Change Patterns and Change Support Features in Process-Aware Information Systems

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

Abstract. In order to provide effective support, the introduction of process-aware information systems (PAIS) must not freeze existing business processes. Instead PAIS should allow authorized users to flexibly deviate from the predefined processes if required and to evolve business processes in a controlled manner over time. Many software vendors promise flexible system solutions for realizing such adaptive PAIS, but are often unable to cope with fundamental issues related to process change (e.g., correctness and robustness). The existence of different process support paradigms and the lack of methods for comparing existing change approaches makes it difficult for PAIS engineers to choose the adequate technology. In this paper we suggest a set of changes patterns and change support features to foster systematic comparison of existing process management technology with respect to change support. Based on these change patterns and features, we provide an evaluation of selected systems.

1 Introduction

Contemporary information systems (IS) more and more have to be aligned in a process-oriented way. This new generation of IS is often referred to as Process-Aware IS (PAIS) [1]. In order to provide effective process support, PAIS should capture real-world processes adequately, i.e., there should be no mismatch between the computerized processes and those in reality. In order to achieve this, the introduction of PAIS must not lead to rigidity and freeze existing business processes. Instead PAIS should allow authorized users to flexibly deviate from the predefined processes as required (e.g., to deal with exceptions) and to evolve PAIS implementations over time (e.g., due to process optimizations or legal changes). Such process changes should be enabled at a high level of abstraction and without affecting the robustness of the PAIS [2].

The increasing demand for process change support poses new challenges for IS engineers and requires the use of change enabling technologies. Contemporary

⋆ This work was done during a postdoctoral fellowship at the University of Twente.

J. Krogstie, A.L. Opdahl, and G. Sindre (Eds.): CAiSE 2007, LNCS 4495, pp. 574–588, 2007.

PAIS, in combination with service-oriented computing, offer promising perspectives in this context. Many vendors promise flexible software solutions for realizing adaptive PAIS, but are often unable to cope with fundamental issues related to process change (e.g., correctness and robustness). This problem is further aggravated by the fact that several competing process support paradigms exist, all trying to tackle the need for more process flexibility (e.g., adaptive processes [3,4,5] or case handling [6]). Furthermore, there exists no method for systematically comparing the change frameworks provided by existing process-support technologies. This, in turn, makes it difficult for PAIS engineers to assess the maturity and change capabilities of those technologies. Consequently, this often leads to wrong decisions and misinvestments.

During the last years we have studied processes from different application domains and elaborated the flexibility and change support features of numerous tools and approaches. Based on these experiences, in this paper we suggest a set of *changes patterns* and *change support features* to foster the comparison of existing approaches with respect to process change support. Change patterns allow for high-level process adaptations at the process type as well as the process instance level. Change support features ensure that changes are performed in a correct and consistent way, traceability is provided, and changes are facilitated for users. Both change patterns and change support features are fundamental to make changes applicable in practice. Finally, another contribution of this paper is the evaluation of selected approaches/systems based on the presented change patterns and change support features.

Section 2 summarizes background information needed for the understanding of this paper. Section 3 describes 17 change patterns and Section 4 deals with 6 crucial change support features. Based on this, Section 5 evaluates different approaches from both academia and industry. Section 6 discusses related work and Section 7 concludes with a summary.

2 Backgrounds

A PAIS is a specific type of information system which allows for the separation of process logic and application code. At run-time the PAIS orchestrates the processes according to their defined logic. Workflow Management Systems (e.g., Staffware [1], ADEPT [3], WASA [5]) and Case-Handling Systems (e.g., Flower [1,6]) are typical technologies enabling PAIS.

