International Journal of Computer Science and Applications,

A PATTERN-BASED METHOD FOR BUILDING REQUIREMENTS DOCUMENTS IN CALL-FOR-TENDER PROCESSES

SAMUEL RENAULT

CITI, CRP Henry Tudor,
Luxembourg, Luxembourg
samuel.renault@tudor.lu

OSCAR MÉNDEZ

CINVESTAV, Instituto Politécnico Nacional,
México D.F., México
oscar@math.cinvestav.mx

XAVIER FRANCH

GESSI Research Group, UPC,
Barcelona, Spain
franch@lsi.upc.edu
http://www.lsi.upc.edu/~franch

CARME QUER

GESSI Research Group, UPC,
Barcelona, Spain
cquer@lsi.upc.edu
http://www.lsi.upc.edu/~cquer

This paper presents our PABRE method for facilitating Requirements Elicitation on the basis of Requirement Patterns with the goal of saving time and reducing errors during this activity. The process presented applies for elicitation in Off-The-Shelf selection projects driven by call-for-tender processes and uses a Requirement Patterns Catalogue. The process selects patterns from the catalogue that apply to the particular selection project, and convert them into the real requirements that finally configure the project Requirements Document. We show some benefits of the pattern approach for requirements engineers and IT consultants, as well as for customers. Finally we discuss the strengths and weaknesses of the proposal and identify some future work.

Keywords: Requirements engineering; Requirements reuse; Off-The-Shelf component selection; Requirement patterns; Requirement document; Call-for-tender process.

1. Introduction

Nowadays, Information Systems (IS) are less and less developed from scratch. Organizations are increasingly choosing Off-The-Shelf (OTS) based technologies to build them [Li et al. (2008)], because this kind of software, including both Commercial (COTS) components and Open Source Software, offers a wide scope of functionalities to support business processes.

There exist many methods for selecting OTS components (see [Mohamed et al. (2007)] for a recent survey). However, as reported in several empirical studies (e.g., [Torchiano and Morisio (2004)][Li et al. (2006)]), these methods are hardly adopted by industry because they propose some techniques and artefacts that either are too complex or exceed the usual resources that companies may invest in OTS selection. Consequently, companies tend to develop their own lightweight selection methods.

Among them, the CITI department of the Public Research Centre Henri Tudor (CPRHT, Luxembourg) has developed a pragmatic approach to select OTS-based solutions for Small- and Medium-sized Enterprises (SME) [Krystkowiak and Bucciarelli (2003)][Krystkowiak et al. (2004)], see Fig. 1. It has been designed to operate in organisations that have no knowledge about Requirements Engineering (RE), and where an external body, usually an Information Technology (IT) consultant, performs the requirements analysis on behalf of this organisation. This approach is based on:

- The joint elicitation of requirements with the customer. The IT consultant conducts the elicitation activity, using his/her skills to extract the requirements that apply to the selection project (see (1) in Fig. 1).
- The design of a Requirements Book (2). The requirements that the IT consultant captures from the customer are structured into a technical document, the requirements book.
- The pre-selection of OTS components based on preliminary functional requirements and availability of local providers able to implement the components.
- The use of the requirements book to conduct a Call-For-Tender process. In a call-for-tender process [Lauesen and Vium (2004)], a question for each requirement in the requirements book is raised. Software and service providers present their bids (3) by answering these questions, stating how their IT solutions achieve the requirements. The answers have to be matched with the requirements book during an evaluation process (4). As a result, an acquisition agreement (5) is written between the customer and the selected supplier (6).
- Final deployment of the OTS-based solution (7) by the selected supplier.

With more than forty projects performed according to this methodology since 2001, the problematic of knowledge capitalisation (i.e., how to transfer knowledge from one project to the next ones) arose at CITI. The first strategy to share knowledge according to experiences of IT consultants was simply to duplicate requirements from former projects as a starting basis for new requirements books. However, this kind of knowledge reuse demonstrated soon its limitations since former requirements were not standardised and were highly dependant on the context of their project and on the engineers that created them. In addition, IT consultants wanting to use this knowledge needed to be aware of all the former requirements books so as to select the ones that were closer to the current project. We concluded that some more powerful conceptual support to the knowledge capitalisation problem was needed.

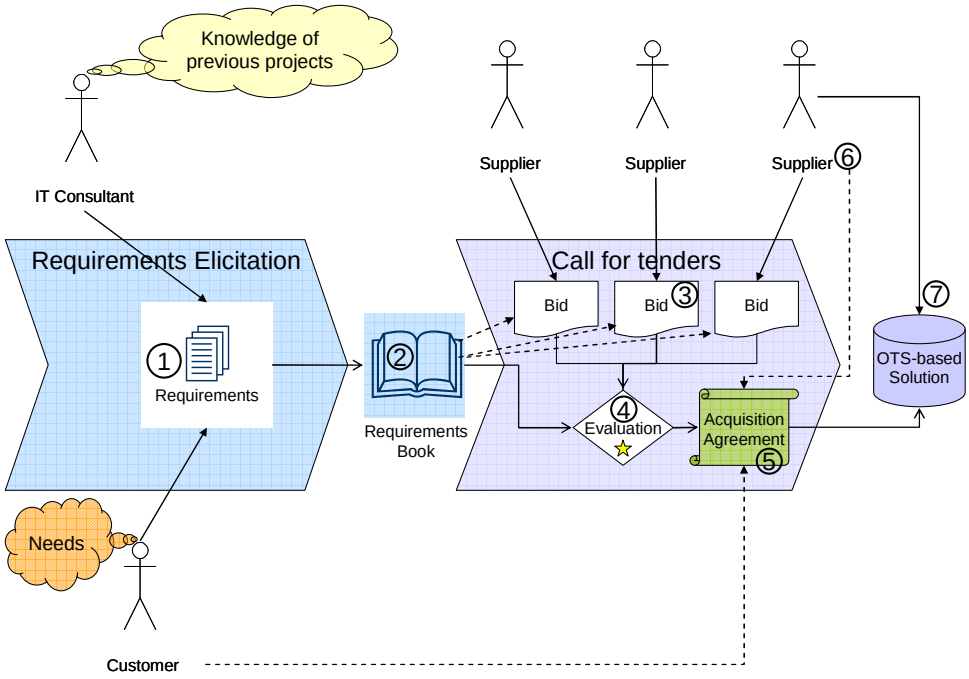

Fig 1. Current call-for-tender process for OTS-based solution selection defined by the CITI - CPRHT

At this point, the notion of pattern [Alexander (1979)] was considered. Patterns have been successfully used in different facets of Software Engineering. Their applicability to the call-for-tender situation seems clear, since requirements that appear over and over in requirements books could be identified as the solution to particular problems in a given context (the classical context-problem-solution scenario of patterns). Therefore, we adopted the notion of Requirement Pattern as the central artifact for requirements reuse.

In this paper we explore the use of a catalogue of requirement patterns as a way to support requirements elicitation by IT consultants (acting as requirement engineers) working with a customer (either the final user or user's representative of the system-to-be). The resulting requirements book is then used as a basis for performing a call-for-tender process, to select the OTS-based solution.

The research undertaken has evolved through the following phases (see Fig. 2):

(1) Consolidation of the existing requirement books: we started our work with a sample of 7 requirement books built for the domains of Content and Document Management Systems. These books contained about 70 non-functional requirements in average (although it is worth to remark that some requirements were not atomic). We aligned these requirements in a very simple way, using a spreadsheet, placing in the same row those requirements that were similar (some requirements appeared at more than one row).

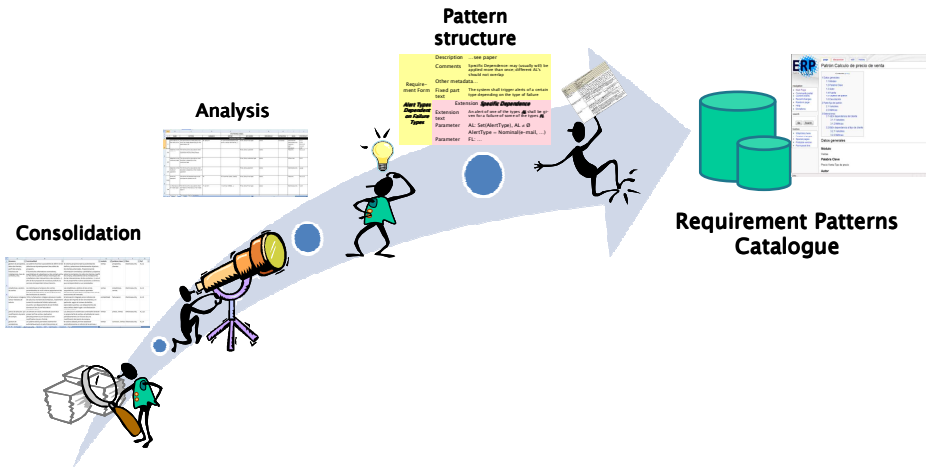

Fig 2. Research method of this work

(2) Analysis of the non-functional requirements: we examined the contents of the spreadsheet as a whole, aiming at finding the fundamental concepts of the analyzed books. Fundamental properties of requirements were also targeted in the search of criteria for designing the structure of the pattern.

