

# Change Support in Process-Aware Information Systems - A Pattern-Based Analysis

Barbara Weber<sup>a,\*</sup>, Stefanie Rinderle-Ma<sup>b</sup> Manfred Reichert<sup>c</sup>

<sup>a</sup>*Department of Computer Science, University of Innsbruck,  
Technikerstraße 21a, 6020 Innsbruck, Austria*

<sup>b</sup>*Inst. Databases and Information Systems, Ulm University, Germany*

<sup>c</sup>*Information Systems Group, University of Twente, The Netherlands*

---

## Abstract

In today's dynamic business world the economic success of an enterprise increasingly depends on its ability to react to changes in its environment in a quick and flexible way. Process-aware information systems (PAIS) offer promising perspectives in this respect and are increasingly employed for operationally supporting business processes. To provide effective business process support, flexible PAIS are needed which do not freeze existing business processes, but allow for loosely specified processes, which can be detailed during run-time. In addition, PAIS should enable authorized users to flexibly deviate from the predefined processes if required (e.g., by allowing them to dynamically add, delete, or move process activities) and to evolve business processes over time. At the same time PAIS must ensure consistency and robustness. The emergence of different process support paradigms and the lack of methods for comparing existing change approaches have made 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 the systematic comparison of existing process management technology with respect to process change support. Based on these change patterns and features, we provide a detailed analysis and evaluation of selected systems from both academia and industry. The identified change patterns and change support features facilitate the comparison of change support frameworks, and consequently will support PAIS engineers in selecting the right technology for realizing flexible PAIS. In addition, this work can be used as a reference for implementing more flexible PAIS.

*Key words:* Workflow-Management, Process-Aware Information Systems, Patterns, Process Change, Process Flexibility

---

## 1 Introduction

In today's dynamic business world the economic success of an enterprise increasingly depends on its ability to react to changes in its environment in a quick and flexible way [22]. Causes for these changes can be manifold and include the introduction of new laws, market dynamics, or changes in customers' attitudes. For these reasons, companies have recognized business agility as a competitive advantage, which is fundamental for being able to successfully cope with business trends like increasing product and service variability, faster time-to-market and business-on-demand.

Process-aware information systems (PAIS), together with service-oriented computing, offer promising perspectives in this respect, and a growing interest in aligning information systems in a process- and service-oriented way can be observed [15,68,79]. In contrast to data- or function-centered information systems (IS), PAIS are characterized by a strict separation of process logic and application code. In particular, most PAIS describe the process logic explicitly in terms of a process model providing the schema for process execution. Usually, the core of the process layer is build by a process management system (PMS), which provides generic functionality for modeling, executing, and monitoring processes. This approach allows for a separation of concerns, which is a well established principle in computer science for increasing maintainability and reducing cost of change [13]. In many cases changes to one layer can be performed without affecting the other layers. For example, modifying the application service which implements a particular process activity does usually not imply any change to the process layer as long as interfaces remain stable, i.e., the external observable behavior of the service remains the same. In addition, changing the execution order of process activities or adding new activities to the process can, to a large degree, be accomplished without touching any of the application services.

The ability to efficiently deal with process changes has been identified as one of the critical success factors for any PAIS [37,27,26]. Through the above described separation of concerns, in principle, PAIS facilitate changes significantly. According to a recent study conducted among several Dutch companies, however, enterprises are reluctant to change PAIS implementations once they are running properly [47]. High complexity and high cost of change are mentioned as major reasons for not fully leveraging the potential of PAIS. To overcome this problem flexible PAIS are needed enabling companies to capture real-world processes adequately without leading to a mismatch between

---

\* Corresponding author.

*Email addresses:* Barbara.Weber@uibk.ac.at (Barbara Weber),  
stefanie.rinderle@uni-ulm.de (Stefanie Rinderle-Ma),  
m.u.reichert@cs.utwente.nl (Manfred Reichert).

computerized processes and those running in reality [61,27]. In particular, the introduction of a PAIS must not lead to rigidity and must not freeze existing business processes [10]. Instead, authorized users must be able to flexibly deviate from the predefined processes as required (e.g., to deal with exceptions) and to evolve PAIS implementations over time (e.g., to continuously adapt the underlying process models to process optimizations). Such changes must be possible at a high level of abstraction without affecting consistency and robustness of the PAIS [49].

### 1.1 Problem Statement

The need for flexible and easily adaptable PAIS has been recognized by both academia and industry and several competing paradigms for addressing process changes and process flexibility have been developed (e.g., adaptive processes [43,2,78], case handling [69], declarative processes [39], and late binding and modeling [2,59,23,21]). However, there is still a significant lack of methods for systematically comparing the change frameworks provided by existing process support technologies. This makes it difficult for PAIS engineers to assess the maturity and the change capabilities of those technologies, often resulting in wrong decisions and expensive misinvestments.

To make PAIS better comparable, *workflow patterns* have been introduced [67]. Respective patterns offer promising perspectives by providing a means for analyzing the expressiveness of process modeling tools and languages in respect to different workflow perspectives. In particular, proposed workflows patterns cover the control flow [67,55], the data flow [53], and the resource perspective [54]. Obviously, broad support for workflow patterns allows for building more flexible PAIS. However, an evaluation of a PAIS regarding its ability to deal with changes needs a broader view. In addition to *build-time flexibility* (i.e., the ability to pre-model flexible execution behavior based on advanced workflow patterns), *run-time flexibility* has to be considered as well [44]. Run-time flexibility is to some degree addressed by the exception handling patterns as proposed in [56]. These patterns describe different ways for coping with the exceptions that might occur during process execution (e.g., activity failures or non-availability of a particular service). Patterns like *Roll-back* or *Redo* allow users to deal with such exceptional situations by changing the state of a running process (i.e., its behavior); usually, they do not affect process structure. In many cases, changing the observable behavior of a running instance is not sufficient, but the process structure has to be adapted as well [44]. In addition, exception handling patterns cover changes at the process instance level, but are not applicable to process schema changes.

Consequently, the existing patterns have to be extended by a set of patterns

suitable for providing a comprehensive evaluation of run-time flexibility as well. In addition, a PAIS’s ability to deal with changes does not only depend on the expressiveness of the used process modeling language and the respective change framework, but also on the features provided by the PAIS to support these changes. Expressiveness only allows making statements whether a particular change can be conducted or not. For example, it provides information on whether or not additional process activities can be added or existing activities can be deleted. However, it does not give insights into how quickly and easily such process changes can be accomplished and whether consistency and correctness are ensured at all time. For example, many of the proposed change frameworks require the user to perform changes at a rather low level of abstraction by manipulating single nodes and edges. This does not only require a high level of expertise, but also slows down the entire change process. In addition, not all of the PAIS supporting dynamic process changes ensure correctness and robustness afterwards, which might lead to inconsistencies, deadlocks or other flaws [73]. Again, methods for a systematic comparison of these frameworks in respect to their ability to deal with changes would facilitate the procedure of selecting an appropriate PAIS.

## 1.2 Contribution

During the last years we have studied processes from different application domains (e.g., healthcare [27], automotive engineering [35], logistics [80], procurement [60], and finance). Further, we elaborated the flexibility and change support features of numerous tools and approaches. Based on these experiences, in this paper, we propose a set of *changes patterns* and *change support features* to foster the comparison of existing approaches with respect to their ability to deal with process changes. Thereby, we focus on control flow changes, and omit issues related to the change of other process perspectives. The extension towards other process aspects (e.g., data flow or resources) constitutes complementary work and is outside the scope of this paper.

Change patterns allow for high-level process adaptations at the process type as well as the process instance level. They allow assessing the expressiveness of change frameworks. Change support features, in turn, ensure that changes are performed in a correct and consistent way, change traceability is enabled, and process changes are facilitated for users. Another contribution constitutes the evaluation of selected approaches from both industry and academia based on the proposed change patterns and change support features.

This paper provides a significant extension of the work we presented in [73]. While in [73] the proposed patterns have been only described very briefly and informally, this paper provides an in-depth description of all identified

change patterns. To obtain unambiguous patterns descriptions, in addition, we enhance their description with a formal specification of their semantics independent of the concrete process meta model they operate on. The discussion of how change patterns can be applied has been also considerably extended. Finally, we include additional patterns and change support features in our comparison framework and we extend the evaluation to a larger set of approaches and tools. Further, this work can be seen as a reference for implementing adaptive and more flexible PAIS. In analogy to the workflow patterns initiative [67], we expect further systems to be evaluated over time and vendors of existing PAIS are expected to extend their current PAIS towards more complete change pattern and change feature support.

