Identifying and Evaluating 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 a detailed analysis and evaluation of selected systems from both academia and industry.

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].

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

The increasing demand for process change support poses new challenges for IS engineers and requires the use of change enabling technologies. Contemporary 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 [3–5] 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.

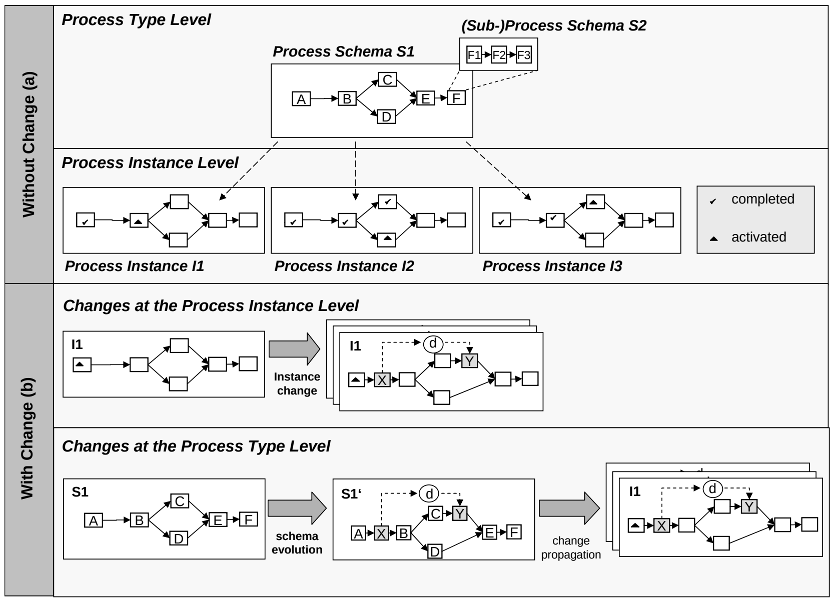


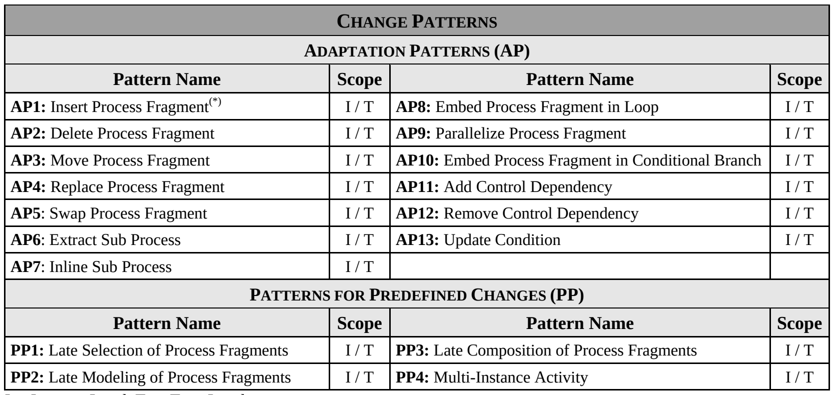
Fig. 1. Core Concepts

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 parameterization 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).

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

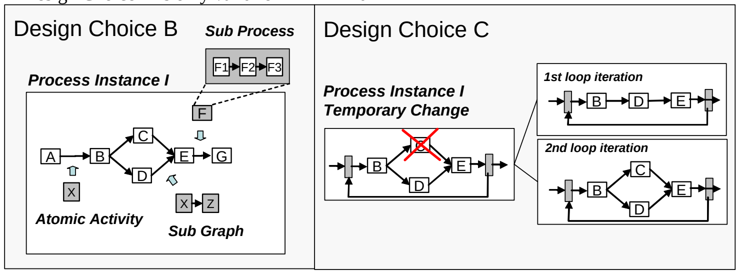


Fig. 3. Design Choices for Adaptation Patterns

In the following all 13 adaptation patterns are described in detail. These 13 patterns allow for the insertion (AP1), deletion (AP2), movement (AP3), and replacement (AP4) of process fragments in a given process schema. They further allow for the 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).

Adaptation Pattern AP1: Insert Process Fragment. The *Insert Process Fragment* pattern (cf. Fig. 5) 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*).

Adaptation Pattern AP2: Delete Process Fragment. The *Delete Process Fragment* pattern, in turn, can be used to remove a process fragment (cf. Fig 6). No additional design choices exist for this pattern. Fig. 5b depicts alternative ways in which this pattern can be implemented.

Adaptation Pattern AP3: Move Process Fragment. The *Move Process Fragment* pattern (cf. Fig. 7) 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.

Adaptation Pattern AP4: Replace Process Fragment. This pattern supports the replacement of a process fragment by another one (cf. Fig. 8). Like the *Move Process Fragment* pattern, this pattern can be implemented based on patterns AP1 and AP2 (*Insert/Delete Process Fragment*).

Adaptation Pattern AP5: Swap Process Fragments. The *Swap Process Fragment* pattern (cf. Fig. 9) allows to swap a process fragment with another one. This adaptation pattern can be implemented based on AP3 (*Move Process Fragment)* or on the combined use of patterns AP1 and AP2 (*Insert/Delete Process Fragment*).

Adaptation Pattern AP6: Extract Sub Process. The pattern *Extract Sub Process* (AP6) allows to extract an existing process fragment from a process schema and to encapsulate it in a separate sub process schema (cf. Fig. 10). This pattern can be used to add an additional hierarchical level in order to simplify a respective process schema or to hide information from process participants. If no direct support of pattern AP6 is provided a workaround could look as follows: The new process schema representing the extracted sub process has to be created manually. As a next step the respective process fragment must be copied to the new process schema and be removed from the original one. In addition, an activity referencing the newly implemented sub process must be added to the original process schema. Further, required input and output parameters must be manually mapped to the sub process. In general, the implementation of pattern AP6 should be based on graph aggregation techniques.

Adaptation Pattern AP7: Inline Sub Process. As opposed to pattern AP6, the pattern *Inline Sub Process* (AP7) allows to inline a sub process schema into the parent process schema, and consequently to flatten the hierarchy of the overall process (cf. Fig. 11). If no direct support for patterns AP7 is provided a couple of manual steps will be required as workaround. First the process fragment representing the sub process has to be copied to the parent process schema. In a next step the activity which has invoked the sub process has to be replaced by the previously copied process fragment. Further, input and output parameters of the sub process have to manually mapped to the newly added activities.

Adaptation Pattern AP8: Embed Process Fragment in Loop. Using this pattern an existing process fragment can be embedded in a loop in order to allow for a repeated execution of the respective process fragment (cf. Fig. 12a). This patterns can be realized based on Patterns AP1 (*Insert Process Fragment*), AP11 and AP12 (*Add / Remove Control Dependency*).

Adaptation Pattern AP9: Parallelize Process Fragments. This patterns enables the parallelization of process fragments which were confined to be executed in sequence (cf. Fig. 12b). If no direct support is provided for AP9, it can be realized by combining patterns AP11 and AP12 (*Add / Remove Control Dependency*).

Adaptation Pattern AP10: Embed Process Fragment in Conditional Branch. Using this pattern an existing process fragment can be embedded in a conditional branch, which consequently is only executed if certain conditions are met (cf. Fig. 13a). AP10 can be implemented based on patterns AP1 (*Insert Process Fragment*), AP11 and AP12 (*Add / Remove Control Dependency*).

Adaptation Pattern AP11: Add Control Dependency. When applying this adaptation pattern an additional control edge (e.g., for synchronizing the execution order of two parallel activities) is added to the given process schema (cf. Fig. 13b). As opposed to the low-level change primitive *add edge*, the added control dependency can be associated with attributes when applying pattern AP11. As an example consider the use of transition conditions. Another parameterization of AP11 will become necessary if different kinds of control dependencies (e.g., loop backs, synchronization of parallel activities) have to be considered. Usually, approaches implementing AP11 also ensure that the use of this pattern meets certain pre- and post-conditions (e.g., guaranteeing the absence of cycles or deadlocks).

Adaptation Pattern AP12: Remove Control Dependency. Using this pattern a control dependency and its attributes can be removed from a process schema (cf. Fig. 14a). Similar considerations can be made as for pattern AP11.

Adaptation Pattern AP13: Update Condition. This pattern allows to update transition conditions in a process schema (cf. Fig. 14b). Usually, an implementation of this pattern has to ensure that the new transition condition correct in the context of the given process schema (e.g., all workflow relevant data elements the transition condition refers to must be present in the process schema).

To put the pattern-based analysis of existing systems on a firm footing we have defined a formal *semantics for adaptation patterns*. The description of control flow patterns [8], for example, is based on Petri Nets. Therefore these patterns already have an inherent formal semantics. Regarding adaptation patterns, in turn, we have to find a semantical description which is independent of a particular process meta model. In our context, we base this description on the behavioral semantics of the respective process schema before and after its change. One way to capture behavioral semantics is to use execution traces. For this purpose, first of all, we provide some preliminary definitions.

Definition 1 (Execution Trace). *Let $\mathcal{A}$ be a set of activities which can be used to specify a process schema S. Let further $\mathcal{Q}$ be the set of all possible execution traces over S. A trace $\sigma \in \mathcal{Q}$ is defined as $\sigma = < a_1, \ldots, a_k >$ where $a_i \in \mathcal{A}$, $i = 1, \ldots, k$, and the order of a_i in σ reflects the order in which activities a_i are completed over S. Then:*

- *traceSucc(a, σ)[1] denotes the function which returns all activities which have been completed after activity a according to trace σ. Formally:*
  traceSucc: $\mathcal{A} \times \mathcal{Q} \mapsto 2^{\mathcal{A}}$ with
  traceSucc(a, $< a_1, \ldots, a_k, a, a_{k+1}, \ldots, a_n >$) = $< a_{k+1}, \ldots, a_n >$
- *analogously tracePred(a, σ) denotes the function which returns all activities which have been completed before activity a according to trace σ.*

Based on this meta-model independent notion of execution traces we exemplarily describe the semantics of selected pattern variants in Figure 4.

