Deriving requirements specifications from the application domain language captured by Language Extended Lexicon

Leandro Antonelli[1], Gustavo Rossi[1],
Julio Cesar Sampaio do Prado Leite[2], Alejandro Oliveros[3]

[1] Lifia, Fac. de Informática, UNLP, calle 50 esq 120, La Plata, Bs As, Argentina
{lanto, gustavo}@lifia.info.unlp.edu.ar
[2] Dep. Informática, PUC-Rio, Rua Marquês de Sâo Vicente 255, Gávea, RJ, Brasil
www.inf.puc-rio.br/~julio
[3] INTEC – UADE, Bs As, Argentina
oliveros@gmail.com

Abstract. Understanding the context of a software system during requirements specification is a difficult task. Sometimes application domains are very complex, other times the limits of the application are fuzzy. Thus, it is difficult to elicit and write the initial set of requirements. This difficulty frustrates requirements engineers and restricts the process of analysis, which could lead to a final software requirement specification of low quality. In such situations technologically outstanding software systems can be built, but they may fail to suit the needs of the client. Hence, clients are unsatisfied and development projects fail. In this paper we propose a strategy to use the application domain language captured by the Language Extended Lexicon in order to obtain different products related to requirements specification. Products vary from classic requirements which state "the system shall…" to products such as Use Cases and User Stories. The strategy focuses on obtaining the initial set of functional requirements. We believe that by minimizing the gap between the problem and the initial set of requirements, we provide engineers with a preliminary product they can work on and refine to reach the quality needed.

Keywords: Requirements specifications, Domain Analysis, Language Extended Lexicon, Requirements statements, User Stories, Use Cases.

1 Introduction

Understanding the context of a software system and specifying requirements can be a hard task for engineers. The contexts of applications can be very complex, so that they can hardly be understood. In this situation it is very difficult to write requirements. Ackoff states that we commonly fail because our solution does not

apply to the problem but not because the solution is not technically well-built [1]. Nowadays, this statement is still true, as several surveys confirm [31] [20].

Software development is a succession of descriptions in different languages where a previous description is necessary for the next [26]. So, if changes are incorporated into a description, previous and succeeding descriptions will have to be changed in order to maintain conformity. For instance, Boehm [5] states that if a mistake occurs in a requirements description and it is corrected in code description, the correction cost could be multiplied by up to 200. Moreover, Mizuno developed the "waterfall of errors" [24] in which he states that in each stage of software development the possibility of occurrences of mistakes is bigger than in the previous one, because each stage relies on products from previous ones.

Thus, it is important to begin a software development with requirements that are as correct and as complete as possible. Although some literature holds the belief that correctness and completeness are two attributes that requirements specifications must satisfy [17], we know that this is unfeasible [12]. However, we have to find ways of diminishing incompleteness and dealing with the possible conflicts that do arise in the requirements context. As such, before the specification of requirements and expectations, it is necessary to understand the context of the application in the broadest way. An approach based on understanding the context through its culture and through the study of the context language has been pointed out as a rewarding strategy, and it is the one we have followed.

The Language Extended Lexicon (LEL) is a technique to specify an application domain (context) language [22]. LEL is a very convenient tool for experts with no technical skills, although people with such skills will obtain more profit from its use. LEL effectively captures and models the application language because it conforms to the mechanism used by the human brain to organize expert knowledge [33]. In particular, the convenience of LEL as a tool arises from 3 significant characteristics: it is easy to learn, it is easy to use and it has good expressiveness. There are several publications which use LEL in complex domains that validate these claims. Gil et al [13] state that "the experience of building a LEL in an application completely unknown to the requirements engineer and with highly complex language can be considered successful, since users stated that requirement engineers have developed a great knowledge about the application". Cysneiros et al [11] state that "the use of LEL was very well accepted and understood by the stakeholders. As these stakeholders were non technical experts from a specific and complex domain, the authors believe that LEL can be suitable to carry out in many other domains". These three characteristics contribute significantly to obtain high quality models, as they allow the actors involved in software development (experts, requirements engineers and developers with different capacities and abilities) to perform the validation of a LEL [19].

Thus, it is very important to develop a LEL previous to other work, because with a LEL we gain knowledge about the context of the application that will be validated and it helps anchor the shared knowledge. It is possible to identify scenarios [15], ontologies [7] and crosscutting concerns [2] from LEL. In this paper we show how to use LEL to obtain requirements statements, User Stories and Use Cases.