(3) Definition of pattern structure: we articulated the properties found in form of a metamodel. Different parts of a pattern were identified and their relationships (causality, optionality, etc.) established. From this metamodel, we designed a template including useful management metadata.

(4) Construction of the catalogue: it embraced two different subphases:

(4.1) We built the first version of the catalogue using the pattern structure, the spreadsheets with the analyzed requirement books, literature review, and expert judgment. Also a preliminary case study was conducted as validation of this first catalogue. The resulting version was constituted by 48 non-functional requirement patterns. The experience is reported in [Renault et al. (2009)].

(4.2) We elaborated further the catalogue with exhaustive expert judgement and analysis of the catalogue from different perspectives. Basically, we eliminated redundancies, some patterns whose existence was not clearly justified, and were leaned to merge some patterns to create new, more structured ones. We executed a second case study for validation. As a result, the catalogue evolved to its current form that contains 29 patterns.

The rest of the paper is organized as follows. After providing the necessary background for discussion (Section 2), we will introduce our notion of requirements pattern and outline the catalogue construction process (Section 3). Then, we will present the Pattern-Based Requirements Elicitation method (PABRE) (Section 4) and describe the validation done up to now (Section 5). Finally, we will provide some analysis of the method (Section 6) and present the conclusions and future work (Section 7).

2. Antecedents

2.1. *OTS Components and OTS-based Solutions*

Off-The-Shelf (OTS) components can be categorized into commercial OTS (COTS) components, and Open Source Software (OSS) components. A COTS component is an external, contracted or licensed, software product [Carney and Leng (2000)], a software subsystem block, which can be used as described in [Galster et al. (2007)], where the source code is controlled by the vendor or provider [Basili and Boehm (2001)]. An OSS component is an external, licensed software product, released by a community of developers, which can be used as described in [Galster et al. (2007)], where the source code is controlled by the community of developers and it is open to changes and customizations [Madanmohan and Rahul (2004)].

Both types of OTS components may be a system or a subsystem by themselves, and can be coarse- or fine-grained, depending on what they were developed for. Usually the interaction with these components is through an application programming interface (API) or through user interfaces, and normally their life cycle is controlled by the owner (public or private, closed or open). In the case of COTS components there is a charge for their use, against the no-charge-at-all for the OSS products (not considering costs for installation, configuration, tuning, customization and maintenance, which are costs that exist in both kinds of components). In the case of COTS components it is usually not possible to access their source code, so, their life cycle is completely controlled by the owner. Instead, OSS components allow accessing and using the source code; then, it is possible to develop a customized version of the component, or participate in the life cycle and development with the original community of developers.

Construction of OTS-based solutions requires some dedicated activities that are different from those appearing in traditional software development. Among them, we mention: selection of OTS components from the marketplace; integration of the selected OTS components into the system; maintenance of the system including the periodical updating of OTS components versions; OTS components dedicated quality assessments. Due to the focus of this paper, we are particularly interested in OTS-based solutions selection projects conducted by call-for-tender processes.

OTS selection methods started to be proposed in mid-90s [Kontio (1996)][Maiden and Ncube (1998)] and still nowadays, new methods are formulated (see [Mohamed et al. (2007)] for a survey). In spite of their differences, all of them share some common principles. Among them, we remark the need to overlap the evaluation of components and the elicitation of requirements. In Fig. 3(a) we show the evolution on time of elicited requirements and candidate components (based on [Maiden and Ncube (1998)]). Precisely, the mentioned overlapping between component evaluation and requirement elicitation cannot be applied in the particular case of projects conducted by call-for-tender processes. In this kind of OTS selection projects, which are usually imposed by legal regulations in public administrations, a document containing the conditions of selection

(that we call in this paper Requirements Book) must be made public to initiate the selection process. Next, interested suppliers send their biddings according to the needs expressed in the requirements book. Evaluation rules are also determined in this book. Selection of candidates, therefore, is carried out once all requirements have been elicited (see Fig. 3(b)). As a consequence, the requirements elicitation activity is required to produce a requirement book of very high quality, since once it is made public, it is difficult to change due to legal regulations.

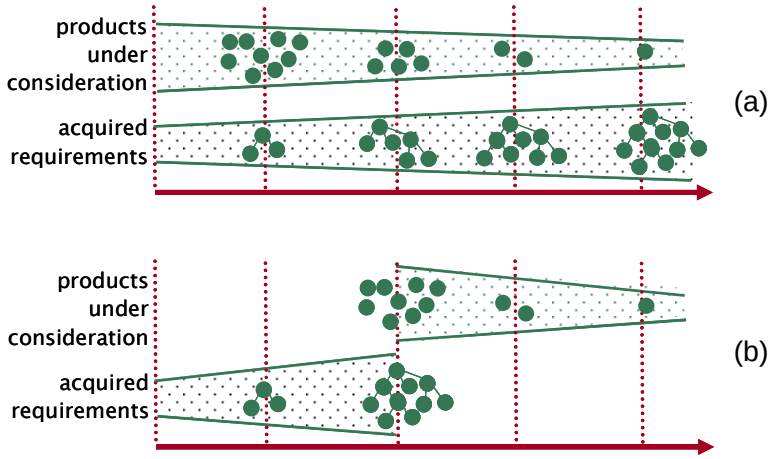

Fig 3. (a) Interleaving of requirements and OTS candidates evaluation in classical OTS selection processes; (b) Strict sequencing of requirements and OTS candidates evaluation in call-for-tender processes.

2.2. *Requirements Elicitation and Requirements Reuse*

Requirements Elicitation is one of the activities that take place during requirements engineering [Sommerville (2005)], consisting on the gathering of requirements from the stakeholders. Requirements elicitation plays a crucial role in traditional software development, because the definition and construction of requirements is the basis of the requirements specification, upon which is based the subsequent design and construction of the IS, including its software-intensive section. In the particular case of OTS-based software development, as highlighted in the previous subsection, the activity is utterly important in the case of call-for-tender processes.

Throughout the years, the techniques for requirements elicitation have evolved (see [Cheng and Atlee (2007)] for a timely state of the art). These techniques are different depending on the type of development or software construction that is carried out. The development of component-based software has its own techniques and methods for requirements elicitation. One of the characteristics of component-based IS construction is speed, so that all activities of this type of development should be fast, including the software requirements elicitation. It is necessary to have a method to conduct the requirements elicitation in the shortest time possible and the most reliable way. Another characteristic of component-based development is reuse, so that all the techniques and

methods that constitute this type of development should contribute to the reuse of components and knowledge, including techniques for gathering requirements.

As part of this reuse need, Requirements Reuse has been subject of research [Lam et al. (1997)][Cybulski and Reed (2000)]. This trend is oriented to use knowledge of prior experiences in requirements elicitation, and this drives to knowledge reuse. Recent proposals of requirements reuse focus on the concept of feature model that stems from product line engineering (e.g., [Pohl et al. (2005)], see [Cheng and Atlee (2007)] for more references). However, we have opted for the concept of Requirement Pattern.

In software engineering, the use of the pattern concept was made popular in the design phase through design patterns [Gamma et al. (1995)] that in fact are, at least in their origins, inspired by the concepts of pattern and pattern language, issued by the architect Alexander and his team [Alexander (1979)]. This concept is about the re-use of knowledge, more precisely, solutions to common and repetitive problems that appear in a particular context, where solutions can be applied over and over to this kind of problems. Some authors have proposed the concept of requirement pattern, either in general [Robertson (1996)][Duran et al. (1999)][Withall (2008)] or in particular contexts like embedded systems [Konrad and Cheng (2002)] or security requirements [25]. Other criterion for classifying approaches is the formalism used to express the requirements: plain natural language [Duran et al. (1999)][Withall (2008)], textual-based artifacts like use cases [Robertson (1996)], object models [Fowler (1997)][Moros et al. (2008)], or formal artifacts like logic-based [Konrad and Cheng (2002)].