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 Process Fragments* (PP1), *Late Modeling of Process Fragments* (PP2), *Late Composition of Process Fragments* (PP3) and *Multi-Instance Activity* (PP4).

Predefined Change Pattern PP1: Late Selection of Process Fragments. The *Late Selection of Process Fragments* pattern (cf. Fig. 15) allows to select the implementation for a particular process step at run-time either based on

[1] Note that in this paper, for the definition of functions traceSucc and tracePred we assume that process schema S is acyclic.

Let T_S denote the set of all execution traces producible on a process schema S. Let further op be an adaptation pattern transforming process schema S into S'. Finally, let x be the associated inserted / deleted / replaced atomic activity.	
AP1: INSERT Activity Design Choice: B[1], D[2a]	*Preliminaries:* op = INSERT(S, x, A, B) where A and B denote activity sets between which x is inserted *Semantics:* $\forall \sigma \in T_{S'}$: x $\in \sigma \Longrightarrow$ A $\subseteq$ tracePred(x, σ) $\wedge$ B $\subseteq$ traceSucc(x, σ)
AP2: DELETE Activity Design Choice: B[1]	*Preliminaries:* op = DELETE(S, x) *Semantics:* $\forall\ \sigma \in T_{S'}$: x $\notin \sigma$
AP3: MOVE Activity Design Choice: B[1], D[2a]	*Preliminaries:* op = MOVE(S, x, A, B) where A and B denote the activity sets between which x is moved starting from its current position *Semantics:* $\forall \sigma \in T_{S'}$: x $\in \sigma \Longrightarrow$ A $\subseteq$ tracePred(x, σ) $\wedge$ B $\subseteq$ traceSucc(x, σ)
AP4: REPLACE Activity Design Choice: B[1]	*Preliminaries:* op = REPLACE(S, x, y) *Semantics:* (replaces activity x by activity y in schema S) $\forall\ \sigma = < a_1, \ldots, a_k, x, \ldots > \in T_S$ $\Longrightarrow$ $\exists\ \mu = < a_1, \ldots, a_k, y, \ldots > \in T_{S'}$
AP5: SWAP Activity Design Choice: B[1]	*Preliminaries:* op = SWAP (S, x, y) *Semantics:* (swaps activity x and y in schema S) $(\forall\ \sigma = < a_1, \ldots, a_k, x, \ldots > \in T_S$ $\Longrightarrow$ $\exists\ \sigma' = < a_1, \ldots, a_k, y, \ldots > \in T_{S'}) \wedge$ $(\forall\ \mu' = < a_1, \ldots, a_l, x, \ldots > \in T_{S'}$ $\Longrightarrow$ $\exists\ \mu = < a_1, \ldots, a_l, y, \ldots > \in T_S)$

Fig. 4. *Semantics of Selected Adaptation Patterns*

predefined rules or user decisions.

Predefined Change Pattern PP2: Late Modeling of Process Fragments. The *Late Modeling of Process Fragments* pattern (cf. Fig. 16) offers more freedom and allows to model selected parts of the process schema at run-time.

Predefined Change Pattern PP3: Late Composition of Process Fragments. Furthermore the *Late Composition of Process Fragments* pattern (cf. Fig. 17) enables the on-the fly composition of process fragments (e.g., by dynamically introducing control dependencies between a set of fragments).

Predefined Change Pattern PP4: Multi Instance Activity. In case of *Multi-Instance Activities* the number of instances created for a particular activity is determined at run-time (cf. Fig. 18). 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 as 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

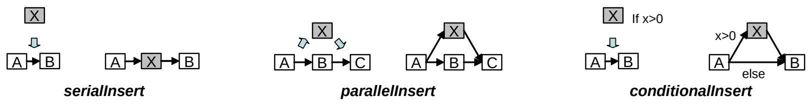
Pattern AP1: Insert Process Fragment	
Description	A process fragment is added to a process schema.
Example	For a particular patient an allergy test has to be added due to a drug incompatibility.
Problem	In a real world process a task has to be accomplished which has not been modeled in the process schema so far.
Design Choices **(in addition to those described 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)
 X A B A X B serialInsert X A B C A X B C parallelInsert X If x>0 A B A x>0 X else B conditionalInsert	
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).

Fig. 5. *Insert (AP1) Process Fragment* pattern

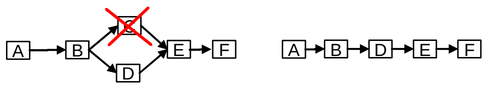
Pattern AP2: Delete Process Fragment	
Description	A process fragment is deleted from a process schema.
Example	For a particular patient no computer tomography is performed due to the fact that he has a cardiac pacemaker (i.e., the computer tomography activity is deleted).
Problem	In a real world process 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. 6. *Delete (AP2) Process Fragment* pattern

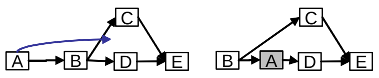
Pattern AP3: Move Process Fragment	
Description	A process fragment is moved from its current position in the process schema to another position.
Example	Usually employees are only allowed to book a flight, after getting approval from the manager. For a particular process instance the booking of a flight 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 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	This adaptation pattern can be implemented based on Pattern AP1 and AP2 (insert / delete process fragment).
Related Patterns	*Swap* adaptation pattern (AP5)

Fig. 7. *Move (AP3) Process Fragment* pattern

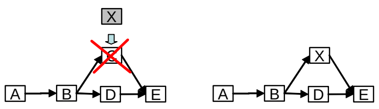
Pattern AP4: Replace Process Fragment	
Description	A process fragment is replaced by another process fragment.
Example	Instead of the computer tomography activity, the X-ray activity shall be performed for a particular patient.
Problem	A process fragment is no longer adequate, but can be replaced by another one. 
Implementation	This adaptation pattern can be implemented based on Pattern AP1 and AP2 (insert / delete process fragment).

Fig. 8. *Replace (AP4) Process Fragment* pattern

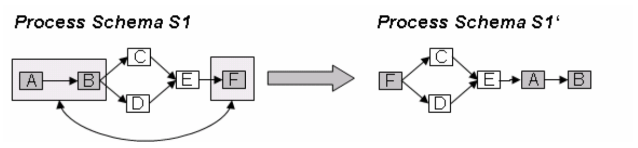
Pattern AP5: Swap Process Fragment	
Description	Two existing process fragments are swapped
Example	Regarding a particular delivery process the order in which requested goods shall be delivered to two customers has to be swapped.
Problem	The predefined ordering of two existing process fragments has to be changed by swapping their position in the process schema.
 Process Schema S1 A B C D E F Process Schema S1‘ F C D E A B	
Implementation	This adaptation pattern can be implemented based on AP3 (*move process fragment*) or on the combined use of patterns AP1 and AP2 (*insert / delete process fragment*).
Related Patterns	*Move Process Fragment* (AP3)

Fig. 9. *Swap Process Fragment (AP5)* pattern

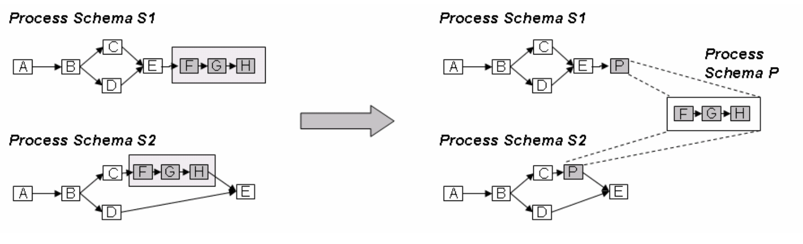
Pattern AP6: Extract Sub Process	
Description	Extract a process fragment from the given process schema and replace it by a corresponding sub process.
Example	A dynamically evolving engineering process has become too large. In order to reduce complexity the process owner extracts activities related to the engineering of a particular component and encapsulates them in a separate sub process.
Problem	*Large process schema.* If a process schema becomes too large, this pattern will allow for its hierarchical (re-)structuring. This simplifies maintenance, increases comprehensibility, and fosters the reuse of process fragments. *Duplication across process schemes.* A particular process fragment appears in multiple process schemes. If the respective fragment has to be changed, this change will have to be conducted repetitively for all these schemes. This, in turn, can lead to inconsistencies. By encapsulating the fragment in one sub process, maintenance costs can be reduced.
 Process Schema S1 A B C D E F G H Process Schema S2 Process Schema S1 P Process Schema P Process Schema S2	
Implementation	In order to implement AP6 graph aggregation techniques can be used. When considering data aspects as well, variables which constitute input / output for the selected process fragment have to be determined and considered as input / output for the created sub-process.
Related Patterns	*Inline Sub Process* (AP7)