It is important to mention that the objective of the strategy proposed is to provide a preliminary set of functional requirements products which must be enriched and validated in later phases. Use Cases in particular have a very complex description because they follow a structure with multiple elements and the strategy provides a partial description of them. In general, the strategy involves transforming the information that a LEL captures into different requirements templates, so that the quality of the requirements obtained depends on the quality of the LEL.

The strategy proposed can be interpreted as a transformation of models [26] and although it does not provide new knowledge, it helps the actors involved by providing a systematic way of transforming the information contained in the LEL. We believe that initial sessions of requirements elicitation are very hard to cope. Requirements engineers may be facing a new application domain from which they must discover the requirements. They must gain knowledge, organize that information and synthesize requirements. In this process, the requirements engineers may misunderstand or forget things. Hence, we propose a strategy to capture the language from the expert and transform that information into an initial set of requirements that must be later on refined, but which are an important starting point to work on.

Requirements engineers who do not use LEL will derive benefits from applying our approach because in spite of the extra work of constructing LEL, it is easier to construct it than to analyze the context of the application and fill in complex requirements templates. Moreover, constructing a LEL before writing requirements will help to solve conflicts which could arise at later stages.

The rest of the paper is organized in the following way. Section 2 presents some background necessary to understand the strategy. Section 3 describes the derivation strategy. Section 4 shows a case study. Section 5 discusses some related works. Finally, section 6 states some conclusions and future works.

2 Background

This section describes the Language Extended Lexicon (LEL), a technique used to capture the language of the application domain. Then, three ways of specifying requirements are described: requirements statements, User Stories and Use Cases.

2.1 Language Extended Lexicon (LEL)

LEL is a glossary whose goal is to register the definition of terms that belong to a domain. It is tied to a simple idea: “understand the language of a problem, without worrying about the problem” [22].

Terms (called *symbols* within LEL) are defined through two attributes: notion and behavioural responses. Notion describes the symbol denotation, which are intrinsic and substantial characteristics of the symbol, while behavioural responses describe connotation, i.e. the relationship between the term which is described and others.

There are two principles that must be followed while describing symbols: the

circularity principle (also called *closure principle*) and the minimal *vocabulary principle*. The *circularity principle* states that the use of LEL symbols must be maximized when describing a new symbol. The *minimal vocabulary principle* states that the use of words that are external to the Lexicon must be minimized. These principles are vital in order to obtain a self-contained and highly connected LEL. Connections among symbols determine that LEL can be viewed as a graph.

Each symbol of LEL belongs to one of four categories: subject, object, verb or state. This categorization guides and assists the requirements engineer during the description of attributes. Table 1 shows each category with its characteristics and how to describe them.

Table 1. LEL categories.

Category	Characteristics	Notion	Behavioral responses
Subject	Active elements which perform actions	Characteristics or condition that subject satisfies	Actions that subject performs
Object	Passive elements on which subjects perform actions	Characteristics or attributes that object has	Actions that are performed on object
Verb	Actions that subjects perform on objects	Goal that verb pursues	Steps needed to complete the action
State	Situations in which subjects and objects can be	Situation represented	Actions that must be performed to change into another state

Some examples of LEL symbols are presented here. The classic bank application domain is used to show symbols from each category. The example consists in a bank which allows clients to open and close accounts. If the account is activated (open) the client can deposit and withdraw money from it. Figure 1 shows a state machine with both states: *activated* and *closed*, and it also shows the conditions which allow transitions: the action *open* allows us to obtain an *activated* account, while the action *close* allows us to *close* the account. Although closed accounts exist, they are blocked from any operation. Then, the operations *deposit* and *withdraw* are related to the state *activated* to show that both operations can be carried out in that state.

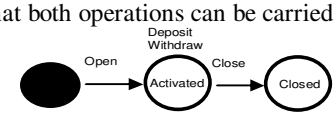

Fig. 1. States and operations of a bank account.

The following symbols from the bank application domain are identified: subject *client* (figure 2); object *account* (figure 3); verbs *open*, *deposit*, *withdraw* and *close* (figure 4); and states *activated* (figure 5) and *closed*. There are underlined words in the descriptions of symbols; these words are expressions that are defined in the LEL too (circularity principle). They represent a kind of link which can be navigated to explore the definition of the other word. It is important to mention that the behavioural responses of *client* (figure 2) and *account* (figure 3) are the same because they describe the same actions from opposite points of view.

