Towards flexible Software Processes by using Process Patterns

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 software 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 present 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 complex 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, the inconsistencies of processes can be eliminated and the flexibility of processes can be improved.

Key Words

Software Tools and Techniques, Modelling Languages, Process Modelling, Process Patterns

1. Motivation

A process pattern represents a proven process which solves an frequently recurring problem in a pattern like way (cf. [5] 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 [7]. 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 [11], the German Process Model [16] etc., since process patterns can be used to adapt the software process to the respective project situation [3].

Although the concept of process patterns is very promising, it has not matured yet [10]. Current research focuses rather on the identification of new process patterns, neglecting the identification of a suitable presentation. Present process pattern descriptions reveal two crucial deficiencies:

- 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 [15]). 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 [4]).
- Moreover, process patterns can unfold their full strength only in combination with other process patterns [5]. 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 [12]). In considering the patterns' relationships, more complex process patterns can be modelled therefore in combining single process patterns. This information about possible combinations of patterns must be provided by an explicit description of the patterns' relationships.

Because of the aforementioned reasons we have developed the language PROPEL (Process Pattern Description Language) for the description of process patterns (cf. [9] for a detailed overview). This language is an extension of the Unified Modeling Language (UML) [14]. 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.

* The Chair of Applied Telematics/e-Business is endowed by Deutsche Telekom AG.

In section 2 we describe the PROPEL meta model and illustrate it with some examples. Section 3 presents a process pattern catalog consisting of a set of process patterns and relationships. Similar approaches are characterized in section 4. In section 5 we summarize and evaluate our statements and give some perspectives of our work.

2. The Concepts of PROPEL

In the subsequent sections we present the basic concepts of PROPEL. A detailed description of the underlying meta model in terms of UML class diagrams can be found in [9], an overview over the OCL constraints in [6]. Figure 1 lists some of the notational elements specified within PROPEL, which are used in the subsequent examples.

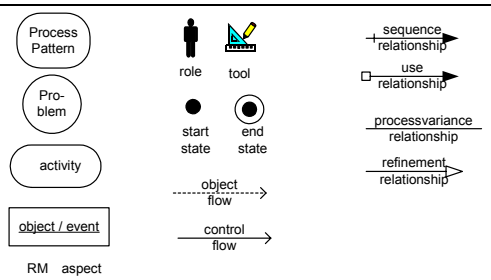

Figure 1: Notational elements of PROPEL

A single *process pattern* consists of a process, which solves a problem which has recurred in a certain context:

A *problem* can be grouped thematically via its aspect. There can be an arbitrary set of solutions (i.e. process patterns) for a problem.

A *context* is a set of objects and and events. It 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 or produced by this process pattern.

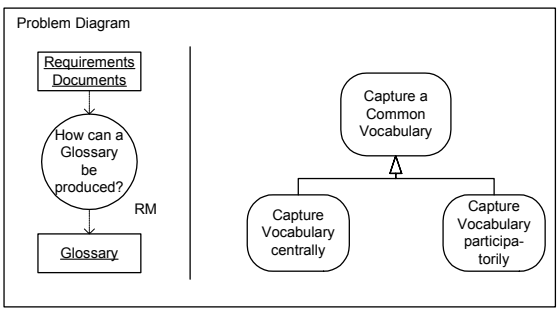

Figure 2: Problem Diagram

As an example, we consider a project for which a project vocabulary is to be developed, in order to improve communication between the project workers and to achieve a clear and unambiguous terminology in the documents. This situation is represented by the problem "How can a Glossary be produced?". Figure 2 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", which solve the problem too. These two process patterns provide a more detailed process and context as the refined process pattern.

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, beneath).

The *process* (modelled by the UML meta class "ActivityGraph") of the process pattern specifies activities which are necessary to solve the problem. It describes the control flow of and the object flow between its activities. A process has to fulfil the conditions specified by the initial and resulting context of the problem to be solved. In our example, figure 3 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.

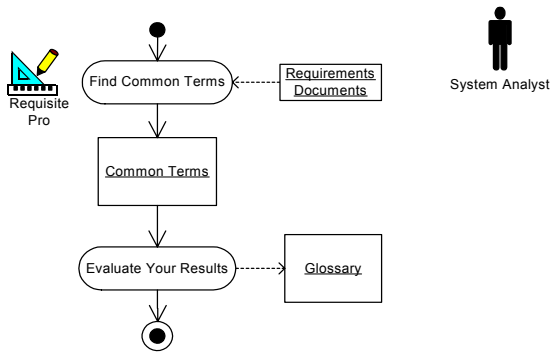

Figure 3: Example process

Furthermore, *roles* can be assigned to activities and processes. A role (e.g. "System Analyst") describes experiences, knowledge and abilities, which are necessary in order to carry out activities. The definition of the meta class "Role" may be irritating, since the UML specification contains the similar meta class "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 meta class Role. The activities and tasks of a Role are described in detail within a profile (not shown here).

