Aktuelles Schlagwort: Process Change Patterns

Barbara Weber[1], Stefanie Rinderle[2], and Manfred Reichert[3]

[1]Quality Engineering Research Group, University of Innsbruck, Austria
Barbara.Weber@uibk.ac.at
[2]Inst. Databases and Information Systems, Ulm University, Germany
stefanie.rinderle@uni–ulm.de
[3]Information Systems Group, University of Twente, The Netherlands
m.u.reichert@cs.utwente.nl

1 Introduction

Process-aware information systems (PAIS) allow for process changes at different levels. In general, respective changes must be enabled at the process type as well as the process instance level [1]. While the former become necessary to deal with the evolving nature of real-world processes (e.g., to adapt the process to legal changes), the latter are needed to cope with exceptional situations. Vendors often promise flexible solutions for realizing adaptive PAIS, but force users to realize process adaptations at a rather low level of abstraction. Apart from this, several competing paradigms exist, all trying to tackle the need for process flexibility (e.g., adaptive process management vs. case handling). So far, there has been no method for systematically comparing the different change frameworks, making it difficult for PAIS engineers to choose the right technology.

We have studied a variety of processes and process change scenarios from different domains (e.g. [2, 3]). We have further elaborated the change support features provided by existing tools. Taking these experiences, we have designed a set of *process changes patterns* to foster the comparison of existing approaches with respect to process change definition. More precisely, change patterns allow for high-level process adaptations at both the process type and the process instance level. In this paper we sketch 17 characteristic patterns we identified as relevant for *control flow changes* (cf. Fig. 1). Adaptations of other process aspects (e.g., data or resources) are outside the scope of this paper. Change patterns reduce the complexity of process change (like design patterns in software engineering reduce system complexity) and raise the level for expressing changes by providing abstractions which are above the level of single node and edge operations.

As illustrated in Fig. 1, we divide change patterns into *adaptation patterns* and *patterns for predefined changes*. Adaptation patterns allow modifying a process schema based on high-level change operations. Generally, adaptation patterns can be applied to the whole process schema; i.e., the region to which they are applied may be chosen dynamically. By contrast, for predefined changes, at build-time, the process engineer defines regions in the process schema where potential changes may be performed during run-time.

For each pattern we provide a name, a brief description, an illustrating example, a description of the problem it addresses, a couple of design choices, remarks regarding its implementation, and a reference to related patterns. *Design Choices* allow for parametrization of patterns keeping the number of distinct patterns manageable. Design choices which are not only relevant for particular patterns, but for a whole pattern category, are described only once at the category level. Typically, existing approaches only support a subset of the design choices in the context of a particular pattern. We denote the combination of design choices supported by a particular approach as a *pattern variant.*

CHANGE PATTERNS			
ADAPTATION PATTERNS (AP)			
Pattern Name	**Scope**	**Pattern Name**	**Scope**
AP1: Insert Process Fragment(*)	I / T	**AP8:** Embed Process Fragment in Loop	I / T
AP2: Delete Process Fragment	I / T	**AP9:** Parallelize Process Fragment	I / T
AP3: Move Process Fragment	I / T	**AP10:** Embed Process Fragment in Conditional Branch	I / T
AP4: Replace Process Fragment	I / T	**AP11:** Add Control Dependency	I / T
AP5: Swap Process Fragment	I / T	**AP12:** Remove Control Dependency	I / T
AP6: Extract Sub Process	I / T	**AP13:** Update Condition	I / T
AP7: Inline Sub Process	I / T		
PATTERNS FOR PREDEFINED CHANGES (PP)			
Pattern Name	**Scope**	**Pattern Name**	**Scope**
PP1: Late Selection of Process Fragments	I / T	**PP3:** Late Composition of Process Fragments	I / T
PP2: Late Modeling of Process Fragments	I / T	**PP4:** Multi-Instance Activity	I / T

I… Instance Level, T … Type Level
(*) A process fragment can either be an atomic activity, an encapsulated sub process or a process (sub) graph

Fig. 1. Change Patterns Overview

2 Adaptation Patterns

Adaptation patterns allow to structurally change process schemes. Examples include the insertion, deletion and re-ordering of activities (cf. Fig. 1). Fig. 2 describes general design choices valid for all adaptation patterns. First, each adaptation pattern can be applied at the process type or process instance level. Second, adaptation patterns can operate on an atomic activity, an encapsulated sub process or a process (sub-)graph (cf. Fig. 2). We abstract from this distinction and use the generic concept *process fragment* instead. Third, the effects resulting from the use of adaptation patterns at the instance level can be permanent or temporary. A *permanent instance change* remains valid until instance completion (unless it is undone by a user). By contrast, a *temporary instance change* is only valid for a certain period of time (e.g. one loop iteration).