For each business process to be supported a process type represented by a *process schema* S has to be defined. In the following, a process schema is represented by a directed graph, which defines a set of *activities* – the process steps – and control connections between them (i.e., the precedence relations between these activities). Activities can either be atomic or contain a sub process (i.e., a reference to a process schema S') allowing for the hierarchical decomposition of a process schema. In Fig. 1a, for example, process schema $S1$ consists of six activities: Activity `A` is followed by activity `B` in the flow of control, whereas `C` and `D` can be processed in parallel. Activities `A` to `E` are atomic, and activity `F` constitutes a sub process with own process schema $S2$. Based on a process

schema S, at run-time new *process instances* $I_1, \ldots, I_n$ can be created and executed. Regarding process instance I_1 from Fig. 1a, for example, activity `A` is completed and activity `B` is activated (i.e., offered in user worklists). Generally, a large number of process instances might run on a particular process schema.

PAIS must be able to cope with change. In general, changes can be triggered and performed at two levels – the process type and the process instance level (cf. Fig. 1b) [2]. Schema changes at the type level become necessary to deal with the evolving nature of real-world processes (e.g., to adapt to legal changes). Ad-hoc changes of single instances are usually performed to deal with exceptions, resulting in an adapted *instance-specific* process schema.

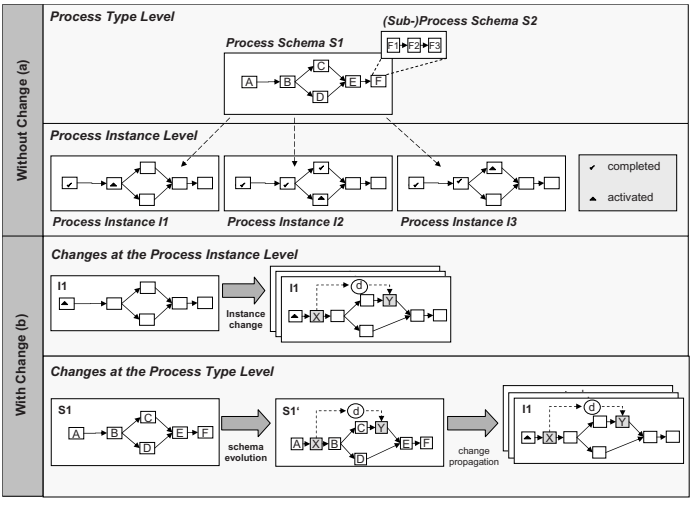


Fig. 1. Core Concepts

3 Change Patterns

In this section we describe 17 characteristic patterns we identified as relevant for *control flow changes* (cf. Fig. 2). 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 [7]) and raise the level for expressing changes by providing abstractions which are above the level of single node and edge operations. Consequently, due to their lack of abstraction, low level change primitives (add node, delete edge, etc.) are not considered to be change patterns and thus are not covered in this section.

As illustrated in Fig. 2, we divide our change patterns into *adaptation patterns* and *patterns for predefined changes*. Adaptation patterns allow modifying the schema of a process type (type level) or a process instance (instance level) using high-level change operations. Generally, adaptation patterns can be

applied to the whole process schema or process instance schema respectively; they do not have to be pre-planned, i.e., the region to which the adaptation pattern is applied can 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. 2. Change Patterns Overview

3.1 Adaptation Patterns

Adaptation patterns allow to structurally change process schemes. Examples include the insertion, deletion and re-ordering of activities (cf. Fig. 2). Fig. 3 describes general design choices valid for all adaptation patterns. First, each adaptation pattern can be applied at the process type or process instance level (cf. Fig. 1b). Second, adaptation patterns can operate on an atomic activity, an encapsulated sub process or a process (sub-)graph (cf. Fig. 3). We abstract from this distinction and use the generic concept *process fragment* instead. Third, the effects resulting from the use of an adaptation pattern at the instance level can be permanent or temporary. A *permanent instance change* remains valid until completion of the instance (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) (cf. Fig. 3).