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.

The *aspect* (e.g. "RM" for Requirements Management) specifies a certain theme (e.g.: project acquisition, management project, risk management etc.) problems and process patterns are belonging to. By using aspects a set of process patterns or problems can be structured into subsets.

The subsequent four figures show the concepts of PROPEL which allow modelling relationships between process patterns.

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

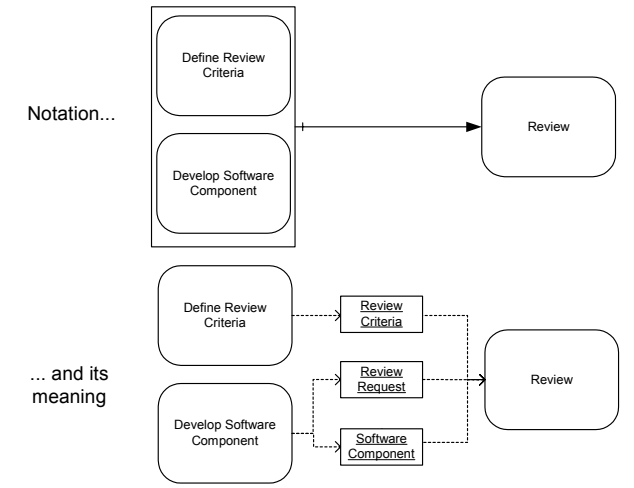

Figure 4: Example sequence relationship

In our example (figure 4), 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.

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 5), 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". This means that the hierarchical composition of a process is established by the mapping of problems onto activities.

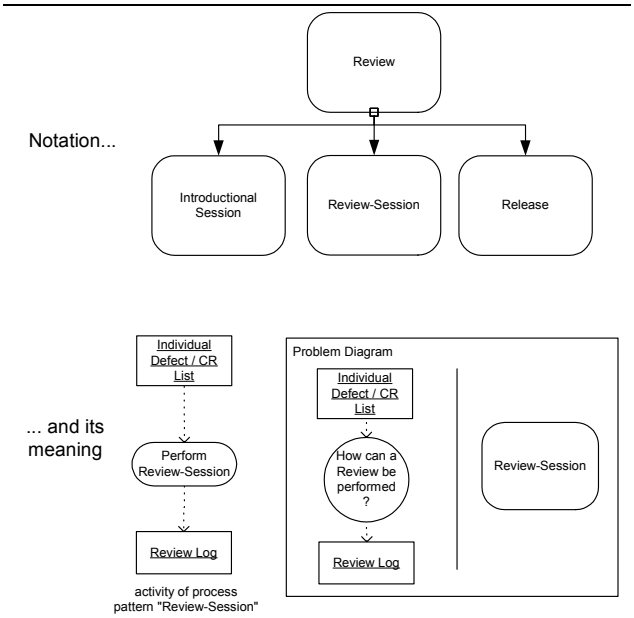

Figure 5: Example use relationship

The *processvariance relationship* associates two variant process patterns. Such a relationship exists, if the variant patterns solve the same problem but with different solutions.

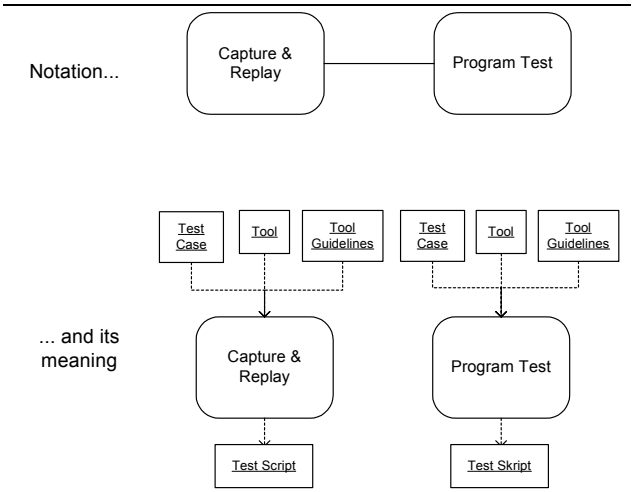

Figure 6: Example processvariance relationship

In our example (figure 6), the two process patterns "Capture&Replay" and "Program Test" are variant process patterns, since they both solve the same problem, ie.e. 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 refine-

ment relationship. In our example (figure 7), 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".

