IO, five patterns of Formalized Sentential Requirements (FSRs) are provided, as shown in Fig. 1 (a), in the example of the *Prescription* IO. Each pattern includes a function (Create, Alter, Read, Erase, Notify - all derived from relevant linguistic verbs) that also denotes the type of the FSR, functional conditions (Instrument, Amount, Time, Location – for simplicity, they do not appear in the NALASS screenshots), and functional (semantic) roles (e.g. Creator, Accompaniment) that are related to each function and are also attributes of the IO. The aforementioned FSR functions are decomposed to sub-functions and constraints (e.g. *Create* is decomposed to the *Add* and *Compare* sub-functions. *Compare* checks if the value to be assigned to an IO attribute satisfies the constraints about that attribute). Hence, the FSR facilitate the formalization of functions, business roles that replace the functional roles, data attributes, functional conditions and constraints of the IO. As another step, for the formalization of additional types of attributes of each IO, the NLSSRE methodology makes use of the genitive case, the adjective and other types of attributes. Some of these types of attributes will be illustrated in section 4 through the description of the OO component of NALASS. After the definition of FSRs and attribute types, the c*reation of questions for each IO* should take place. Questions are derived from the elements of the FSR patterns (Fig. 1 (b)) and the predefined types of attributes (not shown in figure 1(b) for simplicity). Following, the answers to these questions (Fig. 1 (c)) feed the FSR patterns (e.g. *Creator* takes the value *Doctor*), and, hence, create complete requirements (Fig. 1 (d)) in the form of formalized sentences (FSRs) that can be used, as the last step, to create diagrammatic notations such as DFDs (Fig. 4. 5. 6.), Class diagrams (Fig. 7) and Use Case diagrams (Fig. 8) and specifications (Table I), as well as the SRS document (Fig. 9. 10). The entire procedure is supported and automated by the NALASS tool, and it will be clarified further through the description of the tool in Section IV.

It has been illustrated that NLSSRE uses syntax (IS elements of a requirement are written in the correct order in a formalized sentence) and semantics (genitive case types, adjective types, etc.) of NL to formalize the IS requirements, through the stages of RD, RA and RS. Especially the use of predefined questions guides users to provide specific answers without ambiguities, vagueness and redundancies. Additionally the use of NL gives expressiveness to the formalization of requirements and makes them easily understood by the users, analysts and programmers. There is a common terminology based on a consisted and common language of writing, without ambiguities and redundancies, and, furthermore, this controlled language is computer-processed and translated automatically into diagrammatic notations, use case descriptions and the SRS document, as already mentioned.



(a) Automatic creation of FSRs for each IO

Prescription

<Creator, Accompaniment> <Create> <Prescription> :: <System> <Notifies> <Creator, Accompaniment, Intended Recipient, Notifiee>
<Alterator, Accompaniment> <Alter> <Prescription> :: <System> <Notifies> <Alterator, Accompaniment, Intended Recipient, Notifiee>
<Erasor> <Erase> <Prescription> :: <System> <Notifies> <Erasor, Creator, Notifiee>
<Experiencer> <Read> <Prescription>

Drug

<Creator, Accompaniment> <Create> <Drug> :: <System> <Notifies> <Creator, Accompaniment, Intended Recipient, Notifiee>
<Alterator, Accompaniment> <Alter> <Drug> :: <System> <Notifies> <Alterator, Accompaniment, Intended Recipient, Notifiee>
<Erasor> <Erase> <Drug> :: <System> <Notifies> <Erasor, Creator, Notifiee>
<Experiencer> <Read> <Drug>

(b) Questions

Prescription

1. Who creates the Prescription?
2. Who accompanies the Creator?
3. Who is the Intended Recipient of the Prescription?
4. Who else is notified for the creation of the Prescription?
5. Who alters the Prescription?
6. Who accompanies the Alterator?
7. Who is notified for the alteration of the Prescription?
8. Who erases the Prescription?
9. Who is notified about the erasure of the Prescription?
10. Who Reads the Prescription?