Fig. 10. *Extract Sub Process (AP6)* pattern

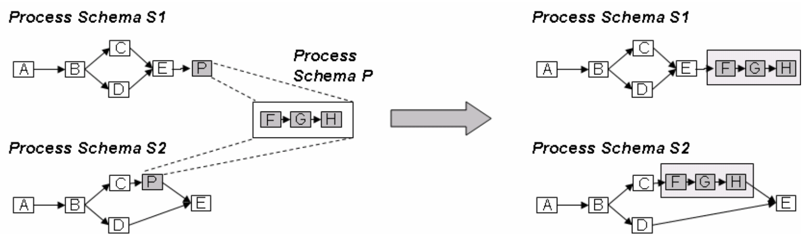
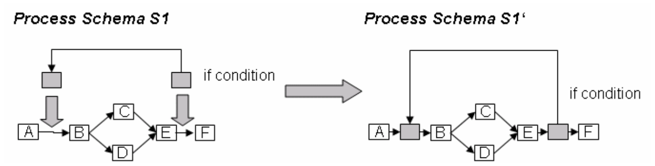
Pattern AP7: Inline Sub Process	
Description	A sub process to which one or more process schemes refer is dissolved. Accompanying to this the related sub process graph is directly embedded in the parent schemes.
Example	The top level of a hierarchically structured engineering process only gives a rough overview of the product development process. Therefore, the chief engineer decides to lift up the structure of selected sub processes to the top level.
Problem	*Too many hierarchies in a process schema*: If a process schema consists of too many hierarchy levels the inline sub process pattern can be used to flatten the hierarchy. *Badly structured sub processes*: If sub processes are badly structured the inline pattern can be used to embed them into one big process schema, before extracting better structured sub-processes (based on AP6).
 Process Schema S1 A B C D E P Process Schema P F G H Process Schema S2 A B C D P E Process Schema S1 A B C D E F G H Process Schema S2 A B C D F G H E	
Implementation	The implementation of this pattern can be based on other adaptation patterns (e.g., AP1). When also dealing with data aspects, the data context of the sub process and its current mapping to the parent process has to be transferred to the parent process schema.
Related Patterns	*Extract Sub Process* (AP6)

Fig. 11. *Inline Sub Process (AP7)* pattern

a) Pattern AP8: Embed Process Fragment in Loop	
Description	Adds a loop construct to a process schema surrounding an existing process fragment
Example	Regarding the treatment process of a particular patient a lab test shall be not only performed once (as in the standard treatment procedure), but shall be repeated daily due to special risks associated with this patient.
Problem	A process fragment is actually executed at most once, but needs to be executed recurrently based on some condition.
 Process Schema S1 if condition A B C D E F Process Schema S1' if condition A B C D E F	
Implementation	This adaptation pattern can be implemented based on Patterns AP1 (insert process fragment), AP11 and AP12 (add / remove control dependency)
Related Patterns	*Embed Process Fragment in Conditional Branch* (AP10)

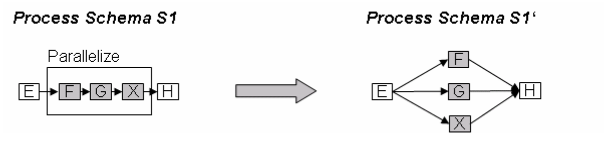
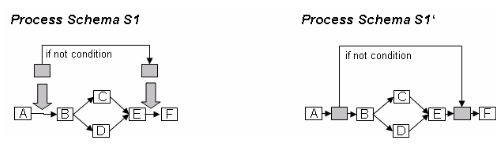
b) Pattern AP9: Parallelize Process Fragments	
Description	Process fragments which were confined to be executed in sequence are parallelized.
Example	For a running production process the number of resources is dynamically increased. Thus, certain activities which were ordered sequentially can now be processed in parallel.
Problem	Ordering constraints predefine for a set of process fragments turn out to be too strict and shall therefore be removed.
 Process Schema S1 Parallelize E F G X H Process Schema S1' E F G X H	
Implementation	This adaptation pattern can be implemented based on Patterns AP11 and AP12 (add / remove control dependency)

Fig. 12. *Embbed Process Fragment in Loop (AP8) and Parallelize Process Fragments (AP9)* patterns

a) Pattern AP10: Embed Process Fragment in Conditional Branch

Description	An existing process fragment shall be only executed if certain conditions are met.
Example	So far, in company *XY* the process for planning and declaring a business trip has required travel applications for both national and international trips. This shall be changed by asking for a respective travel applications only when going for an international trip.
Problem	A process fragment should only be executed if a particular condition is met.
	 Process Schema S1 if not condition A B C D E F Process Schema S1' if not condition A B C D E F
Implementation	This adaptation pattern could be implemented based on patterns AP1 (insert process fragment), AP11, and AP12 (add / remove control dependency)
Related Patterns	*Embed Process Fragment in Loop* (AP9)

b) Pattern AP11: Add Control Dependency

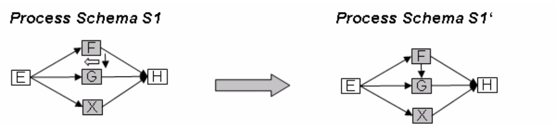
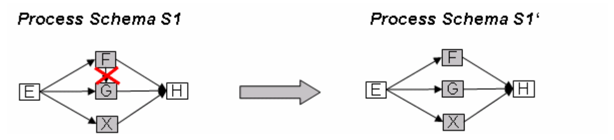
Description	An additional control edge (e.g., for synchronizing the execution order of two parallel activities) is added to the process schema
Example	For a running production process the number of resources is dynamically decreased. Thus, certain activities which were ordered in parallel now have to be processed in sequence.
Problem	An additional control dependency is needed in the process schema
	 Process Schema S1 E F G X H Process Schema S1' E F G X H
Related Patterns	*Remove Control Dependency* (AP12), *Parallelize Process Fragment* (AP9)

Fig. 13. *Embbed Process Fragment in Conditional Branch (AP10) and Add Control Dependency (AP11)* patterns

a) Remove Control Dependency	
Description	A control edge is removed from the process schema
Example	Assume that for a medical treatment procedure, usually, test A has to be finished before test B may be started. In an emergency situation, these two tests shall be performed in parallel in order to quickly treat the respective patient
Problem	An existing control dependency is not needed anymore in the process schema
	 Process Schema S1 E F G X H Process Schema S1' E F G X H
Related Patterns	*Add Control Dependency* (AP11), *Parallelize Process Fragments* (AP9)

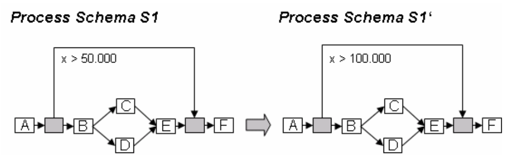
b) Pattern AP13: Update Condition	
Description	A transition condition in the process schema is updated.
Example	In a loan approval process the manager will have to approve a loan if the amount is larger than 50.000 Euro. Starting from January next year only loans above 100.000 Euros have to be approved by the manager.
Problem	A transition condition has to be modified as it is no longer valid.
	 Process Schema S1 x > 50.000 A B C D E F Process Schema S1' x > 100.000 A B C D E F
Related Patterns	*Embed Process Fragment in Loop* (AP8), *Embed Process Fragment in Conditional Branch* (AP10)

Fig. 14. *Remove Control Dependency (AP12) and Update Condition (AP13)* patterns

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. A detailed description of this pattern can be found in [8].

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?

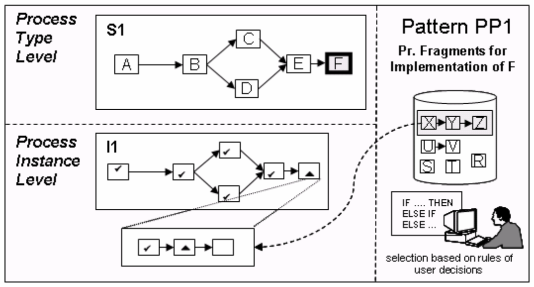
   1. Automatically based on predefined rules
   2. Manually by an authorized user
   3. Semi-automatically: options are reduced by applying some predefined rules; user can select among the remaining options

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 *Late Modeling of Process Fragments* (PP2)

Fig. 15. Late Selection of Process Fragments (PP1)

Pattern PP2: Late Modeling of 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.

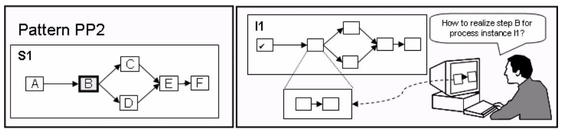
Design Choices

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. Then, the process fragment is dynamically invoked as a sub process. The assignment of the respective process fragment to the placeholder activity is done through late binding.

Related Patterns

Late Selection of Process Fragments (PP1)

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

Fig. 16. Late Modeling of Process Fragments(PP2)

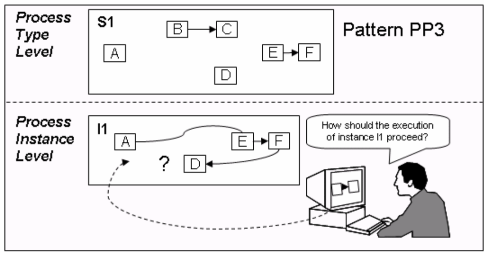
Pattern PP3: Late Composition of Process Fragments	
Description	At build time a set of process fragments is defined out of which a concrete process instance can be composed at run time. This can be achieved by dynamically selecting fragments and adding control dependencies on the fly (cf. Fig. 6).
Example	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.
	 Process Type Level S1 B C A E F D Pattern PP3 Process Instance Level I1 A E F ? D How should the execution of instance I1 proceed?

