Process Patterns - a Means to Describe Processes in a Flexible Way

Mariele Hagen, Volker Gruhn
Chair of Applied Telematics / e-Business
Department of Computer Science
University of Leipzig
{hagen, gruhn}@ebus.informatik.uni-leipzig.de

Abstract

Process patterns allow the modular modelling and adaptable application of business processes. Present descriptions of process patterns show defects like non-uniform and unequivocal description forms and missing relationship definitions. These defects disadvantageously affect the effective usage of process patterns. In this work we introduce the language PROPEL (Process Pattern Description Language), which provides concepts for the semiformal description of process patterns and relationships between process patterns. With the help of PROPEL single process patterns can be modelled and, by definition of relationships, be composed to more complicated processes. With the representation of different views of a process pattern catalog the process patterns and their relationships can be shown clearly. An example illustrates how a process pattern catalog and the contained process patterns are modelled. It is shown that in applying PROPEL the complexity of a process model can be reduced and inconsistencies of processes be eliminated.

1. Introduction

A process pattern represents a proven process which solves an frequently recurring problem in a patternlike way (cf. [3] for an introduction to patterns). The problem embodies a concrete situation which may emerge e.g. during the software development. The process describes a set of activities, which can be executed to solve the problem. Process patterns offer the documentation of proven knowledge by abstracting from details and facilitate the communication [5]. Process patterns especially enable a flexible process support since one can select and apply a suitable process pattern according to the present situation. For this reason process patterns are considered as an alternative to complex and heavyweight process models like Rational's Unified Process [9], the German Process Model [13], Objectory Process [8] etc., since process patterns can be used to adapt the software process to the respective project situation [1].

Although the concept of process patterns is very promising, it has not matured yet [7]. Current research focuses rather on the identification of new process patterns, neglecting the identification of a suitable presentation. Process patterns are not described in a precise way, nor are process pattern relationships sufficiently described.

Up to now patterns were described in an informal way by natural language. This is on the one hand an advantage, since no knowledge about notation, syntax and semantics is necessary for understanding a pattern. On the other hand, natural language produces ambiguous and inexact expressions (e.g., the relationships „is prerequisite of" and „may contain" in [12]). In addition to the informal description of relationships the descriptions of the processes themselves are surprisingly informal. A step wise description of the process is often missing, and there are even process patterns which contain no process as a solution (e.g. in [2]).

Moreover, process patterns can unfold their full strength only in the combination with other process patterns [3]. Unfortunately, present descriptions of pattern relationships are not formally specified and give no information about their syntactic or semantic meaning (e.g., „pattern X uses pattern Y" or „pattern A can be combined with pattern B"). In considering the patterns' relationships, more complicated process patterns (processes) could be modelled therefore in combining single process patterns (processes). This information about possible combinations of patterns must be provided by an explicit description of the patterns' relationships.

In section 2. we outline our approach. Section 3. then deals with the PROPEL metamodel und illustrates it with some examples. Section 4. presents a process pattern catalog consisting of a set of process patterns and relationships. In section 5. we finally summarize and evaluate our statements and give some perspectives of our work.

2. Approach

Because of the aforementioned reasons we have developed the language PROPEL (Process Pattern Description Language) for the description of process patterns (cf. [6] for a detailed overview). This language is an extension of the Unified Modeling Language (UML) [11]. Hence, many proven and widespread modelling concepts of the „Lingua Franca“ of the software engineering domain can be reused. The UML already offers concepts like activity diagrams for modelling processes which we make use of for modelling the process element of a process pattern. PROPEL suggests a uniform description schema for process patterns and defines relationships between process patterns. Herewith PROPEL allows to describe process patterns and their relationships in a semiformal way.

3. The PROPEL Metamodel

In the following, we present the metamodel of PROPEL through class diagrams. A detailed description of the metamodel can be found in [6], an overview over the OCL constraints in [4]. Furthermore, we give some examples to clarify the purpose of the PROPEL concepts. Figure 1 lists the notational elements specified within PROPEL and used in the examples.