(c) Answer Questions

Prescription

1. Who creates the Prescription? Doctor
2. Who accompanies the Doctor? Nurse
3. Who is the Intended Recipient of the Prescription? Pharmacist
4. Who else is notified in the creation of the Prescription? Patient
5. Who alters the Prescription? Doctor
6. Who accompanies the Doctor in Alteration? None
7. Who else is notified in the alteration of the Prescription? Patient
8. Who erases the Prescription? Doctor
9. Who is notified in the erasure of the Prescription? None
10. Who Reads the Prescription? Pharmacist

(d) FSRs

Prescription

1. <Doctor, Nurse> <Create> <Prescription> :: <System> <Notifies> <Doctor, Nurse, Pharmacist, Patient>
2. <Doctor> <Alter> <Prescription> :: <System> <Notifies> <Doctor, None, Pharmacist, Patient>
3. <Doctor> <Erase> <Prescription> :: <System> <Notifies> <Doctor, None, None>
4. <Doctor, Nurse, Pharmacist> <Read> <Prescription>

Fig. 1. The predefined questions (b) created automatically by the FSR patterns (a), and the resulting complete FSRs (d) created automatically by the answers of the users (c), for the Prescription IO – screenshots are taken from NALASS that automates and supports the NLSSRE methodology.

IV. The NALASS Tool in Depth

NALASS is a software tool that supports the NLSSRE methodology by providing a user-friendly graphical user interface and automating the application of the methodology. It includes 7 components as depicted in Fig. 2:

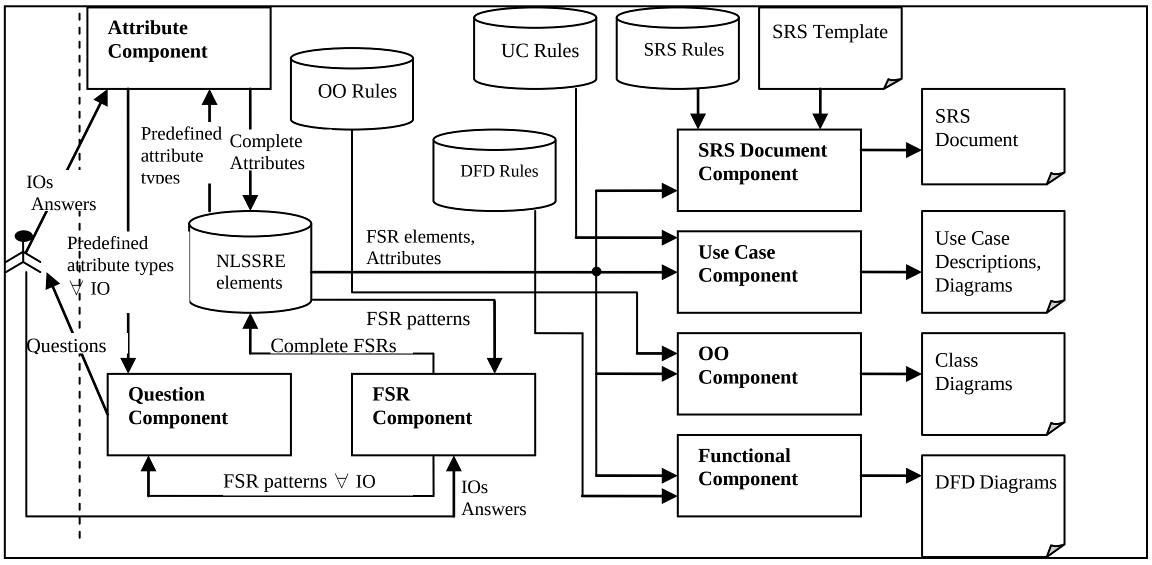
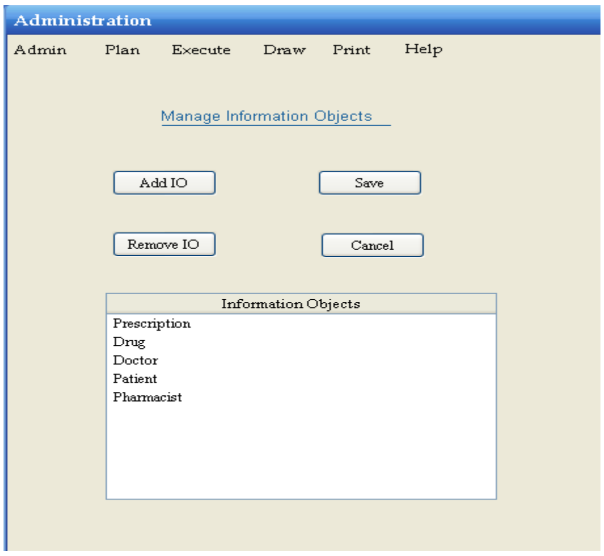


