NL-OOPS: from natural language to object oriented requirements using the natural language processing system LOLITA

L. MICH

Department of Computer and Management Science, University of Trento, Via Inama 5, 38100 Trento, Italy

(*Received 17 April 1996; revised 24 July 1996*)

Abstract

This paper describes NL-OOPS, a CASE tool that supports requirements analysis by generating object oriented models from natural language requirements documents. The full natural language analysis is obtained using as a core system the Natural Language Processing System LOLITA. The object oriented analysis module implements an algorithm for the extraction of the objects and their associations for use in creating object models.

1 Introduction

In this paper we describe an NLP-based CASE (Computer Aided Software Engineering) prototype – NL-OOPS, an acronym for Natural Language – Object Oriented Production System – that supports natural language requirements analysis by extracting the objects and their associations for use in creating object models. It is a prototype developed in a joint project between the Computer and Management Science Department of Trento University (I) and the Laboratory for Natural Language Engineering, Department of Computer Science of Durham University (UK). As a core system for the CASE tool, we are using the Natural Language Processing System LOLITA (Large-scale Object-based Linguistic Interactor Translator Analyser), a system developed at Durham University (Long and Garigliano, 1994).

1.1 The problem

It is widely recognised that the process of developing software systems is time and resources consuming. It requires both a systematic and a creative approach. There is a trade-off between these dimensions: small projects could be managed in a less 'formalised' manner, but as the size of the project increases, it becomes more and more necessary to follow common guidelines to manage the complexity of the system. To increase the productivity of the system development process it is necessary both to adopt a new paradigm and to use suitable CASE tools. In recent years, the demand

https://doi.org/10.1017/S1351324996001337 Published online by Cambridge University Press

of large system development has led to the extension of a relatively new approach – the Object Oriented (OO) paradigm – from the programming domain to cover the full development life cycle, including analysis and design. However, there are few CASE tools able to support the first steps of systems development, requirements elicitation and definition, which are generally considered too hard to structure. The main reason is that most of the input consists of natural language documents. The Artificial Intelligence results in the field of Natural Language (NL) suggest a fruitful approach, that is using a Natural Language Processing (NLP) system to support system development starting from the earlier phases of requirements analysis. Moreover, there is a strong relationship between natural language and object oriented structures, and many authors have proposed a kind of NL analysis as a starting point for conceptual analysis.

OO analysis is supported by NL-OOPS from the very beginning of the system development process, that is starting with a pre-processing activity based on the LOLITA functionalities, correcting and normalising the input requirements texts. Information is memorised in a particular form of conceptual graph (Sowa, 1983), LOLITA's SemNet, which can then be used to build the various models required by the OO methodology adopted. This modelling activity is based on an algorithm for the extraction of the objects and their associations. The output can also be used for a requirement validation if necessary, gearing it to the level of linguistic competence of the user or expert. The validation process is based on the automatic text generation from sections of the conceptual graph (Smith, Garigliano and Morgan, 1994).

1.2 Structure of the paper

The paper is organized as follows. Section 2 describes the role played by Natural Language in the development of CASE tools devoted to the initial phases of system development and in OO modelling. Section 3 gives an overview of the project, describing the requirements for the CASE tool and the role of LOLITA as a *core* for an NLP-based CASE tool. Section 4 provides a description of the OO analysis module and a complete case study, followed by a presentation of some implementation issues for NL-OOPS. The main related works are presented in section 5. Finally, in the last section both the open problems and the future developments of the project are presented.

2 Natural language and object oriented requirements analysis

2.1 Role of natural language in requirements analysis

The requirements elicitation phase is critical and developed mainly through dialogue between the users and the systems analyst.[1] Usually many people are involved in the requirements analysis, therefore communicative processes are the starting point for the interpretation process necessary to define and model system requirements.

[1] For a book about the fundamental concepts related to system development – life cycle phases, requirements analysis, CASE tools – readers may refer to Whitten, Bentley and Barlow (1994).

https://doi.org/10.1017/S1351324996001337 Published online by Cambridge University Press

Whatever technique is used for requirements elicitation – questionnaires and interviews, work observation, brainstorming, focus group, etc. – by far the largest and most important section of available information consists of NL documents, often describing the system in terms of how it should perform, what functionalities it should support and what items should exist in the system. Moreover, many OO methodologies advise on the use of linguistic criteria for object identification and for OO model construction.

In recent years a certain number of NLP-based CASE tools have been realised (see section 5), but we are not going to review all the advantages of a CASE tool based on a real NLP system: the point is that some of these features are a matter of particular interest in OO modelling. All the more so because, in our opinion, we 'build' the world through the language. According to a neo-pragmatist position (Garigliano and Nettleton, 1997), we see the world as a *continuum* and we build descriptions using a form of language, depending on our sensations and on our history. Language is used mostly for describing the world in terms of objects as entities sufficiently stable that we recognise in the continuum and relations between them. Objects are identified on the basis of use and convenience.[2] This way of describing reality (Plato) is predominant in respect of the one describing functions, changes, movements, etc. (Heraclit).

For these reasons NL plays a predominant role in abstraction processes to develop OO models and the role of 'creativity' – understood as the ability to abstract suitable objects, and hence to interpret natural language descriptions coming from different sources – is often considered to be crucial in the OO modelling process.

These observations might explain the difficulties that are encountered in the development of non-NLP-based CASE tools devoted to the initial phases of system development. Without entering the debate on the final possibility of automatic production of 'creative' behaviour, it is clear from recent advances in natural language engineering that major in roads are now possible in the interpretation of NL descriptions.

The tools needed would have to guide the analyst in the gathering and interpretation of key messages, while keeping track of the interactions with a number of users/experts, during which the analyst's point of view contributes as much as anybody else's to determine the 'universe of discourse' of the problem under discussion. It is also important to underline here that fulfilling these requirements is not only a problem of technological maturity, but one that is intrinsic to the deep nature of the processes involved in the analysis phase, and as such, one that presents itself whatever the technology used.

2.2 Extracting objects from natural language documents

One of the most critical activities in the whole object oriented development process is object and association identification. At the conceptual level we cannot assume that "…in the physical or abstract reality being modeled, the objects are just there for

[2] See also Winograd and Flores (1986).

https://doi.org/10.1017/S1351324996001337 Published online by Cambridge University Press

picking!'' (Meyer, 1988). A more profitable position is to consider that ''Every representation is an interpretation and … objects and their properties are not inherent in the world, but arise only in event of *breaking down* in which they become *present at hand*'' (Winograd and Flores, 1986). A detailed classification of the approaches to *discover* classes and objects can be found in Booch (1994). In Booch's survey, two approaches – Structured Analysis and Informal English Description – are cited as alternative techniques. The first one uses Data Flow Diagrams as input to derive candidate objects and classes (Bailin, 1989; Wieringa, 1991), but systems development experience suggests that an OO approach should be adopted in all of the life cycle phases, to avoid changing technique. Informal English descriptions, that is NL requirements statements, are the input first used by Abbott (1983), who applies simple syntactic rules to extract objects and operations.

Not surprisingly, the nature of the abstractions used for the construction of OO models has led, from the beginning, towards the development of heuristics based on a kind of analysis of NL requirements. The linguistic theories adopted have been evolved from syntactic linguistic criteria to more sophisticated text analysis using variants of the Case Grammar by Fillmore (1968). To avoid the problems related to the complexity of free language, many methods use a subset of the NL, or impose on the user a standard form for the requirements input (grammar constraints).

A critical point for using a NLP system as the core of an OO analysis CASE tool is that the knowledge representation has to be independent of the grammatical form of the NL requirements. For example, the sentences ''John kissed Mary'' and ''John gave a kiss to Mary'' have to correspond to the same knowledge representation items.[3] This problem has been outlined by Booch (1994): ''… any noun can be verbed, and any verbs can be nouned; therefore, it is easy to skew the candidate list to emphasize either objects or operations''. In other words, it is not correct to assume a complete isomorphism between syntax and semantics, that is between the role of the words in the sentences and their meaning. This fact is reflected in the rules proposed by all the authors. For example, in the rules for object identification looking for nouns in the problem statement, one finds expressions like 'suggest', 'typically', etc.[4]

Given the nature and the complexity of the requirements analysis, and of OO modelling in particular, it is our contention that a CASE tool for generating OO models from textual requirements documents has to use a 'real' NLP system. In other words, if we do not assume that the language used by analysts and users has to be a sub-set of NL, it is not possible to use *ad hoc* small NLP systems, and a successful approach demands a system able to tackle the full range of NL problems. A clear example which illustrates that it is mandatory to have a real NLP system is represented by the quantification problem. For instance, given the following two sentences, ''John owns a dog'' and ''All men own a dog'', the system has to correctly

[3] This example is further detailed in section 4.1.