Using patterns, a requirements book may be constructed by identifying which patterns apply in the call-for-tender project and adapting them to the specificities of the system being procured. The knowledge embraced by requirement patterns comes from post-mortem analysis of requirement specifications, from modeling and analysis of domains in software areas, and from specialized documentation from those areas.

3. Structure of the Pattern Catalogue

As mentioned in the introduction, the goal of the paper is defining the PABRE method for building requirement books with the use of a requirement patterns catalogue. In this section we focus on the patterns from a dual point of view. On the one hand, we first describe the structure that the method imposes on the patterns and the catalogue, which is based on [Mendez et al. (2008)]. On the other hand, we outline the process followed to obtain the current contents of the catalogue. At this respect, we remark that describing the concrete contents of the catalogue (i.e., the patterns stored therein) is not an objective of the paper since the method is mostly content-independent.

It is important before introducing the structure of the patterns and the catalogue, to have a sample of the type of requirements that CITI finds in their projects. Table 1 shows four examples.

Table 1. Examples of requirements found in CITI projects.

[For a Document Management System.] An easy-to-use mechanism shall allow binding a publication to an event. The pre-publishing actions (control, validation, etc.) will be identified for each type of workflow.
The system must be available 22 hours per day and 7 days per week. The system should not stop more than 1 hour per working day. The solution's availability rate should be 98% minimum.
The solution should permit to trace all the user actions. The data to trace are: user name, date, accessed or modified data.
The solution should have a graphical interface for all the functionalities presented. A Web interface for external access is also necessary

3.1. *Structure*

The template that we use for requirements patterns is shown in Fig. 4 filled with a particular example, the *Failure Alerts* pattern.

3.1.1. *Pattern metadata*

The first set of attributes defines the metadata about the pattern itself: the name of the pattern; its description; its author; comments included by its author and its users; its goal; the sources from where it was obtained (e.g., the requirement books and projects from which it was identified and included in the repository); and some keywords to facilitate searches in the repository. We highlight the important role that the goal attribute plays since it will help to decide whether the pattern is applicable to the project at hand (see Section 4): a pattern will be applied in a call-for-tender process if the customer needs to achieve its goal. In the case of the *Failure Alerts* pattern, the stated goal is that the users of the OTS-based solution want to be alerted when some failure occurs. In other words, the goal plays the role of the "problem" part of the pattern, whilst the "solution" is encapsulated in the pattern forms.

3.1.2. *Pattern forms*

A requirement pattern, when used in different projects to achieve the same goal, may be written differently, thus the template allows declaring several forms in a pattern. Normally the number of different forms in a pattern will be very low. For instance, the *Failure Alerts* pattern has two forms that differ on the granularity of information needed: if the customer needs some specific types of alerts when some specific types of failures occur (*Alert Types Dependent on Failure Types* form) or not (*Failure Alerts Provided*). Each form has some metadata similar to the one of the pattern, so we have as attributes: the name of the pattern form; its description; its author; comments included by its author and its users; its version or data in which it has been changed for the last time; and the

sources from where it was obtained[*]. Finally, every form has exactly one fixed part and may have extensions.

3.1.3. *Fixed Part*

Fixed parts of a form are usually quite abstract: the inclusion in a requirements book of a requirement obtained from the application of a fixed part, states that the procured system has to achieve the goal of the requirement pattern, but it does not state how this goal is achieved. In case of *Failure Alerts Provided* form, the fixed part states that failures will be informed by means of alerts.

3.1.4. *Extensions*

Since the fixed part of a form is abstract, it is usual to know some extra-information or constraints about how to achieve the goal of the requirement pattern. Form extensions ("extended part" in the template) allow stating this information. Extensions may be defined either by rewriting the fixed part or by restricting it. In case of the *Failure Alerts Provided* form, we define extensions that establish the type of alerts that are required by the customer, and the type of failures that need to be informed of. The use of extensions, therefore, allows including more detailed information in the requirements book when applying the pattern.

3.1.5. *Form text*

Every fixed and extended part of a pattern is specified by a form text. This text is expressed as a short sentence written in natural language that may include one or more parameters that indicate those parts that may vary in different projects. In the case of the *Failure Alerts* pattern, we have parameters in the extended parts of the forms. When a pattern is selected and a form applied, the parameters that appear in the text will be substituted by values. In order to define the valid values that a parameter may take, each parameter will be bound to a metric and optionally will also have a correctness condition. Metrics may be enumerated values (e.g., names of middleware platforms), integer (e.g., for stating number of connections supported), real numbers (e.g., for measuring response time) and Boolean values (e.g., for knowing if some protocol is supported). In our example, the parameter in the first extended part of the *Failure Alerts Provided* form, will take as values the (non-empty) set of types of alerts that the customer whishes that the solution provides.

[*] Tool support may allow inhering metadata from the pattern to the forms whenever needed, e.g. author and sources of *Failure Alerts Provided* could be easily created by default inheriting from the pattern.

Requirement Pattern *Failure Alerts*	**Description**	*This pattern expresses the need of a software solution for having the capability to inform its users about failures*	
	Comments	*The alert is supposed to be issued at the moment the failure occurs.*	
	Pattern goal	*Alert the users about failures*	
	Author	*Oscar Mendez-Bonilla*	
	Sources (0..*)	• *Requirement books from CITI: CITIxxx-aa, CITIyyy-bb* • *Specialized literature: Pressman chap. 15, …*	
	Keywords (0..*)	*Alert, Failure, Crash*	
	Dependencies (0..*)	*IMPLIES: Failure Reports*	
Requirement Form *Failure Alerts Provided*	**Description**	*This form does not establish any relationship among the types of alert and failure.*	
	Comments	*Each extension may be applied just once*	
	Version	*Wed, 26/11/2008 - 2:25am*	
	Author	*Oscar Mendez-Bonilla*	
	Sources (0..*)	Same as above	
	Fixed Part	**Form Text**	*The solution shall give an alert in case of failure*
		Question Text	*Can your solution give an alert in case of failure?*
	Extended Part *Alert Types*	**Form Text**	*Alerts provided by the solution shall be: AL*
		Question Text	*Can the alerts provided by the solution be: AL?*
		Parameter	**Metric**
		AL: is a non-empty set of alert types	*AL: Set(AlertType)* *AlertType: {E-mail, SMS, Page, Fax, Skype, IM, …}*
	Extended Part *Failure Types*	**Form Text**	*Failures to be alerted of shall be: FL*
		Question Text	*Can the failures to be alerted be: FL?*
		Parameter	**Metric**
		Requirement Form *Failure Alerts Provided*	*FL: Set(FailureType)* *FailureType: {Server Crash, Network Crash, …}*
Requirement Form *Alert Types Dependent on Failure Types*	**Description**	*This form establishes a dependency among the types of alert and of failure that occurs.*	
	Comments	*The extensions may be applied more than once*	
	Version	*Wed, 26/11/2008 - 2:45am*	
	Author	*Carme Quer*	
	Sources (0..*)	Same as above	
	Fixed Part	**Form Text**	*The solution shall give alerts of a certain type depending on the type of failure*
		Question Text	*Can your solution give alerts of a certain type depending on the type of failure?*
	Extended Part *Specific Dependence*	**Form Text**	*An alert of one of the types AL shall be provided for a failure of some one of the types FL*
		Question Text	*Can your solution give alerts of one of the types AL for a failure of some of the types FL?*
		Parameter	**Metric**
		AL: is a non-empty set of alert types	*AL: Set(AlertType)* *AlertType: {E-mail, SMS, Page, Fax, Skype, IM, …}*
		FL: is a non-empty set of failure types	*FL: Set(FailureType)* *FailureType: {Server Crash, Network Crash, …}*

Fig. 4. Template and requirements pattern example

3.1.6. *Question text*

Also, each fix or extended part has associated a question text. The question text corresponds to a rewriting of the form text of the part in an interrogative style and will be used in case of selecting a requirement pattern part for being considered during the call-for-tender process. From the point of view of the structure of the question, it must be a natural language interrogative sentence that must include the same parameters than its corresponding form. For example, in the case of the extended part *Specific Dependence* the question text includes the same two parameters of the form text of this extended part.