Fig. 2. Configuration of NALASS.

(i) the FSR component that uses the predefined FSR patterns and the identified IOs to automatically generate, on one hand, the FSR patterns for each IO, and, on the other hand, the complete FSRs fed with the received answers; (ii) the Attribute component that uses the predefined types of attributes and the identified IOs to automatically generate the attributes types for each IO, on one hand, and the complete attributes formed by the received answers, on the other hand; (iii) the Question component, which processes the elements of the FSR patterns and attributes for each IO, to automatically create the question sets (for each IO) to be submitted to the user; and (iv), (v), (vi) and (vii) the Documentation, Use Case, Object Oriented and Functional components that process the elements of the completed FSRs, the completed attributes and specific rules to automatically generate the SRS document, Use Case specifications and diagrams, Class diagrams, and DFDs, respectively.

Below we illustrate, with examples, the 7 components through the description of the two basic interfaces of the tool, named Plan and Execution.

Plan. The first step for the analyst, in the Plan section, is to use a particular guide to identify and add the Information Objects of the IS. The screen in Fig. 3 shows some of the IOs of a Hospital Information System.



Information Objects
Prescription
Drug
Doctor
Patient
Pharmacist

Fig. 3. Adding information objects .

Subsequently, for each IO, the five patterns of FSRs (fig. 1a – Create, Alter, Read, Erase, Notify) as well as predefined types of attributes (not shown in the figure for simplification), as a first step, and specific question sets (fig. 1b), as a second step, are created automatically by NALASS. The questions are derived from the processing of the predefined types of attributes for each IO and from the elements of each FSR pattern, since they need to be fed with a value – for example, for the *Creator* element of the *Creation* FSR for the *Prescription* IO, the question *"Who creates the prescription?"* is derived. This formalization in providing specific questions that are linked to the analysis and organisation of requirements is the difference from most, if not all, of the approaches which use formalism in NL RE. Such approaches try to develop and formalize requirements that are already written in existing documents. We consider them as being inefficient, since requirements in such documents are often poorly written and organized; sentences do not necessarily follow the correct form of syntax, while there may exist redundant words, fuzzy and complicated meanings, etc. As such, it is rather precarious and difficult to apply linguistic rules on such documents.

Execution. In the 'Execution' section:

The analyst is in the user's environment submitting questions to the users and noting down the answers (figure 1c). The answers to the questions feed the FSR patterns as they are the values of the constituent elements of the FSR patterns, as shown in figure 1(d) (e.g. Creator takes the value Doctor). The answers also feed the types of attributes.

Subsequently the FSRs and their constituent elements, as well as the IO attributes, with the use of specific rules are transformed to DFDs, Class diagrams, Use case specifications and diagrams, and the SRS document. Below we introduce this transformation through the description of the relevant components and some indicative rules:

Functional component: Within this component, the FSRs for each IO are grouped under one comprehensive function with the heading *Manage IO*. For example, for the *Prescription* and *Drug* IOs, the FSRs of *Prescription* and *Drug*, as appear in Fig. 1 (a), will be grouped under *Manage Prescription* and *Manage Drug*. The *Manage* functions for each IO are the functions of the 1st level DFD (Fig. 4), the *Create, Alter, Read, Erase* and *Notify* functions for each *Manage IO* are the functions of the 2nd level DFD (Fig. 6). For the 3rd level functions, the second level functions are decomposed to *Compare*, *Add*, and *Remove* (Fig. 5).