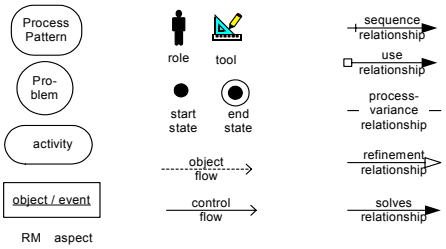

Figure 1: Notational elements of PROPEL

Figure 2 presents the parts of PROPEL that deal with modeling a single process pattern. Figure 3 shows how these concepts are used for modeling purposes. A process pattern consists of a process, which solves a problem which has recurred in a certain context.

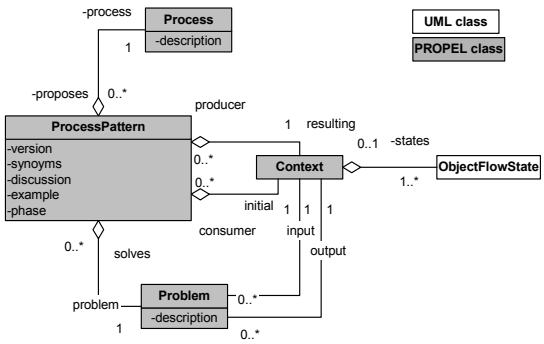

Figure 2: PROPEL syntax - part 1

A problem can be grouped thematically via its aspect. The association solves expresses that there can be an arbitrary set of solutions (i.e. process patterns) for a problem. The detailed textual description of the problem is expressed by the attribute description.

The context of a process pattern defines the conditions which must be fulfilled before and after application of the process pattern. A context is a part of a process pattern. It is either consumed (s. association consumer) or produced (s. association producer) by the process pattern. A context is a set of objects and and events (states). An object is represented by the metaclass ObjectFlowState, whose type of classifier is an artifact. An event is likewise represented by the stereotype <<SignalFlowState>>, the type of the classifier is then a signal.

In our example (figure 3) the process pattern "Capture a Common Vocabulary" solves the problem "How can a Glossary be defined?", which recurs in the initial context consisting of the object "Requirements Documents" and in the resulting context consisting of the object "Glossary". That means that the context specifies objects and events which emerge before and after the execution of the ProcessPattern.

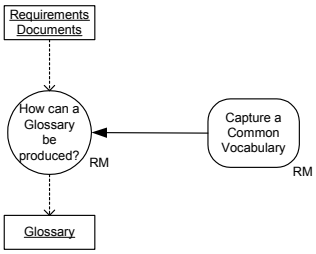

Figure 3: Example problem and process pattern

We use the UML metaclass "ActivityGraph" to model a process (cf. figure 4). A process is a set of activities (metaclass "ActionState"). It describes the control flow of and the object flow between its activities. A process can be an element of an arbitrary set of process patterns. If a process is an element of more than one process patterns, we speak of processvariant process patterns (cf. relationship processvariance).

Furthermore, roles can be assigned to activities and processes. This may be irritating, since the UML specification contains the similar metaclass "Actor". However, the association of actors to model elements is limited to the system which is to be developed, i.e. only associations to uses cases, subsystems and classes are permitted. However, for the modelling of software development processes we need a broader term and therefore we introduce the metaclass Role. The activities and tasks of a Role are described in detail within a profile (not shown here).

A role (e.g. "System Analyst") describes experiences, knowledge and abilities, which are necessary in order to carry out activities. A tool (e.g. "Requisite Pro") supports the execution of an activity and appear often as a software system. Different goals are linked with the usage of a tool, these goals can differ from activity to activity.