3.1.7. *Dependencies*

Requirements patterns do not live isolated; they may be interrelated in the catalogue. We have identified two types of dependencies, among requirement patterns and among parameters. Dependencies among requirement patterns generalize the well-known idea of having dependencies among requirements [Egyed and Grünbacher (2004)][ISO/IEC Standard 9126-1]. These dependencies may be used during the elicitation process (e.g., to help determining the application order) and also they may be propagated to the requirements specification to improve traceability, e.g, if the requirements pattern catalogue has reported that the achievement of a requirement influences on the achievement of another one.

On the other hand, dependencies among parameters may help simplifying the process and enforcing the correctness of the resulting requirements books, since they will declare relationships among the values of different parameters that must be fulfilled. For example, given a requirement pattern "The users' manual shall be written in <*name-of-manual-language*>" and "The help-desk service shall give assistance in <*name-of-desktop-language*>", being both parameters declared of type *OfficialLanguage* (enumerated metrics with values {English, Spanish, …}), the relationship may state as default value of the second parameter the same language for the first one chosen.

3.1.8. *Classification schemas*

To facilitate their comprehension and reuse during the elicitation process, the patterns in the catalogue need to be indexed following some hierarchical classification schema. Currently we have two of these hierarchies introduced in the repository, which are the ISO 9126-1 catalogue [Robertson and Robertson (1999)] and a classification schema, based on the Volere approach [32] and on empirical experiments of CITI, but we could add other schemas (see Fig. 5). The reason of having several classification schemas is for improving both the usability and portability of the repository: usability, because the same catalogue may be used with different classification schemas by the same requirements engineer; portability, because different requirements engineers, used to other standards or even their own, customized classification schemas, may view the requirements patterns catalogue with their own perspective.

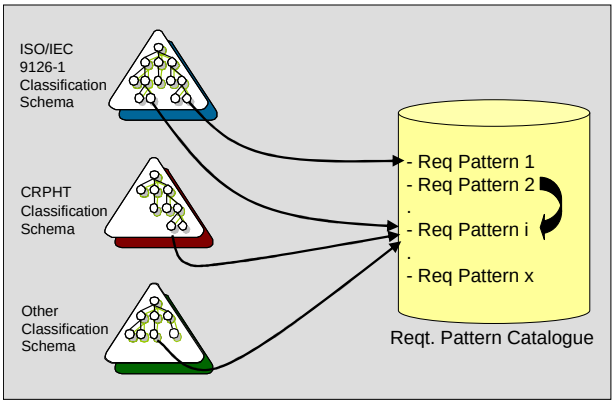

Fig. 5. Organization of the Requirements Patterns Catalogue: Patterns, Dependencies and Classification Schemas

3.2. *Construction of the Requirement Pattern Catalogue*

Reflecting the typical distinction among functional and non-functional requirements, we have built a pattern catalogue that is composed of functional and non-functional requirements patterns. Non-functional patterns are mainly domain-independent, whilst functional patterns depend on the domain. In this first stage, we have focused on (1) non-functional patterns, and (2) functional patterns for the particular software domain of ERP systems, and more specifically for the Sales module of ERP systems. At the moment of writing this paper, the non-functional part is more stable than the functional one, thus our comments are applicable just for this type of patterns.

We have obtained this first version of the non-functional part of the patterns catalogue from the analysis of 7 requirements books coming from projects driven by the CITI. We used a generalization process to obtain each requirement pattern: when several requirements were identified in different requirement books intending to state the same (or very similar) goal in the resulting software system, we defined a requirement pattern. The final definition of this first version of the patterns catalogue was decided taking into account not just the requirements books, but also the description of the specific approach used for the elicitation of requirements, literature review about the particular type of requirement addressed by the pattern, and expert judgment from requirement engineers of the CITI.

It is worth to remark that we do not aim at defining a closed catalogue, but an evolving one, that will take profit of the information obtained from the use of the patterns in industrial projects. This will facilitate the inclusion of new patterns, the evolution of others (especially adding new forms and extensions when needed), and the elimination of those patterns that finally become useless. A good example of this fact has been introduced in the research methodology as presented in the first section of the paper, where the current catalogue has been obtained as an evolution of the former one.

4. PABRE: A Method for Pattern-Based Requirements Elicitation

In this section we present the method that explores the requirement pattern catalogue to produce a requirements book to be used in call for tender processes.

4.1. *Assumptions*

We have designed a method to apply to patterns from the catalogue in order to extract requirements based on those patterns. The requirements are collected in the requirements book. Although it is expected that most of the requirements will come directly from the instantiation of the patterns, other situations may occur: (1) a pattern has to be slightly modified when becoming a requirement, probably because some forms or extension of the pattern is missing; (2) some requirement cannot be created as a pattern instantiation, either because the requirement is very specific of the project, or because the catalogue is still not complete enough.

Requirements elicitation becomes a process of search in, and pick-up from, the requirement patterns catalogue. Eventually, this process could be basically executed by the customer him/herself, but in our case, it is performed through interviews between an IT consultant and a customer (or a representative of the customer). All decisions are agreed between the IT consultant and the customer.

The requirement patterns, as information entities, are atomic (they cannot be partially applied), do not overlap and cannot be merged into one single requirement.

The classification schemas are comprehensive in the sense that their leaves cover all the relevant types of non-functional requirements (according to literature and to our experience in OTS selection projects).

Before starting the exploration of patterns, the IT consultant chooses a classification schema that will guide his/her exploration. Usually, if not always, the IT consultant will choose the classification schema he/she is most familiar with.

At the beginning of the process, the IT consultant explains to the customer the procedure that will be followed. This information makes the customer more aware of what's going on. In particular, the customer must get two messages: (1) requirements flow: patterns will be explored according to the selected classification schema: "standardized requirements" (i.e., patterns) will be proposed for a given scope of non-functional requirements (i.e., a set of related classifiers) and when all "standardized requirements" are explored for a given scope, then the scope will change and new "standardized requirements" will be proposed; (2) individual processing: for each pattern there will be a well-defined sequence of steps that will take place systematically.

4.2. *Steps of the Process*

Here we describe the different steps of the process, which are represented in Fig. 6. Two phases are distinguished: the requirements elicitation itself, which consists of five steps iteratively applied decomposed into activities (Fig. 7 shows their detail); and the catalogue evolution phase, which maintains the pattern-related knowledge up-to-date.

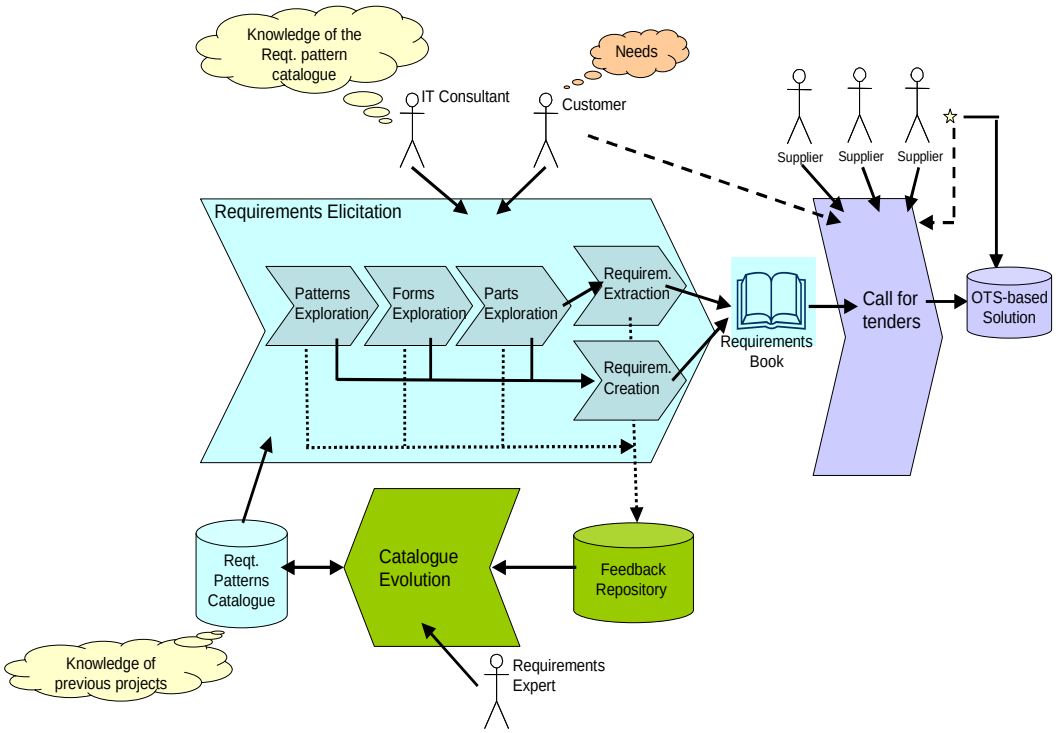

Fig. 6. Pattern-based process for OTS-based solution selection.

