A Novel Software Tool for Supporting and Automating the Requirements Engineering Process with the use of Natural Language

[1]Marinos G. Georgiades, [2]Andreas S. Andreou
[1]Dept. of Computer Science, University of Cyprus, Nicosia, Cyprus
[2]Dept. of Computer Engineering and Informatics, Cyprus University of Technology, Limassol, Cyprus

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 documentation. 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, such as object related data flow diagrams and class diagrams, which are presented in this paper.

Keywords
Requirements Engineering tool; automated RE; natural language RE

I. Introduction

Recent studies show that the least understood parts of systems' development are the stages of requirements discovery, analysis, and specification (e.g. The Standish group [1]). The problem observed is that there is an enormous 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 Georgiades and Andreou [3], 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 [4]. 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 Object Related Data Flow Diagrams (ORDFDs – defined later), Class Diagrams, Use case specifications and diagrams, and the Software Requirements Specification (SRS) Document. The generation of the first two types of diagrams is discussed in this paper. The rest of the paper is organized as follows: Section II examines relevant literature on RE tools and describes how NALASS differs from similar propositions. Section III provides a summary of the NLSSRE methodology and its application within the tool, while section IV offers a detailed description of the tool. 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 in general 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 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. Fabbrini et al. [5] propose 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 [6] and Circe [7] parse a set of structured requirements in natural language to generate specific models (ER, DFD, OO design, etc.) The common characteristic of these and other related parsing tools is that they are mostly used and applied to pre-existing documents with disorganized text, redundancies and ambiguities. As a result, the retrieval approach is not particularly reliable, as explained earlier in this section. Other tools, such as the one reported by Kassel and Malloy [8], are not

parsers and offer the user the capability to enter the requirements from scratch, but they also lack specific types of questions (for RD) linked to the identification of data and functions of an IS. In contrast, NALASS implements the NLSSRE methodology and provides specific predefined requirement patterns, specific categories of data, functional conditions and business rules, from which automatically derives specific sets of questions. 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 that is why current approaches and tools often result to inadequate requirements and diagrams. Additionally, NALASS may be conceived as a complete toolset that can generate ORDFDs, Class Diagrams, Use Case specifications and diagrams, as well as a well-structured NL-SRS document that covers the essential parts outlined by the IEEE SRS template [4].

III. Methodology Overview

The NLSSRE methodology introduced by Georgiades and Andreou [3] 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 Library IS) that needs to produce, change or present electronic information about documents or other physical entities (e.g., student, book, etc.) The first step of NLSSRE guides the analyst to identify specific discrete data entities, called Information Objects (IO). An Information Object (IO) is defined as a digital representation of a tangible or intangible entity—described by a set of attributes—which the users need to manage through Creating, Altering, Reading, and Erasing its instances, and be Notified (CAREN) by the messages each instance (IOi1) can trigger. The next step—which is central in utilizing the methodology—involves the application of specific functions on every IO, as well as the written specification, in the form of formalized sentences (Formalized Sentential Requirements – FSRs), of the IO, its functions, the involved business roles, and the functional conditions. NLSSRE provides specific FSR patterns, based on which it guides the user to derive specific questions to identify the business roles and possible values of the functional conditions; the answers to these questions assist in forming the complete FSRs. Writing the requirements as formalized sentences does not only help to make expression of requirements more disciplined, understandable and organized, but also leads to the identification of entities (business roles and functional conditions) that are involved during the application of a function on an IO. Furthermore, such formalization makes easier the transformation of requirements into diagrammatic notations and specifications.

Fig. 2(a) shows the most basic FSR patterns, namely Create, Alter, Read, and Erase, which are derived from the corresponding CAREN (create, alter, read, erase, notify) functions. CAREN functions are parts of each IO and they are decomposed to sub-functions (at the system user's level, not the programmer's), as depicted in fig.1.

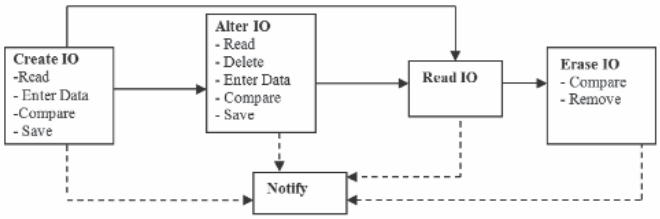

Fig.1 : CAREN - A recommended set of functions and sub-functions applied on an IO, and the notifications produced.