Section 2 summarizes background information needed for the understanding of this paper. Section 3 describes 18 change patterns sub-dividing them into adaptation patterns and patterns for changes in predefined regions. The formal semantics of these patterns is provided by Section 4. Section 5 deals with 7 crucial change support features. Taking these change patterns and features, Section 6 evaluates different approaches. Section 7 presents related work. We conclude with a summary and outlook in Section 8.

## 2 Background Information

In this section we describe basic concepts and notions used in this paper.

### 2.1 Basic Notions

A PAIS is a specific type of information system which provides process support functions and allows for the separation of process logic and application code. For this purpose, at build-time the process logic has to be explicitly defined based on the constructs provided by a process meta model (e.g., Workflow Nets [65], WSMNets [48]). At run-time the PAIS then orchestrates the processes according to their defined logic and allows integrating users and other resources. Workflow Management Systems (e.g., Staffware [15], ADEPT [43], WASA [78]) and Case-Handling Systems (e.g., Flower [15,69]) are typical technologies enabling PAIS.

In a PAIS, for each business process to be supported (e.g., booking a business trip or handling a medical order), a *process type* represented by a *process schema*  $S$  has to be defined. For one particular process type several process schemes may exist representing the different versions and the evolution of this type over time. In the following, a single process schema corresponds to a

directed graph, which comprises a set of *nodes* – representing process steps (i.e., *activities*) or control connectors (e.g, XOR-Split, AND-Join) – and a set of control edges between them. Control edges specify the precedence relations between the different nodes. Furthermore, activities can either be *atomic* or *complex*. While an atomic activity is associated with an invokable application service, a complex activity contains a sub process or, more precisely, a reference to a sub process schema  $S'$ . This allows for the hierarchical decomposition of process schemes.

Most of the patterns introduced in this paper are not only applicable to atomic or complex activities, but also to sub process graphs with single entry and single exit node (also denoted as *hammocks* in graph literature [82]). In this paper, we use the term *process fragment* as a generalized concept covering atomic activities, complex activities (i.e., sub processes) and hammocks (i.e., sub graphs with single entry / exit node). If a pattern is denoted as being applicable to a process fragment, it can be applied to all these objects.

The above described meta model is shown in Fig. 1, whereas an abstract example of a process schema based on this meta model is given in Fig. 2. As a graphical illustration we use the BPMN notation [38]. In Fig. 2 process

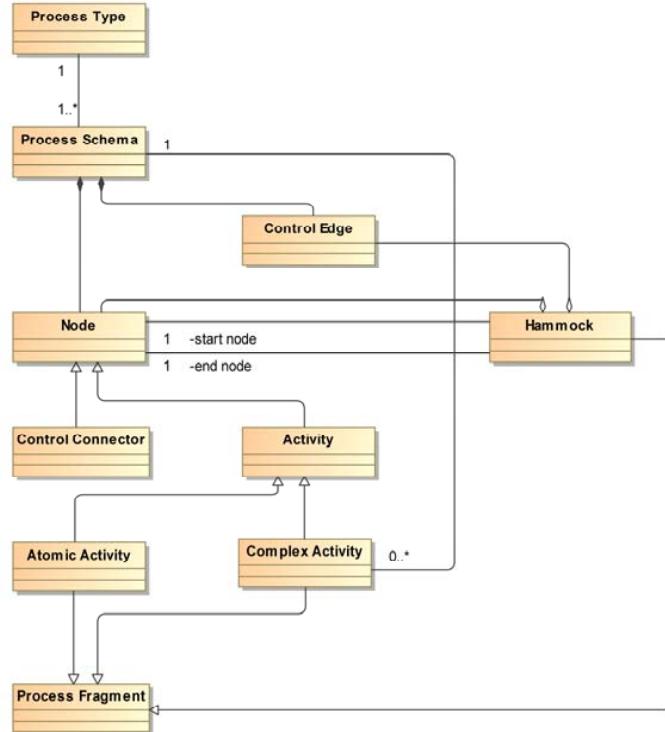


Fig. 1. Basic Notions - Process Meta-Model

schema  $S_1$  consists of six activities and two control connectors: Activity A is followed by activity B in the flow of control, whereas activities C and D can be processed in parallel. Activities A to E are atomic, and activity F constitutes a

complex activity (i.e., sub process with own process schema  $S_2$ ). The region of the process schema containing activities B, C, D and E as well as the control connectors AND-Split and AND-Join constitutes an example for a hammock, i.e., process sub graph with single entry and single exit point. The term *process fragment* covers all of the above mentioned concepts and can either be an atomic activity (e.g., activity A), an encapsulated sub process like process schema  $S_2$ , or a hammock (e.g., the sub graph consisting of activities B, C, D, E and the two connector nodes). Based on process schema  $S$ , at run-time new *process instances* can be created and executed. Regarding process instance  $I_1$  from Fig. 2, for example, activity A is completed and activity B is activated. Generally, a large number of instances, all being in different states, might run on a particular process schema.

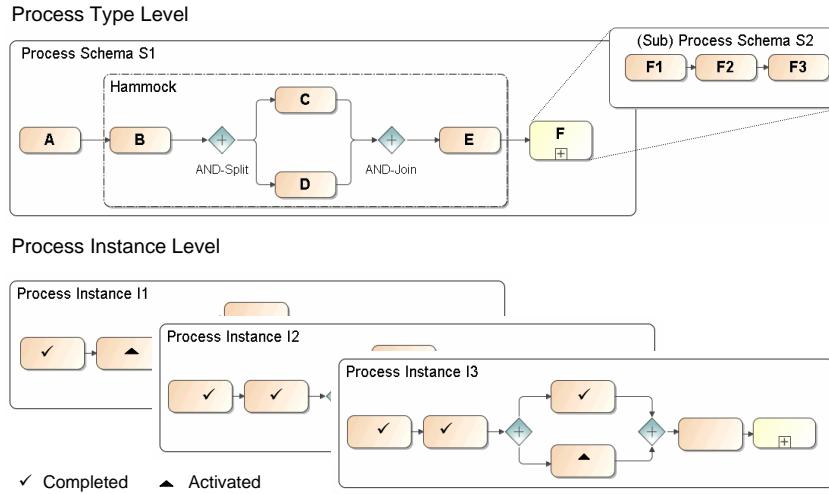


Fig. 2. Core Concepts - An Example

## 2.2 Process Flexibility

To deal with evolving processes, exceptions and uncertainty, PAIS must be flexible. This can either be achieved through structural process adaptations (cf. Fig. 3) or by allowing for loosely specified process models, which can be refined by users during run-time according to predefined criteria (cf. Fig. 4).

**Process Adaptation.** In general, process adaptations can be triggered and performed at two levels – the process type and the process instance level (cf. Fig. 3) [49]. *Process schema changes at the type level* (in the following called *schema evolution*) become necessary to deal with the evolving nature of real-world processes (e.g., to adapt them to legal changes). Such a schema evolution often necessitates the propagation of respective changes to ongoing process instances (with respective type), particularly if these instances are long-running. For example, let us assume that in a patient treatment process, due to a new

legal requirement, patients have to be educated about potential risks before a surgery takes place. Let us further assume that this change is also relevant for patients for which the treatment has already been started. In such a scenario, stopping all ongoing treatments, aborting them and re-starting them is not a viable option. As a large number of treatment processes might be running at the same time, applying this change manually to all ongoing treatment processes is also not a feasible option. Instead efficient system support is required to add this additional information step to all patient treatments for which this is still feasible (e.g., if the surgery has not yet started). *Ad-hoc changes* of single process instances, in turn, are usually performed to deal with exceptions or unanticipated situations, resulting in an adapted *instance-specific* process schema [43,78,32]. The effects of such ad-hoc changes are usually instance-specific, and consequently do not affect any other ongoing process instance. In a medical treatment process, for example, a patient's current medication may have to be discontinued due to an allergic reaction of this particular patient.