Fig. 17. Late Composition of Process Fragments (PP3)

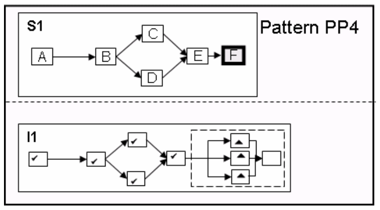
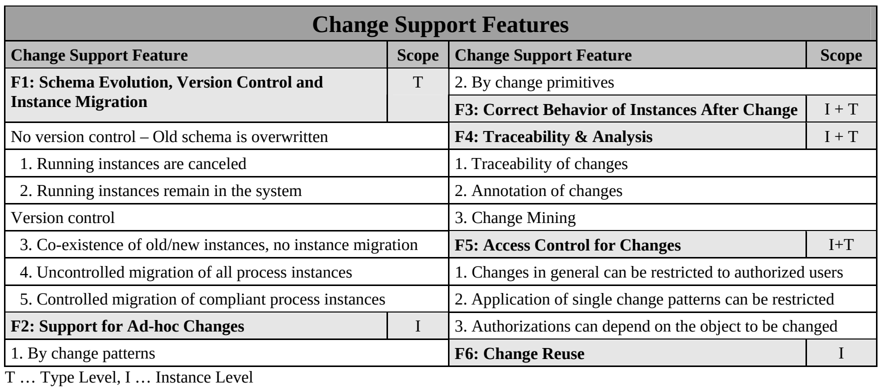
Pattern PP4: Multi Instance Activity	
Description	This pattern allows the creation of multi instances of a respective activity during run-time.
Example	The scanning activity has to be repeated until all parcels are scanned. The number of parcels is not known at build-time.
Problem	A respective activity has to be executed several times. The number of instances that have to be created are not at build-time.
	

Fig. 18. Multi-Instance Activity (PP4)

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. 19). Relevant change support features include *process schema evolution* and *version control*, change correctness, change traceability, access control and change reuse[2]. As illustrated in Fig. 19 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 [9].



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. 19. Change Support Features

4.1 Schema Evolution, Version Control and Instance Migration (Change Feature F1)

In order to support changes at the process type level, version control for process schemes should be supported (cf. Fig. 19). In case of long-running processes, in 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. 20).

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. 20a). In the latter case running process instances can either be withdrawn from the run-time environment or, as illustrated in Fig. 20a, they

[2] 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

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. 20a 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.

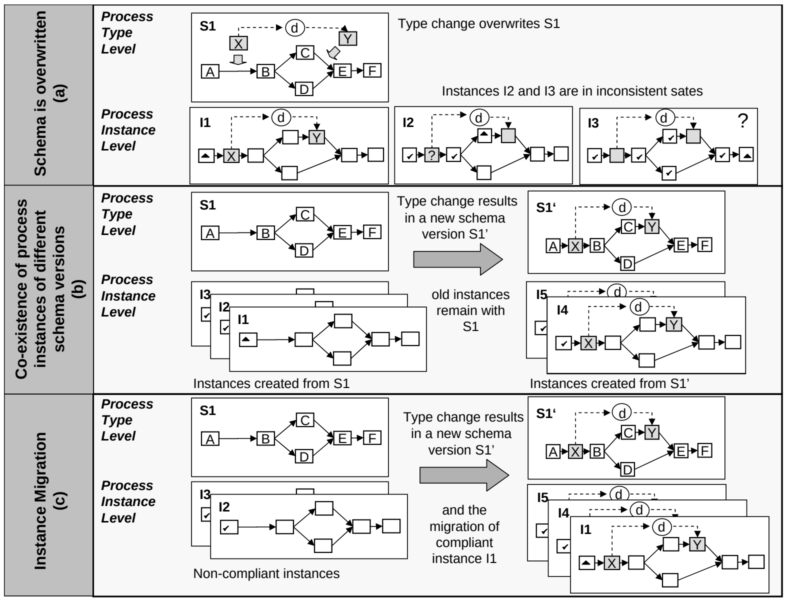
By contrast, if a PAIS provides explicit version control two support features can be differentiated: running process instances remain associated with the old schema version, while new instances will be created on the new schema version. This approach leads to the co-existence of process instances of different schema versions (cf. Fig. 20b). Alternatively a migration of a selected collection of process instances to the new process schema version is supported (in a controlled way) (cf. Fig. 20c). The first option is shown in Fig. 20b 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. 20c illustrates the controlled migration of process instances. Only those instances are migrated which are *compliant*[3] 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 (Change Feature F2). 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 (Change Feature F3). 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). In particular, different formal criteria (see [2]) have been introduced to ensure that process instances can only be updated to a new schema if they are compliant with it. Depending on the used process meta model, in addition, (formal) constraints of the respective formalism have to be taken into account

[3] 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]).



Schema is overwritten (a)
Process Type Level
Process Instance Level
S1
Type change overwrites S1
Instances I2 and I3 are in inconsistent sates
I1
I2
I3
?
Co-existence of process instances of different schema versions (b)
Type change results in a new schema version S1'
old instances remain with S1
S1'
Instances created from S1
Instances created from S1'
Instance Migration (c)
and the migration of compliant instance I1
Non-compliant instances

Fig. 20. Version Control

as well when conducting process changes.

Traceability and Analysis (Change Feature F4). 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 [10] 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 [9]. Finally, change mining allows for the analysis of changes, for instance, to support continuous process improvement [10].

Access Control for Changes (Change Feature F5) 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 (Change Feature F6). In the context of ad-hoc changes "similar" deviations (i.e., combination of one or more adaptation patterns) can occur more than once [11]. 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, 9]. 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 is mainly based on a comprehensive literature study. 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. This has also provided us with valuable insights into the implementation of change patterns and change features in respective approaches. In detail, the evaluated approaches are ADEPT [3, 13–15], CBR-Flow [12, 9], WIDE [16, 17], Pockets of Flexibility [18–20], Worklets/Exlets [4, 21], MOVE [22], HOON [23], 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] are considered in the present evaluation.

Evaluation results are aggregated in Fig. 21. If a change pattern or change support feature is not supported at all, the respective table entry will be labeled with "-" (e.g., no support for adaptation patterns AP4 and AP5 is provided by ADEPT). Otherwise, a table entry describes the exact pattern variants as supported by listing all available design choices. As an example take change pattern PP1 (Late Selection of Process Fragments) of the Worklet/Exlet approach [4, 21]. In Fig. 21 the string "A[1,2], B[1,2], C[2]" indicates that this change pattern (cf. Fig. 15) is supported for the Worklet/Exlet approach with design choices A, B and C. Further, it indicates for every design choice the exact options available. For example, for design choice A, Options 1 and 2 are supported. Taking the description from Fig. 15 this means that the selection of the activity implementation can be be done automatically (based on predefined rules) or manually by a user. Finally, in case no design choice exists for a supported change pattern, the respective table entry is simply labeled with "+" (e.g., support of change pattern PP4 by WIDE). Partial support is labeled with "◦" (e.g., the Worklet/Exlet approach supports change feature F3 partially).

Generally, 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 associated with process changes 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.

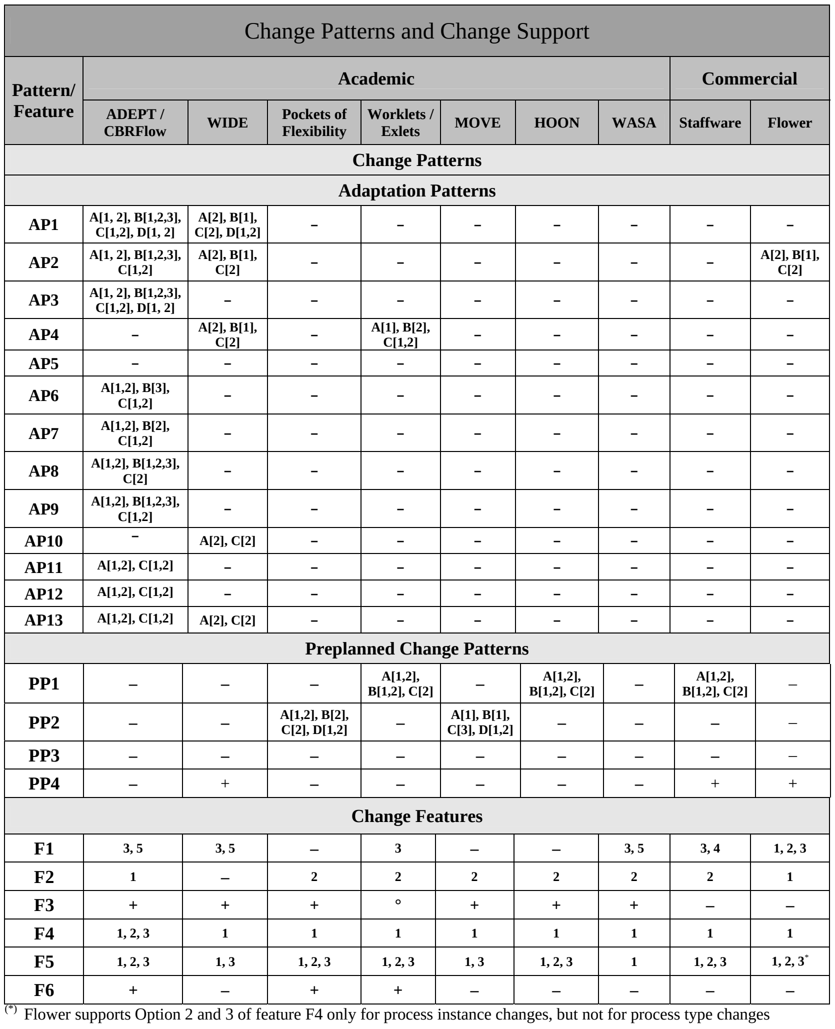
Certain adaptation patterns (e.g., AP3 or AP4) could be implemented by applying a combination of the more basic ones (e.g., AP1, AP2, AP10 and AP11). Again, a given approach will only qualify for a particular adaptation pattern, if it supports this pattern directly (i.e., it offers one change operation for realizing the respective adaptation pattern). For instance, providing support for adaptation patterns AP1 (Insert Process Fragment) and AP2 (Delete Process Fragment) allows to implement pattern AP3 (Move Process Fragment) in a straightforward way. However, moving activities by using adaptation patterns AP1 and AP2 in combination with each other is more complicated when compared to the direct application of AP3 (and does also lead to less meaningful change logs).