2.2 Requirements Specification

IEEE standard 830-1998 states that requirements must describe clearly what the software system must do [17]. Thus, they recommend using the expression “the

system shall…" because it states clearly the functionality and the obligatory condition that the software system must implement. In this sense it is important to use the word "shall" instead of using other weak expressions as "should" or "could" [28] [32]. The following example shows a requirement from the bank application.

Subject: client
Notion
Person that operates an account.
Behavioral responses
The client can open an account.
The client can deposit money into his account.
The client can withdraw money from his account.
The client can close an account.

Fig. 2. Client symbol description.

Object: account
Notion
The account has a balance.
Behavioral responses
The client can open an account.
The client can deposit money into his account.
The client can withdraw money from his account.
The client can close an account.

Fig. 3. Account symbol description.

Verbs: close
Notion
Act of ceasing to operate the account.
Behavioral responses
The client withdraws money from his account.
The bank denies any account operation.

Fig. 4. Close symbol description.

State: Activated
Notion
Situation where the client is ready to use an open account.
Behavioral responses
The client can close the account and he will have a closed account.

Fig. 5. Activated symbol description.

The system shall close an account.

Fig. 6. Requirement statement.

2.3 User Stories

A user story is a description in natural language that captures what the user wants to achieve. User stories are used with agile software development methodologies and generally adjust to a template which considers three attributes: a *role*, a *goal/desire* and a *reason* [9]. The *goal/desire* represents the requirement that the application must fulfill. The *role* defines the user who interacts with the application in order to use the feature described by the *goal/desire*. Both these attributes refer to elements within the scope of the application. In contrast, the *reason* belongs to the context of the application and it states why the user requires that the application provide the functionality described in *goal/desire* (Figure 7).

We can identify four User Stories from the bank application, one for each verb: *open an account*, *close an account*, *withdraw money* and *deposit money*. The role is the same in all the User Stories: the client. Then, the reason must be stated according to the verb. We provide an example of a User Story according to the *close account* operation (Figure 8).

As a <role>,
I want <goal/desire>
so that <reason>.

Fig. 7. User Story description.

As a client,
I want to close an account
so that I cease to operate the account.

Fig. 8. *Close an account* User Story.

2.4 Use Cases

Jacobson developed a way of specifying the behaviour of an object oriented application describing its use [18]. Use Cases can be specified with different levels of abstraction. They vary from conceptual diagrams with many types of relationships to textual descriptions with different levels of granularity. Cockburn [8] identifies three levels of detail in writing use cases. First, there is the brief use case, which consists of a few sentences summarizing the objective of the use case. Then, there is the casual use case, which consists of a few paragraphs of text, describing the sequence of main actions of the use case. Finally, there is the fully dressed use case, which is a formal document based on a detailed template with various sections. This is what is most commonly understood as use case. The full description includes a description of the main success scenario as well as alternatives or variants. In this paper we concentrate only on the main success scenario and some other attributes such as *name,* which is a short statement of the action, and *goal,* which is a goal in the context, so it is in fact the reason. Then, there is a description of how to implement the functionality. There are descriptions of the state of the world before and after the execution of the Use Case, which define the condition that must be validated previous to execution and the situation that must be achieved after the execution. These attributes are the *precondition* and *success end condition.* Finally, there is a description of the role the user must fulfill while interacting with this functionality as well as a description of the actions the system and user perform during the execution of the functionality. This template is summarized in Figure 9.

We can identify four Use Cases from the bank application, one for each verb: *open an account*, *close an account*, *withdraw money* and *deposit money*. The primary actor is the same in all the Use Cases: the client. The rest of the attributes must be stated according to the verb. We provide an example of a Use Case related to the *close account* operation (Figure 10).

Use Case: <name: goal as a short verb phrase>
Goal in Context: <a longer statement of the goal >
Preconditions: < the state of the world to allow the execution of the use case>
Success End Condition: <the state of the world upon successful completion>
Primary Actor: <role of the primary actor >
Main success scenario
<actions description>

Fig. 9. Use Case Description.

Use Case: Close an account
Goal in Context: Cease to operate an account.
Preconditions: The account must be activated
Success End Condition: The account will be closed
Primary Actor: Client
Main success scenario
The client withdraws money from his account.
The bank denies any account operation

Fig. 10. Close Use Case.