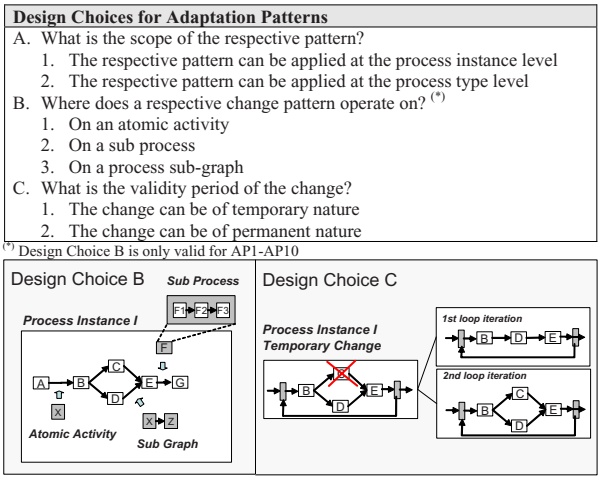


Fig. 3. Design Choices for Adaptation Patterns

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. 4a) can be used to add process fragments to a process schema. In addition to the general options described in Fig. 3, 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 [3]. 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 4b). No additional design choices exist for this pattern. Fig. 4b depicts alternative ways in which this pattern can be implemented.

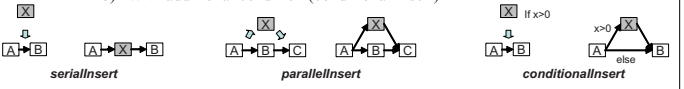
The *Move Process Fragment* pattern (cf. Fig. 5a) allows to shift a process fragment from its current position to a new one. Like for the *Insert Process Fragment* pattern, an additional design choice specifies the way the fragment can be embedded in the process schema afterwards. Though the *Move Process Fragment* pattern could be realized by the combined use of AP1 and AP2 (*Insert/Delete Process Fragment*), we introduce it as separate pattern as it provides a higher level of abstraction to users. The latter also applies when a process fragment has to be replaced by another one. This change is captured by the *Replace Process Fragment* pattern (cf. Fig. 5b).

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),

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):

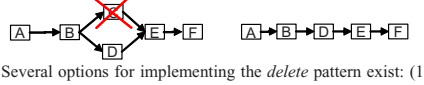
D. How is the additional process fragment X embedded in the process schema?
1. X is inserted between 2 directly succeeding activities (serial insert)
2. X is inserted between 2 activity sets (insert between node sets)
   a) Without additional condition (parallel insert)
   b) With additional condition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

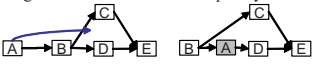


Implementation: Several options for implementing the *delete* pattern exist: (1) The fragment is physically deleted (i.e., corresponding activities and control edges are removed from the process schema), (2) the fragment is replaced by one or more null activities (i.e., activities without associated 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. 4. 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.



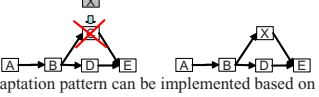
Design Choices:

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. 5. Move (AP3) and Replace (AP4) Process Fragment patterns

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 description of these patterns can be found in [8].

3.2 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. At run-time changes are only permitted within these parts. In this category we have identified 4 patterns, *Late Selection of Proces Fragments* (PP1), *Late Modeling of Process Fragments* (PP2), *Late Composition of Process Fragments* (PP3) and *Multi-Instance Activity* (PP4) (cf. Fig. 6). The *Late Selection of Process Fragments* pattern (cf. Fig. 7) allows to select the implementation for a particular process step at run-time either based on predefined rules or user decisions. The *Late Modeling of Process Fragments* pattern (cf. Fig. 8a) offers more freedom and allows to model selected parts of the process schema at run-time. Furthermore the *Late Composition of Process Fragments* pattern (cf. Fig. 8b) enables the on-the fly composition of process fragments (e.g., by dynamically introducing control dependencies between a set of fragments).