4.2.1. *Pattern Exploration*

At each iteration, the IT consultant starts by selecting the next applicable pattern according to the current classification criterion (step S.0 in Fig. 7), and once checked the comments about this pattern (if any), he/she explains the description and goal of the pattern to the customer (S.1). Based on this explanation, the IT consultant asks the customer to define the importance of the pattern (decision D.1 in Fig. 7). If the customer considers the pattern as not important for him/her, the IT consultant determines with the customer whether the pattern matches customer's needs (D.2). This decision allows a quick way to skip the pattern without processing its elements entirely (S.2).

Skipping patterns at this step is only allowed when the customer considers that the pattern goal does not match any customer need (by rating the importance as "low"). When patterns are skipped, the IT consultant collects the reason for skipping for further qualitative analysis of the patterns catalogue (feeding the feedback repository, see Fig. 6). Then the IT consultant proceeds to the next pattern according to the classification. Eventually, some other information could be used for choosing the next pattern (dependencies, keywords, or even the dynamics of the elicitation process), but we have not explored this issue yet.

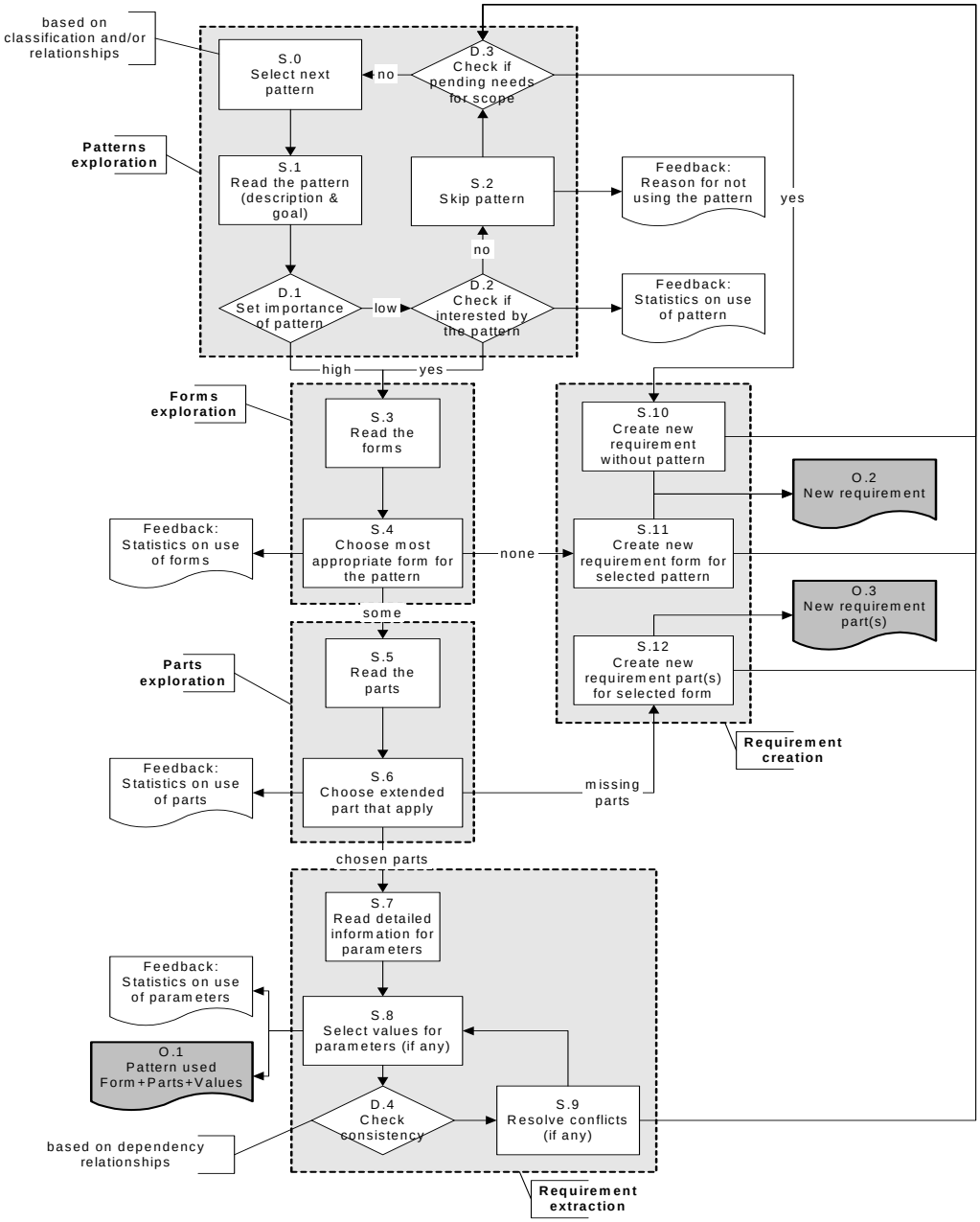

Fig. 7. Detail of the activities that take place during the requirements elicitation phase.

If the pattern selected in this iteration is the last one bound to the current scope (in subsection A we defined "scope" as a set of related classifiers), before changing scope, the IT consultant will ask the customer if there are still some needs related to this scope that have not been covered with the patterns (D.3). If this is the case, it is necessary to create one or more requirements from scratch (see Requirement Creation activity below).

4.2.2. *Forms Exploration*

If the customer chooses the pattern, this means that some requirement(s) bound to the pattern goal must appear in the requirements book. The IT consultant explains the different forms of the pattern, based on their descriptions (S.3). The customer then chooses the most appropriate form to achieve the goal of the pattern according to his/her context (S.4). The description of the chosen form can already be considered as part of the requirement that will be included in the requirements book. If no existing form suits the customer, the IT consultant will need to elaborate the requirement(s) in order to satisfy the pattern goal; this information is also considered as feedback for the Catalogue Evolution phase.

4.2.3. *Parts Exploration*

If some existing form has been selected, the IT consultant explains the different extended parts composing the chosen pattern form (S.5). The consultant briefly explains each parameter and gives example of possible values (found in the metrics description). The customer chooses the most convenient parts to achieve the patterns goal according to its context (S.6). The customer may choose more than one extended part for a specific form, even one extended part can be applied more than once with different assignments of values to parameters (examples of both situations may be found in the template example presented in Fig. 4). Eventually some extension not existing in the catalogue may be needed, again it becomes necessary to elicit the missing needs separately.

4.2.4. *Requirement Extraction*

For the chosen parts, the IT consultant gives more details about the parameters that apply, e.g. details on possible correctness conditions, dependencies to/from other parameters, and explains the exhaustive list of values for each parameter (S.7). Then the customer chooses the values for the parameters (S.8). The requirement is extracted by applying the pattern text of the selected parts with the parameters' values that have been agreed between customer and consultant. Several pattern texts from the different extensions applied can be concatenated to extract one requirement. Using the template example of Fig. 4, a possible requirement coming from the first form and applying the two extensions once, could be: "The solution shall give an alert in the case of failure. Alerts provided by the solution shall be: E-mail, SMS. Failures to be alerted of shall be: ServerCrash, NetworkCrash."

During the choice of values, the dependency relationships will be checked, in order to verify the consistency between parameters and even between patterns (D.4). The different

types of dependencies have to be taken into account (in the case of patterns: conflicts, synergies, etc.; in the case of parameters: value dependencies and arbitrary formulae), as well as their direction (from which pattern to which other, i.e. the dependency may be upon a pattern already considered or to some pattern still not considered during the process). When the IT consultant detects a conflict or an inconsistency, he/she warns the customer and they try to solve the conflict (S.9). Conflict resolution may be not straightforward and may even force to reconsider requirements already agreed; we do not tackle this issue here.