3 Derivation Strategy

The derivation strategy is inspired by Hadad's strategy for deriving Scenarios from LEL [15]. In Hadad's strategy verbs correspond to Scenarios which have actors who perform the actions. These actors correspond with the symbols of the category

subject. Scenarios represent behaviour which will be implemented in a software application, as requirements statements, User Stories and Uses Cases represent functionality. User Stories and Use Cases have actors or roles in their descriptions, so subjects must be considered since they are naturally linked to verbs, because subjects have a description of the actions they perform in their behavioural responses, and these actions are the verbs which originate the requirements. Use Cases also include information about pre and post conditions. These conditions are obtained from state symbols. The following section describes in detail the derivation of each product.

Since LEL describes the application domain, it is necessary to identify the sections of the LEL which are included in the scope of the software system. This task consists in identifying which symbols are within the scope of the software application and which are beyond it. Then, the derivations detailed in the following sections are applied to the symbols which belong to the scope of the software system. This distinction may not be clear-cut, because a symbol may have a part of its description within the scope of the application and a part beyond it. Nevertheless, the aim of the strategy is to produce the initial set of requirements as an aid in the first sessions of elicitation. Later, once the preliminary set of requirements are refined and enriched in further sessions of analysis, the limit of the application will be precise and completely established. Moreover, the derivation proposed is a simple transformation of product structure, and some text may need rewriting in order to make sense in the templates or in order to translate domain oriented description from LEL into software oriented description of requirements products.

3.1 From LEL To Requirements

Since verbs are actions within the scope of the software system, they are candidates for requirements that the software system must implement. The strategy can be described in the following way using ATL transformation [3] (Figure 11) and with a diagram (Figure 12).

```
rule LEL2RequirementStament  {
from s : Symbol (s.isVerb())
to r : RequirementStament (statement <-
'The system shall '+ s.name) }
```

Fig. 11. ATL for derivation of requirements.

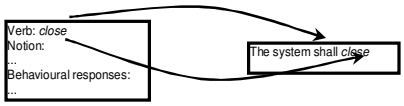

Fig. 12. Graphic derivation of requirements.

Derivation can be exemplified with Figure 4 which defines the verb symbol and figure 6 that shows the requirement statement.

3.2 From LEL To User Stories

User Stories have three attributes: a role ("As a…"), a requirement ("I want…") and a reason ("so that…"). The "I want" attribute must be related to verbs according to the reasoning of previous derivations. Then, a role is necessary to perform the action. Subjects are naturally related to verbs, because the behavioural responses of

subjects include the actions that they perform. Thus, the attribute "As a" is the subject who performs the action stated by the verb. Finally, the attribute "So that" is a reason, an objective; verb notion has this information. The strategy can be described in the following way using ATL transformation (Figure 13) and with a diagram (Figure 14). It is important to mention that Figure 13 does not include the fixed text: "As a… I want to… so that…" because of space limitations and figure clarity.

```
rule LEL2UserStory  {
from s : Symbol (s.isVerb())
to u : UserStory (
   u.role <-
s.referencedInBehaviouralResponsesFrom()
-> select (x| x.isSubject()) -> first()
   u.goal/desire <- s.name
   u.reason <- s.notion }
```

Fig. 13. ATL for Derivation of User Stories.

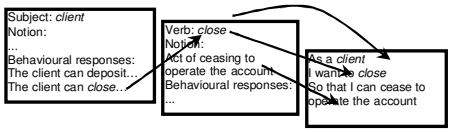

Fig. 14. Graphic derivation of User Stories

.

Derivation can be exemplified with Figure 2 which shows the subject *client*, Figure 4 which defines the verb *close* and Figure 8 which shows the User Stories.

3.3 From LEL To Use Cases

Use Cases represent interactions with the application. Since LEL verbs represent actions within the scope of the application, every verb must be derived into a Use Case. The id of the Use Case must be the name of the verb. As verb symbols have a goal in their notion, this notion is used to describe the goal in the context of the Use Case. The behavioural responses of verb symbols describe the actions needed to reach the goal, so these behavioural responses are used to describe the main success scenario. Then, a role is necessary to perform the action. Subjects are naturally related to verbs, because the behavioural responses of subjects include the actions that they perform. Thus, the primary actor is the subject who performs the action stated by the verb. State symbols are candidates for pre and post conditions [27]. It is necessary to identify which states are related to the verb used to describe the Use Case, and both related states must be used as pre and post conditions. It is important to mention that the LEL may not have states related to each verb, so in this situation pre or post conditions or both could be left blank. The strategy can be described in the following way using ATL transformation (Figure 15) and with a diagram (Figure 16) which shows symbols of the state category in circles and of the other categories in rectangles. Derivation can be exemplified with figure 2 which shows the subject *client*, then with figure 4 which defines the verb *withdraw*. The state of Figure 5 shows how an *account* is transformed from *activated* to *closed*. Finally, Figure 10 describes the Use Case.

```
helper LEL def: StateUsingVerbAsTransition(): Symbol =
(self.referencedInBehaviouralResponsesFrom()->select (x|x.isState())-> first ())

rule LEL2UseCase  {
from s : Symbol (s.isVerb())
to u : UseCase (
   u.useCase <- s.name
   u.goalInContext <- s.notion
   u.preconditions <- (StateUsingVerbAsTransition).name
   u.successEndCondition <- (StateUsingVerbAsTransition) .behaviouralResponses() ->
select (x|x.isState()) -> first()).name
   u.primaryActor <- s.referencedInBehaviouralResponsesFrom() -> select
(x|x.isSubject()) -> first()
   u.mainSuccessScenario <- s.behaviouralResponses }
```