[4] ''A common noun in the informal strategy *suggests* a data type. A proper noun or direct reference *suggests* an object.'' (Abbott, 1983). ''*Typically*, the objects we identify in this step derive from the nouns we use in describing the problem space.'' (Booch, 1986). ''Common nouns in the problem statement *can help* in identifying classes and verb phrases the associations between them.'' (Wieringa 1991). ''Classes … *often* correspond to nouns.'' (Rumbaugh *et al*., 1991). ''The objects become *typically* a noun which exists in the domain.'' (Jacobson *et al*., 1992).

https://doi.org/10.1017/S1351324996001337 Published online by Cambridge University Press

understand that we refer to a particular element of the set 'dog' in the first sentence, and to a set of dogs in the second. Furthermore, it is necessary to account for ambiguities, contradictions, omissions, redundancies, etc., and this can be done only if the inferential system is embedded in the NLP system. Most of these features, at different levels of development, can be found in LOLITA, the system used for the NL-OOPS project.

3 The NL-OOPS project

Starting considerations for the project were:

- Requirements analysis plays a key role in the system development life cycle: errors and omissions in this phase may result in failure of the whole system.
- Formalisation can intervene only to specify requirements and cannot help to elicit them: there are a few specification languages, but the identification of the key elements is left to the analyst's experience and intuition.
- Supporting requirements elicitation and definition needs a general-purpose, domain independent NLP system which will not place restrictions on the language used.

Our goal was to implement a system capable of generating OO requirements specifications analysing the input requirements in NL and extracting the objects, and their relationships needed to create an OO model of the requirements (see figure 1).[5] The activities of NL-OOPS can be classified in three groups: the first one contains all the direct interactions with the NLP system LOLITA; the second group includes the activities related to the definition of a *coherent* requirements set, that is the requirements document, or – following Rumbaugh (Rumbaugh *et al.*, 1991) – the problem statement; the last group of activities concerns the real OO analysis.

3.1 The logical architecture of NL-OOPS

An adequate CASE tool for OO system development should:

- deal with all the life cycle phases, including the very early requirement elicitation phaser and the validation phase;
- allow information handling using both text and graphics without requiring the user/analyst to learn new languages;
- support different OO methodologies.

Fundamental points are these: the documents should be in their normal NL form; the user should not be pestered with continuous requests for clarification, and he/she should be required to engage only in necessary and creative activities; the system should be able to deal in some measure with problems in the documents, be they linguistic, logical or due to a clash of perspectives; the final OO model output should not miss essential contents, or contain elements that are not strictly needed to represent requirements, and it should indeed be reusable for successive systems analysis.

[5] For an annotated bibliography on the main topics related to the project, see Mich and Giuliani (1995).

https://doi.org/10.1017/S1351324996001337 Published online by Cambridge University Press

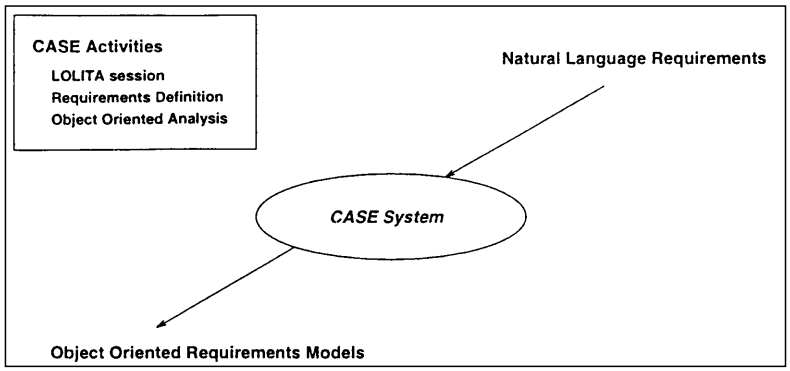


Fig. 1. NL-OO requirements analysis.

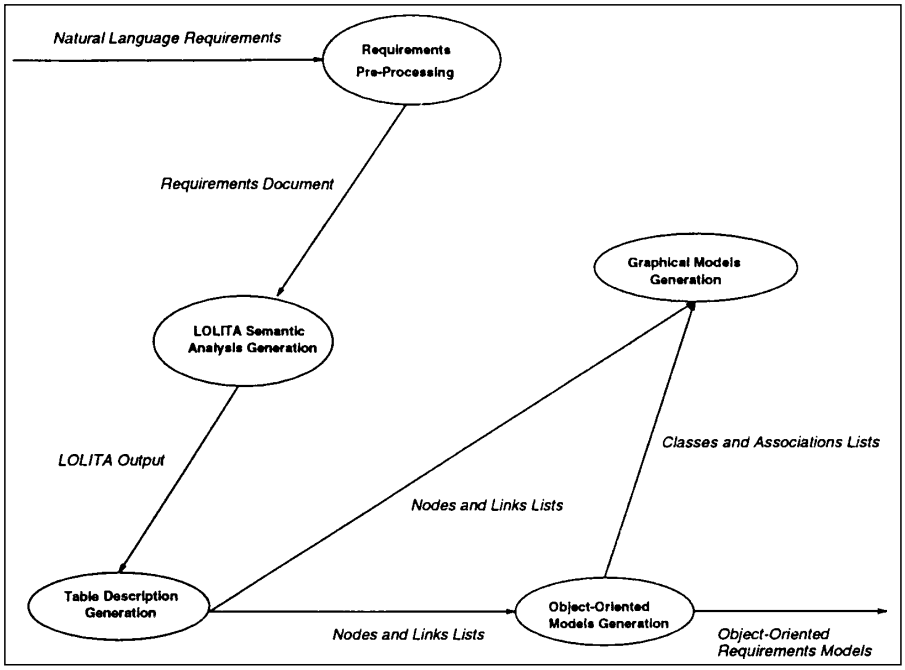


Fig. 2. Flows and activities of the CASE.

The NL-OOPS prototype has been designed keeping in mind that it had to be interactive, providing both support for the requirements generation and an explanation capability to allow the analyst to ask specific questions about the model extracted from the requirements.

The main flows and activities of NL-OOPS are represented in figure 2.

NL-OOPS supports the requirements analysis process by generating OO models starting from NL requirements documents which have been pre-processed to produce a 'coherent' set of NL requirements (see section 4.1), which is the input for the full NL analysis realised by LOLITA. The results of the analysis are added to the knowledge base of the NLP system, that is to the LOLITA semantic network, which

https://doi.org/10.1017/S1351324996001337 Published online by Cambridge University Press

is called SemNet (see the next section). To have a representation of SemNet suitable for the OO analysis, all the tokens of SemNet used in this step are rearranged into tables, containing descriptions of nodes and arcs, by a module of NL-OOPS. These tables are then stored in a database, whose primitives are used to implement the OO analysis module. Both LOLITA SemNet subsections and OO models can be visualised as texts or diagrams at different points of the analysis process. In this way better support to an incremental system development is given and the analyst maintains the control of the requirements modelling process.

As for the OO methods, we use the Object Modeling Technique (OMT) (Rumbaugh *et al*., 1991), both because it encompasses virtually all the life cycle phases, and it is also sufficiently representative of the OO approach as not to invalidate the generality of the argument.[6]

3.2 LOLITA as core for an NLP-based CASE

For the NL-OOPS project we are using LOLITA[7], a large scale NLP system built according to the principles of natural language engineering, whose goal is to produce systems which are large in scale, allow easy integration and expansion, are feasible both in terms of speed and of memory, are maintainable, robust and such that the intended users are able and willing to use them (Garigliano *et al*., 1995).

All these features are relevant for the implementation of a CASE tool supporting early system development activities such as requirements definition and anaysis. LOLITA meets most of these criteria, with the noticeable exception of robustness one.

As core for a tool supporting an analyst in requirements analysis, LOLITA is sufficiently robust in many aspects.[8] For instance, about 90% of NL sentences are analysed at various levels of accuracy, depending also on their length, as serious problems arise for sentences with more than 40 words. It follows that the quality of the analysis depends on the text, but usually the resulting output contains most of the information necessary for the modelling activities.