4.2.5. *Requirement Creation*

From the phases above, it is clear that in different situations, the extraction of a requirement from the catalogue is not direct. To sum up, we have identified the following situations:

- The patterns bound to some scope do not cover all the customer needs related to that scope. The IT consultant uses the information about the classification criteria of the scope to guide a classical requirements elicitation process (S.10).
- A pattern has been considered applicable but none of its forms fits well the needs of the customer. The IT consultant will work with the customer until a good way of expressing the goal of the pattern matching the needs of the customer is found (S.11).
- A pattern form has been chosen but some detail is needed that is not captured by any existing extensions. The IT consultant will work with the customer eliciting all the details needed to complete the requirement with all the relevant information (S.12).

In all the cases, the requirement (or part of requirement) added to the requirement book is provided as feedback to the requirements expert for the next phase, Catalogue Evolution. Also, a question for that requirement must be considered in the call-for-tender process, typically by rephrasing the requirement into an interrogative form. As an example, the typical pattern text written as: “The system shall provide X”, will usually be converted into “Is your system able to provide X”.

4.2.6. *Catalogue Evolution*

After the IT consultant has driven the requirements elicitation process and the requirements book for the call-for-tender process is complete, the knowledge gained in this project must be capitalized in the requirement pattern catalogue. As remarked in the activities above, the IT consultant will collect the information useful for this purpose, both failures and success on patterns application. For failures, the last bullet (Requirement Creation) summarizes the situations that may be encountered in which the catalogue does not contain all the information needed in this project at hand. For successes, each application of a pattern is registered. We remark that there are cases in the middle, e.g. when a requirement is applied but there is not form that captures its goal in the appropriate terms, this is a success from the pattern point of view (i.e., the pattern

has been chosen and applied), but a failure from the form point of view (i.e., a form was still missing).

The different actions that the requirements engineer may take to enlarge the catalogue, and the situations in which these actions may be taken, are:

- Promote a requirement into requirement pattern. When a requirement has been written from scratch (see first bullet in Requirements Creation activity).
- Create a new form. When a requirement comes from a pattern goal but without applying any existing form (see second bullet in Requirements Creation activity).
- Create a new extension. When a requirement is expressed using some form but the details needed were not contained in any existing extension (see third bullet in Requirements Creation activity).
- Extend some existing extension. Remarkably when an extension contains some metric of a domain defined by enumeration of its values, and some other values not in domain have arose in the project.

But not just enlargement is possible, also some other operations can be applied over the catalogue after updating the statistics about it with the data of this project:

- Removal of unused patterns, forms or extensions. When after this project some threshold has been exceeded, the removal from the catalogue of those pieces of information that do not seem to be relevant in call-for-tender processes may be considered.
- Refactoring of the catalogue. For instance, changing the order in which forms of a pattern are considered (most used forms first), or even more fundamental changes like splitting or joining some patterns.

The decision on whether or not to take the actions is up to the requirements expert, probably checking with IT consultants before taking the decision. At the current stage of our research, we still do not have concrete advices about the conditions to apply these actions.

5. The Use of the Requirement Patterns Catalogue in the PABRE Process

At the time being, the validation carried out has been twofold: on the one hand, internal validation with requirements engineers; on the other hand, external validation in an industrial project. As already mentioned, the current form of the catalogue embraces 29 patterns that have been classified according to two classification schemas, the ISO/IEC 9126-1 standard and the experience-based classification schema developed at CITI. The current coverage of both classification schemas is quite similar: the catalogue covers 13 out of the 27 subcharacteristics of the ISO/IEC 9126-1 software quality model, and 29 out of 58 of the experience-based classification schema. Table 2 provides a more detailed view of one of the two cases. It may be observed that one of the patterns has been classified in two different subcharacteristics. We may expect other cases where this fact will appear.

Table 2. Pattern classification using ISO/IEC 9126-1.

Functionality	
Suitability	-
Accuracy	Precision
Interoperability	Data Exchange, Interoperability with External Systems
Security	Authentication, Authorization, Automatic Logoff, Stored Data Protection, Data Transmission Protection
F. Compliance	-
Reliability	
Maturity	Failure Alerts
Fault Tolerance	Alternative Data Storage, Downtime, Uptime, Availability
Recoverability	Log, Backup
R. Compliance	-
Usability	
Understandability	Interface Language, Interface Type
Learnability	Online Help, Interface Learnability, Documentation
Operability	Installation Procedures, Recovering Procedures, Update Procedures, Failure Alerts
Attractiveness	-
U. Compliance	-
Efficiency	
Time Behaviour	Interface Load Time, Concurrent Users Capacity
Resource Behaviour	Data Capacity, Users Capacity
E. Compliance	-
Maintainability	
Analyzability	-
Changeability	-
Stability	-
Testability	-
M. Compliance	-
Portability	
Adaptability	Development Language
Installability	Platform
Coexistence	-
Replaceability	-
P. Compliance	-

5.1. *Presentation to Requirement Engineers*

We first presented the patterns, their structure and their classification criteria to requirements engineers with a wide experience in OTS-based selection processes. We proposed to them an initial version of the PABRE method.

They considered that we have a lot of interesting information about each pattern, but that most of that information should not be shown during the requirements elicitation, if it is not required. As for example metadata used for traceability and history purposes (sources from where the patterns were obtained, keywords or author of the pattern) are not necessary during the elicitation process.

Another conclusion was that it would be necessary to have a support tool for helping in the presentation and browsing of the catalogue and to hide unnecessary information to the IT consultant.

They contributed to the catalogue structure by noticing the possible dependencies among values of parameters of different requirement patterns. We already had dependencies among patterns, but we have not considered dependencies among parameter values (see dependencies in Section III). They also required us that the process and its support tool could guide the IT consultant in these cases. For example, advising the IT consultant which would be the most appropriate value for a parameter, taking into account the value previously assigned to another parameter.

5.2. *Case study 1: Non functional requirements for a digital library project*

At the early development of the non-functional patterns, we experienced a trial use of the PABRE method in an industrial project. For this project we acted as IT Consultant Company whose mission was the design of a requirements book for the renewal of a digital library system for a customer.

The method was tested in one meeting dedicated to the elicitation of requirements. Before using the patterns catalogue in this meeting, the customer had already identified the scope, the goal and the future actors of the digital library system.

During the meeting, a research engineer took the role of IT consultant. From the customer side, the project manager, a usability expert, a business expert and the head of the IT department took part in the meeting. The classification schema chosen to browse the patterns was the experience-based schema, since both the IT consultant and the customers were used to it.

During this meeting 21 of the 48 patterns of the first version of the catalogue were explored. From them, 17 patterns generated one or more requirements. Specifically 3 of them were applied twice. Also 2 new requirements, not coming from the pattern catalogue, were added in relation to classifiers of the experience-based classification schema. In other words, from 22 requirements included in the requirements book after this first meeting, 20 came from the catalogue, i.e. more than 90%. This was a very positive indicator of the quality of the patterns initial catalogue.

If we talk specifically about the PABRE method, first of all as productivity rate, producing these 22 requirements took 80 minutes, which does not seem a bad figure

especially taking into account that also for the IT consultant it was the first real application of the method. The customer team also suggested of utmost importance to have tool support to automatically generate the requirements book from the patterns application. This was considered a key success factor.

This first case study helped us to improve both the contents and the structure of the patterns catalog. As for the content improvement, redundancies were eliminated by merging patterns (thus reducing the catalog from 48 to 29 patterns). Regarding the structure, rules were added to control and drive the usage of the pattern's forms and parts.

5.3. *Case study 2: Non functional requirements for a CRM SaaS project*

The second case study took place after improving the patterns catalog structure and contents. We started with the catalog of 29 patterns already available as refinement of the first version. Again, we acted as an IT Consultant Company to elicit non-functional requirements. In this project the customer was willing to design a requirements book to drive the selection of a Customer Relationship Management (CRM) solution. An important constraint in the project was the necessity to select a CRM solution hosted by the provider (Software as a Service, SaaS, or Application Service Provider model).

Before carrying out the elicitation of non-functional requirements, we helped the customer to identify the main goals and functional requirements for the CRM solution.

The method was tested in one meeting dedicated to the elicitation of requirements. For some organisational reasons (working language different from the language of the catalogue), the process was not applied as-is during the meeting with the customer, but rather applied *a posteriori* after the meeting. However we used the patterns catalogue as a basis to drive the interview meeting. As a first step for the IT consultant, we discarded 8 patterns that were not applicable due to the project's context of SaaS solution selection (step S.2 in the process). We chose the experience-based classification schema to browse the catalogue.