Note that missing support for adaptation patterns does not necessarily mean that no run-time changes can be performed. As long as feature F2 (Support for Ad-hoc Changes) is provided, ad-hoc changes to running process instances are possible (for details see [24]). 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) changes to predefined regions in the process can be performed during run-time. In addition, the need for structural changes of the process schema can be decreased making feature F3 less crucial.

All evaluation results are summarized in Fig. 21. A detailed description of the evaluated approaches is provided in the following sections.

5.1 Evaluation Details: ADEPT / CBRFlow

Support for Adaptation Patterns. Generally, ADEPT enables the application of adaptation patterns at both the process type and the process instance level (Design Choice A[1,2]). Supported adaptation patterns may operate on atomic activities, sub processes, and process sub graphs (Design Choice



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]	–	–	–	–	–
AP5	–	–	–	–	–	–	–	–	–
AP6	A[1,2], B[3], C[1,2]	–	–	–	–	–	–	–	–
AP7	A[1,2], B[2], C[1,2]	–	–	–	–	–	–	–	–
AP8	A[1,2], B[1,2,3], C[2]	–	–	–	–	–	–	–	–
AP9	A[1,2], B[1,2,3], C[1,2]	–	–	–	–	–	–	–	–
AP10	–	A[2], C[2]	–	–	–	–	–	–	–
AP11	A[1,2], C[1,2]	–	–	–	–	–	–	–	–
AP12	A[1,2], C[1,2]	–	–	–	–	–	–	–	–
AP13	A[1,2], C[1,2]	A[2], C[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. 21. Change Patterns and Change Support Features in Practice

B[1,2,3]).[4] Most adaptation patterns can be applied in both a temporary or a permanent manner (Design Choice C[1,2]).

In detail: ADEPT supports the insertion of process fragments (AP1), which can be added serially, conditionally, or in parallel (Design Choice D[1,2(a+b)]) to a process schema or process instance schema respectively. Furthermore, it is possible to delete process fragments (AP2) or to move them to another position (AP3). Adaptation patterns AP4 (Replace Process Fragment) and AP5 (Swap

[4] There exist some restrictions in this context. For example, adaptation pattern AP6 (Extract Process Fragment) is only applicable to an existing process fragment. The use of adaptation pattern AP7 (Inline Sub Process), in turn, makes only sense in connection with sub processes.

Process Fragment) are not directly supported by the current ADEPT system, but can be realized based on adaptation patterns AP1, AP2, and AP3. With AP6 (Extract Sub Process), AP7 (Inline Sub Process), and AP8 (Embed Process Fragment in Loop) more complex patterns are supported as well. Adaptation pattern AP9 (Parallelize Process Fragment) is implemented in ADEPT as a variant of pattern AP3 (Move Process Fragment). Adaptation pattern AP10 (Embed Process Fragment in Conditional Branch) is not directly supported, but can be realized with patterns AP1 and AP3. Finally, adaptation patterns AP11 (Add Control Dependency), AP12 (Remove Control Dependency), and AP13 (Update Condition) are supported. Details about respective ADEPT change operations (e.g., their formal semantics, their implementation, etc.) can be found in [15, 14].

It is important to mention that the block-structured modeling approach used in ADEPT (for details see [3]) significantly facilitates the implementation and use of adaptation patterns.

Support for Predefined Change Patterns. No support for predefined change patterns is provided.

Schema Evolution, Version Control and Instance Migration (Change Feature F1). ADEPT enables advanced version control. If a process schema is changed at the type level, a new process schema version is created. Further it is checked which process instances can be correctly migrated to the new schema version, and which instances remain running on the old schema version (F1[3,5]). In this context, ADEPT uses a well-defined correctness criterion for deciding on the compliance of process instances with a modified schema version. This criterion is independent of the ADEPT process meta model and is based on a relaxed notion of trace equivalence (see [25]). It considers all kinds of control flow changes and works correctly in connection with loop backs as well. In order to enable efficient compliance checks, for each high level change operation (or adaptation pattern respectively) ADEPT provides precise and easy to check compliance conditions (for details see [25]). Finally, efficient procedures exist for correctly adapting the states of compliant process instances when migrating them to the new schema version.

Ad-hoc Changes (Change Feature F2). From the very beginning, ADEPT has supported ad-hoc changes at the process instance level (see [3]). These ad-hoc changes are based on the aforementioned adaptation patterns (F2[1]). In particular, the introduction of ad-hoc changes does not lead to an unstable system behavior, i.e., none of the guarantees (e.g., absence of deadlocks) achieved by formal checks at build-time are violated due to the ad-hoc change at run-time. In ADEPT this is achieved based on well-defined pre- and post-conditions for the high-level change operations. Finally, when introducing an ad-hoc change, all complexity associated with the adaptation of instance states, the remapping of activity parameters, or the problem of missing data (e.g., due to activity deletions) is hidden from users.

Correctness of Changes (Change Feature F3). One of the major design goals of the ADEPT approach was to ensure correctness and consistency when migrating process instances to a new process schema version or when applying an ad-hoc change to a particular process instance (for details see [3, 25]). This goal has been achieved based on the aforementioned compliance rules as well as on operation-specific pre- and post-conditions (F3[+]).

Traceability and Analysis (Change Feature F4). The ADEPT process management system enables change traceability by maintaining comprehensive change logs. These change logs comprise both syntactical and semantical information about the performed process changes (F4[1,2]). While the former capture data about the applied adaptation patterns and their parameterizations, the latter cover contextual knowledge about the changes (e.g., change reason and change performer). ADEPT provides powerful support for maintaining, purging, and utilizing such logs, and for annotating log entries (for details see [26]). In the ProCycle project, the ADEPT system has been integrated with the conversational case-based reasoning component of CBRFlow [12]. Among other things, this integration allows to enrich change logs with contextual information [27, 9, 28]. Finally, in the MinADEPT project first techniques and tools for analyzing and mining ADEPT change logs (F4[3]) have been provided [10].

Access Control (Change Feature F5). In respect to access control ADEPT allows to restrict changes to authorized users (F5[1]). In addition, authorizations can be defined at the level of single change patterns. For instance, a particular user might be authorized to insert, but not delete activities (F5[2]). Authorization can also depend on the object to be changed. For instance, a particular user might be authorized to insert only selected activities (F5[3]). A detailed description of the access control model provided by ADEPT can be found in [**?**].

Change Reuse (Change Feature F6). The integration of ADEPT with case-based reasoning techniques (see above) also enables change reuse. Whenever an ad-hoc modification becomes necessary the user is assisted in searching for similar, previously performed changes, which he then can reuse [9, 28, 11]. If for a particular situation no change reuse is possible, the user can specify a new ad-hoc modification using the adaptation patterns provided by ADEPT.

5.2 Evaluation Details: WIDE

Support for Adaptation Patterns. Generally, WIDE enables the application of adaptation patterns only at the process type level (Design Choice A[2]) and a change pattern operates on atomic activities (Design Choice B[1]). All change patterns are applied in a permanent manner (Design Choice C[2]). In detail, WIDE supports the insertion of process fragments (AP1) which can be added in a serial and conditional manner (Design choice D[1,2(b)]). Furthermore, it is possible to delete existing process fragments (AP2) or to replace them (AP4). The embedding of a process fragment in a conditional branch (AP10) and the updating of conditions (AP13) are supported as well. Regarding the adaptation patterns supported by WIDE it is important to mention that one design goal was *minimality*; i.e., a change operation will be only provided if it cannot be realized by the combined use of a set of other change operations [17]. Obviously, in WIDE non-supported adaptation patterns (e.g., AP3 and AP5) could be easily implemented by the combined use of existing patterns. For a more detailed description of the supported adaptation patterns see [17, 16].

Support for Predefined Change Patterns. In terms of predefined change patterns WIDE provides support for multi-instance activities (PP4) (see [16]). Thereby the number of activity instances can be fixed during build-time or depend on workflow relevant data, which becomes available at run-time.

Schema Evolution, Version Control and Instance Migration (Change Feature F1). In WIDE version control of process schemes is supported: If a process schema is changed, a new process schema version is created (Design Choice F1[3]) and it is checked which process instances can migrate to the new version according to the so called *compliance criterion* (Design Choice F1[5]).

Ad-hoc Changes (Change Feature F2). WIDE does not address ad-hoc changes of process instances.

Correctness of Changes (Change Feature F3). Correctness and consistency of (compliant) process instances is guaranteed when migrating them to a new process schema version (process schema evolution). For this, the aforementioned compliance criterion is used (F3[+]) (for details see [17]).

Traceability and Analysis (Change Feature F4). Process type changes can be tracked back through the existence of process schema versions (F4[1]). Change annotations and change mining are not addressed in WIDE.

Access Control (Change Feature F5. WIDE provides a role-based access control model, which allows restricting access to authorized users. Authorizations can depend on the process schema to be changed (F5[1,3]).

Change Reuse (Change Feature F6). WIDE does not support change reuse (F6[-]).

5.3 Evaluation Details: Pockets of Flexibility (PoF)

Basic to this approach is the *Pocket of Flexibility (PoF)*. Essentially, a PoF constitutes a placeholder activity (within a process schema) which can be substituted by a dynamically modeled process fragment during run-time [19, 20].

Support for Adaptation Patterns. The provision of adaptation patterns is not in the focus of the PoF approach. None of the described adaptation patterns is supported through high-level change operations and changes can only be applied within pre-planned regions. The latter are covered by change pattern PP2.