LOLITA[9] is built around a large graph – SemNet, a particular form of conceptual graph (Sowa, 1983) – which holds knowledge that can be accessed, modified or expanded using NL input. SemNet contains more than 100,000 connected nodes, and has been merged with WordNet from Princeton University. A fundamental point of LOLITA is that it moves away as soon as possible from the superficial linguistic structures and reaches deep structures independent of each specific NL language and close to more generic cognitive forms.

[6] A good and compehensive description of the OO approach is given in Graham (1991). For a comparison of the main OO methodologies see, for example, de Champeaux and Faure (1992).

[7] A collection of papers about LOLITA can also be found via anonymous ftp (ftp.dur.ac.uk) or in the www page of the Laboratory for Natural Language Engineering (ftp://ftp.dur.ac.uk/pub/comp-sci/lolita/).

[8] LOLITA is one of only six NLP systems that have participated in all the tasks for the MUC-6 competition (Morgan, Garigliano and the LOLITA Group, 1996).

[9] The current version of LOLITA is mostly written in the pure functional programming language Haskell (Davie, 1992), and it comprises 300 separate modules (approx. 45,000 lines of code, corresponding to about 470,000 lines of an imperative language such as, for example, C) and 1900 grammatical rules.

https://doi.org/10.1017/S1351324996001337 Published online by Cambridge University Press

The core of LOLITA has a *pipeline* structure. As SemNet contains both lexical and semantic information, all its components are linked to it. A simplified diagram of the core is given in figure 3, with its two main facilities: a group of modules used to

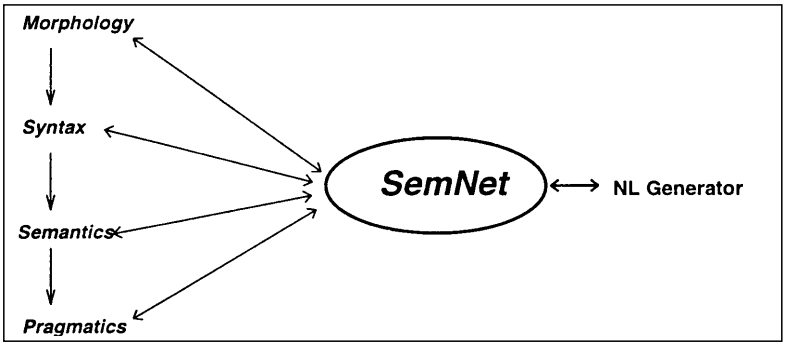


Fig. 3. Diagram of LOLITA *core*.

transform NL text into SemNet and the NL Generation module that converts information represented in SemNet into NL text.

NL documents are morphologically, syntactically and then semantically analysed. Semantic analysis assigns a part of a graph to each construction. This is then passed to the pragmatic analysis, which before adding the new information checks if it is consistent with the rest of SemNet.[10]

Concepts are represented in SemNet by two basic types of nodes: *entity* nodes and *event* nodes. Simple relationships between concepts correspond to arcs in SemNet. More complex relationships are implemented using event nodes, which have a frame-like structure representing the various components of the event: the subject, the action, the object – if the action is transitive – and other information depending on the kind of event.

Each node has a set of *controls* (control variables) and a set of *arcs* (Garigliano *et al.*, 1992).[11]

The most important controls for the OO analysis algorithm are:

rank Gives quantification information: *universal* for general sets; *bounded existential* for instances of a concept where the instance depends on the particular instance of some other universal concept;[12] *individual* for anonymous instances of a concept; *named individual* for named instances of a concept (see, for example, the rank values in figure 5).

type Concepts are sorted into a type lattice. The main values for this type control are *entity*, *relation*, *event* (see, for example, the node in figure 9).

family This control is used to classify nodes into the semantic and pragmatic group

[10] A detailed description of the analysis stages can be found in Morgan *et al.* (1996).
[11] There are about 50 controls and 60 different arcs.
[12] For example, a rank *bounded existential* is associated with hotel for the sentence "In each city there is a hotel".

https://doi.org/10.1017/S1351324996001337 Published online by Cambridge University Press

```
    * event: 96007 *
generalisation_:
  event - 7688 - rank: universal
subject_:
  guest - 95970 - rank: universal
action_:
  call - 11692 - scale: instantaneous
object_:
  hotel - 96064 - rank: universal
time_:
  present_ - 20989 -
date:
  27 March 1995
source_:
  roberto - 19845 - rank: named individual
status_:
  suspended_ - 29025 -
instrument_of:
  event - 96077 - rank: universal
**********************************
event:
  Guests call hotels.
```

Fig. 4. Event node example (text version).

to which they belong. Examples of values for family control are *living*, *human*, *inanimate man-made*, *inanimate*.

Event nodes in LOLITA are classified into four categories:

- static
- cyclic
- dynamic
- instantaneous

depending on the action *scale* control. Static events correspond to unchangeable situations referring to the usual temporal framework. Recurrent situations are related to cyclic actions, as, for example, 'to manage a company'. Otherwise, changes correspond to dynamic events, if they span over a temporal interval (for example, 'to run a race') or to instantaneous events, if a very short time is involved (for example, 'to win a race').

As for the arcs – corresponding to relationships between concepts – a relevant role is played by those used to specify hierarchies (*specialisation*, *generalisation*, etc.) and the event arcs used for representing their components, for example, *subject*, *action*, *object*, etc.

Another arc that is important for the OO analysis algorithm is the *status* arc. As LOLITA's object-based approach requires that each node be representable on its own, event nodes are marked *suspended* until they are definitely stored in SemNet, while event nodes referring to epistemic actions are marked with the status *hypothesis*.

Entity nodes can also have a *size* arc, if quantitative information is needed (for instance, to specify the number of rooms in a hotel).

Other arcs indicate the source of the information, the date on which it has been analysed, and so on.

https://doi.org/10.1017/S1351324996001337 Published online by Cambridge University Press

An event node example of SemNet created for the case study described in section 4.3 is given in figure 4 and a piece of SemNet for the first sentence of the case study – "Sogno is a company which owns and manages hotels" – is given in figure 5.

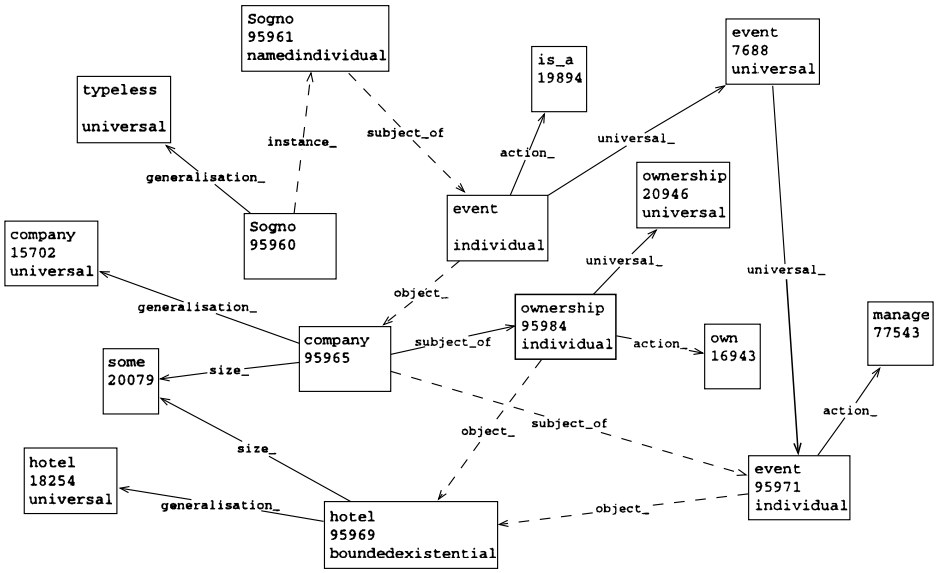


Fig. 5. SemNet fragment.

LOLITA also makes use of prototypical events, which define restrictions on events by providing 'templates' for them, for example, imposing the criterion that in an 'ownership' event only humans can take the subject role (Morgan *et al.*, 1996).

A point to underline here about LOLITA knowledge representation in SemNet is that it fulfills an essential requirement for an NLP-based CASE tool: if no restriction is given for the language in the documents to be processed for the OO requirements analysis, a semantic representation that is independent from the grammatical form of the NL requirements is needed (see also section 2.2).

