A Metamodel for Software Requirement Patterns†

Xavier Franch[1], Cristina Palomares[1], Carme Quer[1], Samuel Renault[2], François De Lazzer[2]

[1]Universitat Politècnica de Catalunya (UPC)
UPC – Campus Nord, Omega building, 08034 Barcelona (Spain)
{franch | cpalomares | cquer}@essi.upc.edu
[2]CITI, CRP Henri Tudor
29 avenue John F Kennedy, Luxembourg (Luxembourg)
{samuel.renault | francois.delazzer}@tudor.lu

Abstract. **[Context and motivation]** Software Requirement Patterns (SRP) are a type of artifact that may be used during requirements elicitation that also impact positively in other activities like documentation and validation. In our experiences, SRP show a great percentage of reuse for the non-functional requirements needed in call-for-tender requirement specifications. **[Question / problem]** We are facing the need of formulating the accurate definition of SRP for their use in call-for-tender processes to allow reasoning rigorously and know more about their semantics and applicability. **[Principal ideas / results]** In this paper we present a metamodel for SRP around three main concepts: 1) the structure of SRP themselves; 2) the relationships among them; 3) the classification criteria for grouping them. **[Contribution]** We provide a rigorous definition that shows the concepts that are of interest when defining and applying SRP.

Keywords: software requirement patterns, requirements reuse, metamodel.

1 Introduction

Reuse is a fundamental activity in all software development related processes. Of course, requirements engineering is not an exception to this rule [1]. The reuse of software requirements may help requirement engineers to elicit, validate and document software requirements and as a consequence, obtain software requirement specifications of better quality both in contents and syntax [2].

There are many approaches to reuse. Among them, patterns hold a prominent position. According to their most classical definition, each pattern describes a problem which occurs over and over again, and then describes the core of the solution to that problem, in such a way that it can be used a million times over, without ever doing it the same way twice [3]. Software engineers have adopted the notion of pattern in several contexts, remarkably related with software design (e.g., software design and architectural patterns), but also in other development phases, both earlier

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

and later. We are interested in the use of patterns for the software analysis stage, namely Software Requirement Patterns (SRP).

As [4] shows, there are not much proposals for SRP in the literature, in fact their exhaustive review lists just 4 catalogues out of 131, compared to 47 design catalogues and 39 architecture catalogues. Our own literature review has found some more approaches but still this unbalance is kept. The existing approaches differ in criteria like the scope of the approach, the formalism used to write the patterns, the intended main use of patterns and the existence of an explicit metamodel. Table 1 shows the classification of these approaches with respect to the mentioned criteria. In the last row we describe our own method as general-purpose, representing patterns in natural language, aiming at writing sofware requirements specifications (SRS) and metamodel-based.

About the two approaches that propose a metamodel, [8] focus on reuse of semi-formal models (e.g., UML class diagrams and sequence diagrams), thus the kind of concepts managed are quite different. Concerning [6], their focus is on variability modeling for handling the different relationships that requirements may have. From this point of view, it is a very powerful approach, but other aspects that we will tackle here, like the existence of different forms that a pattern may take, or multiple classification criteria, are not present in their metamodel.

Table 1. Comparison of approaches to software requirement patterns.

	Scope	*Notation*	*Application*	*Metamodel?*
[5]	General purpose	Natural language	Req. elicitation	Just templates
[6]	General purpose	Object models	Variability modeling	Yes
[7]	Business applications	Event-Use case	Identify patterns	No
[8]	General purpose	Semi-formal models	Writing req. models	Yes
[9]	Embedded systems	Logic-based	From informal to formal reqs.	No
[10]	Security requirements	UML class diagrams	Security goals elicitation	No
[11]	Security requirements	Natural language	Req. elicitation in SOC	No
[12]	General purpose	Natural language	Writing SRS	Just template
[13]	General purpose	Problem frames + *i**	Knowledge management	No
Ours	General purpose	Natural language	Writing SRSs	Yes

The idea of using SRP for reusing knowledge acquired during this stage arose from the work of the CITI department of the Centre de Recherche Publique Henri Tudor (CRPHT) on helping SME with no background in requirements engineering to handle requirements analysis activities and to design SRS in order to conduct call-for-tender processes for selecting Off-The-Shelf (OTS) solutions [14]. More than 40 projects ran successfully following the CITI methodology, but the only technique of reuse they applied was starting a new project by editing the most similar requirement book. These techniques demonstrated their weaknesses especially in relation to mobility of IT experts and consultants. It became necessary to provide better means to capitalize requirements in a high-level manner by creating reusable artifacts like patterns, supporting consultants' need of creating new SRS.