Create : Creation is the most significant function, since during Creation the attributes of an IOi take their initial values which are the basis for further processing by the remaining functions. Creator is the entity that creates the IO, Accompaniment is the entity that assists the Creator in the creation of the IO, Intended Recipient is the entity for which the IO is created and which will utilize the IO within the IS, and Notifiee is the entity that needs to be notified for the creation of the IO (this entity will not use the IO in any way that will cause any interaction within the system). On the right of the symbol "::" the syntax of the Notification function follows, which is triggered after the execution of the function on the left.

Alter : During Alteration, the value of one or more of the attributes of an IOi changes. A significant attribute that changes during alteration is the attribute State. When the IO corresponds to a procedure (e.g., examination) or event (e.g., appointment), the State value may change from Start to Ongoing/ Pending to Finished/ Completed or Cancelled, or even Expired or Archived; when the IO is an inanimate physical object (e.g., book, drug) then State may change from InStock to Sold/Lent, and when the IO is an animate object State usually takes values according to the IOs business role (e.g., Student IO State may be new, studying, graduated, suspended, or Patient IO State may be ill, under treatment, cured); and when the IO corresponds to a document (usually in electronic form, e.g., prescription, voucher), State may take values such as stored, archived, cancelled, edited/reviewed or retrieved. The change from one state to another (e.g., from Pending to Complete), for a particular IO, often derives a new alteration function, such as Cancel,

Read : The meaning of this function may be conceived in two ways: the first, which is the one that concerns requirements analysis, is about what a user wants to read regarding a particular IO per se or from its relations with other objects. It mainly concerns the presentation (optical or acoustical) of notifications and forms regarding the IO per se (e.g., Appointment form), or the presentation of reports of the IO with related objects (e.g., report of a patient's monthly appointments). The second concept for Read concerns the way the data will be presented, including drawings, graphics, video, multimedia, etc.; the first meaning of this concept falls in RE, but the detailed procedures of implementing methods of presentation concerns the Design which is outside of the scope of RE. Experiencer is the entity that experiences IO through viewing it, listening to it, etc.