During the meeting, two research engineers took the role of IT consultants to elicit non-functional requirements. From the customer side, the project manager, and five users representative were present. In comparison to the first case study, none of the customer's individuals was particularly skilled in IT.

The meeting lasted 2 hours during which we explored 16 patterns. 8 Patterns were not considered due to the lack of time (pending to explore in a next meeting).

28 parts out of 17 forms were applied as is (O.1 resulting from S.8 in the process) and 5 requirements were extracted by adding an extra parameter in the extracted part (O.1 resulting from S.9 before S.8 in the process). For 3 patterns, new requirements were identified at the level of a pattern's part (Step S.13 in the process). S.10 and S.11 were not applied at all, giving us a good confidence in the completeness of the current pattern catalogue.

Usage rules were overridden 3 times: for 3 patterns the fixed parts were not used because of their lack of precision; the extended parts were preferred. Further research for considering this fact is needed.

This case study confirmed the necessity to have a tool supporting the exploration of patterns and extraction of requirements so that to improve the efficiency of the process.

6. Analysis of the Method

6.1. *Possible benefits*

The possible benefits for consultants are the reduction of time spent to perform the elicitation of the requirements and the improvement of the quality of the requirements book obtained.

For requirements engineering experts, a benefit from using this method is a database of requirements elicitation experiences that might be used for statistical analysis.

6.1.1. *Faster requirements elicitation process*

IT consultants use the former requirements' elicitation process in consultancy projects over 10 man-days for the requirements elicitation part (and between 15 to 25 man-days for the overall OTS selection, including call-for-tender processes). With this patterns-based requirements' elicitation process we aim at downsizing consultancy project time down to 4 or 5 man-days for the requirements elicitation part.

The reduction of time comes from the fact that patterns offer "ready to use" requirements and that the catalogue covers the most common non-functional requirements. Then the consultant spends less time on the elicitation of the requirements. Also, the catalogue and the process have been designed to help the engineer and the user to choose requirements in a faster way, since the most frequent output (use of the pattern standard or ancestor forms) has the shortest decisional path.

6.1.2. *Higher quality of the requirements book*

The IEEE-830 standard [IEEE Recommended Practice for Software Requirements Specification (1998)] describes recommended approaches for the specification of software requirements (SRS). Among the recommendations it gives a set of characteristics that must have a good SRS. These characteristics are: Correctness, Unambiguity, Completeness, Consistency, Importance and/or Stability Ranking, Verifiability, Modifiability, and Traceability. In this subsection we justify how the use of the PABRE method and the pattern catalogue may drive to a good requirements book taking into account the characteristics of this standard (see Table 3 for a summary).

Correctness. As the standard states, the achievement of this characteristic does not depend on any tool or any procedure. However, the participation of the customer in the elicitation process enforces its achievement.

Unambiguity. The idea is that the patterns catalogue will be unambiguous, since it is the result of the study of multiple requirements books after a reviewing and a rewriting process. Taking into account this pre-processing, we could eventually guarantee that the requirements extracted from patterns will very rarely have any ambiguity; goals may play

an important part in this validation. However, this is not possible to be ensured for all the requirements books, since it may include new requirements created during the elicitation process not directly coming from patterns.

Completeness. If we provide a complete patterns catalogue, we may contribute to obtain complete requirement books. This could be possible after some time of applying the catalogue, and once arrived to a stable catalogue. However, we think that we may never have a strictly complete patterns catalogue since there may always exist the need of requirements that are very specific of projects, and it would not have sense have them as patterns.

Table 3. Summary of quality characteristics addressed by the use of patterns

IEEE-830 Characteristics		**Addressed**
Correctness	An SRS is correct if, and only if, every requirement stated therein is one that the software shall meet.	No
Unambiguity	An SRS is unambiguous if, and only if, every requirement stated therein has only one interpretation.	Partially
Completeness	An SRS is complete if, and only if, it includes: all significant requirements, definition of the responses of the software to all realizable classes of input data in all realizable classes of situations and definition of terms and full labels and references to all figures, tables, and diagrams.	Partially
Consistency	An SRS is consistent if, and only if, no subset of individual requirements described in it conflict.	Partially
Importance and/or Stability Ranking	An SRS is ranked for importance and/or stability if each requirement in it has an identifier to indicate either the importance or stability of that particular requirement.	No
Verifiability	An SRS is verifiable if, and only if, every requirement stated therein is verifiable. A requirement is verifiable if, and only if, there exists some finite cost-effective process with which a person or machine can check that the software product meets the requirement.	Yes
Modifiability	An SRS is modifiable if, and only if, its structure and style are such that any changes to the requirements can be made easily, completely, and consistently while retaining the structure and style.	Yes
Traceability	An SRS is traceable if the origin of each of its requirements is clear and if it facilitates the referencing of each requirement in future development or enhancement documentation.	No

Consistency. In our patterns catalogue, that we have pre-processed to guarantee that is consistent, we explicit any dependency that can exists among patterns, or parameter

values of patterns. If these dependencies are taken into account during the elicitation process, we may guarantee that the subset of requirements of the requirement book that have been extracted from the catalogue are consistent. However, as in the case of unambiguity, we may not guarantee the consistency of the whole book, since there may have requirements created during the elicitation process.

Importance and/or Stability Ranking. This characteristic does not depend of the use of requirement patterns. However, this classification was already done in the previous process, and it is maintained in the new one.

Verifiability. The way in which we propose to write the requirement patterns will drive to verifiable requirements that do not contain any non-definable or non-evaluable terms. Specifically, when a pattern is applied the parameters of the pattern must take some concrete value that will ensure the verifiability of the corresponding requirements.

Modifiability. The requirements book obtained will have an easy-to-use organization corresponding to the chosen classification schema, and the use of the patterns catalogue will guarantee that all requirements that will be extracted from a pattern of the catalogue are not intermixed.

Traceability. On the one hand, the backward traceability of requirements in the requirement book, which were extracted from the patterns catalogue, is partially guaranteed by the catalogue because in the catalogue we maintain the sources from were the patterns were derived. However, the other references to the origin taking into account earlier documents of the project, or the backward traceability of new requirements, do not depend on the use of the catalogue. On the other hand, the forward traceability with a unique name or reference number given to requirements was already done in the previous process, and it is maintained in the new one.

6.1.3. *Statistics on Requirements Engineering projects*

The fact of using a systematic process for performing requirements elicitation allows the collection data for statistical analysis.

For a given requirement engineering project, the divergence of a project can be identified when patterns are often skipped or new requirements are created apart from pattern elements.

For a set of requirements engineering projects using the pattern catalogue, quality of a pattern can be identified when the first form is preferred to the other form.

For a set of requirements engineering projects using the patterns catalogue, completeness of the patterns catalogue can be identified when the pattern forms are preferred to the creation of new requirements.

6.2. *Possible Drawbacks*

6.2.1. *Heaviness of the process*

The process may be “heavy” for inexperienced IT consultants that discover the catalogue and that are more used to collect requirement in a less driven manner. It is then necessary

to plan an initial training on the concept of requirements patterns and on the navigation throughout the catalogue. For this last necessity, goal matching or faceted descriptions using keywords could be explored in the future.

Even experienced consultants can find inefficient the fact of processing the entire catalogue during one interview rather than identifying requirements in an exploratory way. To tackle this issue one may consider pre-selecting patterns and parameters before the interview, on the basis of information regarding the current IT infrastructure and IT strategy of the customer. These pieces of information are usually collected before the requirements elicitation interviews. After this "pre-selection" of patterns the consultant only needs to confirm his/her analysis with the customer. However this may introduce a bias since requirements are no more elicited from the customer but deducted by the consultant.

7. Conclusion and Future Work

In this paper we have presented PABRE, a pattern-based method to ease the requirements elicitation process for OTS selection projects. Early feedback from IT experts and an ongoing case study give us confidence in the fact that this approach can increase efficiency of requirements elicitation as well as quality of the produced requirements.