Support for Predefined Change Patterns. The PoF approach enables flexibility through change pattern PP2 (Late Modeling of Process Fragments) [19, 20]. Optionally, constraints can be defined restricting the activities that can be used for late modeling to a subset of the activity repository (Design Choice A[1,2]). In addition, the order of these activities can be restricted by pre-defined constraints. Finally, for the late modeling of a PoF during run-time only a restricted set of modeling elements is available, i.e., only sequential and parallel routing of added activities is supported (Design Choice B[2]). Note that this facilitates late modeling of process fragments for end users and thus increases user acceptance.

Late modeling starts when the placeholder activity (so called PoF) is instantiated (Design Choice C[2]). Following this the user can define a corresponding process fragment using a restricted set of modeling elements. Upon completion of the late modeling the newly defined process fragment is instantiated. In this context the modeling of the placeholder activity either can be done from scratch (Design Choice D[1]) or in case that the placeholder activity contains a predefined template by adjusting this template (Design Choice D[2]).

Schema Evolution, Version Control and Instance Migration (Change Feature F1). Change feature F1 is not supported by the PoF approach. Instead, the existence of placeholder activities allows to individually model parts of the process model or even the whole process model during run-time. Though this individualization reduces the need for structural changes of the process model, the need for changing the "core process" (i.e., the toplevel process) cannot be completely discarded. This problem is not addressed in the PoF approach.

Ad-hoc Changes (Change Feature F2). As mentioned above the PoF approach does not provide direct support for any adaptation pattern as no high-level change operations are provided and changes can only be performed within pre-planned regions. Furthermore, modeling and execution is not interwoven in the PoF approach, i.e., the modeling of the process fragment associated with a placeholder activity has to be completed before before the execution of the respective process fragment may start.

In particular, this approach does not allow for (ad-hoc) changes of a process fragment once it has been instantiated, unless this fragment itself contains placeholder activities.[5] In the latter case the functionality provided by adaptation pattern AP1 (Insert Process Fragment) can be partially "simulated". Generally,

[5] Generally, it is not always possible for end users to anticipate all future changes during modeling time.

a placeholder activity can be positioned between two fragments or parallel to an existing one. By substituting this placeholder activity with a concrete (sub) process fragment during runtime, in principle, a serial or parallel insertion becomes possible (cmp. Design Choice D[1,2] of pattern AP1).

Any fragment substituting a placeholder activity has to be modeled with a conventional process editor, which only provides basic change primitives (e.g., insert/delete node, insert/delete connector) to the user. Another limitation of this workaround results from the fact that insertion is restricted to pre-modeled placeholder activities. Finally, adaptation patterns other than AP1 cannot be simulated with the PoF approach.

Correctness of Changes (Change Feature F3). The PoF approach ensures change correctness as the process fragment resulting from late modeling is validated before it gets instantiated.

Traceability and Analysis (Change Feature F4). Each process fragment resulting from late modeling is stored in the process repository as process variant. An advanced querying interface for retrieving process variants from this repository is offered. Thus, change traceability can be easily ensured (F4[1]). Change annotations and change mining are outside the focus of the PoF approach.

Access Control (Change Feature F5). Regarding access control PoF allows restricting changes to particular users by associating the placeholder activity with a particular role (F1[1]). In addition, the PoF approach allows for the definition of constraints for a placeholder activity. Consequently, the kind of changes that can be applied by an authorized user can be partially restricted (F1[2]). For each placeholder activity different authorizations can apply (F1[3]).

Change Reuse (Change Feature F6). Change reuse is supported by providing a querying component for process fragments. So far, this component is solely based on control flow aspects [18].

5.4 Evaluation Details: Worklets/Exlets

Basic to the Worklet/Exlet approach is the notion of worklets. Essentially, a worklet is a small, discrete process fragment that acts as a late-bound sub process for an enabled activity [4]. In turn, exlets are exception handling processes for parent process instances, which are invoked if specific events occur [21].

Support for Adaptation Patterns. The Worklet/Exlet approach provides direct support for adaptation pattern AP4 (Replace Process Fragment). This is based on the ability to substitute worklet-enabled activities with a process fragment (see also PP1) [4]. The respective pattern can be applied at the process instance level (Design Choice A[1]) and operates on atomic worklet-enabled activities (Design Choice B[1]). The pattern is applied in a permanent manner (Design Dhoice C[2]). Other adaptation patterns are not supported directly, as no high-level change operations are provided. However, certain ad-hoc changes can be realized by using change primitives (see Change Feature F2).

Support for predefined change patterns. Primarily, the Worklet/Exlet approach enables process flexibility by supporting late selection of process fragments, i.e., change pattern PP1 is provided [4]. In particular, process activities can be associated with a worklet selection service. Such a "worklet-enabled" activity does not constitute a placeholder (in contrast to approaches like HOON, PoF or MOVE), but a valid activity with standard implementation, which can be (optionally) substituted by a whole process fragment (i.e., worklet) during run-time (if appropriate). If this does not happen, the worklet-enabled activity is executed as "ordinary" task. In general, respective worklets are selected automatically following a rule-based approach (Design Choice A[1]). However, if not appropriate the proposed selection can be rejected by users and another worklet can be chosen instead by (dynamically) adding a new selection rule (Design Choice A[2]). A worklet itself refers to a sub process fragment, consisting of one or more activities; it is treated as a separate process instance during run-time (Design Choice B[1,2]). Worklet selection takes place when the worklet-enabled activity becomes activated (Design Choice C[2]).

Schema Evolution, Version Control and Instance Migration (Change Feature F1). The Worklet/Exlet approach itself does not address the problem of process type changes. However, it has been implemented as part of the YAWL engine, which support version control for process. Thereby, current instances remain running on the old schema version until their completion, whereas the execution of new instances is based the new model version (F1[3]). Note that the used late selection approach reduces the need for structurally changing the toplevel process (i.e., the "parent" schema).

Ad-hoc Changes (Change Feature F2). The Worklet approach provides direct support for adaptation pattern AP4 (see above) by supporting the substitution of worklet-enabled activities through worklets. For worklet-enabled activities, new worklets can be added or existing ones can be changed as long as no worklet (i.e., sub-process) is selected and instantiated for them. In principle, even changes to (parts of) a running process instance will be possible if the respective instance contains worklet-enabled activities (see below). As one limitation of the Worklet/Exlet approach, process fragments at the lowest hierarchical level cannot be changed once they have been instantiated, since they only contain ordinary activities, but no worklet-enabled ones. Finally, the Exlet extension provides powerful exception handling mechanisms which allow to cope with both expected and unexpected exceptions [21].

Pattern AP4 constitutes the only adaptation pattern directly supported in the Worklet/Exlet approach. However, other adaptation patterns can be realized based on workarounds and change primitives:

– *Adaptation Pattern AP1.* In principle, the Worklet/Exlet approach allows to realize parts of the functionality covered by adaptation pattern AP1 through

its late selection concept. Except for AP4 this is the only adaptation pattern whose functionality can be realized using Worklets without Exlets. The approach allows for substituting a worklet-enabled task with a process fragment, which basically corresponds to a serial insert (Design Choice D[1]). As the insertion is restricted to worklet-enabled activities, the position of the process fragment to be inserted must be pre-planned. In addition, the approach allows to conditionally insert fragments (Design Choice D[3]) as well as to insert them in parallel to the worklet-enabled activity (Design Choice D[2]). Inserting a process fragment in parallel to an existing branch (consisting of several (worklet-enabled) activities) can only be achieved if there is an optional placeholder activity defined in parallel to the respective branch.
Using exlets [21] in addition to worklets allows for additional workarounds with respect to AP1. For example, exception handling processes can be invoked at the occurrence of particular events. The approach allows to dynamically add exlets as well as respective selection events. Therefore it is appropriate for handling expected as well as unexpected exceptions. Exlets can be associated with a particular process or activity instance (i.e., the exlet is invoked if pre-constraints or post-constraints of the process or activity instance are met). However, they can be also triggered at any point in the process through user intervention (e.g., realized as external trigger). This allows to insert additional process fragments at any position in the process during run-time. In order to realize a serial insert of activity X before activity B and after activity A, for example, an exlet has to be invoked after A has been completed, and B has to be suspended. After the completion of the exlet containing activity X, activity B can be executed (Design Choice D[1]). The exlet extension allows for more complex applications of AP1 as well. However, through the lack of high-level operations changes can become pretty complex.

– *Adaptation Pattern AP2.* Like AP1, adaptation pattern AP2 can be realized through workarounds. An enabled activity Y can be skipped through an Exlet using the *Force Fail WorkItem* primitive. Following this the execution of Y is stopped and its status is set to failed.
– *Adaptation Pattern AP3 and AP5.* Patterns AP3 and AP5 can be implemented through a combination of AP1 and AP2.
– *Adaptation Pattern AP4.* The worklet selection service directly allows for substituting worklet-enabled tasks with a process fragment. Therefore we consider AP4 as being directly supported in the Worklet/Exlet approach.
– *Adaptation Pattern AP6 and AP7.* No direct support.
– *Adaptation Pattern AP8.* Using the Exlets extension a process fragment (i.e. a worklet) can be repeatedly run while a constraint is satisfied.
– *Adaptation Pattern AP9.* A sequence of worklet-enabled activities can be parallelized using an exlet as workaround. The respective exlets sets the execution state of the respective activities to "failed". Using a compensation primitive in the exlet then allows executing any number of worklets in parallel.

- *Adaptation Pattern AP10.* AP10 can be realized through an exlet. If a particular condition does not hold, this exlet fails the respective activities.
- *Adaptation Pattern AP11 - AP13.* These patterns can be realized through exlet rules creating the necessary constraints. In respect to AP13, selection criteria for worklets can be dynamically added, edited or removed during run-time.