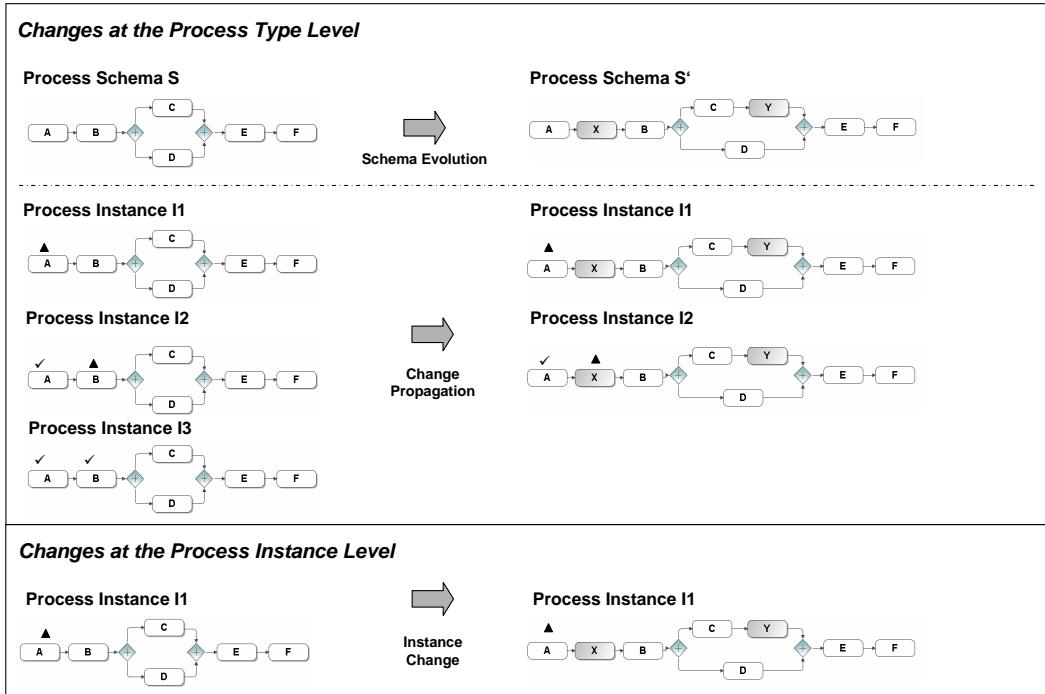


Fig. 3. Process Adaptation

**In-built Flexibility.** Flexibility can be also achieved by leaving parts of the process model unspecified at build-time and by adding the missing information during run-time (cf. Fig. 4) [2,39,59,23,21]. This approach is especially useful in case of uncertainty as it allows deferring decisions from build-time to run-time, where more information becomes available. For example, when treating a cruciate rupture for a particular patient we might not know in advance which treatments will be exactly performed in which execution order. Therefore, this part of the process remains unspecified during build-time and the physician decides on the exact treatments at run-time.

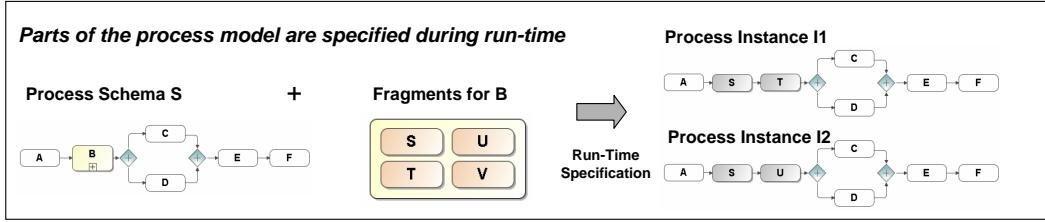


Fig. 4. In-built Flexibility

### 3 Change Patterns

In this section we describe 18 characteristic patterns we identified as being relevant for *control flow changes* in PAIS (cf. Fig. 6). Adaptations of other process aspects (e.g., data or resources) are outside the scope of this paper, but are addressed in our complementary work on change patterns. Like design patterns in software engineering, change patterns aim at reducing system complexity [16] by raising the level of abstraction for expressing changes in PAIS. The change patterns described in this paper constitute solutions to commonly occurring changes in PAIS. As an example consider the insertion of an additional process fragment into a given process schema. Most of our patterns have been identified based on several case studies investigating changes of real-world processes in different domains (cf. Section 1.2). Furthermore, we have analysed change facilities offered by existing PAIS-enabling technologies to complete the set of relevant change patterns.

We divide the change patterns into two major categories: *adaptation patterns* and *patterns for changes in predefined regions*. Thereby, adaptation patterns allow for structural process adaptations, whereas patterns for changes in predefined regions allow for in-built flexibility (cf. Section 2.2).

**Adaptation Patterns** allow structurally modifying a process schema at the type or instance level by using high-level change operations (e.g., to add an activity in parallel to another one) instead of low-level change primitives (e.g., to add a single node or delete a single edge). Though process adaptations can be performed using low-level change primitives as well, these primitives are not considered as real change patterns due to their lack of abstraction. Generally, adaptation patterns can be applied to the whole process schema, i.e., the region to which the adaptation pattern is applied can be chosen dynamically. Therefore, adaptation patterns are well suited for dealing with exceptions.

**Patterns for Changes in Predefined Regions.** By contrast, patterns for changes in predefined regions do not enable structural process adaptations, but allow process participants to add information regarding unspecified parts of the process model (i.e., its process schema) during run-time. For this purpose, the application of these patterns has to be anticipated at build-time.

This can be accomplished by defining regions in the process schema where potential changes may be performed during run-time. As process schema changes or process schema expansions can only be applied to these predefined regions, respective patterns are less suited for dealing with arbitrary exceptions [43]. Instead they allow for dealing with situations where, due to uncertainty, decisions cannot be made at build-time, but have to be deferred to run-time. Fig. 5 gives a comparison of these two major pattern categories.

	<b>Adaptation Pattern</b>	<b>Patterns in Changes to Predefined Regions</b>
<b>Structural Process Change</b>	YES	NO
<b>Anticipation of Change</b>	NO	YES
<b>Change Restricted to Predefined Regions</b>	NO	YES
<b>Application Area</b>	Unanticipated exceptions, unforeseen situations	Address uncertainty by deferring decisions to run-time

Fig. 5. Adaptation Patterns vs. Patterns for Changes in Predefined Regions

Fig. 6 gives an overview of the 18 patterns described in detail in the following. 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. In particular, *design choices* allow for parametrizing change patterns keeping the number of distinct patterns manageable. While the deletion of such a process fragment constitutes a change pattern, a design choice allows parametrizing whether the pattern can be applied to atomic activities, complex activities or hammocks (cf. Section 2). Design choices not only relevant for a particular pattern, but for a set of patterns, are described only once for the entire set. 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*.

### 3.1 Adaptation Patterns

Adaptation patterns allow users to structurally change process schemes. In general, the application of an adaptation pattern transforms a process schema  $S$  into another process schema  $S'$ . For doing so two different options exist, which can be both found in existing systems (cf. Section 6).

On the one hand, structural adaptations can be realized based on a set of *change primitives* like `add node`, `remove node`, `add edge`, `remove edge`, and `move edge`. Following this obvious approach, the realization of a particular adaptation (e.g., to delete an activity or to add a new one) usually requires the application of multiple change primitives. To specify structural adaptations

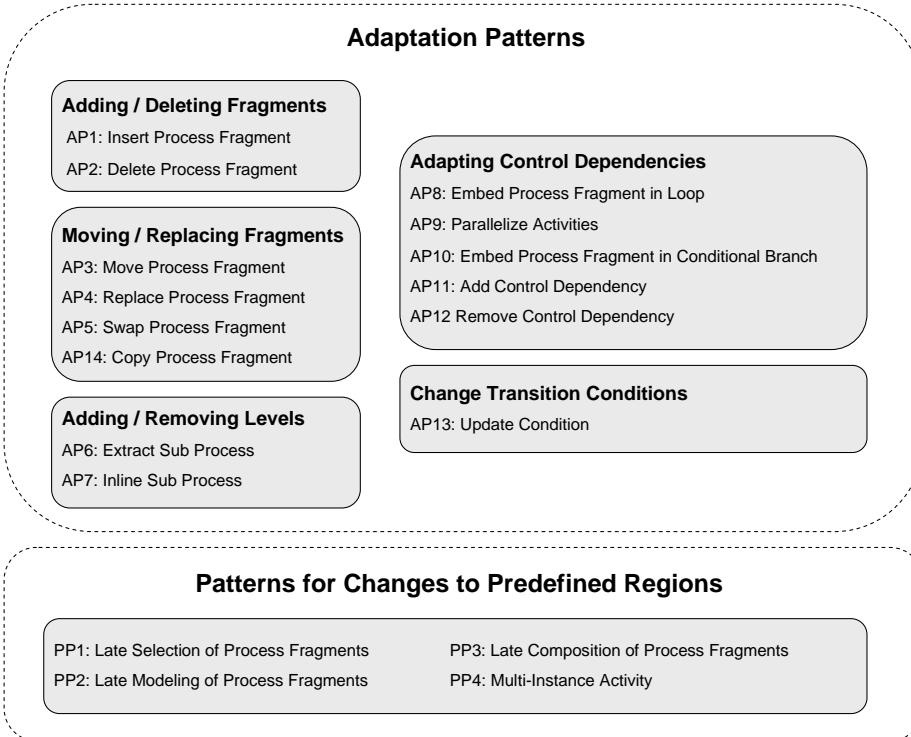


Fig. 6. Change Patterns Overview

at this low level of abstraction, however, is a complex and error-prone task. Further, when applying a single change primitive, soundness of the resulting process schema (e.g., absence of deadlocks) cannot be guaranteed. Therefore, for more complex process meta models it is not possible to associate formal pre-/post-conditions with the application of single primitives. Instead, correctness of a process schema has to be explicitly checked after applying the respective set of primitives. A detailed discussion on how change complexity and cost of change are affected when using change primitives is given in Section 3.3.