The main contributions of our approach are:

- PABRE is an OTS selection method customized to the particular case of call for tender projects. This characteristic makes it different from other well-known existing methods like OTSO, PORE, CARE or others filling thus an existing gap in the current state of the art.
- PABRE is a method that tries to fit the reality of industrial OTS selection projects. It has been designed with the knowledge acquired from practice and does not try to impose methods or techniques that may be difficult to adopt in practice like multi-criteria decision-making techniques, goal-oriented reasoning, etc.
- PABRE supports knowledge capitalization by means of a reuse infrastructure based on practice. "Based on practice" means that the knowledge reused comes from past experiences, and the updating of this knowledge is integrated into the PABRE process itself.
- PABRE is highly customizable to the specific needs of IT companies and organizations. On the one hand, we have used a particular pattern structure but others could be equally valid. On the other hand, the separation among the catalogue itself and the classification schemas used to browse it allows applying it to different realities.

Future work focuses especially on gaining experience. In spite of the early feedback mentioned above, we need to assess the efficiency of the PABRE method and the completeness of the catalogue with more case studies. To do so, we plan to gradually propose the use of the PABRE method to IT consultants members of the CASSIS network (a network of certified IT consultants in Luxembourg and surrounding area

[Renault et al. (2007)]), before generalizing it to every project of software selection performed by an IT consultant of this network.

Eliciting non-functional requirements has been a good starting point for capitalizing knowledge, since these requirements do not vary much from a project to another. In a near future we plan to extend the catalogue with patterns related to non-technical requirements and functional requirements. Given the nature of non-technical requirements [Carvallo et al. (2006)], we think that the situation will be similar to non-functional ones, i.e., the identified patterns will be mostly domain-independent (although distinctions in the type of software, e.g. OSS vs. COTS components) may yield to some variability. As for functional requirements patterns, we have already mentioned that they are domain-dependant. We are currently focusing on the ERP systems domain, starting by one particular module (Sales module), given that this type of component is focus of many consultant-assisted projects. Other business-application-related domains, i.e. CRM or DM systems, will be next targets.

Some ideas regarding the improvement of the catalogue have been proposed in the catalogue evolution part of section IV. As soon as we will have a sufficient amount of feedback from usages of the catalogue, we intend to formalize a more precise method for managing the evolution of patterns in the catalogue with the feedback material repository.

Acknowledgments

This work has been partially supported by the Spanish project ref. TIN2007-64753.

References

C. Alexander. *The Timeless Way of Building*. Oxford Books, 1979.

V.R. Basili, B. Boehm. "COTS-based Systems Top 10 List". *IEEE Computer*, 34(5), May 2001.

D. Carney, F. Leng. "What do you mean by COTS? Finally, a useful answer". *IEEE Software*, 17(2), March-April 2000.

J.P. Carvallo, X. Franch, C. Quer. "Managing Non-Technical Requirements in COTS Components Selection". In *Proceedings of the 14th IEEE International Requirements Engineering Conference* (RE'06), 2006.

B.H.C. Cheng, J.M. Atlee. "Research Directions in Requirements Engineering". In *Proceedings of the 29th IEEE International Conference on Software Engineering* (ICSE'07), Minneapolis, Minnesota (USA), 2007.

L. Chung, B. Nixon, E. Yu, J. Mylopoulos. *Non-functional Requirements in Software Engineering*. Kluwer Publishing, 2000.

J.L. Cybulski, K. Reed. "Requirements Classification and Reuse: Crossing Domain Boundaries". *In Proceedings of the 6th International Conference on Software Reuse* (ICSR-6), Vienna (Austria), 2000.

A. Durán, B. Bernárdez, A. Ruíz, M. Toro. "A Requirements Elicitation Approach Based in Templates and Patterns". In *Proceedings 2nd Workshop on Requirements Engineering* (WER'99), 1999.

A. Egyed, P. Grünbacher. "Identifying Requirements Conflicts and Cooperation: How Quality Attributes and Automated Traceability Can Help". *IEEE Software*, 21(6), November-December 2004.

M. Fowler. *Analysis Patterns*. Addison-Wesley, 1997.

M. Galster, A. Eberlein, M. Moussavi. "Matching Requirements with Off-the-shelf Components at the Architectural Level". In *Proceedings of the 2nd International OTS-Based Development Methods Workshop* (IOTSDM'07), Banff, Alberta (Canada), 2007.

E. Gamma *et al*. *Design Patterns*. Addison-Wesley, 1995.

ISO/IEC Standard 9126-1. *Software Engineering – Product Quality – Part 1: Quality Model*, 2001.

S. Konrad, B.H.C. Cheng. "Requirements Patterns for Embedded Systems". In *Proceedings 10th IEEE International Requirements Engineering Conference* (RE'02), Essen (Germany), 2002.

J. Kontio. "A case study in applying a systematic method for COTS selection". In *Proceedings of the 18th IEEE International Conference on Software Engineering* (ICSE'96), Berlin (Germany), 1996.

M. Krystkowiak, B. Bucciarelli. "COTS Selection for SMEs: a Report on a Case Study and on a Supporting Tool". In *Proceedings of the 1st International Workshop on COTS and Product Software: Why Requirements are so Important* (RECOTS'03), Monterey, California (USA), 2003.

M. Krystkowiak, V. Betry, E. Dubois. "Efficient COTS Selection with OPAL Tool". In *Proceedings of the 1st International Workshop on Models and Processes for the Evaluation of COTS Components* (MPEC'04), Edinbourgh, Scotland (UK), 2004.

W. Lam, T.A. McDermid, A.J. Vickers. "Ten steps towards systematic requirements reuse". In *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering* (ISRE'97), Annapolis, Maryland (USA), 1997.

S. Lauesen, J.-P. Vium. "Experiences from a tender process". In *Proceedings of 10th Workshop on Requirements Engineering: Foundation for Software Quality* (REFSQ'04), Riga (Latvia), 2004.

J. Li, F.O. Bjornson, R. Conradi, and V. By Kampenes. "An Empirical Study of Variations in COTS-Based Software Development Processes in the Norwegian IT Industry". *Journal of Empirical Software Engineering*, 11(3), 2006.

J. Li *et al*. "A State-of-the-Practice Survey of Risk Management in Development with Off-the-Shelf Software Components". *IEEE Transactions Software Engineering* 34(2), 2008.

T.R. Madanmohan, D. Rahul. "Open Source Reuse in Commercial Firms". *IEEE Software*, 21(6), November-December 2004.

N. Maiden, C. Ncube. "Acquiring Requirements for COTS Selection". *IEEE Software*, 15(2), 1998.

D. Matheson, I. Ray, I. Ray, S. H. Houmb. "Building Security Requirement Patterns for Increased Effectiveness Early in the Development Process". In *Proceedings of Symposium on Requirements Engineering for Information Security* (SREIS'05), 2005.

O. Mendez, X. Franch, C. Quer. "Requirements Patterns for COTS Systems". In *Proceedings of the 7th International Conference on Composition-Based Software Systems* (ICCBSS 2008), 2008.

A. Mohamed, G. Ruhe, A. Eberlein. "COTS Selection: Past, Present, and Future". In *Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems* (ECBS'07), Tucson, Arizona (USA), 2007.

B. Moros, C. Vicente, A. Toval. "Metamodeling Variability to Enable Requirements Reuse". In *Proceedings of 13th International Workshop on Exploring Modeling Methods for Systems Analysis and Design* (EMMSAD'08), Montpellier (France), 2008.

K. Pohl, G. Böckle, F. van der Linden. *Software Product Line Engineering*. Springer Verlag, 2005.

S. Renault, B. Barafort, E. Dubois, M. Krystkowiak. "Improving SME trust into IT consultancy: a network of certified consultants case study." In EuroSPI 2007 Industrial proceedings, 10.1 - 10.8., Potsdam, Germany, 2007.

S. Renault, O. Méndez, X. Franch, C. Quer. "PABRE: Pattern-Based Requirements Elicitation." In *IEEE Proceedings of the 3rd International Conference on Research Challenges in Information Systems* (RCIS), Fès (Morocco), May 2009.

S. Robertson. "Requirements Patterns Via Events/Use Cases". In *Proceedings Pattern Languages of Programming* (PLoP'96), Washington University Technical Report (#wucs-97-07), 1996.

S. Robertson, J. Robertson. *Mastering the Requirements Process*. Addison-Wesley, 1999.

IEEE Recommended Practice for Software Requirements Specifications. IEEE Computer Society, 1998.

I. Sommerville. "Integrated Requirements Engineering: A Tutorial". *IEEE Software*, 22(1), January-February 2005.

M. Torchiano, M. Morisio. "Overlooked Aspects of COTS-Based Development". *IEEE Software* 21(2), 2004.

S. Withall. *Software Requirements Patterns*. Barnes & Noble, 2008.