These workarounds indicate that ad-hoc changes are possible when using the Worklet/Exlet approach. However, it should be clear that these low-level modifications will become pretty complex and error prone if no direct support for adaptation patterns exists. Like in software engineering, the presence of change patterns will assist users in dealing with recurrent problem situations at a semantically high level.

Correctness of Changes (Change Feature F3). All process fragments in the Worklet repository are YAWL process models. If only Worklets are used correctness of changes is ensured through the inbuilt verification and validation feature of the YAWL Process Editor. In case that exlets are used in addition, it cannot be guaranteed that the respective process instance can terminate properly. All compensatory worklets launched from an exlet are executed as distinct process instances - thus deadlocks are not an issue. However, it is possible for an exlet to contain a primitive to suspend a process instance and to leave it in that state (i.e. there is no subsequent un-suspend primitive in the exlet). This limitation will be corrected in the next version.

Traceability and Analysis (Change Feature F4). Traceability is ensured through maintaining a process log (F4[1]). Change annotation is supported through annotating newly added rules with a description of the process instance which triggered the rule creation (F4[2]). Change mining is currently not supported.

Access Control (Change Feature F5). The Worklet/Exlet approach allows restricting changes to particular users. The addition of rules can only be accomplished by the administrator (F5[1]). Furthermore, the process fragments that can be chosen with the worklet selection service are restricted to the worklet repertoire of the respective worklet-enable activity (F5[2]); i.e., for each worklet-enabled activity a repertoire (i.e., a collection) of worklets is maintained (i.e., a worklet repertoire) (F5[3]).

Change Reuse (Change Feature F6). The Worklet/Exlet approach supports the reuse of changes by supporting the incremental evolution of selection rules. In addition, worklets themselves may be reused in different worklet repertoires.

5.5 Evaluation Details: MOVE

Support for Adaptation Patterns. MOVE does not provide direct support for adaptation patterns as changes are restricted to placeholder activities and consequently have to be pre-planned. All changes that can be performed within the pre-planned region are covered by pattern PP2.

Support for predefined change patterns (PP2). Similar to the PoF approach MOVE only allows for changes in restricted and pre-defined process areas through late modeling. However, the MOVE approach is less powerful in terms of supported design choices. In general, all activities from the process repository can be chosen for late modeling (Design Choice A[1]). In contrast to PoF no additional constraints can be defined. For modelling the placeholder activity the same constructs are used than for modelling the rest of the process schema (Design Choice B[1]). More precisely, modelling is based on a high-level Petri Net formalism (FunSoft Nets) [29, 30]. The late modelling is triggered when a particular state in the process is reached (Design Choice C[3]). It can then be accomplished with the standard process editor either by starting from scratch (Design Choice D[1]) or by loading a pre-modeled process template and adapting it (Design Choice D[2]).

Schema Evolution, Version Control and Instance Migration (Change Feature F1). Like in the PoF approach the need for structural changes of the process model can be decreased through the late modeling capabilities of MOVE. However, structural changes of the toplevel process cannot be completely excluded. The challenges of process type changes and process schema evolution are not addressed in MOVE.

Ad-hoc Changes (Change Feature F2). Like the PoF approach MOVE does not support any of the adaptation patterns, but still allows for a certain degree of run-time flexibility through its late modeling capabilities. Regarding support for ad-hoc changes similar considerations can be made than for the PoF approach (see Section 5.3, Change Feature F2).

Correctness of Changes (Change Feature F3). All late-modeled process fragments are FunSoft process models. Due to the use of this high-level Petri Net formalism, model correctness can be easily ensured using the FunSoft Process Editor and its model checking features.

Traceability and Analysis (Change Feature F4). Late-modeled process fragments are stored in the process repository. As no ad-hoc changes are supported this is sufficient for traceability (F4[1]). Change annotations and change mining are not supported.

Access Control (Change Feature F5). Regarding access control MOVE allows restricting changes to particular users through assigning the placeholder activity to a particular role (F5[1]). Despite of this, no further restrictions can be specified in MOVE. For different placeholder activities different authorizations can apply (F5[3]).

Change Reuse (Change Feature F6). No explicit change reuse is supported in MOVE.

5.6 Evaluation Details: HOON

Support for Adaptation Patterns. HOON [23] does not provide direct support for any of the described adaptation patterns as changes are restricted to placeholder activities and no high-level change operations are considered.

Support for Predefined Change Patterns. HOON enables process flexibility by supporting late selection of process fragments, i.e., the ability to select a respective activity implementation at run-time. Thus change pattern PP1 is supported. In particular, this allows for the modification of not yet instantiated sub-processes. Regarding supported design choices, HOON is comparable with the Worklets/Exlet approach. The respective activity implementation can be selected in a fully automated way based on well-defined procedures and workflow runtime data (Design Choice A[1]), but can be also done manually by users (Design Choice A[2]). The activity implementation refers to a sub process consisting of one or more activities (Design Choice B[1+2]). The decision which activity implementation shall be selected is made when the placeholder activity is enabled (Design Choice C[2]).

Schema Evolution, Version Control and Instance Migration (Change Feature F1). HOON does not support structural changes of the "toplevel" process schema. However, in principle, the late selection support of HOON allows to dynamically add new activity implementations or to change existing ones before associating them with a particular workflow activity during run-time. Thus, the need for structural changes is lower when compared to process management systems like ADEPT, WASA, or Staffware.

Ad-hoc Changes (Change Feature F2). Although HOON does not support any of the adaptation patterns, in principle, modifications of running process instances become possible through the late selection of process fragments (Option 2).

The late selection mechanism allows to dynamically select an activity implementation at the process instance level. Once this has happened and the respective sub process has been instantiated, changes will be only possible if the respective sub process itself contains placeholder activities. In any case, the late selection support reduces the need for structural changes of a process model as process fragments can be dynamically added and selected during run-time

In terms of structural changes, HOON allows realizing part of the functionality described in AP1 through workarounds. It allows selecting an activity implementation for a placeholder activity, which basically corresponds to a serial insert (Design Choice D[1]). However, the insertion is restricted to the placeholder activity. If no placeholder activity is available right after the activity where the process fragment should be added, no insertion can be performed. If a process fragment shall only be conditionally inserted, the placeholder activity will have to be embedded in a conditional branch (Design Choice D[3]). Inserting a process fragment in parallel to an existing branch can only be achieved, if there is an optional placeholder activity defined in parallel to the respective branch (Design Choice D[2]).

Correctness of Changes (Change Feature F3). Correctness of changes or, more precisely, correctness of newly defined or adapted process schemes is ensured in HOON (through formal analyses of the respective HOON nets). In case that an activity implementation is not available, alternative activity implementations will be automatically assigned, or the user will be involved to manually assign an activity implementation.

Traceability and Analysis (Change Feature F4). Traceability of changes is supported in HOON through execution logs (F4[1]). Change annotations and change mining not supported.

Access Control (Change Feature F5). Besides the automated selection of activity implementations, HOON allows to restrict changes to particular users or user roles (F5[1]). Further, the set of process fragments that may be selected for a placeholder activity can be restricted based on the used HOON Net formalism (F5[2]). Generally, for each placeholder activity different kind of authorizations can be realized (F5[3]).

Change Reuse (Change Feature F6). No change reuse is supported in HOON.

5.7 Evaluation Details: WASA

Support for Adaptation Patterns. The design of high-level change operations was out of the scope of the WASA project. Thus, no direct support for any of the adaptation patterns is provided (see F2). Nevertheless, WASA enables structural process changes at both the process type and the process instance level using change primitives (Design Choice A[1,2]). All change primitives can operate on atomic activities and sub processes due to the object-oriented approach followed by WASA (Design Choice B[1,2]). Finally, changes are applied in a permanent manner (Design Choice C[2]).

Support for predefined change patterns (PP1). No support for predefined change patterns is provided.

Schema Evolution, Version Control and Instance Migration (Change Feature F1). In WASA advanced support for version control of process schemes is provided: If a process schema is changed, a new process schema version is created (design choice F1[3]) and it is checked which process instances can migrate to the new version according to a well-defined correntness criterion (called valid mapping correctness criterion in WASA) (Design Choice F1[3, 5]).

Ad-hoc Changes (Change Feature F2). In WASA ad-hoc changes can be performed using a workflow editor and change primitives (e.g., insert/delete node, insert/delete connector) (F2[2]).

Correctness of Changes (Change Feature F3). The correctness of dynamically changed process instances is ensured by the valid mapping correctness criterion (F3[+]). Based on this well-defined correctness criterion both control and data flow correctness can be guaranteed.

Traceability and Analysis (Change Feature F4). Traceability in ensured as for every process instance the process schema version which controlled instance execution is known. In addition, execution logs are provided (F4[1]).

Access Control (Change Feature F5). Access control for applying change patterns is realized by role-based access control in WASA. For example, only users with role 'process administrator' are allowed to perform change patterns (F5[1]).

Change Reuse (Change Feature F6). Change reuse is not supported in WASA (F6[-]).

5.8 Evaluation Details: Staffware

In terms of commercial systems Staffware and Flower have been evaluated. The evaluation of Staffware is based on version 10 of Staffware.

Support for Adaptation Patterns. Staffware does not support any of the adaptation patterns with high-level change operations.

Support for Predefined Change Patterns Regarding predefined change patterns Staffware provides support for PP1 (*Late Selection of Process Fragments*) and PP4 (*Multi-Instance Activity*). The Late Selection of Process Fragments is supported through the *Graft Activity* which is a feature of the Staffware Process Orchestrator. The selection of the activity implementation can either be automated or manually done by the user (Design Choice A[1,2]). The activity implementation refers to a sub process consisting of one or more activities (Design Choice B[1+2]). The decision which activity implementation shall be selected is made when the placeholder activity (i.e., the Graft Activity) is enabled (Design Choice C[2]). Pattern PP4 (*Multi-Instance Activity*), in turn, is supported through the *Dynamic Sub-Procedure Step*, which is also provided by the Staffware Process Orchestrator. The number of activity instances can either be fixed during build-time or be defined based on workflow relevant data, which becomes available at run-time.