For example, the sentences "John kissed Mary" and "John gave a kiss to Mary" correspond to the same SemNet nodes. In particular, in this case the normal form is the first one, and it is not necessary to use the object node *kiss* for their disambiguation. In contrast, for "Guests reserve a room" and "Guests make a reservation", LOLITA uses an object node *reservation*, because *reservation* exists independently of the event related to these sentences, that is *reservation* continues to exist after the reservation process has finished.

At a different level, the feasibility of the NL-OOPS project was based on the capabilities of the LOLITA system to support many applications in different domains. The facilities provided by the core of LOLITA have been used to build a number of prototype applications, including, for example, an information-extraction application that fills in pre-defined templates according to the content of newspaper articles, an intelligent language tutor interface and a dialogue module.

https://doi.org/10.1017/S1351324996001337 Published online by Cambridge University Press

4 OO requirements analysis using NL-OOPS

To develop a CASE prototype for generating OO requirements from NL documents, it is necessary to address two important areas of requirements analysis:

- Detection of ambiguity, inconsistency and incompleteness in the documents used to elicit the requirements, defining a 'coherent' set of requirements.
- Determination of the entities in the problem statement and how they are related, building adequate OO models.

4.1 Requirements pre-processing

Requirements analysis consists of two main activities:

- Identification of requirements
- Modelling of requirements.

To identify the user requirements for an application, an analyst can use various techniques: interviewing, group meetings and discussions, questionnaires, and so on. The resulting documentation is usually stored in a project depository in a narrative format. Before starting requirements specification, the analyst has to check these documents: they may contain ambiguities, inconsistencies, redundancies, etc.

To support this step, as the core of NL-OOPS, LOLITA could intervene by analysing input texts at several levels in order to correct, select and normalise them, so as to generate a requirements document, which in turn is the starting point for the conceptual analysis. There are functionalities in LOLITA which allow text correction and completion, elimination of style difference (possibly even differences in the actual language used)[13] and ambiguity resolution (at grammatical, semantic, pragmatic, discourse and dialogue levels). All these phenomena occur very frequently in real and serious text, and require a real general purpose base NL system; which is why so many other problem-specific systems use a pseudo-natural language.[14]

An application for LOLITA similar to requirements pre-processing is under development in the judicial domain. A number of statements have to be checked to identify correspondences, contradictions, etc. As for the NL-OOPS project, at the moment we are working with single original NL requirements texts: they are analysed by LOLITA to produce the correspondent SemNet structures, which in turn are the input of the OO module described in the next section.

4.2 The OO analysis algorithm

The second part of the process starts from the SemNet, in which some redundancies and inconsistencies have already been solved or flagged for the user's attention, and involves the construction of OO models. At the end of requirements pre-processing,

[13] For instance, switching the style parameters for the generator, passive sentences could be transformed into active ones.

[14] These issues are not described further here, because they are mostly linguistic and readers may refer to the papers about LOLITA for more information and further references (see, for example, Long and Garigliano (1994)).

https://doi.org/10.1017/S1351324996001337 Published online by Cambridge University Press

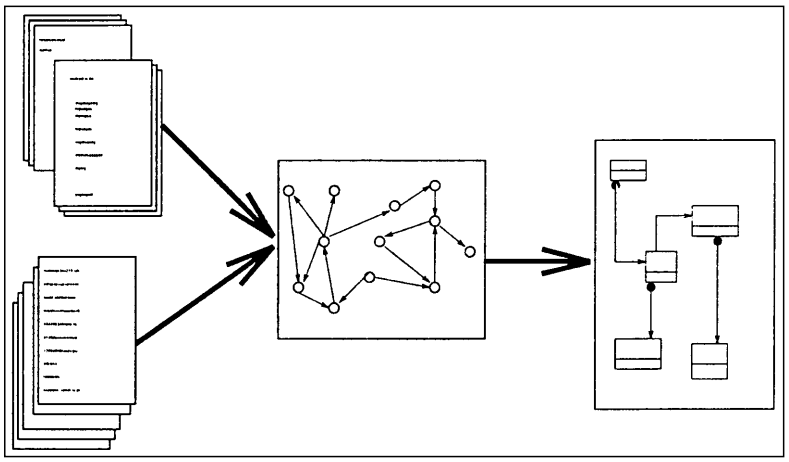


Fig. 6. Requirements elicitation and modelling.

all the information is in SemNet (see figure 6). The OO analysis algorithm has to extract 'elements' – objects, associations, etc. – that are 'relevant' for OO modelling.

OO analysis contains, in some order, the following activities (Jacobson *et al.*, 1992):

- finding the objects
- organising the objects
- describing how the objects interact
- defining the operations of the objects
- defining the objects internally.[15]

The central problem is connected to the steps involving identification activities. The OO analysis algorithm implemented in NL-OOPS addresses these steps. The algorithm has to translate LOLITA's representation of the NL requirements into an adequate Object model.

In this section the abstract structure of the algorithm is given.[16] The algorithm steps (figure 7) are explained in more detail in the next section using a case study, so that it is possible to follow its application.

The OO analysis algorithm is based on two different phases: the first phase is *context independent* and the second *context dependent* (figure 7).

4.2.1 The context independent phase

In this phase, SemNet nodes that are not useful for a domain analysis are filtered and a list of candidate object classes is produced. Some of the nodes are deleted, while others are marked, for further analysis in the second part of the algorithm.

The input of this phase is the table of all the nodes in SemNet used by LOLITA to process the requirements text. The table contains both new nodes created to

[15] With regard to the 'Object Model' for the OMT (Rumbaugh *et al.*, 1991), an analysis of how the steps needed can be achieved using LOLITA is given in Mich and Garigliano (1994).

[16] See also Mich and Garigliano (1995a).

https://doi.org/10.1017/S1351324996001337 Published online by Cambridge University Press

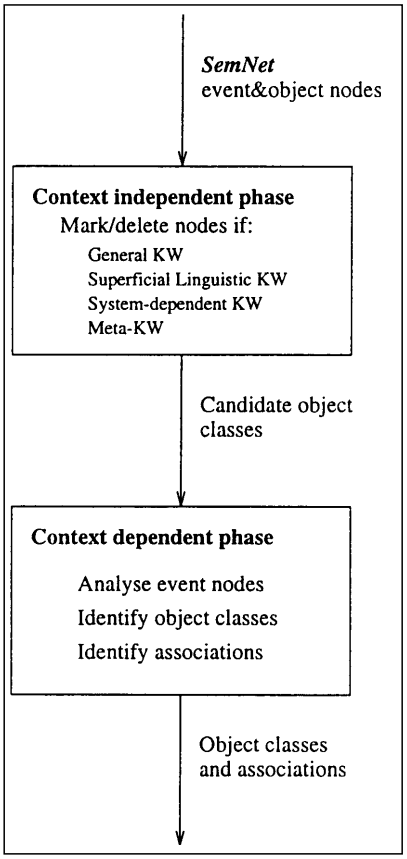


Fig. 7. OO analysis algorithm.

disambiguate the requirements, and existing nodes used to connect them to the rest of SemNet. The output is a list of candidate object classes, which is a subset of the entity nodes.

In the context independent phase we then mark or eliminate, depending on the case, the nodes corresponding to:

General Knowledge General knowledge related nodes are not useful in this phase of the requirements analysis, because it is a domain analysis, and nodes such as *entity* or *something happens* cannot add information to the OO model.

The criteria used to filter nodes related to general knowledge are the following:

- Delete nodes at the highest levels of hierarchies, for example, *groups*, *something*, *things*, etc.
- Delete nodes used by LOLITA to implement its spatial logic, which is quite sophisticated.
- Delete nodes used to process temporal knowledge, corresponding to the temporal logic implementation of LOLITA.

https://doi.org/10.1017/S1351324996001337 Published online by Cambridge University Press

- Mark existing nodes whose type is not *entity*, for example, nodes with type *attribute*, *relation*, etc.

Superficial Linguistic Knowledge The second group of filtered nodes are related to the superficial structure of knowledge, and it contains nodes such as *pair* or *set* created to resolve plural anaphorical references, and the internal events[17] whose action is not language isomorphic, which have to be eliminated together with their subject and object nodes if they do not occur in other events.

System-dependent Knowledge Nodes corresponding to the particular way in which LOLITA represents some concepts in respect to the other NLP systems are eliminated, as they cannot be part of a requirements model. A typical example for this group is nodes created by LOLITA to manage proper nouns. These are entity nodes with a *typeless generalisation* and an *instance* node with rank *named individual* (see, for example, figure 10).