We describe four selected adaptation patterns in more detail. These four patterns allow for the insertion, deletion, movement, and replacement of process fragments in a given process schema. The *Insert Process Fragment* pattern (cf. Fig. 3a) can be used to add process fragments to a process schema. In addition to

Design Choices for Adaptation Patterns

A. What is the scope of the respective pattern?
   1. The respective pattern can be applied at the process instance level
   2. The respective pattern can be applied at the process type level

B. Where does a respective change pattern operate on? (*)
   1. On an atomic activity
   2. On a sub process
   3. On a process sub-graph

C. What is the validity period of the change?
   1. The change can be of temporary nature
   2. The change can be of permanent nature

(*) Design Choice B is only valid for AP1-AP10

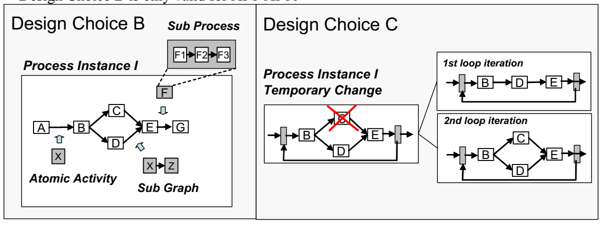


Fig. 2. Design Choices for Adaptation Patterns

the general options described in Fig. 2, one major design choice for this pattern (Design Choice D) describes the way the new process fragment is embedded in the respective schema. There are systems which only allow to serially insert a fragment between two directly succeeding activities. By contrast, other systems follow a more general approach allowing the user to insert new fragments between two arbitrary sets of activities. Special cases of the latter variant include the insertion of a fragment in parallel to another one or the association of the newly added fragment with an execution condition (*conditional insert*). The *Delete Process Fragment* pattern, in turn, can be used to remove a process fragment (cf. Fig 3b). No additional design choices exist for this pattern. Fig. 3b depicts alternative ways in which this pattern can be implemented.

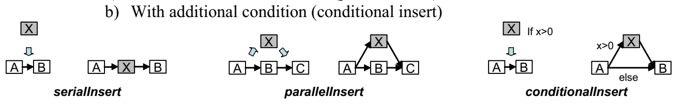
The *Move Process Fragment* pattern (cf. Fig. 4a) allows to shift a process fragment from its current position to another one. Like for the *Insert Process Fragment* pattern, an additional design choice specifies the way the fragment can be embedded in the process schema afterwards. Though the *Move* pattern could be realized by the combined use of patterns AP1 and AP2, we introduce it as separate pattern as it provides a higher level of abstraction to users. The latter also applies when a fragment has to be replaced by another one. This change is captured by the *Replace Process Fragment* pattern (cf. Fig. 4b).

We have only described the most relevant adaptation patterns. Additional patterns we identified are: swapping of activities (AP5), extraction of a sub process from a process schema (AP6), inclusion of a sub process into a process schema (AP7), embedding of an existing process fragment in a loop (AP8), parallelization of process fragments (AP9), embedding of a process fragment in a conditional branch (AP10), addition of control dependencies (AP11), removal of control dependencies (AP12), and update of transition conditions (AP13). A detailed description of these patterns can be found in [4, 5].

a) Pattern AP1: Insert Process Fragment

Description: A process fragment is added to a process schema.
Example: For a particular patient an allergy test has to be added due to a drug incompatibility.
Problem: In a real world process a task has to be accomplished which has not been modeled in the process schema so far.
Design Choices (in addition to the ones in Fig. 3):

