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

Leandro Antonelli¹, Gustavo Rossi¹,
Julio Cesar Sampaio do Prado Leite², Alejandro Oliveros³

¹Lifia, Fac. de Informática, UNLP, calle 50 esq 120, La Plata, Bs As, Argentina 
{lanto, gustavo}@lifia.info.unlp.edu.ar
²Dep. Informática, PUC-Rio, Rua Marquês de Sào Vicente 255, Gávea, RJ, Brasil 
www.inf.puc-rio.br/~julio
³INTEC – 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 determine which are requirements of the software system and which are beyond its scope. 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 must...” to products such as Use Cases and User Stories.

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

1 Introduction

It is very hard to understand the context of a software system and specify requirements. The context of applications can be very complex, so it 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 and not because the solution is not technically well-built [1]. Nowadays, this statement is still true, as several surveys confirm [32][29][21].

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 [6] 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 every stage of software development the possibility of occurrences of mistakes is bigger than in the previous one, because every stage relies on products of previous ones.

Thus, it is important to begin a software development with requirements that are the most correct and the most complete as possible. Although some literature hold the belief that correctness and completeness are two attributes that requirements specifications must satisfy [18], we know that this is unfeasible [13]. However, we have to use ways of diminishing the incompleteness, as well as dealing with the possible conflicts that do happen in the requirements context. As such, previous to the specification of the requirements and expectations, it is necessary to understand the context of the application in the most broader way. Using an approach based on understanding the context through its culture based on the study of the context language has been pointed as a rewarding strategy, and is the one we have followed.

The Language Extended Lexicon (LEL) is a technique to specify an application domain (context) knowledge [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 is a convenient and effective tool to capture and model the application language because LEL conforms to mechanisms used by 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 [14] 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 [12] 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 positively to obtain models of quality as they allow to the actors involved in software development (experts, requirements engineers and developers) who have different capacities and abilities to perform the validation of an LEL [20].

Thus, it is very important to develop a LEL previous to other piece of work, because with an LEL we gain knowledge about the context of the application that can be validated and it helps in anchoring the shared knowledge. It is possible to identify scenarios [16], ontologies [8] and crosscutting concerns [2] from LEL. In this paper we show how to use the 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 requirements products which must be enriched and validated in later phases. Use Cases in particular have a very complex description and the strategy provides a partial description of them. In general, the strategy involves
transforming the information that the LEL captures into different templates of requirements, 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 for the transformation of information contained in the LEL. People who do not use LEL will obtain benefits because it is easier to construct LEL than to fill in complex requirements templates. Moreover, constructing LEL before writing requirements will help to solve conflicts which could arise in requirements.

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”.

Terms (which are called symbols within the 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 behavioral 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 the LEL belongs to one of four categories: subject, object, verb and 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.

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

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 provides the chance of opening and closing 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.

![States and operations of a bank account.](image1)

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*. Descriptions of symbols have underlined words; these words are expressions that are defined in LEL too (circularity principle). They represent a kind of link which can be navigated to explore the definition of the other word.

**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.

![Client symbol description.](image2)

**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.

![Account symbol description.](image3)

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

![Close symbol description.](image4)

**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.

![Activated symbol description.](image5)
2.2 Requirements Specification

IEEE states that requirements must describe clearly what the software system must do. Thus, they recommend using the expression “the system must...” because it states clearly the functionality and the obligatory condition that the software system must implement it. In this sense it is important to use the word “must” instead of using other weak expressions as “should” or “could” [28]. The following example shows a requirement from the bank application.

The system must 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 [10]. The goal/desire represents the requirement that the application must fulfill. The role defines the user who interact with the application in order to use the feature described by the goal/desire. 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 needs that the application provides 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 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 stop operating 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 [19]. 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 [9] identified three levels of detail in writing use cases:

(i) Brief use case: it consists of a few sentences summarizing the objective of the use case.

(ii) Casual use case: it consists of a few paragraphs of text, describing the sequence of main actions of the use case.
(iii) Fully dressed use case: it 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 only concentrate on the main success scenario and some other attributes explained as follows. Name which is a short statement of the action which in fact is the requirement that the application must implement. Goal, which as is a goal in the context, so it is in fact is 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 action the system and user perform during the execution of the functionality. The template is summarized in Figure 9.