On the other hand, structural adaptations can be based on a set of high-level change operations (e.g., to insert process fragment between two sets of nodes), which abstract from the concrete schema transformations to be conducted. Instead of specifying a set of change primitives the user applies one or few high-level change operations to realize the desired process schema adaptation. Approaches following this direction often associate pre- and post-conditions with the high-level operations, which allows to guarantee soundness when applying the respective operations [43,8]. Note that soundness will become a fundamental issue if changes are to be applied by end-users or – even more challenging – by automated software components (i.e., software agents [49,36]). For these reasons we only consider high-level operations as adaptation patterns; more precisely, an adaptation pattern comprises exactly one high-level operation. Its application to a process schema will preserve soundness of this schema if certain pre-conditions are met. A detailed discussion on how

change complexity is reduced when applying adaptation patterns is given in Section 3.3.

In total, 14 out of the 18 identified patterns can be classified as adaptation patterns. In the following all 14 adaptation patterns are described in detail (cf. Fig. 6). Adaptation patterns AP1 and AP2 allow for the insertion (AP1) and deletion (AP2) of process fragments in a given process schema. Moving and replacing fragments is supported by adaptation patterns AP3 (Move Process Fragment), AP4 (Replace Process Fragment), AP5 (Swap Process Fragment) and AP14 (Copy Process Fragment). Pattern AP6 and AP7 allow adding or removing levels of hierarchy. Thereby, the extraction of a sub process from a process schema is supported by AP6, whereas the inclusion of a sub process into a process schema is supported by AP7. AP8-AP12 support the adaptation of control dependencies: embed an existing process fragment in a loop (AP8), parallelize a process fragment (AP9), embed an existing process fragment in a conditional branch (AP10), and add / remove control dependencies (AP11, AP12). Finally, pattern update of transition conditions (AP13) allows for changes in the transition conditions.

Fig. 7 describes two general design choices, which are valid for all adaptation patterns and which can be used for their parametrization. Additional design choices, only relevant in the context of a specific adaptation pattern, are provided with the detailed descriptions of the respective patterns (cf. Fig. 8-21). The design choices of Fig 7 are shortly described in the following. First, each adaptation pattern can be applied at the process type and/or process instance level (cf. Fig. 2). If an adaptation pattern is supported at the process type level, users may edit the respective process model at build-time using a graphical process editor and a high-level change operation implementing the respective pattern. In principle, adaptation patterns can be simulated based on change primitives. However, support for adaptation patterns at the process type level facilitates process modeling by raising the level of abstraction and by reducing complexity (cf. Section 3.3). In case that a respective pattern is also applicable to the process instance level, run-time changes of single instances can be accomplished. Second, adaptation patterns can operate on an atomic activity, an encapsulated sub process or a hammock (cf. Fig. 7).

In the following all identified patterns are described in detail.

**Adaptation Pattern AP1: Insert Process Fragment.** The *Insert Process Fragment* pattern (cf. Fig. 8) can be used to add process fragments to a process schema. In addition to the general design choices described in Fig. 7, one major design choice for this pattern (Design Choice C) describes the position at which the new process fragment is embedded in the respective schema. There are systems which only allow users to serially insert a process fragment between two directly succeeding activities [25]. By contrast, other systems

General Design Choices for Adaptation Patterns	
A.	What is the scope of the pattern?
1.	The respective patterns can be applied at the process instance level
2.	The respective patterns can be applied at the process type level
B.	Where does the change pattern operate on?
1.	On an atomic activity
2.	On a sub process
3.	Or on a hammock
<b>Design Choice B (illustrated for adaptation pattern AP1)</b>	
<p>The diagram illustrates a process instance <b>Process Instance I</b> consisting of activities A, B, C, D, E, and F. Activity C is a decision node with two outgoing paths. The top path leads to activity E, which then leads to activity F. The bottom path from C leads to activity D. Activity D is also a decision node with two outgoing paths: one leading to activity E and another leading back to activity C. Below the process instance, there is an <b>Atomic Activity</b> (represented by a rounded rectangle labeled X) and a <b>Hammock</b> (represented by a rounded rectangle labeled Y followed by a dashed arrow labeled Z). Above the process instance, there is a <b>Complex Activity</b> (represented by a rounded rectangle labeled G containing activities F1, F2, and F3). A dashed arrow points from the Complex Activity to activity F, indicating its interaction with the process instance.</p>	

Fig. 7. General Design Choices for Adaptation Patterns

follow a more general approach allowing the user to insert new fragments between two sets of activities meeting certain constraints [43]. Special cases of the latter variant include the insertion of a process fragment in parallel to another one (*parallel insert*) or the additional association of the newly added fragment with an execution condition (*conditional insert*).

**Adaptation Pattern AP2: Delete Process Fragment.** The *Delete Process Fragment* pattern can be used to remove a process fragment (cf. Fig 9). No additional design choices are needed for this pattern. Fig. 9 depicts alternative ways in which this pattern can be implemented. The first implementation option is to physically delete the respective process fragment, i.e., to remove the corresponding nodes and control edges from the process schema. The second implementation option replaces the fragment by one or more silent activities (i.e., activities without associated action). In the third implementation option the fragment is embedded in a conditional branch with condition FALSE (i.e., the fragment remains part of the schema, but will not be executed).

**Adaptation Pattern AP3: Move Process Fragment.** The *Move Process Fragment* pattern (cf. Fig. 10) allows users 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 re-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*) or be based on change primitives, we introduce it as separate pattern since it provides a higher level of abstraction to users.

Pattern AP1: INSERT Process Fragment	
<b>Description</b>	A process fragment X is added to a process schema S.
<b>Example</b>	For a particular patient an allergy test has to be added to his treatment process due to a drug incompatibility.
<b>Problem</b>	In a real world process a task has to be accomplished which has not been modeled in the process schema so far.
<b>Design Choices (in addition to those described in Fig. 7)</b>	<p>C. How is the new process fragment X embedded in the process schema?</p> <ol style="list-style-type: none"> <li>1. X is inserted between 2 directly succeeding activities (<code>serial insert</code>)</li> <li>2. X is inserted between 2 activity sets (<code>insert between node sets</code>) <ul style="list-style-type: none"> <li>a) Without additional condition (<code>parallel insert</code>)</li> <li>b) With additional condition (<code>conditional insert</code>)</li> </ul> </li> </ol> <pre> graph TD     subgraph serialInsert [serialInsert]         A1[A] --&gt; X1[X]         X1 --&gt; B1[B]     end     subgraph parallelInsert [parallelInsert]         A2[A] --&gt; AS1{AND-Split}         AS1 --&gt; X2[X]         X2 --&gt; AJ1{AND-Join}         AJ1 --&gt; C2[C]     end     subgraph conditionalInsert [conditionalInsert]         A3[A] --&gt; OS1{XOR-Split}         OS1 --&gt; C3[C]         OS1 --&gt; X3[X]         X3 --&gt; OJ1{XOR-Join}         OJ1 --&gt; B3[B]         X3 --&gt; C3         cond["if d &gt; 0"] --&gt; OS1     end </pre>
<b>Implementation</b>	This adaptation pattern can be realized by transforming the high level insertion operation into a sequence of low level change primitives.

Fig. 8. *Insert Process Fragment (AP1)* pattern

**Adaptation Pattern AP4: Replace Process Fragment.** This pattern supports the replacement of a process fragment by another one (cf. Fig. 11). Like the *Move Process Fragment* pattern, this pattern can be implemented based on patterns AP1 and AP2 (*Insert/Delete Process Fragment*) or be directly based on change primitives.