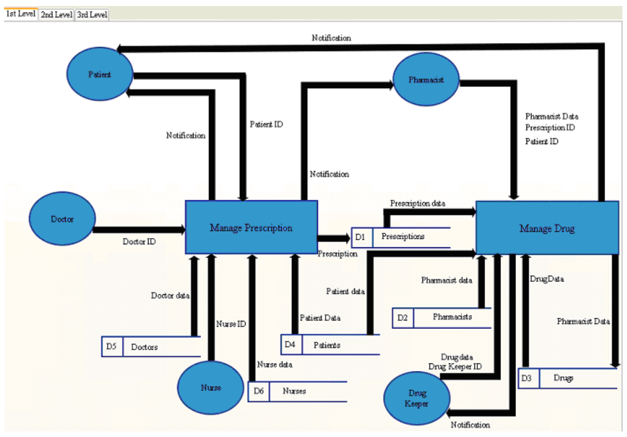


Fig. 4. 1st level DFD, created automatically by NALASS.

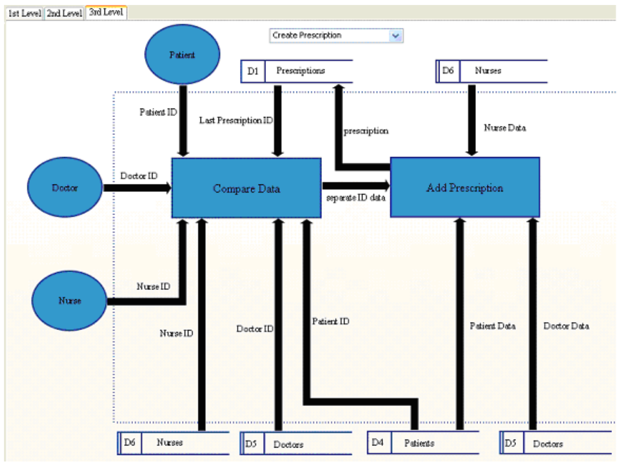


Fig. 5. 3rd level DFD, created automatically by NALASS.

Further rules (not mentioned here due to space limitation) are applied to link functions at the same level. Another indicative rule is that the roles of Creator, Accompaniment, Alterator, Intended Recipient, Experiencer and Notifiee correspond to actors and are represented by a circle. Furthermore, for the functions *Creation, Alteration* and *Erasure*, the role(s) that appear on the left of the name of each function in the corresponding FSR syntax provide data input to the function, hence an arrow from each of these role entities (actors) goes to the relevant function.

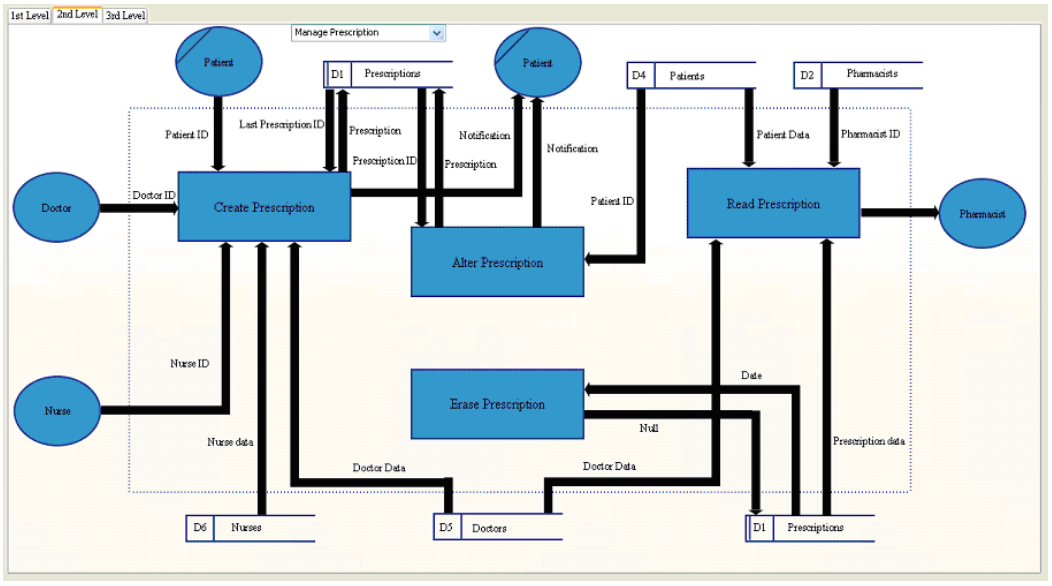


Fig. 6. 2nd level DFD created automatically by NALASS.