As a response to this need, we built an SRP catalogue with 29 patterns. The patterns were all about non-functional requirements since this type of requirements are the less sensitive to changes in the problem domain. The research method used to build this catalogue and the underlying metamodel was based on the study of SRS

from 7 call-for-tender real projects conducted by CITI; experts' knowledge, being these experts: IT consultants, CITI facilitators and UPC researchers; background on requirements engineering literature and especially on requirement patterns. We undertook then a first validation in two real projects. In this paper we focus on the metamodel, that is, the structure of our proposed SRPs and its classification to facilitate the selection of patterns. The PABRE process of application of SRP in the context of CITI and the validation of our current SRP catalogue have been described in [15], therefore neither the process nor the catalogue's content are part of the objectives of this paper.

2 Structure of a Requirement Pattern

The first fundamental question to answer is what the structure of a SRP is. Figure 1 shows an example of SRP that illustrates the most significant components. Note the statement of the goal as a kind of problem-statement of the pattern; goals play a crucial part in the PABRE method built on top of these patterns [15]. SRP metadata (e.g., description, author) are not included for the sake of brevity.

Requirement Pattern Failure Alerts			
Goal Satisfy the customer need of having a system that provides alerts when system failures occur			
Requirement Form *Heterogeneous Failure Alerts*	**Fixed Part**	**Template**	*The system shall trigger different types of alerts depending on the type of failure*
		Extended Parts Constraint	*multiplicity(Alerts for Failure Types) = 0..**
	Extended Part *Alerts for Failure Types*	**Template**	*The system shall trigger %alerts% alerts in case of %failures% failures*
		Parameter	**Metric**
		alerts: non-empty set of alert types	*alerts: Set(AlertType)* *AlertType: Domain of possible types of alerts*
		failures: non-empty set of failure types	*failures: Set(FailureType)* *FailureType: Domain of possible types of failures*
Requirement Form *Homogeneous Failure Alerts*	**Fixed Part**	**Template**	*The system shall trigger an alert in case of failure.*
		Extended Parts Constraint	*multiplicity(AlertsTypes) = 0..1 and* *multiplicity(Failure Types) = 0..1*
	Extended Part *Alert Types*	**Template**	*The solution shall trigger %alerts% alerts in case of failure*
		Parameter	**Metric**
		alerts: non-empty set of alert types	*alerts: Set(AlertType)* *AlertType: Domain of possible types of alerts*
	Extended Part *Failure Types*	**Template**	*The system shall trigger alerts in case of %failures% failures*
		Parameter	**Metric**
		failures: non-empty set of failure types	*failures: Set(FailureType)* *FailureType: Domain of possible types of failures*

Fig. 1. An example of software requirement pattern (parameters appear among '%')

Figure 2 shows the metamodel for SRP. It represents the metaclasses for the basic concepts that appear in the example above and others introduced later. We may observe that the concept represented by a *Requirement Pattern* may take different *Pattern Forms*. Each form is applicable in a particular context, i.e. it is the most appropriate form to achieve the pattern's goal in a particular type of software project. In the example of Fig. 1, the second form is more adequate if the types of alerts that the client wants in the system will be the same for all types of failures, if not the first form must be applied. Applying a SRP, then, means choosing and applying the most suitable form.

At its turn, each form has a *Fixed Part* that characterizes it which is always applied if the form is selected, together with zero or more *Extended Parts* that are optional and help customizing the SRP in the particular project. In general, extended parts must conform to some *Constraint* represented by means of a formula over some predefined operators (e.g., for declaring multiplicities or dependencies among extended parts, as *excludes*, *requires*). For instance, in the example we may see that the first form allows repeated application of its single extended part, whilst the second form allows one application at most of each of its extended parts (since in this form it has not sense to state more than once the types of alerts and failures).