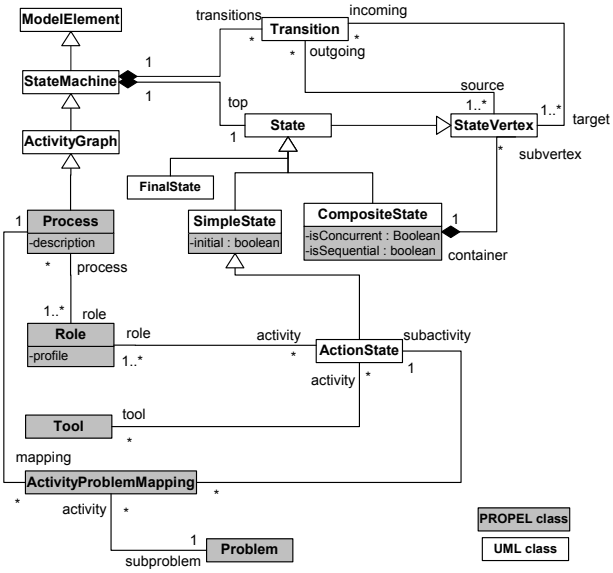

Figure 4: PROPEL syntax - part 2

The process part of the process pattern specifies steps which are necessary to solve the problem. It must therefore fulfil the conditions specified by the initial and resulting context. In our example, figure 5 presents the process of process pattern "Capture a Common Vocabulary". Note that the process takes exactly these objects as input and ouput that correspond to the context specified by the problem. That means that the process is encapsulated by the process pattern. The problem and its solving process patterns are a sufficient information for a pattern user to select a process pattern according to his needs.

The hierarchical composition of a process is established by the mapping of problems onto activities.

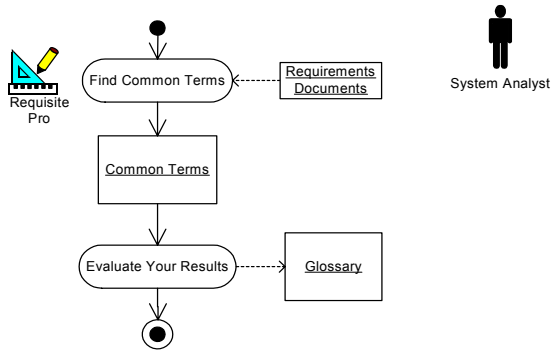

Figure 5: Example process

Figure 6 presents the parts of PROPEL that deal with modeling relationships between process patterns. The following figures 6 to 10 show how these concepts are used for modeling purposes.

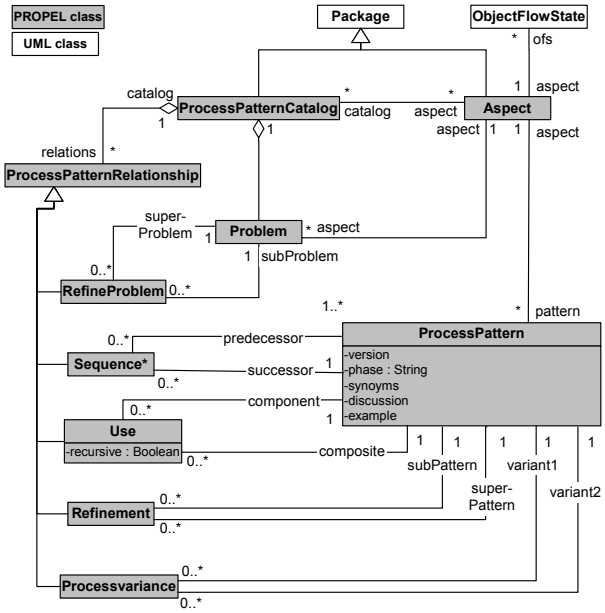

Figure 6: PROPEL syntax - part 3

The *sequence relationship* connects one or more preceding patterns and one succeeding pattern. Such a relationship exists between several process patterns, if the preceding process patterns alltogether produce all objects and events, which the following process pattern needs for its application.

In our example (figure 7), the two process patterns "Define Review Criteria" and "Develop Software Component" together produce all objects ("Review Criteria", "Review Request" and "Software Component") the process pattern "Review" needs. The three process patterns therefore are connected via a sequence relationship.

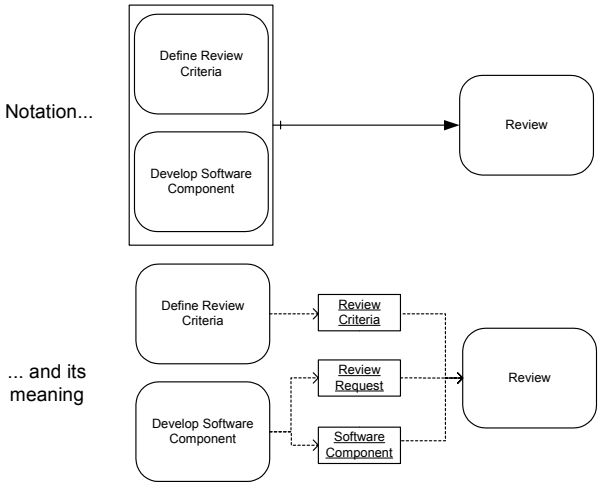

Figure 7: Example sequence relationship

The *use relationship* associates a component pattern and a composite pattern. Such a relationship exists, if the component pattern describes a subprocess of the composite patterns, i.e. a part of the solution. In our example (figure 8), the process pattern "Review" uses three other