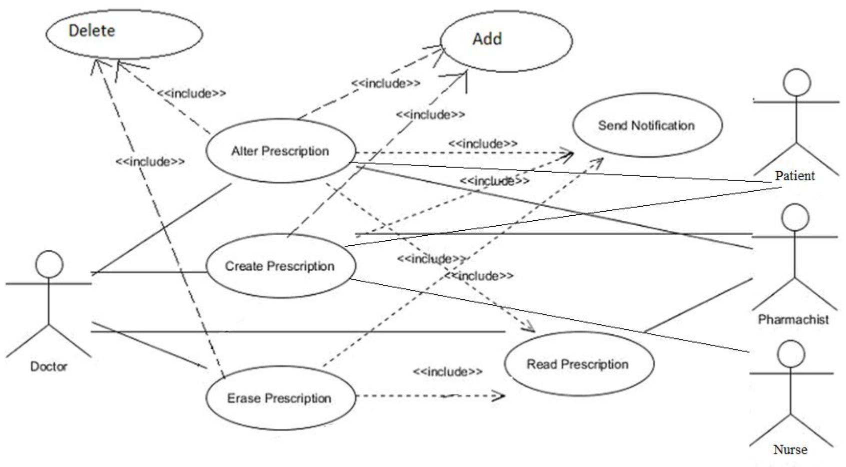
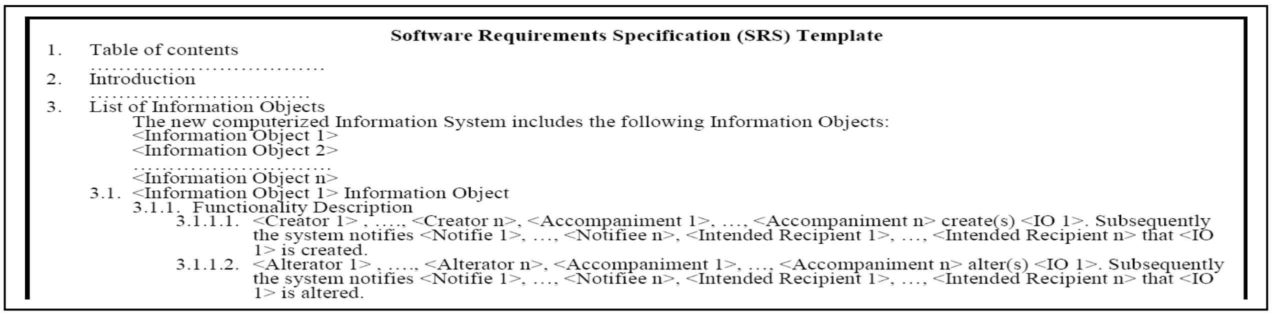


Fig. 8. The use case diagram of the Prescription module, which is created automatically by NALASS by parsing the FSRs of a particular IO.

TABLE I: USE CASE SPECIFICATION EXAMPLE FOR 'CREATE PRESCRIPTION'.