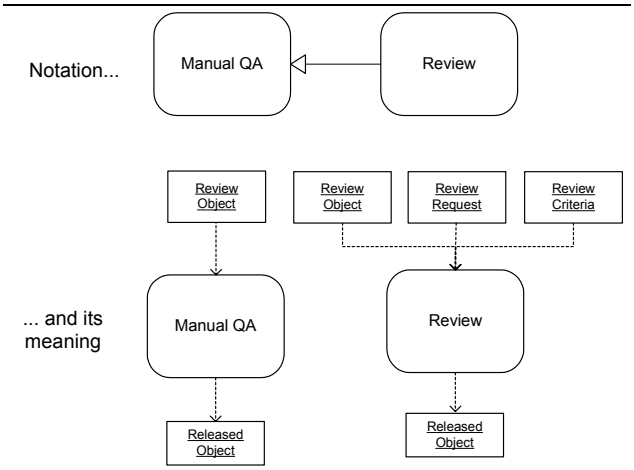

Figure 7: Example refinement relationship

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. We will show details of the concept and usage of process pattern catalogs in section 3.

Finally, we present in figure 8 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).

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 "Capture a Common Vocabulary", it is clear that afterwards the process pattern "Review Requirements" can be applied.

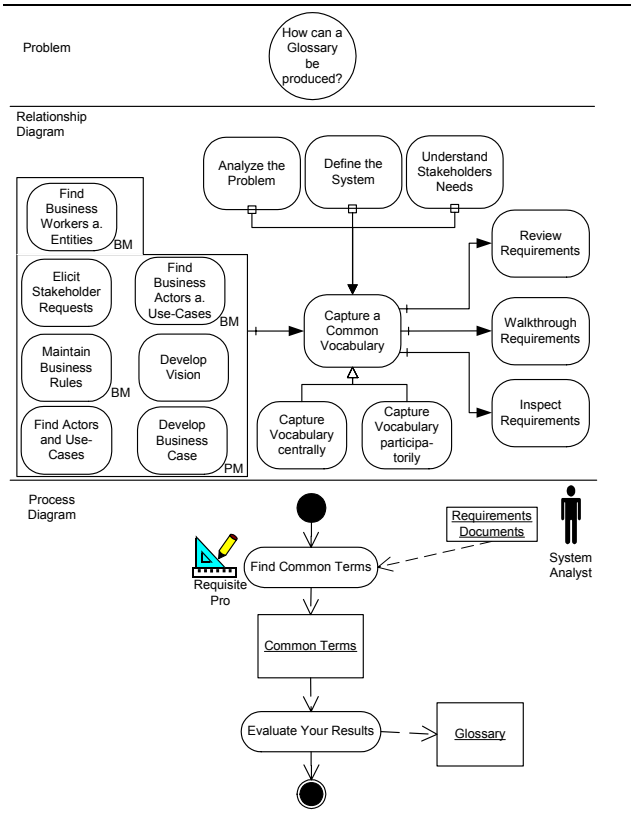

Figure 8: Process Pattern Diagram

3. The Process Pattern Catalog

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) [11]. 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 aspect "Requirements Management". 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 modelling effort was a couple of days. The greatest part of this modelling 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 because of two reasons: We gained consistent process descriptions and could reduce the process model‘s complexity.

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.

For reasons of space we only present the use and the sequence view.

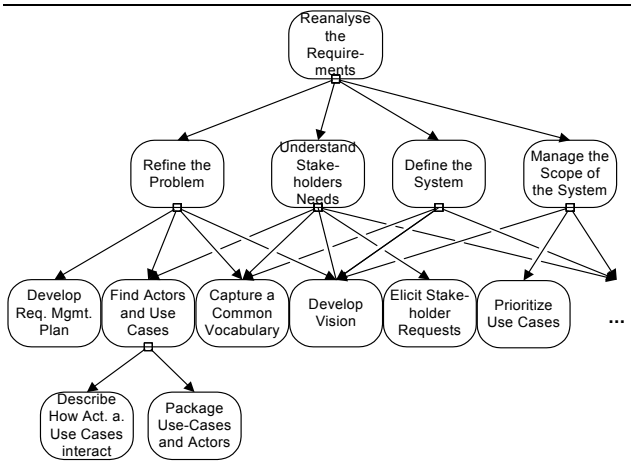

Figure 9: Catalog Diagram - View „Use“

Figure 9 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 10 presents a selection of the existing *sequence relationships*. This information is only implicitly present in the RUP. We made the experience that the number of sequence relationships with one single predecessor pattern is rather small. The reason is that the RUP is divided into nine different disciplines, but processes (i.e. workflow details 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, workflow details and disciplines are closely interlocked, this information is helpful for the user.

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. Inconsistencies between process can be identified and eliminated. Finally, the flexible selection and use of process patterns is ensured.