Meta-Knowledge LOLITA SemNet is built by attaching to each node the source of the information and the date on which it was created. This information could be used to guide model building, but these nodes correspond to meta-properties, and therefore do not have to appear in the candidate classes list.[18] Another kind of meta-knowledge node are those connected to the 'technical goals' of the application under development. These nodes are related to the goals of the application and not to the problem domain. They can be identified by looking for events having *status hypothesis* and involving nodes instantiated independently on the use of the information system.

Some consistency rules are then applied to eliminate isolated elements of the list, for example, event nodes whose subject or object has been deleted.

4.2.2 The context dependent phase

The **second phase** of the algorithm performs a *context-dependent* analysis, and uses LOLITA's events classification to extract classes, associations, attributes (or values of attributes) and operations (or arguments of operations) from the list obtained in the first part of the algorithm.

The main steps of this phase are the following:

1. Events analysis
   - re-classify events
   - associate events to candidate object classes.
2. Object identification
   - extract object classes
   - add attributes to object classes
   - add methods to object classes.

[17] Internal events are used by LOLITA to disambiguate expressions like 'information system', or to represent temporal and spatial concepts.

[18] Real NL texts also bring problems of inconsistency and omission management. Using LOLITA, uncertainty is managed mainly through a Source Control System, based on a method which allows the evaluation of both information and sources together (Garigliano and Bokma, 1987; Mich *et al.*, 1994).

https://doi.org/10.1017/S1351324996001337 Published online by Cambridge University Press

3. Association identification
   - identify association
   - add association multiplicity.

Object and association identification have to be done simultaneously, so these steps are described together.

Events Analysis Event nodes have been stored in SemNet as static, cyclic, dynamic or instantaneous, depending on an absolute temporal framework. As an Object model describes the static structure of the objects and their relationships, a preliminary step is needed in which SemNet events created for a requirements text are *re-classified* according to the current temporal framework. At one extreme, all the events would be dynamic on the universe-age time-scale; at the other extreme, only static events in the time needed to take a picture. For instance, even a typical instantaneous action such as "win" in "John won a race" becomes static in "John has improved, because he has won three races in the last six months"; as in the second sentence, the change is related to the improve action, against the background of constancy given by the three victories.

At the end of this re-classification process, to each candidate object class are *associated* all the events in which it is involved, usually as a subject, object or goal of the events.

Object and association identification Due to the set-based nature of SemNet, candidate object classes are hierarchically connected, so each class is involved in at least a *generalisation* (or *generalisation_of*) relationship (see, for example, the hierarchy in figure 8). For each candidate object class we have to establish its role in the final Object model. The main assumptions are that:

- Static and cyclic events are related either to class associations or to attributes for object classes.
- Dynamic and instantaneous events correspond to operations in the object classes.

Classes involved in only one static or cyclic event correspond to *attribute* values, and classes involved in only one dynamic or instantaneous event are used to specify class *operations*.

As regards an object class, a static and a dynamic characterisation have to be provided,[19] and given the previous rules, *object classes* have to be contained both in static/cyclic events in dynamic/instantaneous events.

Assuming that general problem knowledge is usually given as known in a requirements document, that is, not all the information we need is in the text, we can relax this condition, looking also for classes with more dynamic/instantaneous events, and use LOLITA general knowledge stored in SemNet to complete the Object model with missing associations. In other words, if we require the final Object model to be connected, we can explore SemNet hierarchies, starting from the isolated object classes in the Object model until all of them are connected using existing nodes as close as possible to them. Applying these criteria to each class in

[19] See, for example, de Champeaux and Faure (1993).

https://doi.org/10.1017/S1351324996001337 Published online by Cambridge University Press

the candidate object classes hierarchies, we obtain the class hierarchies for the Object model.

A further analysis of the events is necessary to add attributes and operations to the object classes so far included in the model. For example, an operation identified using a dynamic event has to be added to a class which is a subject or object in the event.

Class *instances* can be identified using the SemNet rank *named individual*: subject nodes in static events with action *is_a* and rank *named individual* are instances of the class playing the *object* role in the event (an example is given in figure 10). An attribute *name* is then added to the class.

At a different level of abstraction, static events with the action *be* and the arc *location* instantiated identify the value for an attribute *position* for an object class.

Other nodes, marked in the first phase of the algorithm, can be used to refine the associations in the model. The arc *size* of an object node gives the information we need to add association multiplicity, that is, to specify how many instances of one object class may relate to a single instance of an associated object class.

Static events with an action *has_part* or *contain* are represented as an aggregation association if the elements involved are classes; otherwise they are used to identify attribute values.

4.3 A case study

The starting point of the conceptual analysis of the case study is the following text on a Hotel Reservation Information System:

Sogno is a company which owns and manages hotels. It needs to modify its information system. The new system should improve the reservation services in order to get a better level of integration, which would improve the guest service. There are three hotels in the chain. All the hotels are located in the same area. The first hotel, Bolzano, is near the cathedral, in the centre of the town. It is a four-star hotel. It has 15 double rooms and 5 single rooms. Each room contains a bath and a small balcony. The hotel Bolzano has a restaurant, a private car park and a garden. The hotel Koenig is the second hotel owned by Sogno. It is not far from the railway station. It is a big new three-star hotel with 55 rooms. The third hotel, Sirena, is by the sea. It has 30 rooms. All rooms have a balcony. 10 rooms look out onto the sea. The hotel Sirena has a garage, a disco and two restaurants. Guests can make a reservation by calling the hotels or the central office of the company, Sogno. Guests are asked about the arrival-departure time and the type of room. They should leave the name, the telephone number and the way of payment. If there isn't a vacant room in the hotel called by guests, the receptionist should propose a room in another hotel of the chain. If they cancel the reservation, they lose the deposit.

It is a complex text, with style changes, containing most of the classical problems of NLP, for example, anaphorical references, words possibly having multiple meanings, structural ambiguity, etc. This text or *problem statement* is the input for the full NL analysis achieved by LOLITA.[20] After the analysis, we have a table containing about 250 nodes, almost the same as the number of words in the problem statement.

Applying the first **context independent phase** of the OO analysis algorithm described in the previous section, we obtain a list of candidate classes containing 83 nodes,

[20] The complete output obtained is reported in Mich and Garigliano (1995b).

https://doi.org/10.1017/S1351324996001337 Published online by Cambridge University Press

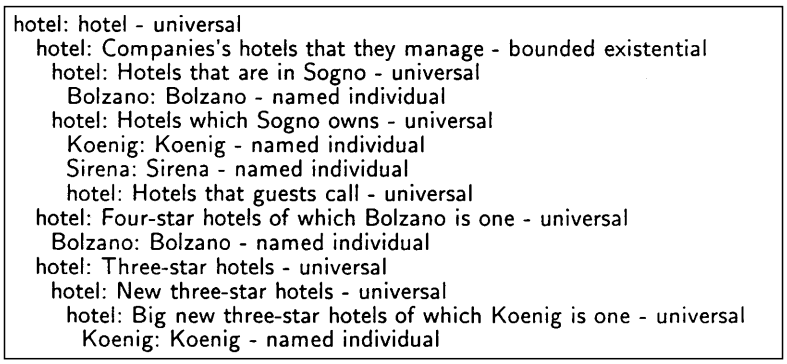


Fig. 8. Hotel hierarchy.

organised in 26 hierarchies (see, for example, the hotel hierarchy in figure 8). Nearly 1/5 of the nodes in the original list have been deleted, and the others are marked for use in the second phase of the algorithm.

The most important point is that all those classifications are obtained automatically, using domain-independent rules. Notice that, for example, the `Koenig` hotel appears twice in the hierarchy, the first time as an instance of the `Hotels which Sogno owns` and the second time as an instance of the set containing `Big new three star-hotels`. The two sets `Hotels that are in Sogno` and `Hotels that Sogno owns` should be the same set. LOLITA could not unify these two objects because to disambiguate that particular meaning of 'are', it is necessary to know that a company can be seen, abstractly, as a set of all its properties, and this is a piece of domain dependent knowledge that happens not to be present.

Using the criteria related to *meta-knowledge*, some of the events are recognised as 'technical goals'. In particular, the algorithm is able to recognise as such the following statements in the original document: "`It needs to modify its information system`" and "`The new system should improve the reservation services in order to get a better level of integration, which would improve the guest service`".

The **context dependent phase**, starting with *events analysis*, uses as upper and lower events for the temporal framework the events "`Sogno is a company that manages its hotels`" and "`Guests call hotels`", respectively. Event reclassification for this text does not affect the application of the following algorithm steps.