Both fixed and extended parts are atomic *Pattern Items* that cannot be further decomposed. Each pattern item contains a *template* with the text that finally appears in the SRS when applied. In this text, some variable information in the form of *Parameters* may (and usually, do) appear. Parameters establish their *Metric*, eventually a correctness condition *inv*, and also may be *related* to other parameters (belonging to other patterns) such that they must have the same value; an example is the parameter *failures* that also appears in some form of other SRP in the catalogue, namely the pattern *Recovery Procedures*.

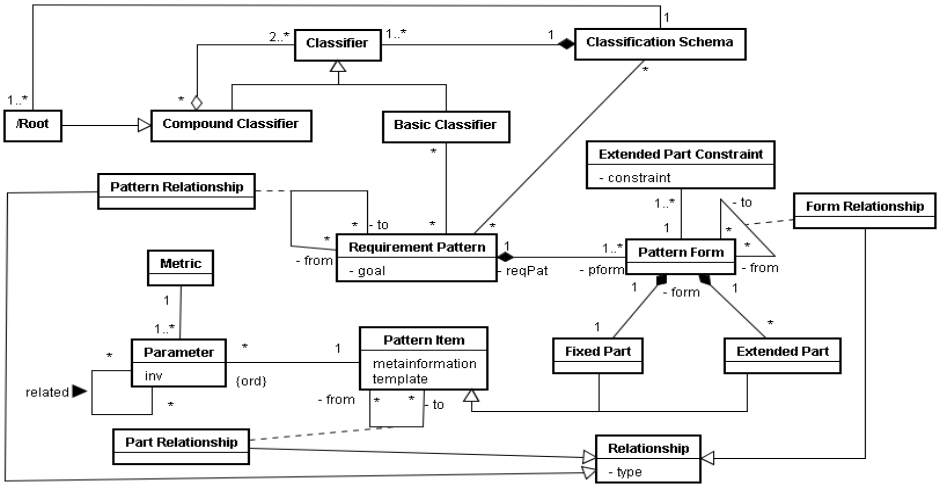

Fig. 2. The metamodel for software requirement patterns.

SRPs are not isolated units of knowledge, instead there are several types of relationships among them. For instance, Withall structures his SRP catalogue using a more detailed proposal of relationships, that may be purely structural like "has", "uses" and "is-a", or with a semantic meaning like "displays" and "is across" [12]. Even generic (unlabelled) relationships are used. A thorough analysis of the SRS written by CITI shows that relationships may appear at three different levels:

- *Pattern Relationship*. The most general relationship that implies all the forms and all the forms' parts of the related patterns.
- *Form Relationship*. A relationship at the level of forms implies all the parts of the related forms.
- *Part Relationship*. The relationship only applies to these two parts.

In any case, if *A* is related to *B* and *A* is applied in the current project, the need of applying or avoiding *B* must be explicitly addressed. The types of relationships are not predetermined in the metamodel to make it more flexible. The superclass *Relationship* includes an attribute to classify each relationship.

A fundamental issue when considering patterns as part of a catalogue is the need of classifying them over some criteria for supporting their search. In fact, it is important to observe that different contexts (organizations, projects, standards, etc.) may, and usually do, define or require different classification schemas. History shows that trying to impose a particular classification schema does not work, therefore we decouple SRPs and *Classifiers* as shown in the metamodel. The catalogue is thus considered as flat and the *Classification Schemas* just impose different structuring schemas on top of it. Classifiers are organized into a hierarchy and then SRP are in fact bound to *Basic Classifiers*, whilst *Compound Classifiers* just impose this hierarchical structure. The use of aggregation avoids cycles without further integrity constraints. Last, a derived class *Root* is introduced as a facilitation mechanism.