**Adaptation Pattern AP5: Swap Process Fragments.** The *Swap Process Fragment* pattern (cf. Fig. 12) allows users to swap a process fragment with another one. The process fragments to be swapped do not have to be directly connected. This adaptation pattern can be implemented based on pattern AP3 (*Move Process Fragment*), the combined use of patterns AP1 and AP2 (*Insert/Delete Process Fragment*), or change primitives.

**Adaptation Pattern AP6: Extract Sub Process.** The pattern *Extract Sub Process* (AP6) allows users to extract an existing process fragment from a process schema and to encapsulate it in a separate sub process schema (cf. Fig. 13). This pattern can be used to add a hierarchical level to simplify a

## Pattern AP2: DELETE Process Fragment

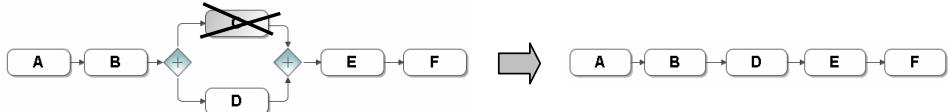
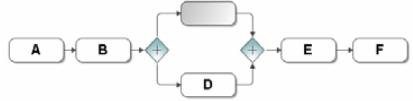
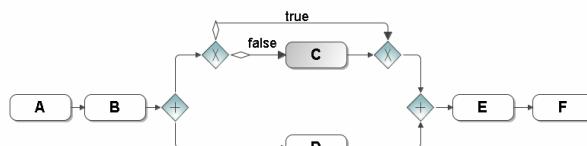
<b>Description</b>	A process fragment is deleted from a process schema S.
<b>Example</b>	For a particular patient a planned computer tomography must not be performed in the context of her treatment process due to the fact that she has a cardiac pacemaker, i.e., the computer tomography activity has to be deleted.
<b>Problem</b>	In a real world process a planned task has to be skipped or deleted.
<b>Implementation</b>	<p>Several options for implementing this pattern exist:</p> <ul style="list-style-type: none"> <li>(1) The fragment is physically deleted (i.e., corresponding activities and control edges are removed from the process schema based on change primitives)</li>  <pre> graph LR     A[A] --&gt; B[B]     B --&gt; D{ }     D --&gt; E[E]     E --&gt; F[F]     </pre>   <li>(2) The fragment is replaced by one or more silent activities (i.e., activities without associated actions)</li>  <pre> graph LR     A[A] --&gt; B[B]     B --&gt; D[ ]     D --&gt; E[E]     E --&gt; F[F]     </pre>   <li>(3) The fragment is embedded in a conditional branch with condition <i>false</i> (i.e., the fragment remains part of the schema, but is not executed)</li>  <pre> graph LR     A[A] --&gt; B[B]     B --&gt; D{ }     D -- true --&gt; C[C]     C --&gt; E[E]     E --&gt; F[F]     D -- false --&gt; D[ ]     D --&gt; E     </pre> </ul>

Fig. 9. Delete Process Fragment (AP2) pattern

process schema or to hide information from process participants. If no direct support for pattern AP6 is provided a possible workaround looks 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 schema and required input and output parameters must be manually mapped to the sub process (not considered in detail here). The implementation of pattern AP6 can be based on graph aggregation techniques [7].

### Pattern AP3: MOVE Process Fragment

<b>Description</b>	A process fragment is moved from its current position in process schema S to another position within the same schema.
<b>Example</b>	Usually employees may only book a flight after it has been approved by the manager. Exceptionally, for a particular process the booking of a flight shall be done in parallel to the approval activity; consequently the <i>book flight</i> activity has to be moved from its current position in the process to a position parallel to the approval activity.
<b>Problem</b>	Predefined ordering constraints cannot be completely satisfied for a set of activities.
<b>Design Choices (in addition to those described in Fig. 7)</b>	<p>C. How is the additional process fragment X embedded in S?</p> <ol style="list-style-type: none"> <li>1. X is inserted between 2 directly succeeding activities (serial move)</li> <li>2. X is inserted between 2 activity sets (move between node sets)             <ol style="list-style-type: none"> <li>a) Without additional condition (parallel move)</li> <li>b) With additional condition (conditional move)</li> </ol> </li> </ol>
<b>Implementation</b>	This adaptation pattern can be implemented based on patterns AP1 and AP2 (insert / delete process fragment) or based on change primitives.
<b>Related Patterns</b>	<i>Swap</i> adaptation pattern (AP5)

Fig. 10. *Move Process Fragment (AP3)* pattern

**Adaptation Pattern AP7: Inline Sub Process.** As opposed to pattern AP6 (Extract Process Fragment), the pattern *Inline Sub Process* (AP7) allows users to inline a sub process schema into the parent process, and consequently to flatten the hierarchy of the overall process (cf. Fig. 14). This might become necessary in case a process schema is divided into too many hierarchical levels or for improving the structure of a process schema. If no direct support for pattern 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 invoking

#### Pattern AP4: REPLACE Process Fragment

<b>Description</b>	A process fragment is replaced by another process fragment in process schema S.
<b>Example</b>	For a particular patient a planned computer tomography must not be performed in the context of her treatment process due to the fact that she has a cardiac pacemaker. Instead of the <i>computer tomography</i> activity, the <i>X-ray</i> activity shall be performed for a particular patient.
<b>Problem</b>	A process fragment is no longer adequate, but can be replaced by another one.
	<pre> graph LR     subgraph S         A1[A] --&gt; X1[X]         X1 --&gt; C1[C]         X1 --&gt; D1[D]     end     subgraph S_prime         A2[A] --&gt; X2[X]         X2 --&gt; C2[C]     end     </pre>

Fig. 11. Replace Process Fragment (AP4) pattern

#### Pattern AP5: SWAP Process Fragment

<b>Description</b>	Two existing process fragments are swapped in process schema S.
<b>Example</b>	Regarding a particular delivery process the order in which requested goods shall be delivered to two customers has to be swapped.
<b>Problem</b>	The predefined ordering of two existing process fragments has to be changed by swapping their position in the process schema.
	<pre> graph LR     subgraph S         A1[A] --&gt; B1[B]         B1 --&gt; C1[C]         C1 --&gt; D1[D]         D1 --&gt; E1[E]         E1 --&gt; F1[F]         F1 --&gt; G1[G]     end     subgraph S_prime         F2[F] --&gt; C2[C]         C2 --&gt; D2[D]         D2 --&gt; E2[E]         E2 --&gt; A2[A]         A2 --&gt; B2[B]         B2 --&gt; G2[G]     end     </pre>
<b>Implementation</b>	This adaptation pattern can be implemented either based on pattern AP3 ( <i>move process fragment</i> ), on the combined use of patterns AP1 and AP2 ( <i>insert / delete process fragment</i> ) or based on change primitives.
<b>Related Patterns</b>	<i>Move Process Fragment</i> (AP3)

Fig. 12. Swap Process Fragment (AP5) pattern

the sub process has to be replaced by the previously copied process fragment. Further, input and output parameters of the sub process have to be 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 to allow

### Pattern AP6: EXTRACT Process Fragment to Sub Process

<b>Description</b>	From a given process schema S a process fragment is extracted and replaced by a corresponding sub process.
<b>Example</b>	A dynamically evolving engineering process has become too large. To reduce complexity the process owner extracts activities related to the engineering of a particular component and encapsulates them in a separate sub process.
<b>Problem</b>	<p><i>Large process schema.</i> 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.</p> <p><i>Duplication across process schemes.</i> 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 (see figure below).</p>
<b>Implementation</b>	To implement pattern 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 must be considered as input / output for the created sub process.
<b>Related Patterns</b>	<i>Inline Sub Process (AP7)</i>

Fig. 13. *Extract Sub Process (AP6)* pattern

for the repeated execution of the respective fragment (cf. Fig. 15). This pattern can be realized based on patterns AP1 (*Insert Process Fragment*), AP11 (*Add Control Dependency*) and AP12 (*Remove Control Dependency*). However, pattern AP8 offers the advantage that the number of operations needed for accomplishing such a change can be reduced (cf. Section 3.3).