Object and association identifications produces a first version of the Object model, in which we have as object classes, candidate classes involved both in static and dynamic events – `Company`, `Hotel`, `Room` – and candidate classes with many dynamic events, such as `Guest` and `Reservation`, that have been further analysed using general knowledge in SemNet in order to obtain a *connected* Object graph.

In particular, the associations between `Reservation` and the other object classes are derived by visiting SemNet hierarchies to find out an event, in this case the prototype event 'Reservation', that unifies them. In more detail, the prototype event for Reservation is the following:

https://doi.org/10.1017/S1351324996001337 Published online by Cambridge University Press

Some humans book some services from some providers

A post-condition event of this prototype is that:

Some humans have a reservation

where reservation in this case is a specialisation of *authorisation*, *permission*, *mandate*.

The first association extracted from SemNet is the one between `Reservation` and `Guest`, as we find that *guest* in SemNet is a specialisation of *humans*.

The action *reserve*, used for the requirements sentence `Guests can make a reservation by calling the hotels or the central office of the company, Sogno`, is connected to the action *book* in the prototype with a *similar* arc, as we can see from the output obtained using the *lc* command of LOLITA, which shows the control information associated with a node (figure 9).

```
77401: To book. (= To reserve. , To hold. =) To register. )
  type: relation
  emotional value: indifferent
  language: english
  relation type: transitive
```

Fig. 9. Node controls example.

Given that *hotel rooms* is a specialisation of *hired rooms*, which in turn is a specialisation of *goods use*, that is a specialisation of *services*, while *hotels* is a specialisation of *providers*, following the prototype and its post-condition, we have that in the Object model the object class `Reservation` is connected with the `Room` and `Hotel` classes.

Static events analysis produces two associations, i.e. `owns` between `Company` and `Hotel`, and an aggregation association `has` been `Hotel` and `Room`.

Searching for the instances of the classes in the model, Sogno is recognised as an instance of the class `Company`, while Bolzano, Koenig and Sirena are instances of the class `Hotel`. To both `Company` and `Hotel` classes an attribute name is then added. In figure 10 the event used to assign Sogno to the object class `Company` is given.

```
    * event: 95972 *
universal_:
  event - 7688 - rank: universal
subject_:
  Sogno - 95961 - rank: named individual
action_:
  is_a - 19894 - scale: static
object_:
  company - 95965 - rank: universal
  ...
**********************************
event:
  Sogno is a company that manages its hotels.
```

Fig. 10. *Instance* event node.

https://doi.org/10.1017/S1351324996001337 Published online by Cambridge University Press

```
    * event: 95992 *
universal_:
  event - 7688 - rank: universal
subject_:
  Sirena - 95995 - rank: named individual
action_:
  be - 78091 - scale: static
location_:
  position_ - 95979 - rank: individual
  ...
**********************************
event:
Sirena is by a sea.
```

Fig. 11. *Position* event node.

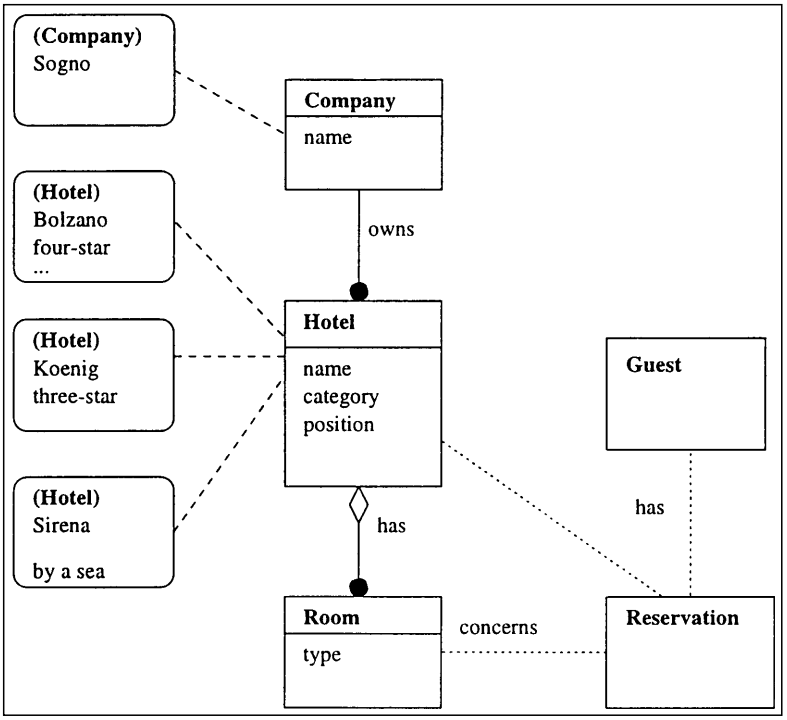


Fig. 12. Object model.

In the static event `Sirena has restaurants, a disco and a garage` – with action *has_part* – restaurant, disco and garage are only attribute values because they are all involved only in it. The names of the attributes can be searched in LOLITA's SemNet comparing the entity nodes related to each class instance. For example, we find that Hotel Sirena has a garage, and Hotel Bolzano has a car-park: garage and car-park in SemNet have a common generalisation – `parking` – that can be used to name the attribute. In the same way we have single and double rooms as an attribute `type` for the room class. Also, `four-star hotels` and `three star hotels` correspond to values of an attribute `category` for the class `Hotel`.

https://doi.org/10.1017/S1351324996001337 Published online by Cambridge University Press

Other static event nodes are used to add an attribute *position* to the `Hotel` object class. In figure 11, an example node representing the *position* of `Sirena Hotel` is given.

At this point it is possible to draw the Object model obtained by applying the second part of the OO analysis algorithm. This model is given in figure 12, using the notation of the Object Modeling Technique (OMT).

Note that the associations extracted from SemNet applying the rule that the Object models have to be connected, are represented with dotted lines.

Using marked nodes related to the *size* arc in LOLITA entity nodes, association multiplicity is obtained. For example, from the node in figure 13 we obtain that the

```
    * hotel: 95969 *
generalisation_:
  hotel - 18254 - rank: universal
size_:
  some - 20079 -
status_:
  suspended_ - 29025 -
object_of:
  event - 95971 - rank: individual
  ownership - 95984 - rank: individual

**********************************
object:
  Companies's hotels that they manage.
```

Fig. 13. Entity node.

association between `Company` and `Hotel` is *1 to many*. The same happens for the association between `Hotel` and `Room`.

Candidate classes useful for identifying operations are involved in only one dynamic event, whose subject or object is the class in the Object model to which an operation is added. For example, `deposit` only appears in the dynamic event `Guests lose a deposit`: an operation `lose` is then added to the class `Guest` with argument deposit. For the object classes `Guest` and `Reservation`, we have the operations in Table 1.

4.4 Implementation issues

The main technical requirements for NL-OOPS were the following:

- The tool had to be modular and integrated with the LOLITA environment.
- Its interface had to support the analyst in a process that requires complex sessions in which many applications have to be used at the same time.
- It had to have a high level of portability, to allow its possible integration with other CASE tools supporting the lower phases of the life cycle.[21]

The first version of NL-OOPS has been implemented in the UNIX operating system, in particular Solaris on Sun Workstations and Linux on the PC architecture.

The modules for the processing of LOLITA output and the creation and

[21] The complete design for NL-OOPS is described in Gaio and Mich (1995).

https://doi.org/10.1017/S1351324996001337 Published online by Cambridge University Press

Table 1. *Operations*

Class	Operation
Guest	lose_deposit call_hotel call_central_office leave_name leave_telephone_number leave_payment_way give_arrival_departure_time give_room_type
Reservation	make cancel

management of the database with SemNet modes are written in Tcl/tk (Ousterhout, 1993).

The user interface has been implemented using the tk graphical toolkit, which allows the quick development of graphical interfaces with a motif-like look and feel.

As for the OO analysis algorithm, it has been implemented using the database primitives for selecting and marking the nodes. New rules can easily be added by the user.

Graph visualisation is made using an existing package – graphed – (Himsolt, 1994), which allows both the visualisation of SemNet fragments and of OO requirements models.

An important concept for the tool is that of 'project': all the files and documents associated with an application are maintained in a project repository, facilitating both documentation and traceability.

5 Related work

In recent years, a certain number of NLP-based CASE tools have been realised in the areas of both software and information system development, and in the construction of conceptual schemas for databases.