The metamodel shows that a SRP may be bound to several classification schemas, and even to more than one classifier in a single classification schema (since no further restrictions are declared). Also note that we do not impose unnecessary constraints that could lead the catalogue to be rigid. For instance, we may mention that a classification schema may not cover all existing SRP (i.e., some SRP may not be classified). Although this situation could be thought as a kind of incompleteness, in fact we are allowing having dedicated classification schemas for particular categories of patterns, e.g. a performance classification schema, a classification schema just for the non-technical criteria [16] and then allowing to compound them for having a multi-source global classification schema. Also we remark that the PABRE method [15] benefits from this existence of multiple classification schemas since nothing prevents changing from one schema to another during catalogue browsing.

3 Conclusions and Future Work

In this paper we have presented a metamodel for software requirement patterns (SRP). This metamodel is the natural evolution of the preliminary proposal of SRP presented at [17] and shows the current concepts used by the PABRE method [15]. The metamodel helps to fix the concepts behind our proposal of SRP, improving the

quality of the current SRP catalogue and process and has been be taken as starting point of the data model of an ongoing support tool. The metamodel has been validated with respect to several software requirement specifications (SRS) written by CITI-CRPHT in the context of call-for-tender processes as well as in two processes themselves. The contents of the catalogue have been validated as explained in [15]; the catalogue itself can be found at the website http://www.upc.edu/gessi/PABRE.

Future work spreads over three main directions. Concerning validation, we are planning to run new case studies to debug all the PABRE components: metamodel, catalogue contents and process. We intend to experiment deeper the application of the SRP catalogue in several contexts (public IT procurement projects and Small- and Medium-Sized companies' projects). We also want to study the suitability of the current presented metamodels for other types of requirement patterns, that is, patterns for functional and non-technical requirements. Last, we will analyze the possibility of converting the current metamodel of a SRP catalogue into a metamodel for a patterns language which would eventually make possible the adoption of the approach in contexts with different needs than those presented here. Although the idea is appealing, it would require more engineering effort and thus needs careful analysis.

References

1. W. Lam, J. A. McDermid, A. J. Vickers. "Ten Steps Towards Systematic Requirements Reuse". REJ 2(2), Springer, 1997.
2. S. Roberson, J. Robertson. Mastering the Requirements Process (2nd ed.). Addison-Wesley, 2006.
3. C. Alexander. The Timeless Way of Building. Oxford Books, 1979.
4. S. Henninger, V. Corrêa. "Software Pattern Communities: Current Practices and Challenges". PLoP 2007.
5. A. Durán, B. Bernárdez, A. Ruíz, M. Toro. "A Requirements Elicitation Approach Based in Templates and Patterns". WER 1999.
6. B. Moros, C. Vicente, A. Toval. "Metamodeling Variability to Enable Requirements Reuse". EMMSAD 2008.
7. S. Robertson. "Requirements Patterns Via Events/Use Cases". PLoP 1996.
8 O. López, M.A. Laguna, F.J. García. "Metamodeling for Requirements Reuse". WER 2002.
9. S. Konrad, B.H.C. Cheng. "Requirements Patterns for Embedded Systems". RE 2002.
10. D. Matheson, I. Ray, I. Ray, S. H. Houmb. "Building Security Requirement Patterns for Increased Effectiveness Early in the Development Process". SREIS 2005.
11. A. Mahfouz, L. Barroca, R. C. Laney, B. Nuseibeh. "Patterns for Service-Oriented Information Exchange Requirements". PLoP, 2006.
12. J. Withall. Software Requirements Patterns. Microsoft Press, 2007.
13. J. Yang, L. Liu. "Modelling Requirements Patterns with a Goal and PF Integrated Analysis Approach". COMPSAC 2008.
14. M. Krystkowiak, B. Bucciarelli. "COTS Selection for SMEs: a Report on a Case Study and on a Supporting Tool". RECOTS 2003.
15. S. Renault, O. Méndez, X. Franch, C. Quer. "A Pattern-based Method for building Requirements Documents in Call-for-tender Processes". IJCSA 6(5), 2009.
16. J.P. Carvallo, X. Franch, C. Quer. "Managing Non-Technical Requirements in COTS Components Selection". RE 2006.
17. O. Méndez, X. Franch, C. Quer. "Requirements Patterns for COTS Systems". ICCBSS 2008.