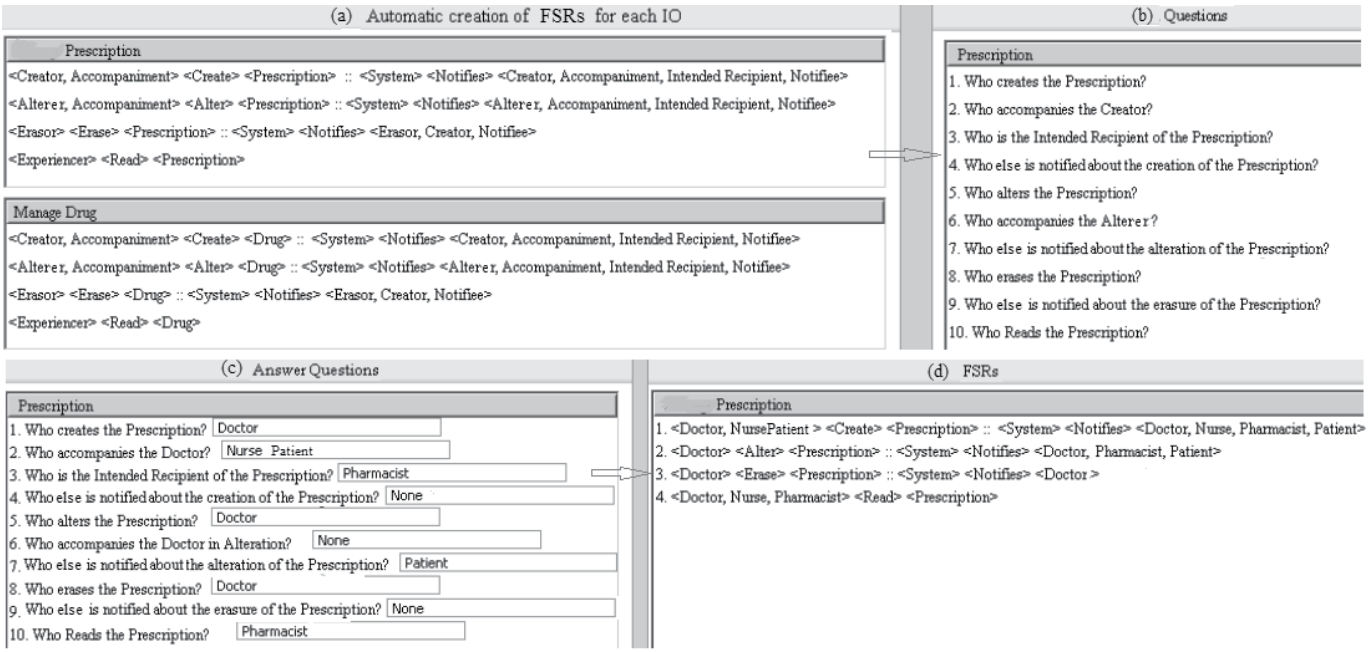

Fig. 2 : The predefined questions (b) created automatically by the FSRs patterns (a), and the resulting complete FSRs (d) created automatically by the answers of the users (c), for the Prescription IO – screenshots are taken from NALASS that automates and supports the NLSSRE methodology.

Erase : Erasure of an IOi means that the IOi is permanently deleted. All of its information about attributes and functions that exist in the context of the IS is deleted.

Notify : At the user level, in a manual, paper-based IS, we encounter the transmission function (from the linguistic verb of transfer of possession), where data is sent from one entity to another. For example, the Doctor gives the Prescription to the Patient, and the Patient gives the Prescription to the Pharmacist. In a computerized IS the transmission of prescription is replaced by the Read function, since the IO (Prescription, in this case) is already stored (after its creation or alteration) in the IS. Hence, the Pharmacist can Read the Prescription IOi by simply retrieving it from the database. However, in a computerized IS, transmission exists at the messaging level, which we call Notification. In particular, when an IOi is created or altered (or even read), then a notification should be sent to the interested parties which are classified into two groups the Intended Recipients (IR) who will have to take an action within the IS as a consequence of the creation or alteration of the IOi (e.g., a Pharmacist is the IR of a Prescription IOi, because, after its creation, s/he will utilize it to create a Drug IOi), and other entities who just need to be informed about the creation or alteration of the IOi, called Notifiees (e.g., patient in the Prescription IOi example). Subsequently, based on the syntax of each FSR pattern and the functional roles involved in each pattern (e.g. Creator, Accompaniment), the tool derives questions, the answers of which are used to feed the FSR patterns. Then the complete FSRs are used by the tool with the attributes for each IO, collected during the third step of the methodology, to build diagrammatic notations, based on specific rules. This paper focuses on the capability of the tool to generate (i) Object-Related Data Flow Diagrams which are defined as data flow diagrams whose functions are applied on information objects (Information Objects). Thus ORDFDs consist of the CAREN functions; and (ii) Class Diagrams. 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.

IV. The NALASS Tool

The tool consists of three main sections: Administration, Plan and Execution. In the Administration section, the analyst can create/ add new types of IS elements, such as FSR patterns and data attribute types that may apply to any project. In the Plan section, the analyst builds the particular elements of a particular project, including its IOs, the FSR patterns of each IO, IO attributes, and questions for each IO. Finally, in the Execution section, the analyst provides answers to the questions, the tool completes the FSRs and attribute values, and it finally uses specific rules to transform the complete FSRs and attributes to diagrammatic notations.

Below we illustrate, with examples, the Plan and Execute sections.

i. The first step for the analyst, in the Plan section, is to use a particular guide, provided by NLSSRE, 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. Subsequently, for each IO, the four main FSR patterns (fig. 1a – Create, Alter, Read, Erase) are created automatically by NALASS. Fig.2a shows the FSR patterns for the information objects Drug and Prescription. The tool also provides other alteration functions, as mentioned previously, such as Cancel, Complete, and Archive, and the analyst can choose which ones to use depending on the category of the IO. For each CAREN function of an FSR, the tool also provides its sub-functions which are depicted in fig.1.