As far as the linguistic theories adopted are concerned, they range from the syntactic linguistic criteria proposed by Abbott (1983) to works that attempt to realise a more sophisticated text analysis using variants of the Case Grammar by Fillmore (1968); however, even in the early work by Abbott, the importance of a semantic analysis and of a real-world knowledge is recognised (under the general heading of 'understanding of the problem domain').

The basic approach by Abbott is developed further in a paper by Booch (1986) which – together with the one by Abbott – is the most quoted with regard to the use of 'linguistic criteria' for the development of models of the external world in terms of interconnected objects.

A process of incremental construction of software modules starting from OO

https://doi.org/10.1017/S1351324996001337 Published online by Cambridge University Press

specifications obtained from informal NL specifications is described in Saeki *et al.* (1989), in which the transformation process takes place interactively and the analyst's role is very important, since the system is able to extract verbs and nouns from informal specifications automatically, but not to determine which words are important for the construction of a formal specification. With respect to the approach by Abbott, the role of verbal patterns is underlined. The NL specifications are analysed separately using two dictionaries, one for nouns and another for verbs, which are classified in a way related to the OO model used for formal specifications. In this way, it is possible to compensate for the lack of a real NLP system: this, however, is achieved at the cost of a considerable loss of information.

To limit this problem, Bailin (1989) proposes a different solution, based on 'filtering' the requirements text to build a requirements database, so as to simplify the search for nouns and noun-phrases. However, no details are given on how to achieve this simplification. As described in this paper, NL-OOPS can also constitute a strong support tool in the initial phase thanks to LOLITA's 'text normalisation' capabilities. Another significant aspect in Bailin's work is the proposal to use Data Flow Diagrams (DFDs) to identify entities in the problem domain (although the author maintains that DFDs are not really part of OO specifications, and can only be used as an intermediate representation). It should, however, be noted that other OO methodologies, such as the Object Modeling Technique which we adopt in this paper, contemplate the use of DFDs as models for the functional aspects. We refer to the paper by Wieringa (1991) for more information on the use of DFDs as part of OO analysis.

A well-known method for the construction of conceptual schemas for relational database development is NIAM (Dunn and Orlowska, 1990). To obtain a domain-independent system, NIAM uses the result of the context-based NLP disambiguation. The analyst interacts with the system on a particular Universe of Discourse, utilising a subset of the NL which is similar to that produced by the LOLITA normaliser: just like other similar NLP approaches, this one lacks in generality, as it was built with that particular application in mind.

Among the tools devoted to the early stages of development support for systems based on AI-style knowledge acquisition, the Requirements Apprentice (RA) (Reubenstein and Waters, 1991) should be mentioned. To avoid the problems related to the complexity of the free language which the user is likely to employ, RA imposes a more restricted command language: this is an approach that may help the analyst, but not the requirement provider. On the other hand, this approach is instrumental in allowing RA to make significant use of domain knowledge, expressed in the 'Clichè Library', a declarative repository of information about domains of particular interest.

The expert system ALECSI-OICSI (Cauvet *et al.*, 1991) allows the user to express the requirements both in NL (French) and in graphic form, and the system builds conceptual structures. ALECSI, just like LOLITA, uses a semantic network to represent the domain knowledge (Rolland and Proix, 1992). The NL interpretation is based on the case semantics approach (Fillmore, 1968), according to which the meaning of a sentence can be extracted from the meaning of the verb and the

https://doi.org/10.1017/S1351324996001337 Published online by Cambridge University Press

recognition of the connected 'cases'. By applying the case concept to the sentence level as well, it is possible to obtain a top-down approach to the interpretation of complex phrases. Furthermore, since the linguistic patterns are independent of any particular model building technique, they can be used with any development methodology. OICSI is based upon the REMORA methodology, which identifies four basic concepts: objects, actions, events and constraints, and four corresponding types of nodes in the semantic net used to implement the conceptual schema needed. There are five types of arcs, representing respectively a relationship between two objects, an action modifying an object, an event triggering an action, an object changing state, and a constraint on a node. Validation phase support is also present, providing for the user a description of the application domain as close as possible to the one initially given to the system. NL sentences are built using a Chomskian approach (Chomsky, 1968): the *deep structures* – corresponding to the semantic content of the sentences – are defined as sets of nodes and arcs of the semantic network describing an entity or an event, and then transformed into *surface structures* using a generative grammar.

For a different validation support tool approach, in which the NL generator is built from a subset of Hobbs coherence relationships, see Dalianis (1992).

An extension of the case grammar is also used in the requirement analysis support system described in Belkhouche and Kozma (1993). The prototype automatically performs a 'semantic case analysis' on sentences based on a subset of English (Analyst Constrained Language: ACL). It uses a form of predicate calculus, rather than a semantic net formalism. Its system architecture is based on four modules, three of which constitute the NLP system, while the fourth is the OO Analysis System (OOAS). The OOAS takes as an input from the NLP modules an annotated tree and case frames. From these two structures, roles, relationships and inheritance rules are derived, which are represented in relational tables. These tables are then used for requirement elicitation. To identify which noun-phrases from the input text represent candidate objects, heuristics are proposed for the analyst to apply. It is also claimed that, as a by-product, the prototype allows the automatic identification of ambiguities, inconsistencies, etc. Compared with the ALECSI-OICSI system, this is a less sophisticated system: rather than use a robust NLP system, it restricts the range of possible input texts, and takes advantage of the fact that the style in technical documents is simpler.

The development of a requirements sub-language – a restricted English natural language, which limits the vocabulary, the syntax and the semantics – is described in Cyre (1995). The language has been designed for an Automatic Specifications Interpreter (ASPIN) system in the domain of digital systems. Restricting both the language and the domain, the system performs an automated requirements analysis using a set of *basis graphs*, simple conceptual graphs which correspond to word sense definition in a conventional dictionary.

Another system based on a subset of English is the Attempto system (Fuchs and Schwitter, 1995, 1996). Requirements specifications are written in 'Controlled English', a controlled NL with a restricted grammar and a domain vocabulary.

https://doi.org/10.1017/S1351324996001337 Published online by Cambridge University Press

Some possible applications of the NL techniques to support the pre-processing activities for writing 'unambiguous formal specifications in English' are described in Macias and Pulman (1993).

A more recent project (Burg and Riet, 1995) combines a linguistically based specification language with graphical modelling techniques, and uses a lexicon to support the modelling process.

As we have seen from this brief survey, there is widespread agreement on the need to move towards NLP modules as part of a requirements analysis tool: however, attempts to date have often been hampered by the need to build *ad hoc*, small NLP systems. The fundamental common drawbacks may manifest themselves as a need for a pseudo-language, or heavy domain dependency, or excessive reliance on user interaction, but can all be traced by to the lack of proper NL facilities (at the analysis, reasoning and generation stages). In fact, it was our contention that a successful approach requires the use of a full NLP system, built to tackle the full range of NL problems. The use of LOLITA for the NL-OOPS project represents a step in this direction.

6 Conclusions

In this paper we have described NL-OOPS, an NLP-based CASE tool that supports natural language analysis by extracting the objects and their associations for use in creating Object models. Using the NLP system LOLITA as a core for the CASE tool, it is possible to help the analyst both with the elicitation and the modelling of user requirements. The OO module is based on an algorithm which filters and translates the object and event nodes of LOLITA's SemNet. The application of the algorithm has been illustrated in this paper with a case study. The resulting Object model is a sound model of the given problem statement. However, if we want to address the complexity of a real systems analysis that typically makes use of many NL documents, two main issues are still to be investigated further. The first group of problems is related to the pre-processing of the requirements documents in order to detect ambiguities, inconsistencies and omissions as soon as possible, producing if necessary a revised NL version of the requirements. These problems will be further investigated by the LOLITA research group. The second problem which arises as the size of the project increases is connected with the selection of the 'relevant' objects. Its solution requires the definition of semantic information measures and the tuning of the rules used to identify object classes from the list of candidates. Future developments of NL-OOPS – a prototype in an advanced state of implementation – concern the use of a speech recognition interface and the development of a more suitable tool for the visualisation of the graphs. Furthermore, as the linguistic issues that motivate the algorithm are sufficiently general and to a great degree language independent, it could be possible to have a tool able to process requirements texts in different languages.

https://doi.org/10.1017/S1351324996001337 Published online by Cambridge University Press

Acknowledgements

I would like to thank the LOLITA group for the invaluable opportunity to use their system and for their crucial support. My grateful thanks also to the research group of my department, which contributed extensively to the design and implementation of NL-OOPS. Finally, a special thank you to the reviewers for their valuable comments and advice.