Fig. 15. ATL for derivation of Use Cases.

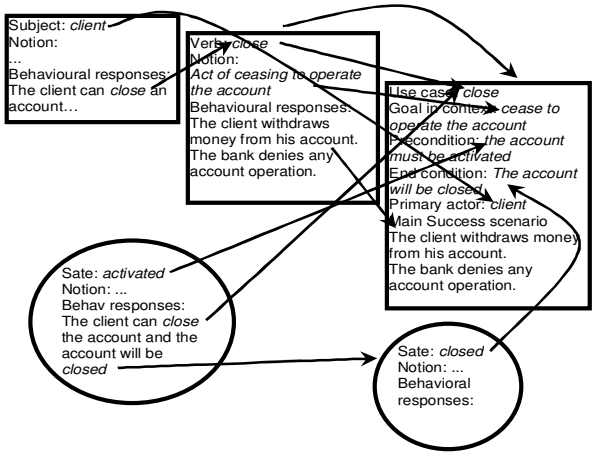

Fig. 16. Graphic Derivation of Use Cases.

4 Case Study

This section describes a particular application domain and a LEL, and discusses the requirements statements, User Stories and Use Cases derived from LEL. The case study involves a real application which was developed for an insurance company by one of the authors of this paper. The LEL was developed previous to software development. The main requirement artifact was requirements statements but User Stories and Use Cases were also developed when more detailed description was needed. In that situation descriptions of requirements were provided intuitively and the strategy described in this paper was not followed.

In this section we describe the application domain and its LEL. We also derive requirements statements, User Stories and Use Cases from LEL in order to contrast the results of the derivation with the requirements intuitively written. We do not include here the requirements products derived because of space limitation, but we

discuss the differences between the requirements products intuitively written and the application of the approach.

4.1 Application Domain

The application domain is an issue tracker which is tailored for a specific organization: an insurance company with an area providing information technology support. The use of this application has two objectives. The main goal is to manage all issues in the organization in order to prevent any issue from getting lost. Also, the tracking of all the issues will be used to improve the business processes within the area. Whatever the objective, the LEL captures the current situation of the application context.

The area is headed by a chief of area, who has 3 sections in charge: development, communication and service desk. Each section has a chief of section and a group of specialists.

The issue is described through the following information: name, description, requester, priority, deadline and category. The basic workflow of an issue consists in the following steps. First, employers from the insurance company create an issue. After that, the issue goes directly to a chief of section if the categories were correctly entered, since they can be used to determine the section. If the categories were not correct, the issue goes to the chief of area who assigns the issue to a chief of section, and the chief of section assigns it to a specialist. Specialists can work on an issue until the issue is finished. They can also pause an issue if they receive another with a higher priority. Another important feature is the possibility of dividing an issue into several sub-issues.

There is an important characteristic about visibility of issues. The Chief of area has the privilege of seeing all the issues, while the chief of section can only see the issues belonging to his section and the specialist can only see his issues.

In order to analyze the performance of the area, it is possible to calculate stats in relation to issues assigned and finished. Based on stats, the chief of area has the privilege of moving specialists from one section to another in order to improve the throughput of the area.

Figure 17 shows a state machine which describes the states in which an issue can be. It is worth mentioning that the state machine is not an input for the strategy. The figure also shows the conditions which allow transitions over the arrows. There are also some actions next to the states, meaning that the action must be performed in that state. Finally, there is one action that can be performed independently of states.