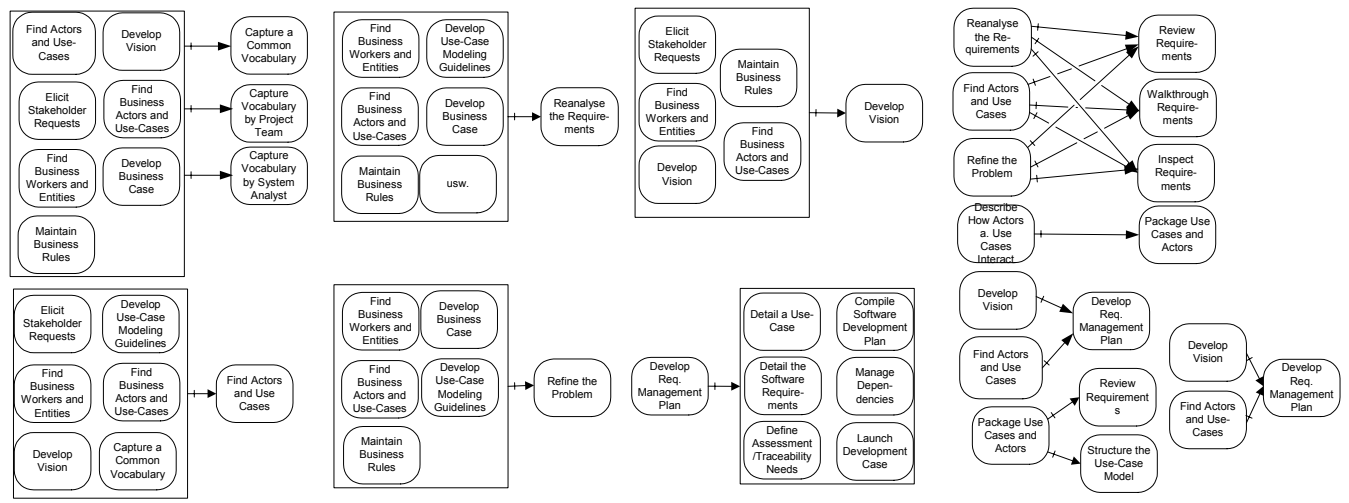

Figure 10: Catalog Diagram - View „Sequence“

4. Related Work

Up to now there are only few approaches exploring the concepts of process patterns. Most publications concentrate on presenting new process patterns and do not bother neither with presentation nor precision (e.g. [4], [2]). Even if relationships are mentioned, their meaning is not specified (e.g. [15], [2]).

Gnatz et. al. identified in [8] the two process pattern relationships „realized by“ and „execute“. The relationship „realized by“ is applied for two variant process patterns that solve the same problem. The relationship „execute“

represents a composition relationship. A formal specification of these relationships is missing.

Bergner et. al. represented in [3] process patterns for component based software development. Instead of process pattern relationships they represent relationships between artifacts. In considering the artifact relationships corresponding process patterns can be selected. How this selection is done remains unclear. Again, the artifact relationships are not specified precisely.

Note that process pattern are not to be mixed up with workflow patterns by van der Aalst [1]. The main goal of workflow patterns is to provide the basis for an comparison of workflow management systems and to provide concepts, how business processes can be implemented. Process patterns instead describe all kinds of business processes independent of any system which supports the process enaction.

5. Conclusions and Future Work

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 specifying 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 process 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 a prototype are available in [13]. A further research question is to log the application of process patterns, 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.

6. References

[1] van der Aalst, W. et al.: Workflow Patterns. Distributed and Parallel Databases, 14(3), pages 5-51, July 2003.

[2] Ambler, S. W.: Process Patterns, Cambridge University Press, 1998, Cambridge.

[3] Bergner, K. et.al.: A Component Methodology based on Process Patterns. TUM-I9823, University Munich, 1998.

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

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

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

[7] 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.

[8] Gnatz, M. et.al.:Towards a Living Software Development Process Based on Process Patterns. In:Software Process Technology, LNCS 2077, Springer, 2001, pp. 182-202.

[9] Hagen, M.; Gruhn V.: Process Patterns - a Means to Describe Processes in a Flexible Way. In: Proc. of ProSim 2004, Edinburgh, Scotland, 2004, pp. 32-39.

[10] Hagen, M.: Support for the definition and usage of process patterns. EuroPloP 2002.

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

[12] Noble, J.: Classifying Relationships Between Object-Oriented Design Patterns. ASWEC '98, 1998.

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

[14] Object Management Group: OMG Unified Modeling Language Specification, March 2003, Version 1.5.

[15] Störrle, H.: Models of Software Architecture. Dissertation, University Munich, 2000.

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