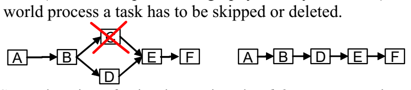
D. How is the additional process fragment X embedded in the process schema?
   1. X is inserted between 2 directly succeeding activities (serial insert)
   2. X is inserted between 2 activity sets (insert between node sets)
      a) Without additional condition (parallel insert)
      b) With additional condition (conditional insert)



Implementation: The *insert* adaptation pattern can be realized by transforming the high level insertion operation into a sequence of low level change primitives (e.g., add node, add control dependency).

b) Pattern AP2: Delete Process Fragment

Description: A process fragment is deleted from a process schema.
Example: For a particular patient no computer tomography is performed due to the fact that he has a cardiac pacemaker (i.e., the computer tomography activity is deleted).
Problem: In a real world process a task has to be skipped or deleted.

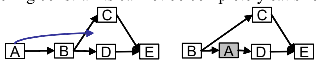


Implementation: Several options for implementing the *delete* pattern exist: (1) The fragment is physically deleted (i.e., corresponding activities and control edges are removed from the process schema), (2) the fragment is replaced by one or more null activities (i.e., activities without associated activity program) or (3) the fragment is embedded in a conditional branch with condition *false* (i.e., the fragment remains part of the schema, but is not executed).

Fig. 3. *Insert (AP1) and Delete (AP2) Process Fragment* patterns

a) Pattern AP3: Move Process Fragment

Description: A process fragment is moved from its current position in the process schema to another position.
Example: Usually employees are only allowed to book a flight, after getting approval from the manager. For a particular process instance the booking of a flight is exceptionally done in parallel to the approval activity (i.e., the book flight activity is moved from its current position to a position parallel to the approval activity).
Problem: Predefined ordering constraints cannot be completely satisfied for a set of activities.



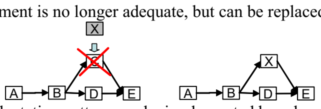
Design Choices:

D. How is the process fragment X embedded in the process schema?
   1. X is inserted between 2 directly succeeding activities (serial move)
   2. X is inserted between 2 activity sets (move between node sets)
      a) Without additional condition (parallel move)
      b) With additional condition (conditional move)

Implementation: This adaptation pattern can be implemented based on Pattern AP1 and AP2 (insert / delete process fragment).
Related Patterns: *Swap* adaptation pattern (AP5) (not detailed in the paper)

b) Pattern AP4: Replace Process Fragment

Description: A process fragment is replaced by another process fragment.
Example: Instead of the computer tomography activity, the X-ray activity shall be performed for a particular patient.
Problem: A process fragment is no longer adequate, but can be replaced by another one.



Implementation: This adaptation pattern can be implemented based on Pattern AP1 and AP2 (insert / delete process fragment).

Fig. 4. *Move (AP3) and Replace (AP4) Process Fragment* patterns

3 Patterns for Predefined Changes

The applicability of adaptation patterns is not restricted to a particular process part a priori. By contrast, the following patterns predefine constraints concerning the parts that can be changed; i.e., at run-time changes are only permitted within these parts. In this category we have identified 4 patterns, *Late Selection* (PP1), *Late Modeling* (PP2), *Late Composition* (PP3) and *Multi-Instance Activity* (PP4) (cf. Fig. 5). The *Late Selection* pattern (cf. Fig. 6) allows to select the implementation for a particular process step at run-time either based on predefined rules or user decisions. The *Late Modeling* pattern (cf. Fig. 7a) offers more freedom and allows to model selected parts of the process schema at run-time. The *Late Composition* pattern (cf. Fig. 7b) enables the on-the fly composition of process fragments (e.g., by dynamically introducing control dependencies between them). Finally, the *Multi-Instance Activity* pattern allows to determine the number of instances created for a particular process activity at run-time.