process patterns, namely "Introductional Session", "Review Session" and "Release". The process of pattern "Review" contains the activity "Perform Review-Session", which needs to be described more detailed. This can be done in specifying a problem (e.g. "How can a Review be performed?") representing the activity. Then, for this problem can be found another solving process pattern, e.g. "Review Session".

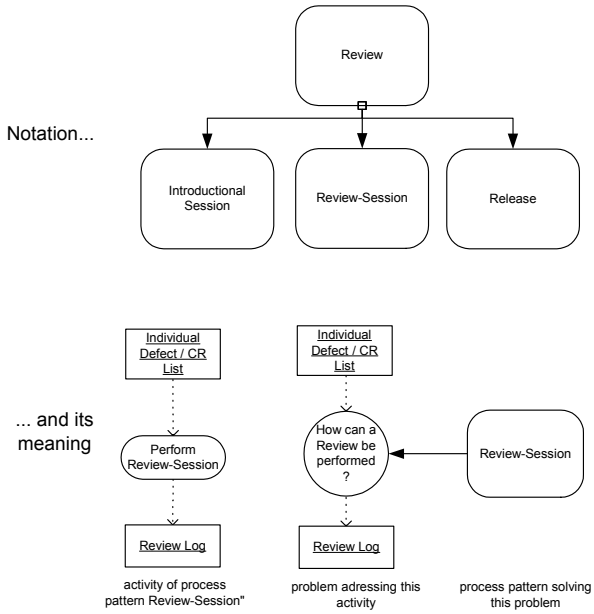

Figure 8: Example use relationship

Problems, which adress a certain activity, can be assigned to an "ActionState" (via the association subactivity). These problems are therefore subproblems of the problem of the superior process pattern. Hence, there may be process patterns which solve a partial step of the superior process pattern. The existence of an one or more "ActivityProblemMappings" is therefore a condition for the hierachical composition of process patterns.

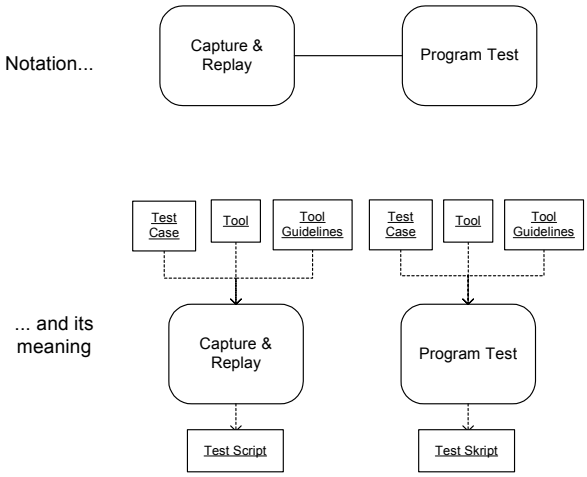

Figure 9: Example processvariance relationship

The *processvariance relationship* associates two variant process patterns. Such a relationship exists, if the variant patterns solve the same problem but with different solutions. In our example (figure 9), the two process patterns "Capture&Replay" and "Program Test" are variant process patterns, since they both fulfill the same context.

The *refinement relationship* associates a superpattern and a subpattern. Such a refinement relationship exists, if the context and process of the superpattern have been refined by the subpattern. This means in addition, that the problems of these two patterns are connected by a refinement relationship. In our example (figure 10), the process pattern "Manual QA" is refined by the process pattern "Review". The initial context of "Review" not only contains the "Review Object", but also "Review Criteria" and "Review Request".