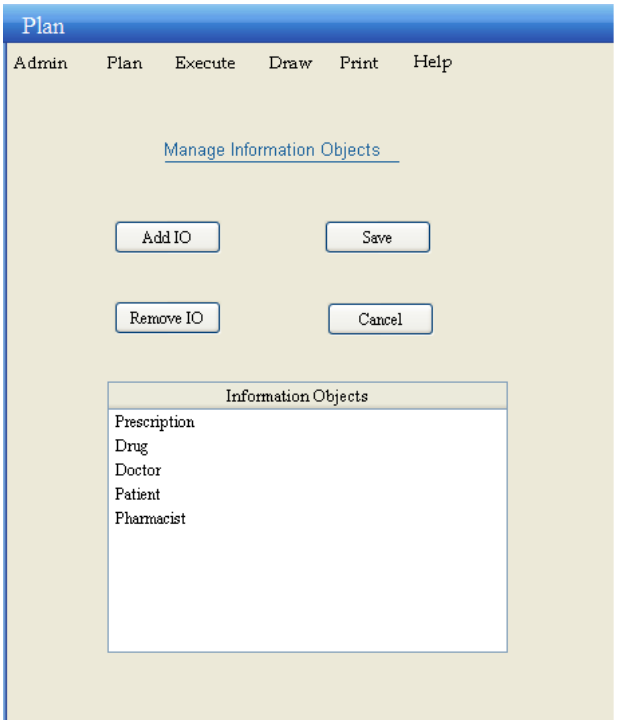

Fig.3 : Adding Information Objects

As the next step, the questions are created automatically by the tool, as shown in fig. 2b, based on the different functional roles (Creator, Accompaniment, etc.) of each FSR pattern, which need to take a value. For example, the Creator of the Prescription needs to take a value, and so we need to ask who the creator of the prescription is. These questions will be submitted to the users of the IS as illustrated in the next paragraph. It is worth noting here that this formalization in providing specific questions that are linked to the analysis and organisation of requirements is the difference from other 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.

II. In the 'Execution' section

The analyst submits the answers received from the users to the form provided by NALASS (fig. 2c). The answers to the questions feed the FSR patterns, as they are the values of the constituent elements (e.g., of functional roles) of the patterns, and generate the complete FSRs as shown in fig.2(d) (e.g. Creator takes the value Doctor which is a business role).

Subsequently the FSRs and their constituent elements, as well as the IO attributes, with the use of specific rules are transformed to ORDFDs, Class diagrams, Use case specifications and diagrams, and the SRS document. In this paper we focus on the transformation to ORDFDS and class diagrams.

A. Transformation to ORDFDs

Within this transformation, 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. 2, 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, and Erase functions for each Manage IO are the functions of the 2nd level DFD (fig. 5). Below we provide in more detail the most basic rules of this transformation:

- The first level ORDFD will include all the Manage IO functions fig.4. Functions are represented by a rectangle.
- The second level ORDFD will include all the 2nd level functions (Create, Alter, Read, Erase) of each first level function (Manage IO) as shown in fig.5.
- For the third level DFD, the second level functions are decomposed to the CAREN sub-functions, according to fig.1. For example, the 2nd level function Create Prescription is decomposed to Enter Data (incorporates the Read and Compare sub-functions) and Save (see fig.6). .
- The functional roles Creator, Accompaniment, Alterer, Intended Recipient, Experiencer and Notifiee correspond to actors (or business actors or business roles) of a traditional DFD and are represented by a circle.
- For the functions Create, Alter and Erase, the business role (s)/ actors (s) that appear on the left of the name of each function, in its syntax, provide data input to the function, hence an arrow from each of these actors goes to the relevant function (e.g. from Doctor to Create Prescription - Fig. 5).
- For the Read function, in the 2nd level of decomposition, the business role of Experiencer receives the IO in a special format/layout for reading (viewing, listening, etc.). Hence an arrow from the Read function goes to the Experiencer actor (business role) in the ORDFD as shown in Fig. 5 (Read Prescription – Pharmacist).
- The Create, Alter and Erase functions create a data flow from the relevant function to the relevant datastore, because the IO is changed and needs to be (re)stored; hence an arrow goes from each function to the datastore (e.g. from Create Prescription to Prescriptions).
- The Create, Alter and Erase functions create data flows from the relevant datastore (which is created because of these functions) to the relevant function, because the function needs to check the IO before altering it; hence an arrow goes from the datastore to each function (e.g. from Prescriptions to Create Prescription).
- The Read function creates a data flow from the relevant datastore to the Read function; hence an arrow goes from the datastore to the function (e.g from Patients to Read Prescription).
- The entities that appear on the right of Notifies in the syntax of the Notification function receive an arrow (data flow) from the relevant function which appears on the left of the Notification function (e.g. from Create Prescription to Doctor, Nurse, Pharmacist, and Patient).
- The tool automatically defines the relation of each function at the highest level (Manage IO). In particular the link of one high-level function to another is created, when the output of one high-level function is an input to another high-level function, e.g., the Manage Drug function uses the Prescription IO which is an output of the Manage Prescription function. In this way, in the ORDFD, the high-level function (in this example Manage Drug) will retrieve the IO from the datastore Prescriptions where the linked high-level function (in this example Manage Prescription) stored it (fig. 4). This procedure is done automatically by the use of the following rule: The Intended Recipient of an IO needs to Read that IO. Thus a link from the relevant datastore of that IO to the Manage IO function of the new IO in which the Intended Recipient is involved as its Creator or Alterer needs to take