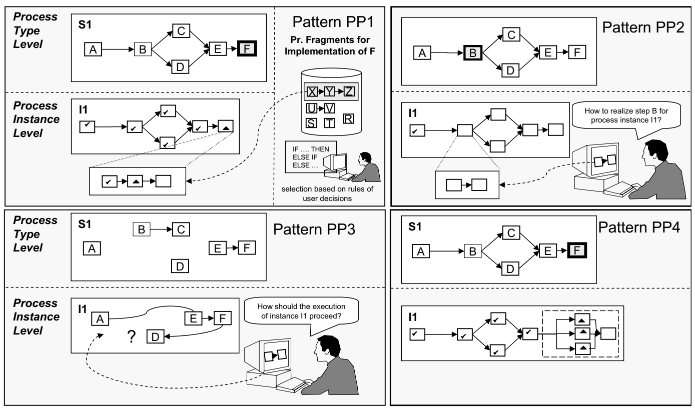
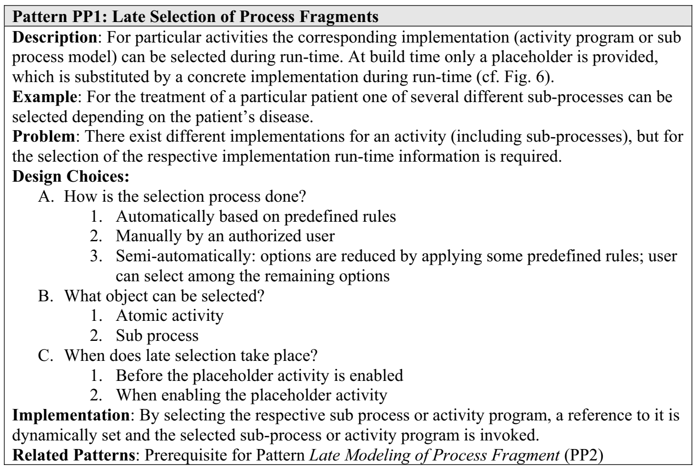

Process Type Level
Process Instance Level
S1
I1
Pattern PP1
Pr. Fragments for Implementation of F
IF .... THEN ELSE IF ELSE ...
selection based on rules of user decisions
Pattern PP2
How to realize step B for process instance I1?
Pattern PP3
How should the execution of instance I1 proceed?
Pattern PP4

Fig. 5. Patterns for Predefined Changes (Overview)

4 Related Work

Patterns were first used to describe solutions to recurring problems by Alexander, who applied them to descibe best practices in architecture [6]. Patterns also have a long tradition in computer science. Gamma et al. applied the same concepts to software engineering [7].

In the area of workflow management, patterns have been introduced for analyzing the expressiveness of process modeling languages (i.e., control flow patterns [8]). In addition, data patterns [9] describe different ways for modeling the data aspect in PAIS. The introduction of these patterns has had significant impact on the design of PAIS and has contributed to their systematic evaluation. However, to evaluate the powerfulness of a PAIS regarding its ability to deal with changes, the existing patterns are important, but not sufficient. In addition, a set of patterns for the aspect of workflow change is needed. Further, the



Pattern PP1: Late Selection of Process Fragments

Description: For particular activities the corresponding implementation (activity program or sub process model) can be selected during run-time. At build time only a placeholder is provided, which is substituted by a concrete implementation during run-time (cf. Fig. 6).
Example: For the treatment of a particular patient one of several different sub-processes can be selected depending on the patient's disease.
Problem: There exist different implementations for an activity (including sub-processes), but for the selection of the respective implementation run-time information is required.
Design Choices:

A. How is the selection process done?
   1. Automatically based on predefined rules
   2. Manually by an authorized user
   3. Semi-automatically: options are reduced by applying some predefined rules; user can select among the remaining options

B. What object can be selected?
   1. Atomic activity
   2. Sub process

C. When does late selection take place?
   1. Before the placeholder activity is enabled
   2. When enabling the placeholder activity

Implementation: By selecting the respective sub process or activity program, a reference to it is dynamically set and the selected sub-process or activity program is invoked.
Related Patterns: Prerequisite for Pattern *Late Modeling of Process Fragment* (PP2)

Fig. 6. Late Selection of Process Fragments (PP1)

degree to which control flow patterns are supported provides an indication of how complex the change framework under evaluation is.

In [10] exception handling patterns are proposed. In contrast to change patterns, exception handling patterns like *Rollback* only change the state of a process instance (i.e., its behavior), but not its schema. The patterns described in this paper do not only change the observable behavior of a process instance, but additionally adapt the process structure. For a complete evaluation of flexibility, both change patterns and exception handling patterns must be considered.