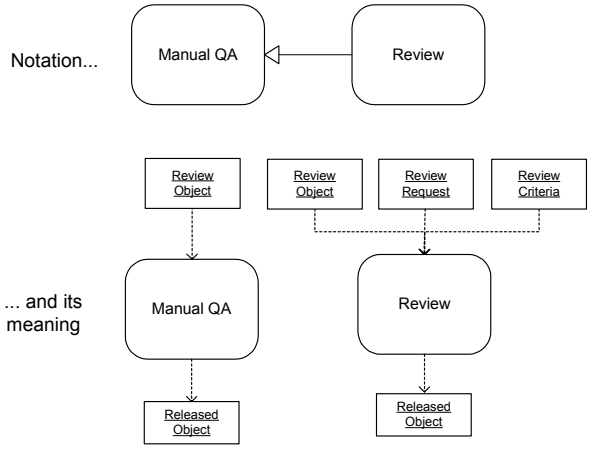

Figure 10: Example refinement relationship

The last two concepts defined in this section are the process pattern catalog and the aspect.

A process pattern catalog represents a set of process patterns and relationships, which tie process patterns together. Process patterns and problems are always assigned to a process pattern catalog (cf. association catalog).

Finally, process patterns and problems can be grouped thematically via their aspect. The aspect is a subclass of the metaclass package and bundles a set of process patterns, problems and objects (via "ObjectFlowState", not shown here). The aspect informs about the subject of these elements (e.g.: project acquisition, management project, risk management etc.).

4. Example

For validating the concepts of PROPEL we developed the process pattern catalog CADS (Catalog for the Development of Software), which illustrates the use of PROPEL. We derived the catalog from the Rational Unified Process (RUP) [9]. Thus we were able to use already proven, widespread processes and concentrate on the form of the description.

The representation of the process patterns specified in this section is limited to process patterns, which belong to the discipline (RUP) or the aspect (PROPEL) "Requirements Management". Process patterns, which belong to other aspects (e.g. „Business Modeling“), carry a special symbol (e.g. „BM“ for „Business Modeling“, cf. process pattern „Find Business Actors and Use Cases“ in figure 12).

Considering the aspect "Requirements Management" we identified 31 problems and 44 process patterns. The difference between the number of problems and process patterns is based on the existence of variant process patterns which belong to the same problem. The modeling effort was a couple of days. The greatest part of this modeling effort was to identify and eliminate inconsistencies of the RUP processes. But with help of the PROPEL relationship definitions we could decide if two processes are consistent, and if not, how this consistency may be established. The effort was reasonable beause of two reasons: We gained consistent process descriptions and could reduce the process model‘s complexity.

4.1. Views of the Process Pattern Catalog

In the following we show a selected part of the process pattern catalog with help of different views. Each view represents an overview of the relationships of a certain type. We could also merge all views into one single view (to represent the catalog as a whole), but this would endanger clarity and understanding.

Figure 11 presents part of the hierarchical view of the catalog CADS with help of the use relationship. This hierarchical structure was taken over as far as possible from the RUP (for the RUP is already hierarchically structured by disciplines (e.g. „Requirements Management“), workflow details (e.g. „Refine the Problem“) and activities („Develop Requirements Management Plan“). Looking at this view it becomes clear that some process patterns are used several times, e.g. the process pattern "Develop vision". The process pattern „Review requirements“ uses itself and is therefore associated with itself via a recursive use relationship. The RUP is lacking such a hierarchical overview.

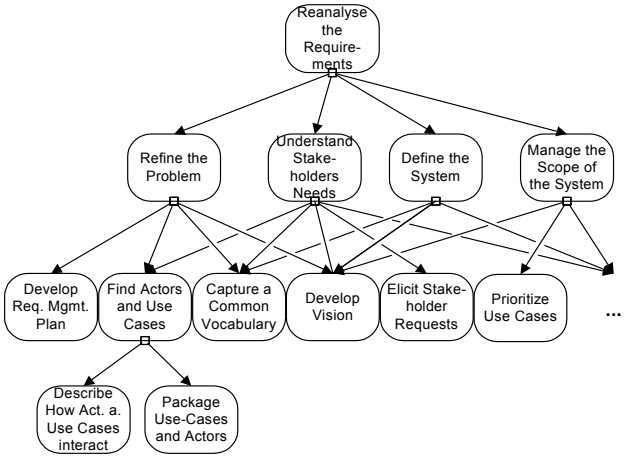

Figure 11: Catalog Diagram - View „Use“

Figure 12 presents a selection of the existing *sequence relationships* (because for lack of space not all relationships instances could be presented here). The sequence relationship signifies that one ore more process patterns provide the (initial) context for another process pattern. It therefore signifies a sequential flow of process patterns. The pattern user is thus informed, which process patterns he can use directly before or after a certain process pattern. If a pattern user would like to use e.g. the pattern "Capture a Common Vocabulary", then he knows that the necessary context can be provided by applying a set of process patterns like „Find Business Workers and Entities“, „Find Actors and Use Cases“ etc. This information is only implicitly present in the RUP.