In case of *Multi-Instance Activities* the number of instances created for a particular activity is determined at run-time. We do not consider multi-instance activity patterns in detail as they constitute some of the workflow patterns described in [9]. Multi-instance activities enable the creation of a particular process activity during run-time. The decision how many activity instances are created can be based either on knowledge available at build-time or on some run-time knowledge. We do not consider multi-instances of the former kind as change pattern since their use does not lead to change. For all other types of multi-instance activities the number of instances is determined based on run-time knowledge which can or cannot be available a-priori to the execution of the multi-instance activity. While in the former case the number of instances can be determined at some point during run-time, this is not possible for the latter case. We consider multi-instance activities as change patterns too, since their dynamic creation works like a dynamic schema expansion.

4 Change Support Features

So far, we have introduced a set of change patterns, which can be used to accomplish changes at the process type and/or process instance level. However, simply counting the number of supported patterns is not sufficient to analyze how well a system can deal with process change. In addition, change support features must be considered to make change patterns useful in practice (cf. Fig. 9). Relevant change support features include *process schema evolution* and *version control*,

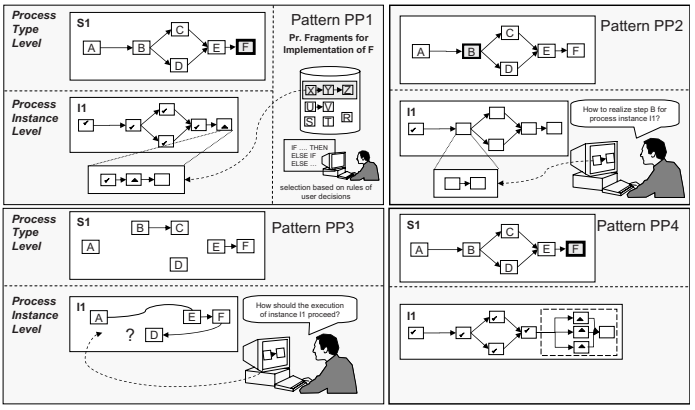


Fig. 6. Patterns for Predefined Changes (Overview)

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. 7. Late Selection of Process Fragments (PP1)

change correctness, change traceability, access control and change reuse[1]. As illustrated in Fig. 9 the described change support features are not equally important for both process type level and process instance level changes. Version control, for instance, is primarily relevant for changes at the type level, while change reuse is particularly useful at the instance level [10].

4.1 Schema Evolution, Version Control and Instance Migration

In order to support changes at the process type level, version control for process schemes should be supported (cf. Fig. 9). In case of long-running processes, in

[1] Again we restrict ourselves to the most relevant change support features. Additional change support features not covered in this paper are change concurrency control and change visualization.

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

Change Support Features			
Change Support Feature	**Scope**	**Change Support Feature**	**Scope**
F1: Schema Evolution, Version Control and Instance Migration	T	2. By change primitives	
		F3: Correct Behavior of Instances After Change	I + T
No version control – Old schema is overwritten		**F4: Traceability & Analysis**	I + T
1. Running instances are canceled		1. Traceability of changes	
2. Running instances remain in the system		2. Annotation of changes	
Version control		3. Change Mining	
3. Co-existence of old/new instances, no instance migration		**F5: Access Control for Changes**	I+T
4. Uncontrolled migration of all process instances		1. Changes in general can be restricted to authorized users	
5. Controlled migration of compliant process instances		2. Application of single change patterns can be restricted	
F2: Support for Ad-hoc Changes	I	3. Authorizations can depend on the object to be changed	
1. By change patterns		**F6: Change Reuse**	I

T … Type Level, I … Instance Level

Fig. 9. Change Support Features

addition, controlled migration of already running instances, from the old process schema version to the new one, might be required. In this subsection we describe different existing options in this context (cf. Fig. 10).