**Adaptation Pattern AP9: Parallelize Process Fragments.** This pattern enables the parallelization of process fragments which were confined to be executed in sequence (cf. Fig. 16). If no direct support for this pattern is provided, it can be simulated by combining patterns AP11 and AP12 (*Add / Remove Control Dependency*) or by using pattern AP3 (*Move Process Frag-*

### Pattern AP7: INLINE Sub Process

<b>Description</b>	A sub process to which one or more process schemes refer is dissolved. Accompanying to this the corresponding sub process graph is directly embedded in the parent schemes.
<b>Example</b>	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 current structure of selected sub processes up to the top level.
<b>Problem</b>	<p><i>Too many hierarchies in a process schema:</i> If a process schema consists of too many hierarchy levels the inline sub process pattern can be used to flatten the hierarchy.</p> <p><i>Badly structured sub processes:</i> 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).</p>
<b>Implementation</b>	The implementation of this adaptation pattern can be based on other adaptation patterns (e.g., AP1). When considering data aspects as well, the data context of the sub process and its current mapping to the parent process have to be transferred to the parent process schema.
<b>Related Patterns</b>	<i>Extract Process Fragment to Sub Process</i> (AP6)

Fig. 14. *Inline Sub Process (AP7)* pattern

ment). However, a separate pattern allows users to accomplish such a change more effectively.

**Adaptation Pattern AP10: Embed Process Fragment in Conditional Branch.** Using this pattern an existing process fragment can be embedded in a conditional branch, which will be only executed if certain conditions are met (cf. Fig. 17). 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 a control edge (e.g., for synchronizing the execution order of two parallel activities) is added to the given process schema (cf. Fig. 18). As opposed to the low-level change primitive *add edge*, the added control dependency shall not violate soundness (e.g., no deadlock causing cycles). Therefore, approaches implementing AP11 usually ensure that the use of this pattern meets certain pre- and post-conditions. Further, the newly added edge

### Pattern AP8: Embed Process Fragment in Loop

<b>Description</b>	Adds a loop construct to a process schema in order to surround an existing process fragment
<b>Example</b>	Regarding the treatment process of a particular patient a lab test shall be not only performed once (as in the standard treatment process), but be repeated daily due to special risks associated with the patient.
<b>Problem</b>	A process fragment is actually executed at most once, but needs to be executed recurrently based on some condition.
<b>Implementation</b>	This adaptation pattern can be implemented based on Patterns AP1 (insert process fragment), AP11 and AP12 (add / remove control dependency). Alternatively, implementation can be based on change primitives.
<b>Related Patterns</b>	<i>Embed Process Fragment in Conditional Branch (AP10)</i>

Fig. 15. *Embedded Process Fragment in Loop (AP8) pattern*

### Pattern AP9: PARALLELIZE Process Fragments

<b>Description</b>	Process fragments which have been confined to be executed in sequence so far are parallelized in a process schema S.
<b>Example</b>	For a running production process the number of resources is dynamically increased. Thus, certain activities which have been ordered sequentially so far can now be processed in parallel.
<b>Problem</b>	Ordering constraints predefined for a set of process fragments turn out to be too strict and shall therefore be removed.
<b>Implementation</b>	This adaptation pattern can be implemented based on Patterns AP11 and AP12 (add / remove control dependency) or based on change primitives.

Fig. 16. *Parallelize Process Fragments (AP9) pattern*

can be associated with attributes (e.g., transitionconds) when applying AP11. 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.

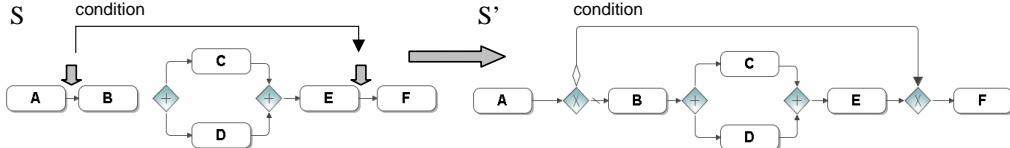
**Adaptation Pattern AP12: Remove Control Dependency.** Using this

### 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 in such a way that respective travel applications are only required for an international trip.

**Problem** A process fragment shall only be executed if a particular condition is met.



**Implementation** This adaptation pattern could be implemented based on patterns AP1 (insert process fragment), AP11, and AP12 (add / remove control dependency) or based on change primitives.

**Related Patterns** *Embed Process Fragment in Loop (AP9)*

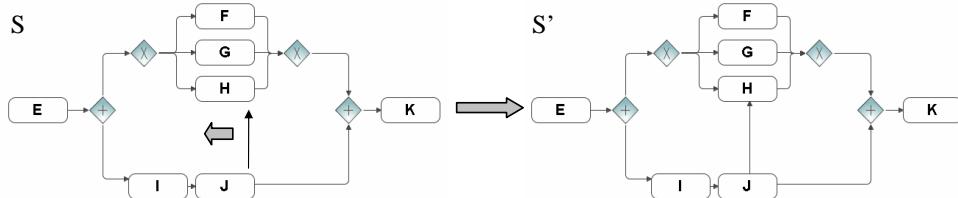
Fig. 17. *Embed Process Fragment in Conditional Branch (AP10)* pattern

### Pattern AP11: Add Control Dependency

**Description** An additional control edge (e.g., for synchronizing the execution order of two parallel activities) is added to process schema S.

**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 process schema S.



**Related Patterns** *Remove Control Dependency (AP12)*

Fig. 18. *Add Control Dependency (AP11)* pattern

pattern a control dependency and its attributes can be removed from a process schema (cf. Fig. 19). Similar considerations as for pattern AP11 can be made.

**Adaptation Pattern AP13: Update Condition.** This pattern allows users to update transition conditions in a process schema (cf. Fig. 20). Usually, an implementation of this pattern has to ensure that the new transition condition

### Pattern AP12: Remove Control Dependency

<b>Description</b>	A control edge is removed from process schema S.
<b>Example</b>	Assume that for a medical treatment procedure test A has to be finished before test B may be started. In an emergency situation, however, these two tests shall be performed in parallel in order to quickly treat the patient.
<b>Problem</b>	An existing control dependency is not needed anymore in process schema S.
<b>Related Patterns</b>	<i>Parallelize Process Fragments (AP9)</i>

Fig. 19. *Remove Control Dependency (AP12)* pattern

is correct in the context of the given process schema. For instance, it has to be ensured that all workflow relevant data elements, which the transition condition refers to, are present in the process schema.

### Pattern AP13: Update Condition

<b>Description</b>	A transition condition in the process schema is updated.
<b>Example</b>	In a loan approval process, currently, the manager has to approve a loan if the amount is larger than 50.000 Euro. Starting from January next year only loans above 100.000 Euros will have to be approved by the manager.
<b>Problem</b>	A transition condition has to be modified as it is no longer valid.
<b>Related Patterns</b>	<i>Embed Process Fragment in Loop (AP8), Embed Process Fragment in Conditional Branch (AP10)</i>

Fig. 20. *Update Condition (AP13)* pattern

**Adaptation Pattern AP14: Copy Process Fragment.** The *Copy Process Fragment* pattern (cf. Fig. 21) allows users to copy a process fragment. In contrast to pattern AP3 (*Move Process Fragment*) the respective process fragment is not removed from its initial position.

#### Pattern AP14: COPY Process Fragment

<b>Description</b>	A process fragment X is copied from its current position in process schema S to another position of the same schema S.
<b>Example</b>	In a reviewing process the papers to be reviewed are sent with the reviewing instruction to the respective reviewers after the submission phase has closed. As the reviewing instructions were erroneous they have to be re-sent to all reviewers.
<b>Problem</b>	A process fragment has to be executed once more.
<b>Design Choices (in addition to those described in Fig. 7)</b>	<p>C. How is the additional process fragment X embedded in the process schema?</p> <ol style="list-style-type: none"> <li>1. X is inserted between 2 directly succeeding activities (serial insert)</li> <li>2. X is inserted between 2 activity sets (insert between node sets)             <ol style="list-style-type: none"> <li>a) Without additional condition (parallel insert)</li> <li>b) With additional condition (conditional insert)</li> </ol> </li> </ol>
<b>Implementation</b>	This adaptation pattern can be implemented based on Pattern AP1 (insert process fragment) or by using change primitives.
<b>Related Patterns</b>	<i>Insert</i> adaptation pattern (AP1), <i>Move</i> adaptation pattern (AP5)

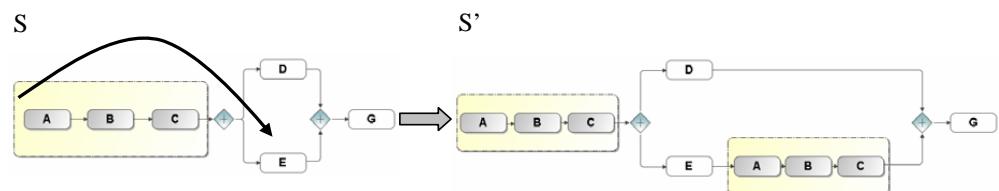


Fig. 21. *Copy Proces Fragment (AP14)* pattern

### 3.2 Patterns for Changes in Predefined Regions

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 or expanded. 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).