Use Case Name:	Create Prescription
ID:	
Description:	The doctor fills out the form for a new prescription and the system sends the prescription electronically to the pharmacist.
Preconditions:	1. Create Examination 2. Examination is at *Complete* state.
Actors:	Doctor, Nurse, Pharmacist, Patient
Post-Conditions:	Prescription is in *Pending* state
Flow of Events:	1. Doctor/Nurse enters Patient ID. 1.1. The System checks Patient ID. 2. Doctor/Nurse enters Pharmacist ID. 2.1. The System checks Pharmacist ID. 3. Doctor/Nurse enters Drug Name. 3.1. The System checks Drug Name. 4. Doctor/Nurse enters Drug Dosage. 4.1. The System checks Drug Dosage. 5. Doctor/Nurse enters Other details. 5.1. The System checks Other details. 6. Doctor/Nurse clicks on the Submit button. 6.1. The System stores the Prescription in the database. 6.2. The System notifies the Doctor, Nurse, Pharmacist, and Patient that Prescription is created.
Exception condition:	1.1. The System displays 'Invalid Patient ID' message, if patient ID is incorrect. Prescription cannot be saved. 2.1. The System displays 'Invalid Pharmacist ID' message, if Pharmacist ID is incorrect. Prescription cannot be saved. 3.1. The System displays 'Invalid Drug Name' message, if Drug Name is incorrect. Prescription cannot be saved. 4.1. The System displays 'Invalid Drug Dosage' message, if Drug Dosage is incorrect. Prescription cannot be saved. 6. The System does not take any action if Doctor/Nurse click on the Cancel button.



Software Requirements Specification (SRS) Template

1. Table of contents
   ……………………………
2. Introduction
   ……………………………
3. List of Information Objects
   The new computerized Information System includes the following Information Objects:
   <Information Object 1>
   <Information Object 2>
   ……………………………
   <Information Object n>

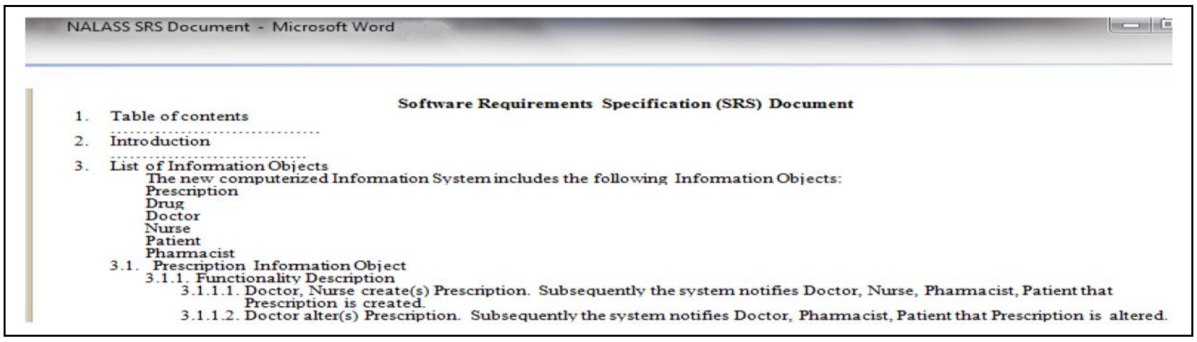
3.1. <Information Object 1> Information Object

3.1.1. Functionality Description

3.1.1.1. <Creator 1> , …., <Creator n>, <Accompaniment 1>, …, <Accompaniment n> create(s) <IO 1>. Subsequently the system notifies <Notifie 1>, …, <Notifiee n>, <Intended Recipient 1>, …, <Intended Recipient n> that <IO 1> is created.

3.1.1.2. <Alterator 1> , …., <Alterator n>, <Accompaniment 1>, …, <Accompaniment n> alter(s) <IO 1>. Subsequently the system notifies <Notifie 1>, …, <Notifiee n>, <Intended Recipient 1>, …, <Intended Recipient n> that <IO 1> is altered.

Fig. 9. SRS document template given as input to NALASS.



NALASS SRS Document - Microsoft Word

Software Requirements Specification (SRS) Document

1. Table of contents
   ……………………………
2. Introduction
   ……………………………
3. List of Information Objects
   The new computerized Information System includes the following Information Objects:
   Prescription
   Drug
   Doctor
   Nurse
   Patient
   Pharmacist

3.1. Prescription Information Object

3.1.1. Functionality Description

3.1.1.1. Doctor, Nurse create(s) Prescription. Subsequently the system notifies Doctor, Nurse, Pharmacist, Patient that Prescription is created.

3.1.1.2. Doctor alter(s) Prescription. Subsequently the system notifies Doctor, Pharmacist, Patient that Prescription is altered.

Fig. 10. Excerpt of the SRS document created automatically by NALASS.