If a PAIS provides no version control feature, either the process designer can manually create a copy of the process schema (to be changed) or this schema is overwritten (cf. Fig. 10a). In the latter case running process instances can either be withdrawn from the run-time environment or, as illustrated in Fig. 10a, they remain associated with the modified schema. Depending on the execution state of the instances and depending on how changes are propagated to instances which have already progressed too far, this missing version control can lead to inconsistent states and, in a worst case scenario, to deadlocks or other errors [2]. As illustrated in Fig. 10a process schema $S1$ has been modified by inserting activities X and Y with a data dependency between them. For instance $I1$ the change is uncritical, as $I1$ has not yet entered the change region. However, $I2$ and $I3$ would be both in an inconsistent state afterwards as instance schema and execution history do not match (see [2]). Regarding $I2$, worst case, deadlocks or activity invocations with missing input data might occur.

By contrast, if a PAIS provides explicit version control two support features can be differentiated: running process instances remain associated with the old schema version, while new instances will be created on the new schema version. This approach leads to the co-existence of process instances of different schema versions (cf. Fig. 10b). Alternatively a migration of a selected collection of process instances to the new process schema version is supported (in a controlled way) (cf. Fig. 10c). The first option is shown in Fig. 10b where the already running instances $I1$, $I2$ and $I3$ remain associated with schema S1, while new instances ($I4$-$I5$) are created from schema $S1'$ (co-existence of process instances of different schema versions). By contrast, Fig. 10c illustrates the controlled migration of process instances. Only those instances are migrated which are *compliant*[2] with $S1'$ ($I1$). All other instances ($I2$ and $I3$) remain running according to $S1$. If instance migration is uncontrolled (as it is not restricted to *compliant* process instances) this will lead to inconsistencies or errors. Nevertheless, we treat the uncontrolled migration of process instances as a separate design choice since this functionality can be found in several existing systems (cf. Section 5).

4.2 Other Change Support Features

Support for Ad-hoc Changes: In order to deal with exceptions PAIS must support changes at the process instance level either through high level changes in the form of patterns (cf. Section 3) or through low level primitives. Although changes can be expressed in both ways, change patterns allow to define changes at a higher level of abstraction making change definition easier.

Correctness of Change: The application of change patterns must not lead to run-time errors (e.g., activity program crashes due to missing input data, deadlocks, or inconsistencies due to lost updates or vanishing of instances).

[2] A process instance I is compliant with process schema S, if the current execution history of I can be created based on S (for details see [2]).

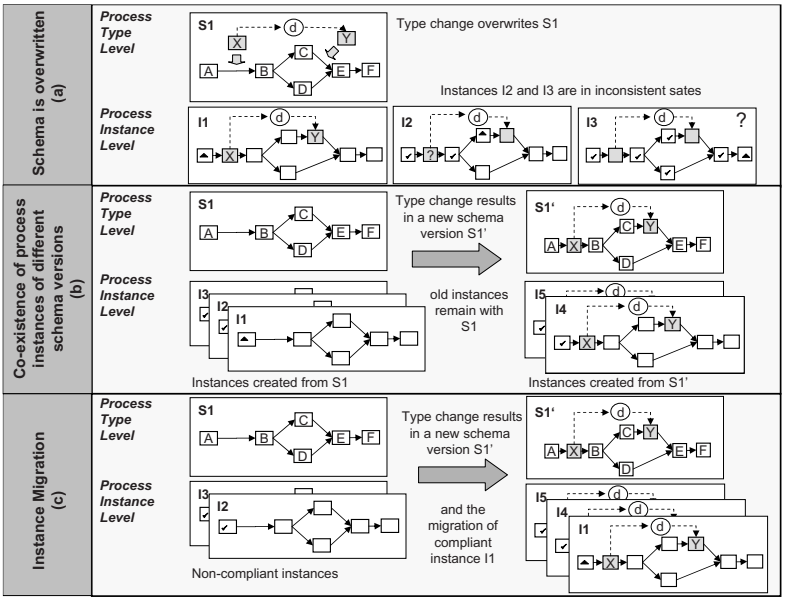


Fig. 10. Version Control

Different criteria (see [2]) have been introduced to ensure that instances can only be updated to a new schema if they are compliant with it.