References

Abbott, R. (1983) Program design by informal English descriptions. *Comm. ACM* **26**(11): 882–894.

Bailin, S. (1989) An object oriented requirements specification method. *Comm. ACM* **32**(5): 608–623.

Belkhouche, B. and Kozma, J. (1993) Semantic case analysis of informal requirements. In S. Brinkkemper and F. Harmsen, (eds), *NGCT'93*, *4th Workshop on the Next Generation of CASE Tools*. Memoranda Informatica 93-32, Universiteit of Twente, Netherlands, pp. 163–182.

Booch, G. (1986) Object-oriented development. *IEEE Trans Software Engineering* **12**(2): 211–221.

Booch, G. (1994) *Object-Oriented Analysis and Design with Application*. Redwood City, CA Benjamin/Cummings.

Burg J. and van de Riet, R. (1995) COLOR-X: linguistically-based event modeling: a general approach to dynamic modeling. *In* K. Iivari and J. Lyytinen and M. Rossi (eds), *6th International Conference on Advanced Information Systems Engineering*, LNCS, Springer-Verlag.

Cauvet, C., Proix, C. and Rolland, C. (1991) ALECSI: An Expert System for Requirements Engineering. *CAiSE'91*, *3rd International Conference on Advanced Information Systems Engineering*, *LNCS* 498, Springer-Verlag, pp. 31–49.

Chomsky, N. (1968) *Language and mind*. New York: Harcourt, Brace & World.

Cyre, W. (1995). A requirements sublanguage for automated analysis. *Int. J. Intelligent Systems* **10**(1): 665–689.

Dalianis, H. (1992). A method for validating a conceptual model by natural language discourse generation. *In* P. Loucopoulos (ed), *CAiSE'92*, *4th International Conference on Advanced Information Systems Engineering*, *LNCS* 593, Springer-Verlag, pp. 425–444.

Davie, A. J. (1992) *An Introduction to Functional Programming Systems using Haskell*. Cambridge, UK: Cambridge University Press.

de Champeaux, D. and Faure, P. (1992) A comparative study of object-oriented analysis. *J. Object Oriented Programming* **5**(1): 21–32.

de Champeaux, D. and Faure, P. (1993) *Object-Oriented System Development*. Reading, MA: Addison-Wesley.

Dunn, L. and Orlowska, M. (1990) A natural language interpreter for construction of conceptual schemas. *In* B. Steinholtz, A. Solvberg and L. Bergman (eds), *CAiSE'90*, *2nd Nordic Conference on Advanced Information Systems Engineering*, *LNCS* 436, Springer-Verlag, pp. 175–193.

Fillmore, C. (1968) The Case for Case. *In* E. Bach and R. Harms (eds), *Universal Linguistics*. Chicago, IL: Rinehart and Winston, pp. 1–90.

Fuchs, N. E. and Schwitter, R. (1995) Specifying logic programs in controlled natural language. *In CLNLP'95*, *Workshop on Computational Logic for Natural language Processing*. Edinburgh, UK.

Fuchs, N. E. and Schwitter, R. (1996). Attempto controlled English. *CLAW'96*, *The First*

https://doi.org/10.1017/S1351324996001337 Published online by Cambridge University Press

International Workshop on Controlled Language Applications, Katholieke Universiteit, Leuven.

Gaio, L. and Mich, L. (1995) Design of a CASE based on the NLPS LOLITA for information systems analysis. Technical Report 5, University of Trento (in Italian).

Garigliano, R. and Bokma, A. (1987) A model for learning by source control. *In* B. Bouchon and R. Yager (eds), *Uncertainty in Knowledge Based System*, *LNCS* 286. Springer-Verlag, pp. 163–170.

Garigliano, R. and Nettleton, D. J. (1997) Neo-pragmatism. *In* The LOLITA Group (eds), *The LOLITA Project: the First Ten Years*, Vol. 3. Springer-Verlag (to appear).

Garigliano, R., Boguraev, B. and Tait, J. (1995) Editorial, *Journal of Natural Language Engineering* **1**(1): 1–7.

Garigliano, R., Morgan, R. and Smith, M. (1992) LOLITA. Technical Report 12, University of Durham.

Graham, I. (1991) *Object Oriented Methods*. Wokingham, UK: Addison-Wesley.

Himsolt, M. (1994) *Graphed user manual*. Innstraße 33, Passau, Germany.

Jacobson, I., Christerson, M., Jonsson, P. and Oevergaard, G. (1992) *Object-Oriented Software Engineering. A case driven Approach*. Reading, MA: Addison-Wesley.

Long, D. and Garigliano, R. (1994) *Reasoning by Analogy and Causality: Model and Applications*. Chichester, UK: Ellis Horwood.

Macias, B. and Pulman, S. (1993) Natural language processing for requirements specifications. *In* F. Redmill and T. Anderson (eds), *Safety-critical systems: Current issues*, *techniques and standards*. London, UK: Chapman & Hall, pp. 67–89.

Meyer, B. (1988) *Object-Oriented Software Construction*. Englewood Cliffs, NJ: Prentice-Hall.

Mich, L. and Garigliano, R. (1994) A linguistic approach to the development of object oriented system using the NL system LOLITA. *In* E. Bertino and S. Urban (eds), *ISOOMS AICA94 International Symposium on OO Methodologies and Systems*, *LNCS* 858. Springer-Verlag, pp. 371–386.

Mich, L. and Garigliano, R. (1995a) Design of an object extraction algorithm from natural language requirements. *Proceedings AICA*, *Associazione Italiana Calcolo Automatico*, *Cagliari*, Cagliari, Italy, pp. 55–62 (in Italian).

Mich, L. and Garigliano, R. (1995b) From natural language requirements to object oriented specification: a case study. Technical Report 4, University of Durham.

Mich, L. and Giuliani, M. (1995) From natural language to object oriented requirements: an annotated bibliography. Technical Report 4, University of Trento.

Mich, L., Fedrizzi, M. and Garigliano, R. (1994) Negotiation and conflict resolution in production engineering. *IPMU'94*, *5th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, Paris, France, pp. 412–417. (Also published in *Fuzzy Logic and Soft Computing*, B. Bouchon-Meunier, R. R. Yager Lofti Z. (eds), World Scientific, 1995, pp. 181–193.)

Morgan, R., Garigliano, R. and the LOLITA Group (1996) University of Durham: Description of the LOLITA system as used in MUC-6. *Proceedings of the Message Understanding Conferences*. San Mateo, CA: Morgan Kaufmann.

Ousterhout, J. K. (1993) *Tcl and the Tk toolkit*. Reading, MA: Addison-Wesley.

Reubenstein, H. and Waters, R. (1991) The Requirements Apprentice: automated assistance for requirements acquisition. *IEEE Trans. Software Engineeering* **17**(3): 226–240.

Rolland, C. and Proix, C. (1992) A natural language approach for requirements engineering. *In* P. Loucopoulos (ed), *CAiSE'92*, *4th International Conference on Advanced Information Systems Engineering*, *LNCS* 593. Springer-Verlag, pp. 257–277.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W. (1991) *Object-Oriented Modeling and Design*. Englewood Cliffs, NJ: Prentice-Hall.

Saeki, M., Horai, H. and Enomoto, H. (1989) Software development process from Natural language specification. *ICSI'89*, *International Conference on Software Engineering*, ACM, pp. 64–73.

https://doi.org/10.1017/S1351324996001337 Published online by Cambridge University Press

Smith, M., Garigliano, R. and Morgan, R. (1994) Natural language generation in LOLITA: an engineering approach. *2nd International Workshop on Natural Language Generation,* Maine, USA.

Sowa, J. (1983) *Conceptual Structures*. Reading, MA: Addison Wesley.

Whitten, J., Bentley, L. and Barlow, V. (1994) *System Analysis and Design Methods*. Homewood, IL: Irwin.

Wieringa, R. (1991) Object-oriented analysis, structured analysis and Jackson system development. *In* F. Van Assche, B. Moulin and C. Rolland (eds), *IFIP TC2/WG8.1 Working Conference on the Object-Oriented Approach in Information Systems*, Elsevier Science, pp. 28–31.

Winograd, T. and Flores, F. (1986) *Understanding Computers and Cognition: a new foundation for design*. Norwood, NJ: Ablex.

https://doi.org/10.1017/S1351324996001337 Published online by Cambridge University Press