**Pattern for Predefined Change PP1: Late Selection of Process Fragments.** The *Late Selection of Process Fragments* pattern (cf. Fig. 22) allows selecting the implementation of a particular process activity at run-time either based on predefined rules or user decisions (Design Choice A). At build-time only a placeholder activity is provided, which is substituted by a concrete implementation (i.e., an atomic activity or sub process) during run-time (Design Choice B). The activity implementation is selected before the placeholder

activity is enabled or when it is enabled (Design Choice C).

**Pattern for Predefined Change PP2: Late Modeling of Process Fragments.** The *Late Modeling of Process Fragments* pattern (cf. Fig. 23) offers more freedom and allows modeling selected parts of the process schema at run-time. Design Choice A specifies, which building blocks can be used for late modeling. Building blocks can either be all process fragments from the repository (without any restrictions), a constraint-based subset of the fragments from the repository, or newly defined activities or process fragments. Design Choice B (cf. Fig. 7) describes whether the user may apply the same modeling constructs during build-time or whether more restrictions apply. Late modeling can take place upon instantiation of the process instance, when the placeholder activity is enabled, or when a particular state in the process is reached (Design Choice C). Depending on the pattern variant users start late modeling with an empty template or they take a predefined template as a starting point and adapt it as required (Design Choice D).

**Pattern for Predefined Change PP3: Late Composition of Process Fragments.** The *Late Composition of Process Fragments* pattern (cf. Fig. 24) enables the on-the fly composition of process fragments from the process repository, e.g., by dynamically introducing control dependencies between a predefined set of fragments. The *Interleaved Routing* pattern [79,55], which has been described as one of the workflow patterns, can be seen as a special implementation of PP3. It allows for the sequential execution of a set of activities, whereby the execution order is decided at run-time and each process fragment has to be executed exactly once. Like in PP3 decisions about the exact control flow are deferred to run-time. However, PP3 does not make any restrictions on how often a particular activity is executed.

**Pattern for Predefined Change PP4: Multi-Instance Activity.** The *Multi-Instance Activity* pattern does not only constitute a change pattern, but is a workflow pattern as well [67]. This pattern allows for the creation of multiple activity instances 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-instance activities 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 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. *Multi-Instance Activities* are considered as change patterns as their usage allows delaying the decision on the number of instances to be created for a particular activity to the run-time (cf. Fig. 25). A detailed description of this pattern can be found in [67].

Pattern PP1: Late Selection of Process Fragments	
<b>Description</b>	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.
<b>Example</b>	For the treatment of a particular patient one of several different subprocesses can be selected depending on the patient's disease.
<b>Problem</b>	There exist different implementations for an activity (including subprocesses), but for the selection of the respective implementation run-time information is required.
<b>Design Choices</b>	<p>A. How is the selection process done?</p> <ol style="list-style-type: none"> <li>1. Automatically based on predefined rules</li> <li>2. Manually by an authorized user</li> <li>3. Semi-automatically: options are reduced by applying some predefined rules; user can select among the remaining options</li> </ol> <p>B. What object can be selected?</p> <ol style="list-style-type: none"> <li>1. Atomic activity</li> <li>2. Sub process</li> </ol> <p>C. When does late selection take place?</p> <ol style="list-style-type: none"> <li>1. Before the placeholder activity is enabled</li> <li>2. When enabling the placeholder activity</li> </ol>
<b>Implementation</b>	By selecting the respective sub process or activity program, a reference to it is dynamically set and the sub-process or activity program is invoked.
<b>Related Patterns</b>	<i>Late Modeling of Process Fragments (PP2)</i>

Fig. 22. Late Selection of Process Fragments (PP1)

### 3.3 Complexity of Changes

In contrast to change primitives change patterns allow reducing cost of process adaptation. In the following we show exemplarily how the effects of applying

## Pattern PP2: Late Modeling of Fragments

<b>Description</b>	Parts of the process schema have not been defined at build-time, but are modeled during run-time for each process instance. 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.
<b>Example</b>	The exact treatment process of a particular patient is composed out of existing process fragments at run-time.
<b>Problem</b>	Not all parts of the process schema can be completely specified at build time.
<b>Design Choices</b>	<ul style="list-style-type: none"> <li>A. What are the basic building blocks for late modeling?             <ul style="list-style-type: none"> <li>1. All process fragments from the repository can be chosen.</li> <li>2. A constraint-based subset of the process fragments from the repository can be chosen.</li> <li>3. New activities or process fragments can be defined.</li> </ul> </li> <li>B. What is the degree of freedom regarding late modeling?             <ul style="list-style-type: none"> <li>1. Same modeling constructs and change patterns can be applied as for modeling at the process type level. Which of the adaptation patterns are supported within the placeholder activity is determined by the expressiveness of the modeling language.</li> <li>2. More restrictions apply than for modeling at the process type level.</li> </ul> </li> <li>C. When does late modeling take place?             <ul style="list-style-type: none"> <li>1. When a new process instance is created.</li> <li>2. When the placeholder activity is instantiated.</li> <li>3. When a particular state in the process (preceding the instantiation of the placeholder activity) is reached.</li> </ul> </li> <li>D. Does the modeling start from scratch?             <ul style="list-style-type: none"> <li>1. Starts with an empty template.</li> <li>2. Starts with a predefined template which can then be adapted.</li> </ul> </li> </ul>
<p><b>Pattern PP2</b></p>	
<b>Implementation</b>	After having modeled the placeholder activity with the editor, the fragment is stored in the repository and is then 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.
<b>Related Patterns</b>	<i>Late Selection of Process Fragments (PP1)</i>

Fig. 23. Late Modeling of Process Fragments(PP2)

a change pattern instead of respective change primitives can be determined.

We use the edit distance to measure the number of operations minimally required to transform a process schema  $S$  to a process schema  $S'$  [33]. Operations

### Pattern PP3: Late Composition of Process Fragments

<b>Description</b>	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.
<b>Example</b>	Several medical examinations can be applied for a particular patient. The exact examinations and the order in which they have to be performed are defined for each patient individually.
<b>Problem</b>	There exist several variants of how process fragments can be composed. In order to reduce the number of process variants to be specified by the process engineer during build time, process instances are dynamically composed out of fragments.

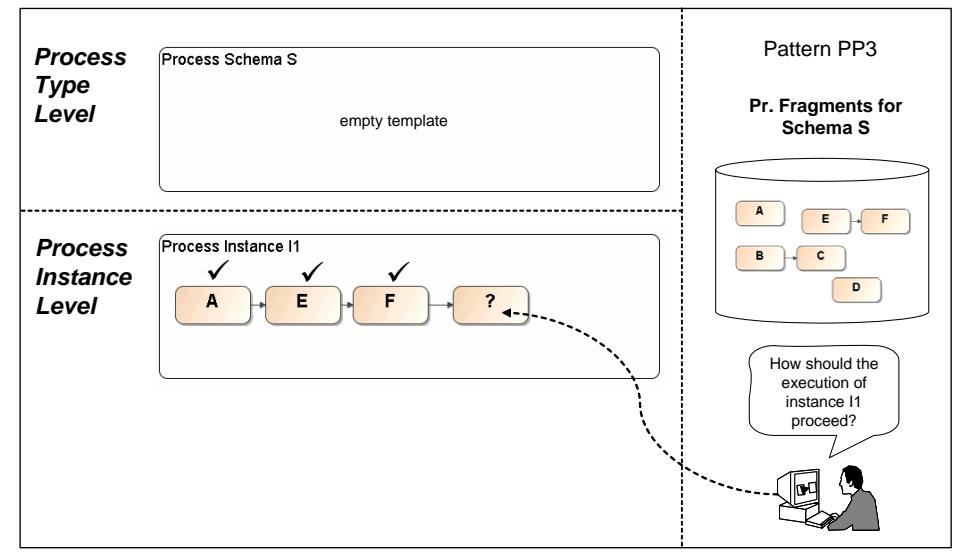


Fig. 24. Late Composition of Process Fragments (PP3)

can either be low-level change primitives (e.g., the insertion and deletion of nodes and edges) or high-level adaptation patterns. By calculating the edit distance the effects of transforming one process schema to another one can be evaluated. In general, when transforming schema  $S$  to  $S'$  using high-level change operations instead of change primitives the edit distance can be shortened (cf. Fig. 26). The effects of using adaptation patterns instead of change primitives can be calculated by comparing the edit distance for transforming  $S$  to  $S'$  with and without adaptation patterns support. Fig. 26 indicates that change patterns allow decreasing the complexity of change by providing high-level change operations. The original process model at the left hand side illustrates a process schema consisting of a single activity A. Assume that a process change should be accomplished inserting an additional activity B in parallel to activity A. Change pattern AP1 (Insert Process Fragment) (Design Choice B2) provides the high-level change operation `parallelInsert (S, B, A, A)` which allows users to insert activity B parallel to A in process schema S. Applying the respective change pattern the user just has to specify a couple of parameters. Internally, this requires 9 change primitives. A new parallel

#### Pattern PP4: Multi Instance Activity

<b>Description</b>	This pattern allows for the creation of multi instances of the respective activity during run-time.
<b>Example</b>	The scanning activity has to be repeated until all parcels are scanned. The number of parcels is not known at build-time.
<b>Problem</b>	A particular activity has to be executed several times. The number of instances that has to be created is not known at build-time.