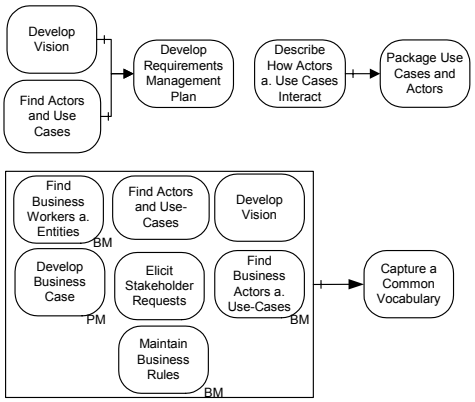

Figure 12: Catalog Diagram - View „Sequence“

The number of sequence relationships with one single predecessor pattern is rather small in this example. The reason is that the RUP is divided into nine different disciplines, but processes (i.e. workflowdetails or activities) of several disciplines cooperate, in order to produce an object or an event. Because of this close linkage of the processes the number of sequence relationships with more than one predecessor pattern is much higher than Sequence relationships with only one predecessor pattern.

The representation of the Sequence relationship is an advantage compared to the RUP, in which the input and the output artifacts of a workflow detail or an activity are indicated, but possible sequences of processes are not described. Since the RUP activities, workflowdetails and disciplines are closely interlocked, this information is helpful for the user.

The *refinement relationship* represents a relationship between a more abstract and more detailed process pattern. For example, there exist two refinements of the process pattern „Capture a Common Vocabulary“ exist, namely the refining process patterns „Vocabulary centrally“ and „Vocabulary participatorily“. These two process patterns describe in a more detailed way than the abstract process pattern, how a project glossary can be provided. The additional information comes along with the refined context and process of the refining process patterns. The user can therefore decide, how much support (abstract pattern - less support, detailed pattern - more support) he needs to solve the problem. The RUP offers no concepts to describe such process refinements.

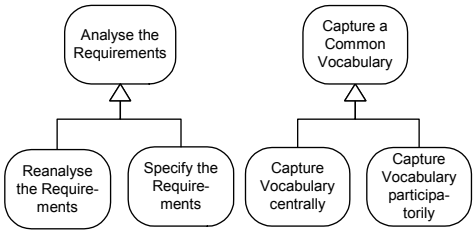

Figure 13: Catalog Diagram - View „Refinement“

Figure 13 presents some occurrences of the *refinement relationship*. Two refinements of the process pattern „Capture A Common Vocabulary“ exist, namely the refining process patterns „Capture Vocabulary centrally“ and „Capture Vocabulary participatorily“. These two process patterns describe in a more detailed way than the abstract process pattern, how a project glossary can be provided. The additional information comes along with the refined context and process of the refining process patterns. (cf. in figure 16 object "Requirements Documents", which is refined by the input of the subproblem; the object "Glossary" is not refined, and is thus existent both in the superproblem and in the subproblem). The user can therefore decide, how much support (superpattern - less support, subpattern - more support) he needs to solve the problem. The RUP offers no concepts to describe such process refinements.

The *processvariance relationship* signifies that processvariant process patterns solve the same problem with a different solution (i.e. the process). Figure 14 presents some occurrences of this relationship. The three process patterns "Review requirements", "Walkthrough Requirements" and "Inspect Requirements" are processvariant patterns. These three process patterns explain, how a manual quality assurance can be done in different ways.

The user can therefore decide, how he would like to solve the problem. The RUP does not deal with the description of variant processes. This is a disadvantage, since the user is always tied to a certain process. Moreover, if a user is aware of a variant process, this variant process cannot be added to the RUP.