5 Summary and Outlook

We designed 17 change patterns which allow to assess the power of a particular change definition framework. Future work will include change patterns for aspects other than control flow (e.g., data or resources) and patterns for more advanced adaptation policies (e.g., the accompanying adaptation of the data flow when introducing control flow changes).

References

1. Rinderle, S., Reichert, M., Dadam, P.: Correctness criteria for dynamic changes in workflow systems – a survey. Data and Knowledge Engineering **50** (2004) 9–34
2. Lenz, R., Reichert, M.: IT support for healthcare processes - premises, challenges, perspectives. Data and Knowledge Engineering (2007) 39–58
3. Mueller, D., Herbst, J., Hammori, M., Reichert, M.: IT support for release management processes in the automotive industry. In: Proc. 4th Int'l Conf. Business Process Management (BPM'06), Vienna (2006) 368–377

a) Pattern PP2: Late Modeling of Process Fragments

Description: Parts of the process schema have not been defined at build-time, but are modeled during run-time for each process instance (cf. Fig. 6). For this purpose, placeholder activities are provided, which are modeled and executed during run-time. The modeling of the placeholder activity must be completed before the modeled process fragment can be executed.
Example: The exact treatment process of a particular patient is composed out of existing process fragments at run-time.
Problem: Not all parts of the process schema can be completely specified at build time.
Design Choices:

A. What are the basic building blocks for late modeling?
   1. All process fragments (including activities) from the repository can be chosen
   2. A constraint-based subset of the process fragments from the repository can be chosen
   3. New activities or process fragments can be defined

B. What is the degree of freedom regarding late modeling?
   1. Same modeling constructs and change patterns can be applied as for modeling at the process type level (*)
   2. More restrictions apply for late modeling than for modeling at the process type level

C. When does late modeling take place?
   1. When a new process instance is created
   2. When the placeholder activity is instantiated
   3. When a particular state in the process is reached (which must precede the instantiation of the placeholder activity)

D. Does the modeling start from scratch?
   1. Late modeling may start with an empty template
   2. Late modeling may start with a predefined template which can then be adapted

Implementation: After having modeled the placeholder activity with the editor, the fragment is stored in the repository and deployed. Finally, the process fragment is dynamically invoked as an encapsulated sub-process. The assignment of the respective process fragment to the placeholder activity is done through late binding.
Related Patterns: necessitates *Late Selection of Process Fragments* (PP1) of the dynamically modified fragment

(*) Which of the adaptation patterns are supported within the placeholder activity is determined by the expressiveness of the used modeling language.

b) Pattern PP3: Late Composition of Process Fragments

Description: At build time a set of process fragments is defined out of which a concrete process instance can be composed at run time. This can be achieved by dynamically selecting fragments and adding control dependencies on the fly (cf. Fig. 6).
Example: Several medical examinations can be applied for a particular patient. The exact examinations and the order in which they are performed are defined for each patient individually.
Problem: There exist several variants of how process fragments can be composed. In order to reduce the number of process variants to be specified by the process engineer during build time, process instances are dynamically composed out of fragments.

Fig. 7. Late Modeling (PP2) and Late Composition of Process Fragments (PP3)

4. Weber, B., Rinderle, S., Reichert, M.: Change patterns and change support features in process-aware information systems. In: Proc. 19th Int'l Conf. on Advanced Information Systems Engineering (CAiSE'07). (2007) 574–588
5. Weber, B., Rinderle, S., Reichert, M.: Identifying and evaluating change patterns and change support features in process-aware information systems. Technical Report Report No. TR-CTIT-07-22, CTIT, Univ. of Twente, The Netherlands (2007)
6. Alexander, C., Ishikawa, S., Silverstein, M.: A Pattern Language. Oxford University Press, New York (1977)
7. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley (1995)
8. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distributed and Parallel Databases **14** (2003) 5–51
9. Russell, N., ter Hofstede, A., Edmond, D., van der Aalst, W.: Workflow data patterns. Technical Report FIT-TR-2004-01, Queensland Univ. of Techn. (2004)
10. Russell, N., van der Aalst, W.M., ter Hofstede, A.H.: Exception handling patterns in process-aware information systems. In: CAiSE'06. (2006)