Object Oriented component: Within this component, the IOs become Classes, and the functions of the FSRs for each IO become the methods of the IO. For attributes, we distinguish several types, examples of which are the Primitive attributes, which are related to the IO per se and usually refer to its physical characteristics (e.g., for the *Patient* IO, primitive attributes include temperature, height, mass), and the Peripheral attributes that refer to other IOs related to the IO under study (e.g., for *Patient*, peripheral attributes include *Doctor, Disease*) and usually appear in the FSR. Specific rules are applied by NALASS to transform the IOs into classes and to define attributes and methods. For example, peripheral attributes are all defined in the FSR patterns. Hence, NALASS reads the FSRs and transforms the roles of the syntactic subject of each FSR (e.g. Creator, Accompaniment) and the Intended Recipient and Notifiee into attributes. Further rules regarding the relationships between classes and cardinality are realized by NALASS. Fig. 7 shows the automatic construction of the *Prescription* and *Drug* classes, with their attributes and relationship.

Use case component: This component reads the FSRs of each IO, and for each IO, it creates a Use case model that includes at least five main use cases, one for each of the 5 FSRs. FSR sub-functions (e.g. *add*, *delete*) correspond to possible “include” use cases or use case actions (figure 8). The *Read* FSR derives a *Read* use case which is <included> in the *Alter,* and *Erase* use cases. Another indicative rule is that the subject roles of *creator* and *alterator* in the FSR are identified as primary actors and positioned on the right of the use case diagram. For the development of the use case specification (Table I), the component uses further rules and also reads the completed attributes for each IO. For example, the sub-functions of each FSR (add/enter, compare/check) for a particular IO, with the IO attributes are used to construct the transactions (flow of events) of each use case of the IO use case model.

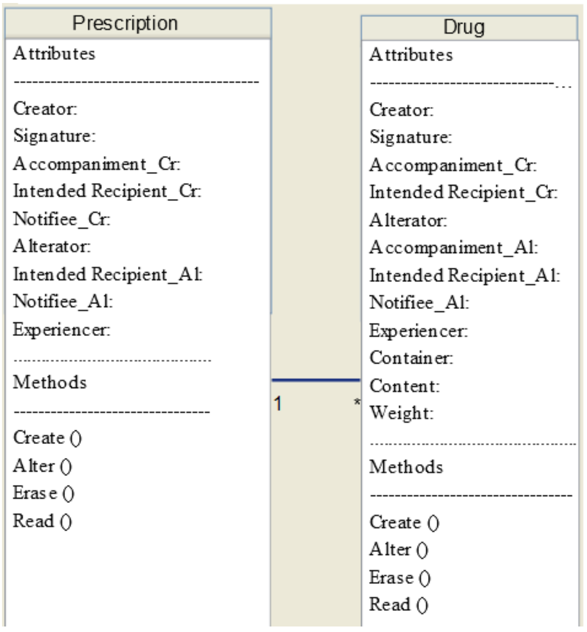


Fig. 7. General form of a class diagram created automatically by NALASS.

The SRS document component: The Documentation component receives as inputs the processed (fed with the answers of the users) elements of the NLSSRE methodology including FSRs, IOs and IO attributes, the SRS template that determines the organization and formatting of the SRS document, and the rules to convert the aforementioned inputs into a well-structured SRS document.

The tool reads the template and applies: (a) formatting rules for the formatting of the new SRS document, by identifying the formatting elements of the template, such as fonts type and size, and line spacing, and applying them to define the format of the new SRS document; (b) substitution rules, by replacing the template variables included in "< >", as shown in Fig. 9 with the values of the components of the corresponding FSRs, IOs and attributes of IOs, as shown in figure 10 (e.g. Information Object 1 is replaced by Prescription, and Creator of IO 1 is replaced by Doctor).