4.2 LEL

The issue tracking LEL has 39 symbols. There are 7 subjects, 12 objects, 12 verbs and 6 states. Subjects can be organized into two groups: a group of roles (4 symbols) and a group of sections (3 symbols). Then, there is a main object (*issue*) and the attributes that the issue has. Some attributes accept several values, so the attribute

(*priority* and *category*) and the possible values (*high / medium / low* and the categories for each section) are described. Verbs are actions that at least one role can perform and States correspond to the situation in which the issues can be. The list of symbols is detailed in table 2.

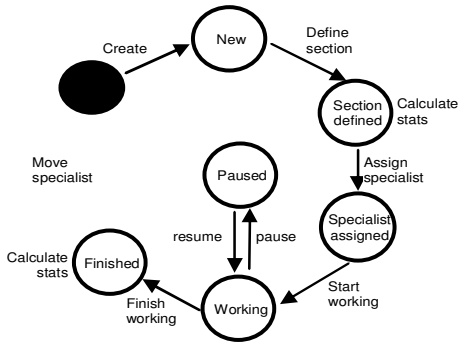

Fig. 17. States and operation of issue tracking application.

Table 2. LEL symbols of issue tracking application.

Subjects	Objects	Verbs	States
Employer of the insurance company	Issue	Create an issue	New
Chief of area	Workload ratio	Define section	Section defined
Chief of section	Priority	Assign issue	Specialist assigned
Specialist	Low priority	Start working	Working
Service desk section	Medium priority	Finish working	Finished
Development section	High priority	Calculate stats	Paused
Communication section	Category	Move specialists	
	Service Desk categories	Create sub-issue	
	Development categories	Edit issue	
	Communication categories	Cancel	
	Deadline	List issues	
	Sub-issue	Change state	

4.3 Requirements

The strategy proposed obtains the requirement statements that were written intuitively while specifying the issue ticket application, since all the symbols identified as verbs correspond to requirements statements.

There are two non functional requirements which are related to authorization and visualization, but they are beyond the scope of the strategy. There are also some business rules which are beyond the scope of the strategy too. For example, a business rule states that "An issue with no category is assigned to the chief of area", which is accordingly described by the symbol *assign issue*. The strategy is right in not identifying it as a requirement because it is not.

Thus, the strategy has obtained all the functional requirements statements needed for the application and nothing more.

4.4 User Stories

The strategy proposed obtains the User Stories that were written intuitively while specifying the issue ticket application, because all the symbols identified as verbs correspond to User Stories.

User Stories provide a more complex description than requirements statements because User Stories add a role and a reason. In general, roles and requirements are easily described without any kind of assistance (i.e. this strategy), but it is sometimes difficult to describe the reason as it refers to something located outside the application but within the context of the application domain. Requirements engineers sometimes find it difficult to cross this boundary.

4.5 Use Cases

The strategy proposed obtains the Use Cases that were written intuitively while specifying the issue ticket application, because all the symbols identified as verbs correspond to Use Cases.

Attributes *Use Case name*, *Goal in context* and *Primary Actor* are in general easy to describe and they represent the same information that is described in User Stories. *Main success scenario* is also easy to describe in general although it requires a level of detail that in many cases is not provided during intuitive description, because the requirement engineer has a lot of attributes to complete and he cannot pay attention to some details. The strategy proposed obtains the main success scenario from the behavioural responses of verbs. This allows the requirements engineer to focus on describing very few attributes, as verbs are described with notion and behavioural responses only. Then the strategy proposed combines some simple descriptions to obtain a complex one such as a Use Case.

The only disadvantage is that LEL does not differentiate between application and context of the application, so the main success scenario does not provide an explicit description of what the system and what the user must do. In contrast, it provides descriptions of what different roles must do and the practitioner reading the Use Case must interpret whether the role is within the system or outside of it.

Finally, there are two attributes that are generally difficult to complete during intuitive description and demand a great effort: *precondition* and *success end condition*. In general these attributes are difficult to identify but in LEL they are directly captured through state symbols. Thus, the strategy identifies those symbols and uses them to describe the Use Case.

5 Related Works

It is very hard to analyze natural language in order to extract requirements, but at the same time natural language is key in Requirements Engineering [4]. There are some approaches which perform text mining on documents to identify verbs and objects which lead to requirements [14]. Other approaches perform text mining to

identify subjects, roles, tasks and objects [31]. These kinds of natural language analyses have a problem of term ambiguity, which is why we decided to work with LEL, a structured glossary constructed from natural language, instead of analyzing natural language documents. By analyzing a LEL we obtain a preliminary version of different specifications products: requirements statements, user stories and use cases. We agree with Ryan [29] who states that validation of requirements must remain an informal and social process, so our approach obtains a preliminary version of requirements products that has to be completed afterwards.