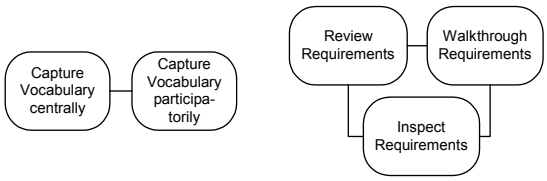

Figure 14: Catalog Diagram - View „Processvariance“

4.2. Detailed View of Problem and Process Pattern

We now present an explanatory detailed view of process patterns and problems in the catalog CADS.

Within a project a project vocabulary is to be developed, in order to improve communication between the project workers and to achieve a clear and unambigious terminology in the documents. This situation is represented by the problem "How can a Glossary be produced?". Figure 15 shows on the left the problem to be solved and its initial ("Requirements Documents") and resulting context ("Glossary"). On the right side the solving process patterns are presented. The problem is solved by the process pattern "Capture A Common Vocabulary". There are two refining process patterns of this pattern, namely process pattern "Capture Vocabulary centrally" and process pattern "Capture Vocabulary participatorily". These two process patterns provide a more detailed process and context as the refined process pattern.

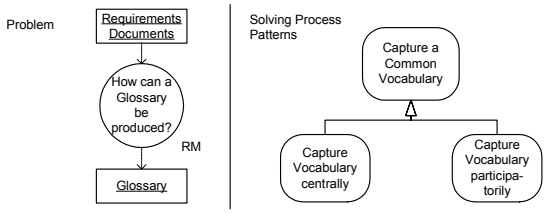

Figure 15: Problem Diagram

Figure 16 shows a refinement of the superproblem "How can a Glossary be produced?". We see on the right side that the context of the superproblem is refined by the context of the subproblem in inheriting all elements of the context and adding further elements if needed. Note that the resulting contexts of both the super- and the subproblem are identical, i.e. the object "Glossary" is their sole element.

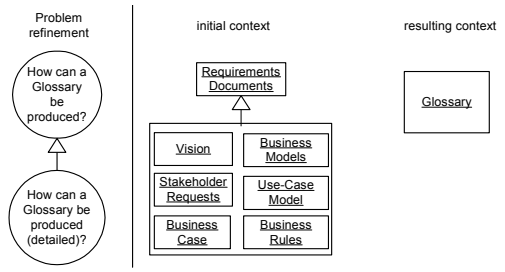

Figure 16: Refinement of Problem

Finally, in figure 17 we present the condensed description of the process pattern "Capture A Common Vocabulary" with the concepts introduced before. This view is also called the process pattern diagram. First, the problem to be solved is named (above). Then, all process patterns the process pattern is connected with via a relationship are presented (middle). Finally, the process of the process patterns is presented (bottom).

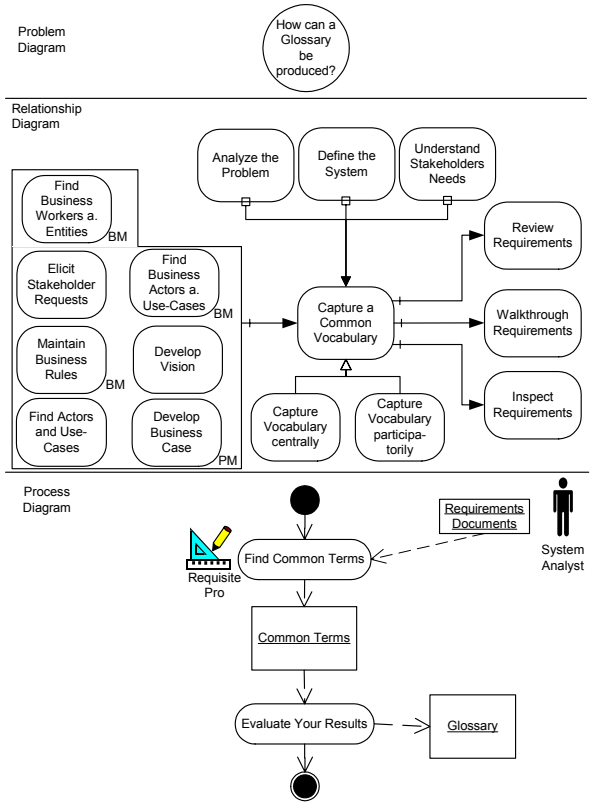

Figure 17: Process Pattern Diagram

As we have seen, the process pattern diagram presents all necessary information of a process pattern as well as the problem diagram presents all necessary information of a problem. In practice, this type of presentation is an advantage since it uses the principle of information hiding.