We can identify four Uses 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 the verb. We provide an example of a Use Case related to the close account operation (Figure 10).

3 Derivation Strategy

The derivation strategy is inspired by Hadad’s strategy for deriving Scenarios from LEL[16]. In Hadad’s strategy verbs correspond to Scenarios which have actors who perform the actions. These actors correspond with the symbols of 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 is included in the scope of the software system. This task must be done with the stakeholders and 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 will be applied to the symbols which belong to the scope of the software system.

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).

\[
\text{rule LEL2RequirementStatement} \\
\text{from } s \text{: Symbol (s.isVerb())} \\
\text{to } r \text{: RequirementStatement (statement <- 'The system must ' + s.name)}
\]

Fig. 11. ATL for 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; and 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).

\[
\text{rule LEL2UserStory} \\
\text{from } s \text{: Symbol (s.isVerb())} \\
\text{to } u \text{: UserStory} \\
\text{u.actor <- s.referencedInBehaviouralResponsesFrom()} \\
\text{-> select (x| x.isSubject()) -> first()} \\
\text{u.requirement <- s.name} \\
\text{u.reason <- s.notion}
\]

Fig. 13. ATL for Derivation of User Stories.

Derivation can be exemplified with Figure 2 which shows the subject client, Figure 4 which defines the verb withdraw 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 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).

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.

Derivation can be exemplified with figure 2 which shows the subject client, then with figure 4 which defines the verb withdraw. The state of Figures 5 shows how an account is transformed from activated to closed. Finally Figure 10 describes the Use Case.

4 Case Study

This section describes an application domain and shows LEL, 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. LEL was developed previous to software development. The main requirement artifact was requirements statements but User Story and Use Cases were also built, when it was needed more detail of a description. In that situation
descriptions of requirements were provided intuitively and the strategy described in this paper it was not followed.

In this section we describe the application domain, its LEL and we also derive requirements statements, User Stories and Uses Cases with the LEL and we contrast results of the derivation with requirements intuitively written.

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 of 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.

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 can go 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. 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 it is described the attribute (priority and category) and the possible values (high / medium / low and the categories for each section are described). The, verbs are actions that at least one role can perform and States correspond to 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.

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

**4.3 Requirements**

The strategy obtains the requirement statements that a requirements engineering would have written, since all the symbols identified as verbs corresponds to requirements statements.

There are two non functional requirements which are related to authorization and visualization and beyond of the scope of the strategy. There are also some business rules which the strategy does cover. For example, a business rule states that “The issue with no category is assigned to the chief of area” and this description is stated in the description of the symbol assign issue and it does not become a requirement.

Thus, the strategy has obtained the functional requirements statements needed for the application and it has not obtained anything more.
4.4 User Stories

The strategy obtains the User Stories that a requirements engineer would have identified, 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 describes 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 obtains the Use Cases that a requirements engineer would have identified, 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 is not provided in many cases 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.

Finally, there are two attributes that are difficult in general to complete in general during intuitively description and demand a huge effort: precondition and success end condition. In general these attributes are difficult to identify and 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 natural language is key in Requirements Engineering [4]. There are some approaches which perform text mining from documents to identify verbs and objects which lead to requirements [15]. Other approaches perform text mining to identify subjects, roles, tasks and objects [31]. These kinds of natural language analyses have the 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 LEL we obtain a preliminary version of different specifications products: requirements statements, user stories and use cases. We agree with Ryan [30] 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 LEL has information about requirements so we can use LEL to derive them. Beum Seuk [5] uses domain knowledge information to enrich requirements, but he needs a previous version of requirements to analyze and combine them with knowledge information so as to provide a richer version of requirements.

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 in their behavioural responses sentences with the structure “subject – verb – direct object” similar to the one used by Niu.

Hadad [17] 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 in a previous product.

Breitman et al [7] 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 [11] 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 previous to 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 specification in a straightforward way 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