Another important distinction is that our approach considers that the application domain knowledge captured by a LEL has information about requirements, so we can use a LEL to derive them. Lee et al [21] use domain knowledge information to enrich requirements, but they need a previous version of requirements to analyze and combine them with knowledge information so as to provide a richer version of them.

Niu et al [25] perform text mining to identify requirements statements in a similar way to our approach. They look for a "verb – direct object" structure. Although we use verb symbols to derive requirements statements, verb symbols have sentences in their behavioural responses with the structure "subject – verb – direct object" similar to the one used by Niu.

Hadad [16] obtains requirements statements from scenarios. She considers episodes of scenarios as requirements statements candidates. In our approach, we use the behavioural responses from LEL as requirements statements candidates, which in fact can be considered predecessors of episodes, so we perform the same identification but on a previous product.

Breitman et al [6] propose a strategy to identify and manage User stories. Nevertheless, the template they use is not "as a… I want… so that…". They use Scenarios that are enriched with risk and priority attributes.

Li et al [23] use natural language to derive use cases. They organize text into subject-verb-object clauses. Then, they generate a UML class diagram. Further analysis allows them to generate use cases. Although subject-verb-object clauses have similarities with descriptions of symbols in LEL, our approach differs from that because we derive Use Cases directly from LEL, while Li obtains a class diagram as an intermediate step.

Cysneiros et al [10] use LEL to identify and register non functional requirements. Then, they obtain use cases from LEL (which contains functional description) and the non functional requirements previously identified. The non functional requirements obtained before Use Case description are used as pre and post conditions, while our work uses states to derive information for pre and post conditions.

6 Conclusions and Future Works

We have presented an approach to produce preliminary requirements specifications straightforwardly from the application domain language captured by the Language Extended Lexicon. Eliciting requirements can be very disappointing if

miscommunication and lack of knowledge permeates the process. Tackling these issues is hard, mainly due to the cultural clash among stakeholders. With the approach proposed we focus on the language of the context, and from there we obtain more complex requirements descriptions through selecting, sorting and combining the basic elements. Moreover, we provide a way of coping with the ambiguity of natural language, as we use a structured and organized product instead of natural language documents. Apart from this, we provide an instrument of traceability. Further work will involve more evaluations based on different cases, but also exploring possible evolutions of the current process.

References

1. Ackoff, R.: Redesigning The Future, Wiley (1974)
2. Antonelli, L., Rossi, G., Leite, J.C.S.P. : Early identification of crosscutting concerns in the domain model guided by states, in proceedings of the 2010 ACM Symposium on Applied Computing, Sierre, Switzerland, ISBN:978-1-60558-639-7, March 22-26 (2010)
3. ATL a model transformation technology, http://eclipse.org/atl/
4. Berry, D.M.: Ambiguity in Natural language Requirements Documents (Extended Abstract), 14th Monterrey Workshop, Monterey, CA, USA, September, 1-7 (2007)
5. Boehm, B.W.: Software Engineering, Computer society Press, IEEE (1997)
6. Breitman, K.K., Leite, J.C.S.P.: Managing User Stories, in proceedings of the International Workshop on Time- Constrained Requirements Engineering (2002)
7. Breitman, K.K., Leite, J.C.S.P.: Ontology as a Requirements Engineering Product, In Proceedings of the 11th IEEE International Conference on Requirements Engineering (RE), IEEE Computer Society, Monterey Bay, California, USA, ISBN 0-7695-1980-6 (2003)
8. Cockburn, A.: Writing Effective Use Cases. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. ISBN 0-201-70225-8 (2001)
9. Cohn, M.: User Stories Applied, Addison Wesley, ISBN 0-321-20568-5 (2004)
10. Cysneiros, L.M., Leite, J.C.S.P.: Driving Non-Functional Requirements to Use Cases and Scenarios, XV Simpósio Brasileiro de Engenharia de Software (2001)
11. Cysneiros, L.M., Leite, J.C.S.P.: Using the Language Extended Lexicon to Support Non-Functional Requirements Elicitation, in proceedings of the Workshops de Engenharia de Requisitos, Wer'01, Buenos Aires, Argentina (2001)
12. Finkelstein, A.C.W., Gabbay, D., Hunter, A., Kramer, J., Nuseibeh, B.: Inconsistency handling in multiperspective specifications, IEEE Transactions on Software Engineering, doi: 10.1109/32.310667, vol.20, no.8, Aug, 569-578 (1994)
13. Gil, G.D., Figueroa, D.A., Oliveros, A.: Producción del LEL en un Dominio Técnico. Informe de un caso, in proceedings of the Workshops de Engenharia de Requisitos, Wer'00, Rio de Janeiro, Brazil (2000)
14. Golding, L., Berry, D.M.: AbstFinder, A Prototype Natural Language Text Abstraction Finder for Use in Requirements Elicitation, Automated Software Engineering, 375-412, (1997)
15. Hadad, G., Kaplan, G., Oliveros, A., Leite, J.C.S.P.: Construcción de Escenarios a partir del Léxico Extendido del Lenguaje, in Proceedings SoST, 26JAIIO, Sociedad Argentina de Informática y Comunicaciones, Buenos Aires (1997)
16. Hadad, G.D.S.: Uso de Escenarios en la Derivación de Software, Tesis Doctoral, Universidad Nacional de La Plata (2008)