Schema Evolution, Version Control and Instance Migration (Change Feature F1). In principle, through the support of change feature F1 process type changes and instance migration are supported. In general, these changes can be accomplished with the Staffware process editor, which only supports low-level change primitives (e.g., the insertion and/or deletion of nodes and control edges). With Staffware the co-existence of process instances of different schema versions is possible (F1[1]). In addition, Staffware allows for the migration of running process instances. However, such an instance migration – through lack of a formal correctness criteria – cannot be restricted to a subset of process instances. This might lead to severe process inconsistencies or even deadlocks in some situations (F1[4]).

Ad-hoc Changes (Change Feature F2). Staffware does not provide support for any of the adaptation patterns as no structural changes to the process instance schema can be performed. Flexibility is provided through patterns PP1 and PP4. Regarding support for ad-hoc changes similar considerations can be made than for the HOON approach (see Section 5.10, Change Feature F2).

In addition, the Staffware Process Orchestrator provides exception handling facilities, which allow for back and forward jumps.

Correctness of Changes (Change Feature F3). Through the absence of a formal process model and the lack of a formal correctness criterion for instance migrations bad surprise during run-time cannot be avoided. When testing the respective instance migration feature of Staffware in our lab, we were easily able to construct deadlocks and program crashes.

Traceability and Analysis (Change Feature F4). As no ad-hoc changes are supported an audit trail is sufficient for traceability of process instances. Additionally, the version management of process schemes allows for traceability of process type changes as well.

Access Control (Change Feature F5). In Staffware changes can be restricted to authorized users (F5[1]). Everyone having access to the process designer can change process schemes and perform instance migrations. In respect to pattern PP1, further restrictions can be defined, e.g., regarding the process fragments that can be selected for a placeholder activity (F1[2]). For each placeholder activity different authorizations can apply (F1[3]).

Change Reuse (Change Feature F6). The reuse of changes is not supported in Stafftware.

5.9 Evaluation Details: Flower

Our evaluation of the Flower case handling system is based on version 3.1 of the software.

Support for Adaptation Patterns. Flower provides explicit support for the *Delete* adaptation pattern (AP2) by allowing the skipping of process steps. No other adaptation patterns are directly supported. The respective pattern can be applied at the process instance level (Design Choice A[1]) and operates on atomic activities (Design Choice B[1]). The change pattern is applied in a permanent manner (Design Choice C[2]).

Support for Predefined Change Patterns. Pattern PP4 (*Multi-Instance Activity*) is supported through the concept of *dynamic subplans*. Like in Staffware and WIDE, the number of activity instances can either be fixed during build-time or be defined based on workflow relevant data, which becomes available at run-time.

Schema Evolution, Version Control and Instance Migration (Change Feature F1). Feature F1 is supported by Flower. First of all, Flower supports the co-existence of process instances of different schema versions (F1[3]). In addition, Flower allows to overwrite an existing process schema version. Thereby the user can remove all running process instances from the system (F1[1]) or let them remain in the system (F1[2]). In the latter case (uncontrolled) overwriting of a process schema cannot be avoided leading to inconsistencies. In summary, both options – removing running instances or overwriting process schemes in an uncontrolled manner – do not provide a satisfactory solution.

Ad-hoc Changes (Change Feature F2). In Flower direct support for the skipping of process activities is provided (F2[1]). In addition, moving activities is indirectly supported. In particular, in Flower users will be enabled to perform a respective activity earlier as planned if all required input data is available and certain other conditions are met.

Correctness of Changes (Change Feature F3). In general, correctness cannot be ensured in all situations. Especially, when overwriting an existing process schema this can lead to severe inconsistencies if ongoing instances are not removed from the system. The latter, in turn, is not an adequate option for practical environments.

Traceability and Analysis (Change Feature F4). Traceability is ensured in Flower as the completed instances are maintained in the system.

Access Control (Change Feature F5). Flower allows restricting changes to authorized users (F5[1]). For AP2 changes can be restricted to a particular user role and activity (F5[2,3]). Process type changes are possible for all users holding the administrator role.

Change Reuse (Change Feature F6). No change reuse is supported in Flower.

5.10 Summary of Evaluation Results

Our pattern-based evaluation of selected approaches shows that there exists no single system which supports all change patterns and features (cf. Fig. 21). 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. Though predefined change patterns allow to "simulate" certain adaptation patterns, change definition takes place at a lower level of abstraction.

6 Related Work

Patterns were first used to describe solutions to recurring problems by Ch. Alexander, who applied patterns to desrcibe best practices in architecture [31]. 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 [8]). In addition, workflow data patterns [32] describe different ways for modeling the data aspect in PAIS and workflow resource patterns [33] 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 [34] 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 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 which meet their flexibility requirements best (or to realize that no system satisfies them at all).

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 Shazia Shadiq, Michael Adams, Matthias Weske and Yanbo Han for their valuable feedback regarding the evaluation of their approaches. In addition, we would like to thank Shazia Shadiq and Michael Adams for the many fruitful discussions, which helped us to significantly improve the quality of this paper.

References

1. Dumas, M., ter Hofstede, A., van der Aalst, W., eds.: Process Aware Information Systems. Wiley Publishing (2005)
2. Rinderle, S., Reichert, M., Dadam, P.: Correctness criteria for dynamic changes in workflow systems – a survey. Data and Knowledge Engineering **50** (2004) 9–34
3. Reichert, M., Dadam, P.: ADEPT$_{flex}$ – supporting dynamic changes of workflows without losing control. JIIS **10** (1998) 93–129
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 (2000) Habil Thesis.
6. van der Aalst, W., Weske, M., Grünbauer, D.: Case handling: A new paradigm for business process support. Data and Knowledge Engineering. **53** (2005) 129–162
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. Rinderle, S., Weber, B., Reichert, M., Wild, W.: Integrating process learning and process evolution - a semantics based approach. In: BPM 2005. (2005) 252–267
10. Günther, C., Rinderle, S., Reichert, M., van der Aalst, W.: Change mining in adaptive process management systems. In: CoopIS'06. (2006) 309–326
11. Weber, B., Wild, W., Lauer, M., Reichert, M.: Improving exception handling by discovering change dependencies in adaptive process management systems. In: Business Process Management Workshops 2006. (2006) 93–104

12. Weber, B., Wild, W., Breu, R.: CBRFlow: Enabling adaptive workflow management through conversational cbr. In: ECCBR'04, Madrid (2004) 434–448
13. Reichert, M., Rinderle, S., Kreher, U., Dadam, P.: Adaptive Process Management with ADEPT2. In: ICDE'05. (2005)
14. Rinderle, S.: Schema Evolution in Process Management Systems. PhD thesis, University of Ulm (2004)
15. Reichert, M.: Dynamic Changes in Workflow-Management-Systems. PhD thesis, University of Ulm, Computer Science Faculty (2000) (in German).
16. Casati, F.: Models, Semantics, and Formal Methods for the design of Workflows and their Exceptions. PhD thesis, Milano (1998)
17. Casati, F., Ceri, S., Pernici, B., Pozzi, G.: Workflow evolution. Data and Knowledge Engineering **24** (1998) 211–238
18. Lu, R., Sadiq, S.W.: Managing process variants as an information resource. In: BPM06. (2006) 426–431
19. Sadiq, S., Sadiq, W., Orlowska, M.: Pockets of flexibility in workflow specifications. In: Proc. Int'l Entity–Relationship Conf. (ER'01), Yokohama (2001) 513–526
20. Sadiq, S., Sadiq, W., Orlowska, M.: A framework for constraint specification and validation in flexible workflows. Information Systems **30** (2005) 349 – 378
21. 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)
22. Hagemeyer, J., Hermann, T., Just, K., Rüdiger, S.: Flexibilität bei Workflow-Management-Systemen. In: Software-Ergonomie '97. (1997) 179–190
23. Han, Y.: Software Infrastructure for Configurable Workflow Systems. PhD thesis, Univ. of Berlin (1997)
24. 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)
25. Rinderle, S., Reichert, M., Dadam, P.: Flexible support of team processes by adaptive workflow systems. Distributed and Parallel Databases **16** (2004) 91–116
26. Rinderle, S., Reichert, M., Jurisch, M., Kreher, U.: On representing, purging, and utilizing change logs in process management systems. In: BPM'06. (2006) 241–256
27. Weber, B., Rinderle, S., Wild, W., Reichert, M.: CCBR–driven business process evolution. In: ICCBR'05, Chicago (2005) 610–624
28. Weber, B., Reichert, M., Wild, W.: Case-base maintenance for ccbr-based process evolution. In: ECCBR'06. (2006)
29. Deiters, W., Gruhn, V.: The funsoft net appoach to software process management. Int'l Journal of Software Engineering and Knowledge Engineering **4** (1994) 229–256
30. Gruhn, V.: Validation and Verification of Software Process Models. PhD thesis, University of Dortmund (1991)
31. Alexander, C., Ishikawa, S., Silverstein, M.: A Pattern Language. Oxford University Press, New York (1977)
32. 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)
33. Russell, N., ter Hofstede, A., Edmond, D., van der Aalst, W.: Workflow resource patterns. Technical Report WP 127, Eindhoven Univ. of Technology (2004)
34. Russell, N., van der Aalst, W.M., ter Hofstede, A.H.: Exception handling patterns in process-aware information systems. In: CAiSE'06. (2006)