V. Conclusions and Future Work

This paper has presented NALASS (Natural Language Syntax and Semantics), a software tool that is intended to automate the application of the NLSSRE methodology (Natural Language Syntax and Semantics Requirements Engineering) as illustrated in [3]-[4]. Like the methodology on which it is based, the tool can be used through the entire Requirements Engineering process to automate large parts of requirements discovery, analysis and specification. NALASS provides a friendly graphical user environment for the Information Systems (IS) analyst, and it reduces the time required for the manual application of the NLSSRE methodology. For the requirements discovery stage, specific sets of questions are automatically created based on the specific predefined types of data attributes and patterns of formalized sentential requirements that are given in advance; for the requirements analysis stage, the requirements are automatically organised and classified according to the same types of data and patterns of formalized sentences; and for the requirements specification stage, the tool can automatically generate Data Flow Diagrams (DFDs), Class Diagrams, Use case specifications and diagrams, and the Software Requirements Specification (SRS) Document.

Our work is still in progress, so future considerations involve (i) expansion of the tool features, such as the automatic generation of activity diagrams, and embedding of DFDs, Class Diagrams and Use case diagrams and specifications (the automatic creation of which is already implemented in NALASS) to the right section of the SRS document (as also indicated in the IEEE SRS template), and (ii) development of a web version of the tool, since now is only available in a desktop version.

References

[1] The Standish group. (23 April 2009). *The CHAOS report*, Press release [Online]. Available: http://www1.standishgroup.com/newsroom/chaos_2009.php

[2] L. Goldin and D. Berry, "Abstfinder: A prototype natural language text abstraction finder for use in requirement elicitation," *Automated Software Engineering*, Kluwer Academic Publishers, Netherlands, 1997, pp. 375–412.

[3] M. Georgiades, A. Andreou, and C. Pattichis, "A Requirements Engineering Methodology basde on Natural Language Syntax and Semantics," in *Proc. of the 13th IEEE International Requirements Engineering Conference*, August 29-September 02, 2005, Paris, France, pp. 473-474.

[4] M. Georgiades and A. Andreou, "A Novel Methodology to Formalize the Requirements Engineering Process with the Use of Natural Language," presented at IADIS International Conference on Applied Computing, October 14-16, Timisoara, 2010.

[5] IEEE Std 830-1998, "Recommended Practice for Software Requirements Specifications," *IEEE Xplore*, 1998.

[6] F. Fabbrini, M. Fusani, S. Gnesi, and G. Lami, "An Automatic Quality Evaluation for Natural Language Requirements," presented at Seventh International Workshop on Requirements Engineering: Foundation for Software Quality, Interlaken, Switzerland, 2001.

[7] F. M. Burg, *Linguistic Instruments in Requirements Engineering*, IOS Press, 1997.

[8] V. Ambriola and V. Gervasi, "Processing natural language requirements,"in *Proc. Of ASE 1997*, pp. 36-45.

[9] N. Kassel and B. A. Malloy, "An Approach to Automate Requirements Elicitation and Specification," in *Proc. of the 7th Int. Conf. on Software Engineering and Applications*, November 3-5, 2003, Marina del Rey, CA, USA, pp. 544-549.

[10] IBM Rational Rose. (July 12, 2010). [Online]. Available: http://www-306.ibm.com/software/rational/

[11] MagicDraw. (July 12, 2010). [Online]. Available: http://www.magicdraw.com/

Marinos G. Georgiades obtained a BSc and a Ph.D. in Computer Science from the University of Cyprus and an MSc in Information Management from the University of Sheffield. His research interests include Software Engineering and more specifically Requirements Engineering with emphasis on the use of Natural Language for the formalization and automation of software requirements elicitation, analysis and specification. He is the recipient of the ISDA 2010 best student paper award.

Andreas S. Andreou studied Computer Engineering and Informatics at the University of Patras, Greece (Diploma, 1993, Ph.D., 2000). Prior to joining the academia he worked in the industry at the posts of Programmer-Analyst, of Director of Requirements Analysis and Development and of IT consultant in Banking Systems. Currently he is an Associate Professor at the Department of Electrical Engineering and Computer Engineering and Informatics of the Cyprus University of Technology. He also served as Software Engineering and IT consultant in several major software projects in Cyprus, including the Integrated Software System for the New Nicosia General Hospital. His research interests include Software Engineering, Web Engineering, Electronic and Mobile Commerce and Intelligent Information Systems.