place. E.g. the Pharmacist is the Intended Recipient of the Prescription as shown in the syntax of Create Prescription (Fig. 2d), and the Pharmacist will Read the Prescription in order to Manage Drug. Hence a link from Prescriptions (datastore) to Manage Drug is created (Fig. 4).

B. Transformation to Class Diagrams

Specific rules are used to transform the FSRs and attributes of each IO to class diagrams. Each IO is transformed to a Class, and its CAREN functions become the methods of the class. Additionally, each IO contains specific attributes according to its IO category (business role, inanimate object, procedure, document, etc.). Some attributes are compulsory and others are optional. Indicatively two of the attribute categories provided by NLSSRE 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, Receptionist, Disease) and usually appear in the FSR patterns of the IO (e.g. (i) "Receptionist,Patient Create Registration"; (ii) "Doctor,Patient Create Prescription"; (iii) "Doctor,Patient Diagnose Disease" – in all these patterns, Patient is an accompaniment). Hence, once information about the attributes of each IO is identified, this information can be refined and codified to the exact attributes of each IO. As an example, information about Doctor, which constitutes one or more attributes of the IO Prescription, can be refined and codified to the specific attributes of Doctor ID, Doctor Signature, Doctor Name, and Doctor Surname. NALASS facilitates this process by providing a grid including all the possible attribute categories for each IO. For the peripheral attributes, NALASS finds all the FSRs in which the IO under study is involved and returns the other participants (e.g., business roles) of the mentioned FSRs. Further rules regarding the relationships between classes and cardinality are realized by NALASS, such as:

- An association relationship exists between the IO and each business role—which actually constitutes an IO, too—in the same FSR.
- There should be:
  (i). A one-to-many association between Creator (e.g., Doctor in the Create Prescription FSR) and IO (Prescription) (apart from rare cases where there could be more than one creators for the same IO)
  (ii). A one-to-many association between the client business role (Patient – otherwise called external accompaniment role) and the IO (Prescription)
- There could be a many-to-many association between Creator (Doctor) and internal Accompaniment (e.g., Nurse or Counselor).

Fig.7 shows the Prescription and Drug classes, with their attributes (types) and relationship, as generated automatically by NALASS.