Traceability and Analysis: To ensure traceability of changes, they have to be logged. For adaptation patterns the applied changes have to be stored in a change log as change patterns and/or change primitives. While both options allow for traceability, change mining [11] becomes easier when the change log contains high-level information about the changes as well. Regarding patterns for predefined changes, an execution log is usually sufficient to enable traceability. In addition, logs can be enriched with more semantical information, e.g., about the reasons and context of the changes [10]. Finally, change mining allows for the analysis of changes (e.g., to support continuous process improvement) [11].

Access Control for Changes: The support of change patterns leads to increased PAIS flexibility. This, in turn, imposes security issues as the PAIS becomes more vulnerable to misuse. Therefore, the application of changes at the process type and the process instance level must be restricted to authorized users. Access control features differ significantly in their degree of granularity. In the simplest case, changes are restricted to a particular group of people (e.g., to process engineers). More advanced access control components allow to define restrictions at the level of single change patterns (e.g., a certain user is only allowed to insert additional activities, but not to delete activities). In addition, authorizations can depend on the object to be changed, e.g., the process schema.

Change Reuse: In the context of ad-hoc changes "similar" deviations (i.e., combination of one or more adaptation patterns) can occur more than once. As

it requires significant user experience to define changes from scratch change reuse should be supported. To reuse changes they must be annotated with contextual information (e.g., about the reasons for the deviation) and be memorized by the PAIS. This contextual information can be used for retrieving similar problem situations and therefore ensures that only changes relevant for the current situation are presented to the user [12,10]. Regarding patterns for predefined changes, reuse can be supported by making historical cases available to the user and by saving frequently re-occurring instances as templates.

5 Change Patterns and Change Support in Practice

In this section we evaluate approaches from both academia and industry regarding their support for change patterns as well as change features. For academic approaches the evaluation has been mainly based on literature. In cases where it was unclear whether a particular change pattern or change feature is supported or not, the respective research groups were additionally contacted. The evaluated academic approaches are ADEPT[3], WIDE [13], Pockets of Flexibility [14], Worklets/Exlets [4,15], CBRFlow [12,10], MOVE [16], HOON [17], and WASA [5]. In respect to commercial systems only such systems have been considered for which we have hands on experience as well as a running system installed. This allowed us to test the change patterns and change features. As commercial systems Staffware [1] and Flower [6] were considered. Evaluation results are summarized in Fig. 11. A detailed description of the evaluated approaches can be found in [8].

If a change pattern or change support feature is not supported at all, the respective table entry will be labeled with "-". Otherwise, it describes the exact pattern variants as supported by listing all available design choices. In case no design choices exist for a particular change pattern, which is supported, the respective table entry is simply labeled with "+". Partial support is labeled with "∘". As an example take change pattern PP1 of the Worklet/Exlet approach [4,15]. The string "A[1,2], B[1,2], C[2]" indicates that design choices A, B and C are supported. Further, it shows for every design choice the exact options available (e.g., for design choice A, Options 1 and 2 are supported).

In particular an adaptation pattern will be only considered as being provided, if the respective system supports the pattern directly, i.e., based on one high-level change operation. Of course, adaptation patterns can be always expressed by means of a set of basic change primitives (like add node, delete node, add edge, etc.). However, this is not the idea behind adaptation patterns. Since process schema changes (at the type level) based on these modification primitives are supported by almost each process editor, this is not sufficient to qualify for pattern support. By contrast, the support of high-level change operations allows introducing changes at a higher level of abstraction and consequently hides a lot of the complexity from the user. Therefore changes can be performed in a more efficient and less error prone way. In addition, in order to qualify as an adaptation pattern the application of the respective change operations must not be restricted to predefined regions in the process.