17. IEEE, IEEE Recommended Practice for Software Requirements Specifications, IEEE Std 830-1998 (Revision of IEEE Std 830-1993)
18. Jacobson, I., Christerson, M., Jonsson, P., Overgaard. G.: Object-Oriented Software Engineering: A Use Case Driven Approach, ACM Press, Addison-Wesley, ISBN 0201544350 (1992)
19. Kaplan, G., Hadad, G., Doorn, J., Leite, J.C.S.P.: Inspeccion del Lexico Extendido del Lenguaje, In: proceedings of the Workshops de Engenharia de Requisitos, Wer'00, Rio de Janeiro, Brazil (2000)
20. Keil, M., Cule, P.E., Lyytinen, K., Schmidt, R.C.: A framework for identifying software project risks, in Communication of the ACM, volume 41, Issue 11, nov (1998)
21. Lee, B. S., Bryant, B.R.: Automation of software system development using natural language processing and two level grammar, In Proceeding of the Workshop Radical Innovations of Software and Systems Engineering in the Future, Monterey, 244-257 (2002)
22. Leite, J.C.S.P., Franco, A.P.M.: A Strategy for Conceptual Model Acquisition, In Proceedings of the First IEEE International Symposium on Requirements Engineering, San Diego, California, IEEE Computer Society Press, 243-246 (1993)
23. Li, K., Dewar, R.G., Pooley, R.J.: Requirements capture in natural language problem statements, Technical report HW-MACS-TR-0023, Heriot-Watt University, Edinburgh, Scotland, UK (2004)
24. Mizuno, Y.: Software Quality Improvement, IEEE Computer, Vol. 16, No. 3, March, 66 – 72 (1983)
25. Niu, N., Easterbrook, S.: Extracting and Modeling Product Line Functional Requirements, in Proceedings of the 16th IEEE International Requirements Engineering Conference, September 08-12, 155-164 (2008)
26. Pons, C., Giandini, R., Pérez, G.: Desarrollo de Software dirigido por Modelos - Conceptos teóricos y su aplicación práctica, Editorial EDULP & McGraw-Hill Educación, Volumen 1, 300 páginas, ISBN: 978-950-34-0630-4 (2010)
27. Rocco, V., Villalba, J.C.: Una heuristica de derivación de LEL a Escenarios, Tesis de grado, Universidad Nacional de La Plata, Mayo (2010)
28. Rosenberg, L., Requirements Engineering. Methodology for Writing High Quality Requirement Specifications and for Evaluating Existing Ones, Software Assurance Technology Center, NASA Goddard Space Flight Center Greenbelt, MD, September 24 (1998)
29. Ryan K.: The Role of Natural Language in Requirements Engineering, In Proceedings of the IEEE International Symposium on Requirements Engineering, San Diego, CA, IEEE Computer Society Press, Los Alamitos, CA, 240-242 (1993)
30. Sawyer, P., Rayson, P., Garside, R.: REVERE: support for requirements synthesis from documents, Information Systems Frontiers, v.4 n.3, September, 343-353 (2002)
31. Standish Group, The Chaos Report, http://www.standishgroup.com/chaos_resources/index.php (1995)
32. Wilson, W. M., "Writing Effective Requirements Specifications", Software Technology Conference (1997)
33. Wood, L.E.: Semi-structured interviewing for user-centered design, Interactions of the ACM, april-may, 48-61 (1997)