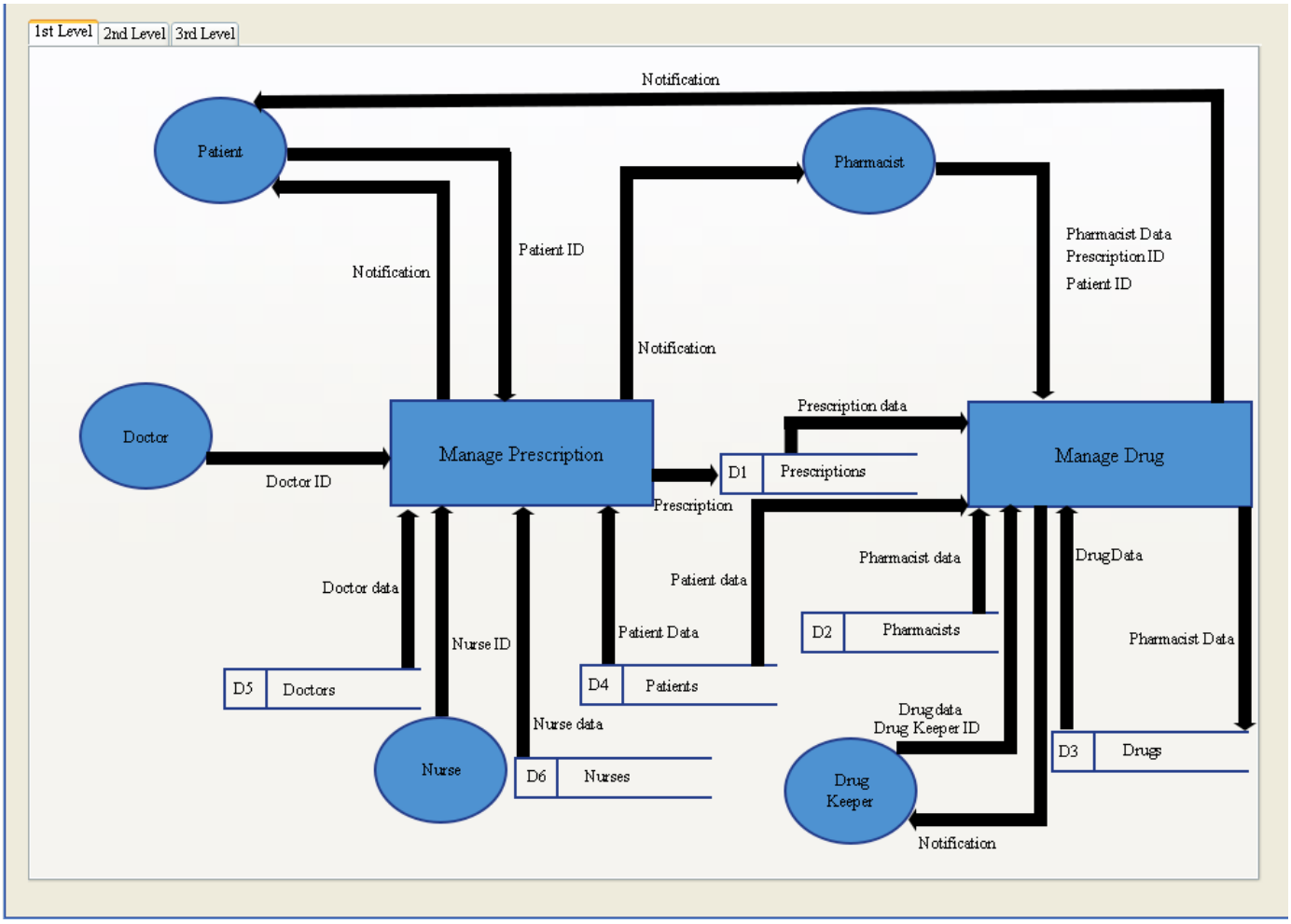

Fig. 4 : First Level ORDFD created automatically by NALASS

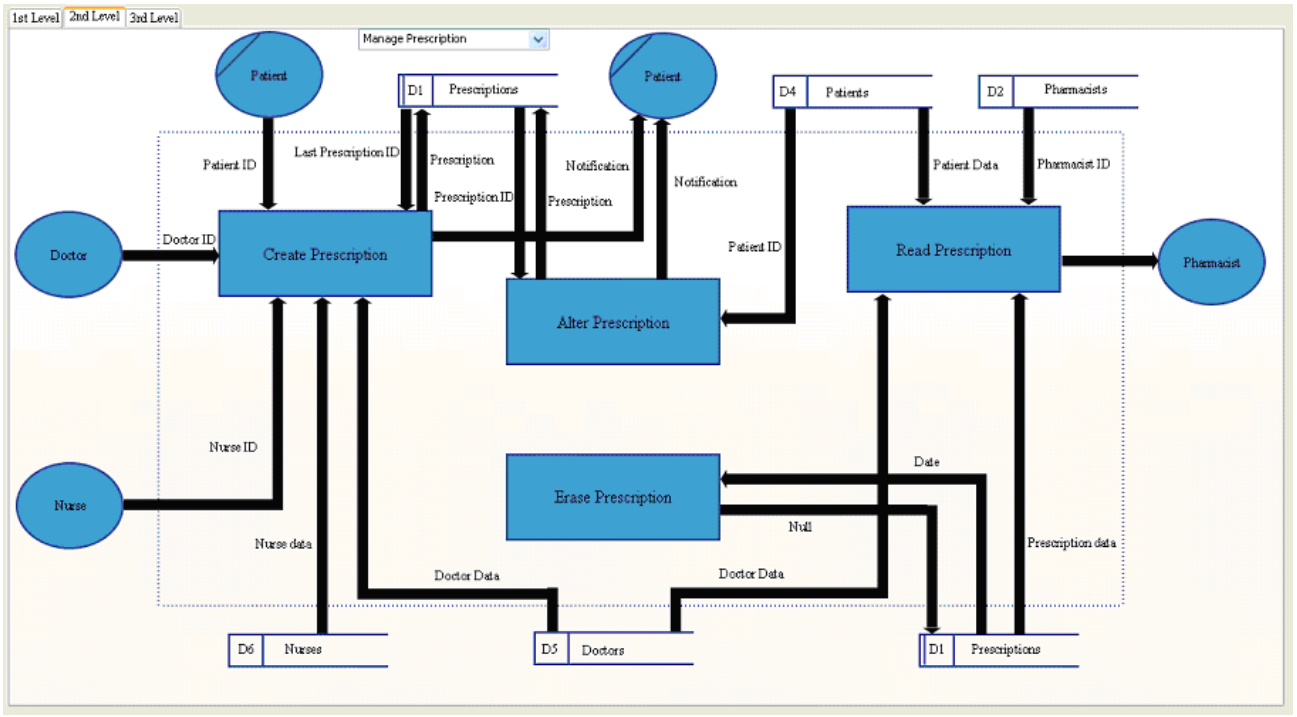

Fig. 5 : 2nd level DFD created automatically by NALASS

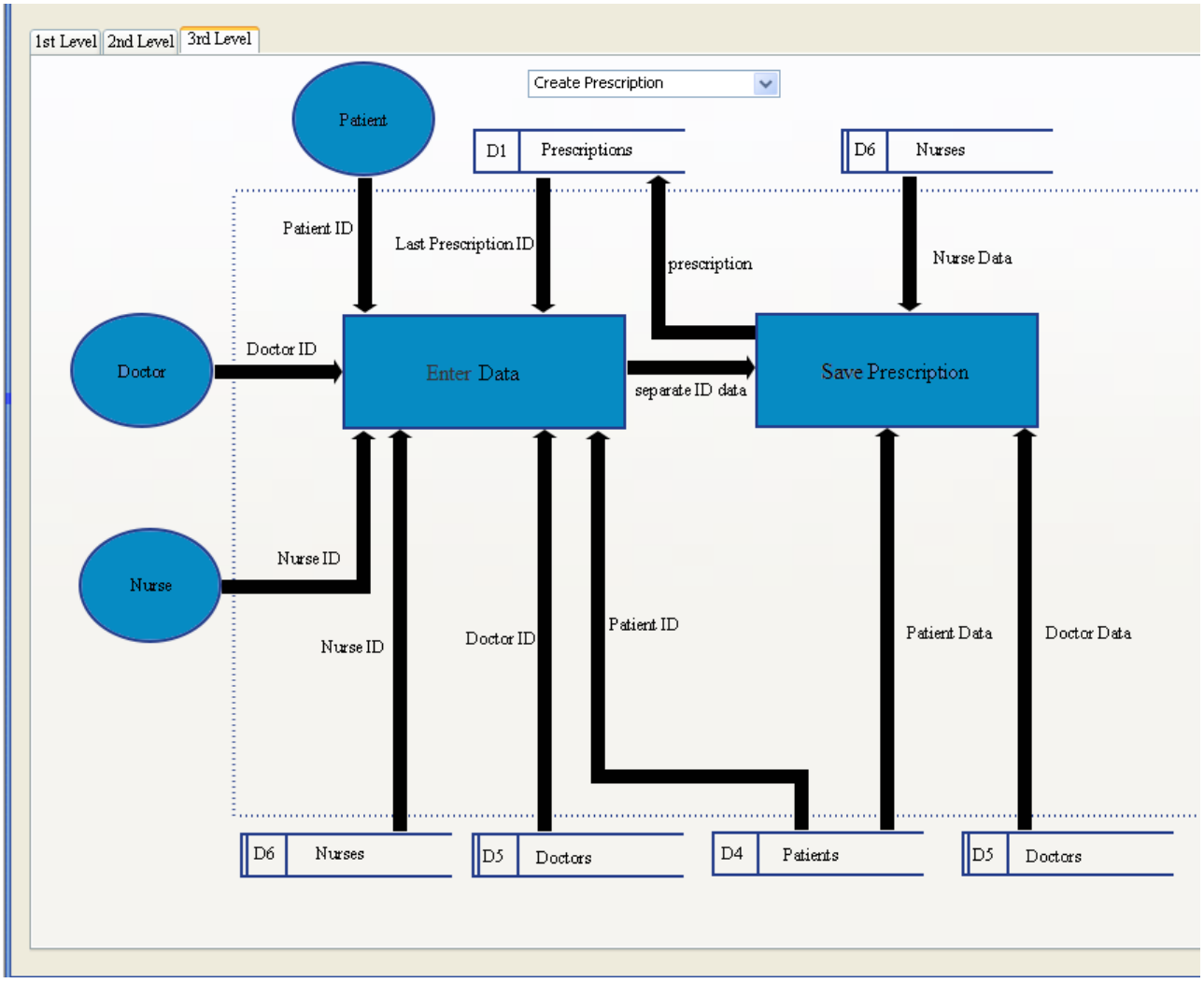

Fig. 6 : Third Level ORDFD for the function Create rescription

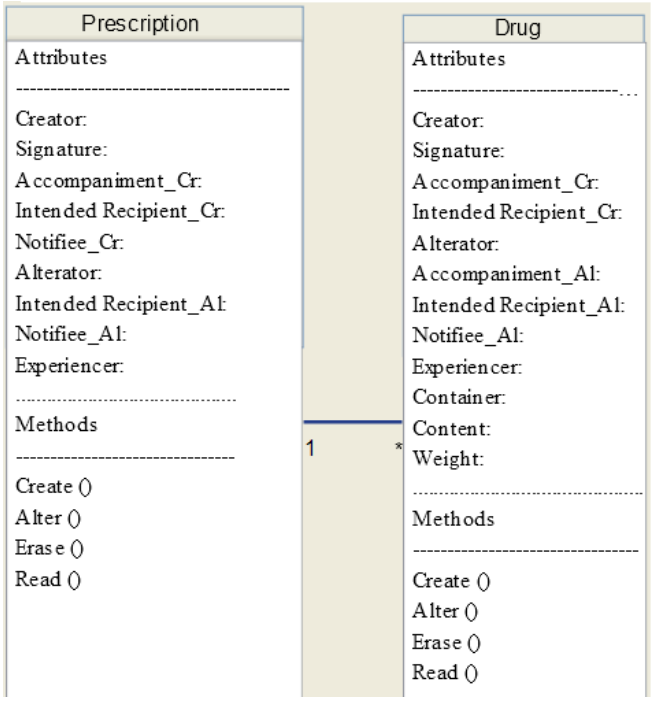

Fig.7 : General form of a Class diagram created automatically by NALASS

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]. 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 diagrammatic notations such as Object-Related Data Flow Diagrams (ORDFDs) and Class Diagrams. Our work is still in progress, so future considerations involve (i) expansion of the tool features, such as the automatic generation of activity diagrams, as well as the improvement of the tool in generating class diagrams, use case diagrams and specifications, (ii) development of a web version of the tool, since now is only available in a desktop version.

References

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

[2] L. Goldin, 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. 2010. "A Novel Methodology to Formalize the Requirements Engineering Process with the Use of Natural Language", In Proceedings of the IADIS Conference on Applied Computing (Timisoara, Romania, October). IADIS Digital Library, pp. 11-18.

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

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

[6] F. M. Burg, "Linguistic Instruments in Requirements Engineering", IOS Press, 1997.

[7] V. Ambriola, V. Gervasi, "Processing natural language requirements," Proc. ASE 1997, pp. 36-45.

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

[9] IBM Rational Rose. [Online] Available : http://www-306.ibm.com/software/rational/

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 / 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.