Several of the adaptation patterns (e.g., AP3 or AP4) can be implemented by applying a combination of the more basic patterns AP1, AP2, AP10 and AP11. However, a given approach will only qualify for a particular adaptation pattern, if it supports this pattern directly (i.e., it offers one respective change operation).

Note that missing support for adaptation patterns does not necessarily mean that no run-time changes can be performed. As long as feature F2 is supported ad-hoc changes to running process instances are possible (for details see [8]). In general, if a respective approach provides support for predefined change patterns like for instance late modeling of process fragments (PP1) or late selection of process fragments (PP2) the need for structural changes of the process schema can be decreased making feature F3 less crucial.

The evaluation of selected approaches shows that there exists no single system which supports all change patterns and features (cf. Table 11). In particular, none of the approaches provides both adaptation patterns and predefined change patterns, which would allow addressing a much broader process spectrum. While predefined change patterns allow to reduce the need for structural changes during run-time by providing more flexible models, adaptation patterns allow for structural changes which cannot be pre-planned. In addition, they make changes more efficient, less complex and less error-prone through providing high-level change operations.

Change Patterns and Change Support									
Pattern/ Feature	Academic							Commercial	
	ADEPT / CBRFlow	WIDE	Pockets of Flexibility	Worklets / Exlets	MOVE	HOON	WASA	Staffware	Flower
Change Patterns									
Adaptation Patterns									
AP1	A[1, 2], B[1,2,3], C[1,2], D[1, 2]	A[2], B[1], C[2], D[1,2]	–	–	–	–	–	–	–
AP2	A[1, 2], B[1,2,3], C[1,2]	A[2], B[1], C[2]	–	–	–	–	–	–	A[2], B[1], C[2]
AP3	A[1, 2], B[1,2,3], C[1,2], D[1, 2]	–	–	–	–	–	–	–	–
AP4	–	A[2], B[1], C[2]	–	A[1], B[2], C[1,2]	–	–	–	–	–
Preplanned Change Patterns									
PP1	–	–	–	A[1,2], B[1,2], C[2]	–	A[1,2], B[1,2], C[2]	–	A[1,2], B[1,2], C[2]	–
PP2	–	–	A[1,2], B[2], C[2], D[1,2]	–	A[1], B[1], C[3], D[1,2]	–	–	–	–
PP3	–	–	–	–	–	–	–	–	–
PP4	–	+	–	–	–	–	–	+	+
Change Features									
F1	3, 5	3, 5	–	3	–	–	3, 5	3, 4	1, 2, 3
F2	1	–	2	2	2	2	2	2	1
F3	+	+	+	°	+	+	+	–	–
F4	1, 2, 3	1	1	1	1	1	1	1	1
F5	1, 2, 3	1, 3	1, 2, 3	1, 2, 3	1, 3	1, 2, 3	1	1, 2, 3	1, 2, 3*
F6	+	–	+	+	–	–	–	–	–

(*) Flower supports Option 2 and 3 of feature F4 only for process instance changes, but not for process type changes

Fig. 11. Change Patterns and Change Support Features in Practice

6 Related Work

Patterns were first used to describe solutions to recurring problems by Ch. Alexander, who applied patterns to descibe best practices in architecture [18]. Patterns also have a long tradition in computer science. Gamma et al. applied the same concepts to software engineering and described 23 patterns in [7].

In the area of workflow management, patterns have been introduced for analyzing the expressiveness of process modeling languages (i.e., control flow patterns [9]). In addition, workflow data patterns [19] describe different ways for modeling the data aspect in PAIS and workflow resource patterns [20] describe how resources can be represented and utilized in workflows. The introduction of workflow patterns has significant impact on the design of PAIS and has contributed to the systematic evaluation of PAIS and process modeling standards. 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 degree to which control flow patterns are supported provides an indication of how complex the change framework under evaluation is. In general, the more expressive the process modeling language is (i.e., the more control flow and data patterns are supported), the more difficult and complex changes become.

In [21] exception handling patterns which describe different ways for coping with exceptions 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 evaluated.