If a user faces a certain problem, he can first search an appropriate problem. Via the problem diagram, (maybe) several process patterns are indicated that solve this problem. In a second step, the user can examine and select one of these process patterns. From this point of view, he can then see via the relationship diagram, which other process patterns he can possibly apply before or after this process pattern, or which variants or refinements are available. If the pattern user has carried out e.g. the process pattern "Develop vision", it is clear that afterwards the process pattern "Develop requirements management plan" can be used.

By this example it becomes clear that a complex process model like RUP can be described by PROPEL process patterns. Furthermore, the complexity of the process model can be reduced and disentangled by identifying process patterns and their relationships.

5. Conclusion

The Process Pattern Description Language PROPEL introduced in this paper offers all necessary means in order to describe process patterns in a uniform way. PROPEL encloses the description of problems, contexts, objects and events, roles and tools. Particularly the possibility of describing relationships between process patterns and thus between processes has to be emphasized. Most process models do not offer the representation of process relationships. The semiformal representation of processes increases the accuracy of the description. It permits to set up rules for the hierarchical structure of process patterns and therefore for defining the use relationship between process patterns. Furthermore, rules for specifing sequences, refinements and variants of process patterns are given.

With the means of PROPEL we described a process pattern catalog based on processes of the RUP. The catalog presents both the process patterns and the relationships between process patterns with the help of different views. We showed that with PROPEL a complex proces model like the RUP can be simplified in modularizing the processes, identifying formerly hidden relationships and eliminating inconsistencies.

To enhance the accuracy of process pattern descriptions we presently work on the formalization of the semantics of PROPEL. We accomplish this by defining a semantic mapping of PROPEL's abstract syntax onto the semantic domain of petri nets. With petri nets we can define the dynamic conditions of process pattern relationships (i.e. not only the structural conditions). For a improved management of process patterns we develop the Process Pattern Workbench, a platform for the documentation and administration of process patterns. Details about the prototype are available in [10]. A further research question is to log the application of process pattern, on the one hand to inform the project workers about which patterns should be and have already been applied, and on the other hand to log statistically which process pattern combinations are more frequently used than others. Furthermore, we develop the process pattern management methodology, in order to identify, document, select and use process patterns in a systematic way.

6. References

[1] Bergner, K.; Rausch, A.; Sihling, M.: A Component Methodology based on Process Patterns. TUM-I9823, University Munich, 1998.

[2] Coplien, J.: A Development Process Generative Pattern Language. In: Proceedings of PLoP 94, 1994.

[3] Coplien, J.: Software Patterns. SIGS Book & Multimedia, 1996.

[4] Dittmann, T.: PPDL - A Description Language for Process Patterns (in German), Diploma Thesis, University Dortmund, 2002.

[5] Dittmann, T., Gruhn, V., Hagen, M.: Improved Support for the definition and usage of process patterns. 1st Workshop on Process Patterns, OOPSLA 2002, Seattle, 2002.

[6] Hagen, M.; Gruhn V.: PROPEL - A Language for the Description of Process Patterns (in German). In: Proc. of Modellierung 2004, LNI, Vol. P-45, Köllen, 2004, pp.203-218.

[7] Hagen, M.: Support for the definition and usage of process patterns. Position Paper, EuroPloP 2002, http://www.haase-consulting.com/workshops/FgEuroplop02/position_papers.html.

[8] Jacobson, I.: Object-Oriented Software Engineering: A Use Case Driven Approach. Addison Wesley Publishing Company, 1992.

[9] Kruchten, P.: The Rational Unified Process, Addison-Wesley Professional, 2000.

[10] Schröder, J.: The Process Pattern Workbench, Thesis (in German), University Dortmund, 2003.

[11] Object Management Group: OMG Unified Modeling Language Specification, March 2003, Version 1.5, formal/03-03-01. http://www.omg.org/docs/formal/03-03-01.pdf.

[12] Störrle, H.: Models of Software Architecture. Design and Analysis with UML. Dissertation, University Munich, 2000.

[13] V-Modell ’97: Entwicklungsstandard für IT-Systeme des Bundes. BWB IT 15, 1997.