7 Summary and Outlook

In this paper we proposed 17 change patterns (and described 8 of them in detail) and 6 change support features, which in combination allow to assess the power of a particular change framework. In addition, we evaluated selected approaches and systems regarding their ability to deal with process changes. We believe that the introduction of change patterns complements existing workflow patterns and allows for more meaningful evaluations of existing systems and approaches. In combination with workflow patterns the presented change framework will enable (PA)IS engineers to choose process management technologies

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) as well as the evaluation of additional systems and approaches.

Acknowledgements. We would like to thank S. Shadiq, M. Adams, M. Weske and Y. Han for their valuable feedback and the many fruitful discussions, which helped us to significantly improve this paper.

References

1. Dumas, M., ter Hofstede, A., van der Aalst, W. (eds.): Process Aware Information Systems. Wiley Publishing, Chichester (2005)
2. Rinderle, S., Reichert, M., Dadam, P.: Correctness criteria for dynamic changes in workflow systems – a survey. Data and Knowledge Engineering 50, 9–34 (2004)
3. Reichert, M., Dadam, P.: ADEPT$_{flex}$ – supporting dynamic changes of workflows without losing control. JIIS 10, 93–129 (1998)
4. Adams, M., ter Hofstede, A.H.M., Edmond, D., v.d.Aalst, W.M.: A service-oriented implementation of dynamic flexibility in workflows. In: Coopis'06 (2006)
5. Weske, M.: Workflow management systems: Formal foundation, conceptual design, implementation aspects. University of Münster, Germany, Habil Thesis (2000)
6. van der Aalst, W., Weske, M., Grünbauer, D.: Case handling: A new paradigm for business process support. Data and Knowledge Engineering. 53, 129–162 (2005)
7. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, New York (1995)
8. 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)
9. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distributed and Parallel Databases 14, 5–51 (2003)
10. Rinderle, S., Weber, B., Reichert, M., Wild, W.: Integrating process learning and process evolution - a semantics based approach. In: BPM 2005, pp. 252–267 (2005)
11. Günther, C., Rinderle, S., Reichert, M., van der Aalst, W.: Change mining in adaptive process management systems. In: CoopIS'06, pp. 309–326 (2006)
12. Weber, B., Wild, W., Breu, R.: CBRFlow: Enabling adaptive workflow management through conversational cbr. In: ECCBR'04, Madrid, pp. 434–448 (2004)
13. Casati, F.: Models, Semantics, and Formal Methods for the design of Workflows and their Exceptions. PhD thesis, Milano (1998)
14. Sadiq, S., Sadiq, W., Orlowska, M.: A framework for constraint specification and validation in flexible workflows. Information Systems 30, 349–378 (2005)
15. Adams, M., ter Hofstede, A.H.M., Edmond, D., v. d. Aalst, W.M.: Dynamic and extensible exception handling for workflows: A service-oriented implementation. Technical Report BPM Center Report BPM-07-03, BPMcenter.org (2007)
16. Th. Herrmann, A.-W., Scheer, H.W. (eds.): Verbesserung von Geschftsprozessen mit flexiblen Workflow-Management-Systemen - Verffentlichungen des Forschungsprojektes MOVE. Bd. 1 - 4. Physica Verlag, Heidelberg (1998)
17. Han, Y.: Software Infrastructure for Configurable Workflow Systems. PhD thesis, Univ. of Berlin (1997)
18. Alexander, C., Ishikawa, S., Silverstein, M. (eds.): A Pattern Language. Oxford University Press, New York (1977)
19. 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)
20. Russell, N., ter Hofstede, A., Edmond, D., van der Aalst, W.: Workflow resource patterns. Technical Report WP 127, Eindhoven Univ. of Technology (2004)
21. Russell, N., van der Aalst, W.M., ter Hofstede, A.H.: Exception handling patterns in process-aware information systems. In: CAiSE'06 (2006)