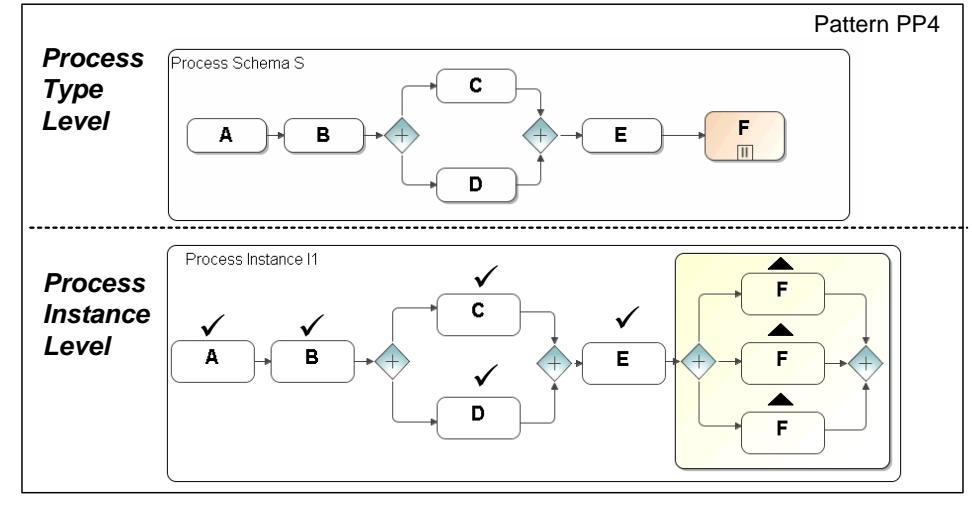


Fig. 25. Multi-Instance Activity (PP4)

block as well as activity B have to be added to the process schema: `Add(Node (B))`, `Add(AND-Split)` and `Add(AND-JOIN)`. Further, both edges of the original schema have to be moved: `MoveEdge((Start, A), (Start, AND-Split))` and `MoveEdge((A, End), (AND-Join, End))`. Finally, 4 new edges have to be added to the new schema: `AddEdge(AND-Split, A)`, `AddEdge(AND-Split, B)`, `AddEdge(A, AND-Join)` and `AddEdge(B, AND-Join)`. In this example, the transformation of the original process schema into the new version requires 9 change primitives resulting in an edit distance of 9. Using adaptation pattern *Insert Process Fragment*, from the perspective of the user, 8 operations can be saved as the entire transformation can be accomplished with 1 high-level change operation.

Fig. 27 depicts the edit distance for selected adaptation patterns and process schemes respectively. This is done exemplarily for ADEPT2 [45] and YAWL [12]. Depending on the structure of process schema  $S$  the implementation of an adaptation pattern with change primitives can result in different edit distances. The edit distance also depends on the used process meta model and the used tool. Note that in this paper we only consider control flow. If data dependencies have to be taken into account as well, the benefits of employing high-level change operations will become even greater [43].

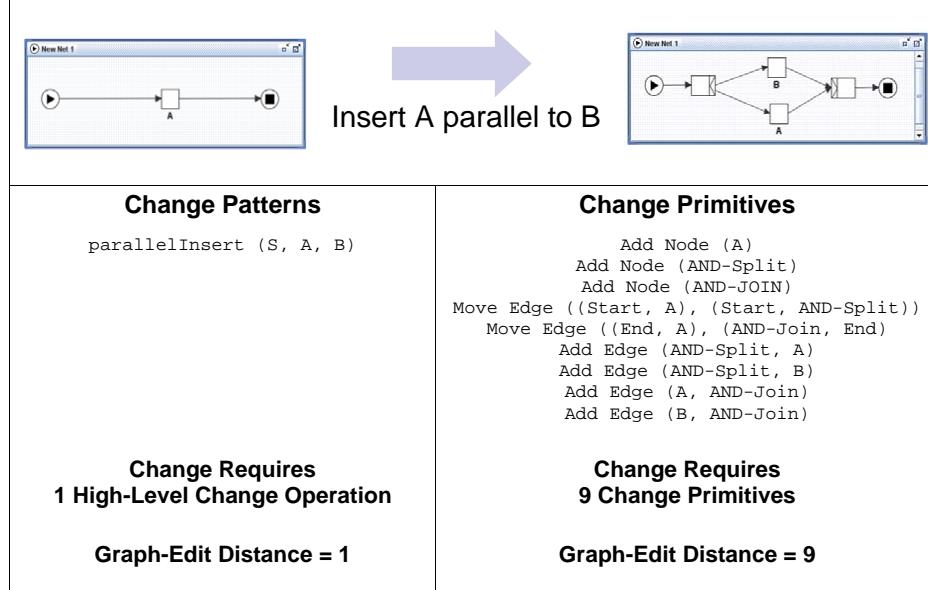


Fig. 26. Change Patterns versus Change Primitives

Although the graph-edit distance does not allow for quantifying how much time is needed to accomplish a respective change it allows assessing its complexity. For both process type and instance level changes, patterns speed up the change process by reducing the efforts for conducting a respective change through raising the level of abstraction. In case of ad-hoc changes not only cost of change can be reduced, but changes also become more applicable for end users as complexity of change is hidden from them.

#### 4 Semantics of Change Patterns

To ground pattern implementation as well as pattern-based analysis of PAIS on a solid basis we provide a *formal semantics* for adaptation patterns (AP1 – AP14). Furthermore, we discuss the semantics of patterns for changes in predefined regions (PP1 – PP4).

##### 4.1 Basic Notions

First of all, we introduce basic formal notions needed for the following considerations. In workflow literature, for example, the formal description of control flow patterns has been based on Petri Nets [67]. Therefore these patterns have an inherent formal semantics. Regarding change patterns, in turn, we aim at a formal description which is independent of a particular process meta model (e.g., Workflow Nets [65], WSMNets [48], or Activity Nets [28]). To achieve

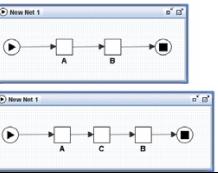
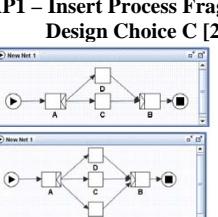
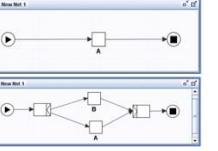
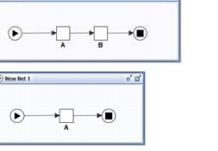
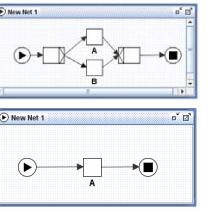
Cost of Change for Selected Scenarios		
Transformation S to S'	ADEPT2	YAWL
<b>AP1 – Insert Process Fragment Design Choice C[1]</b> 	1 Change Pattern SerialInsert(S, C, A, B)	3 Change Primitives AddNode(A) MoveEdge((A,B), (C,B)) AddEdge(A,C)
<b>AP1 – Insert Process Fragment Design Choice C [2a]</b> 	1 Change Pattern ParallelInsert(S, D, A, B)	3 Change Primitives AddNode(E), AddEdge(A,E), AddEdge(E,B)
<b>AP1 – Insert Process Fragment Design Choice C [2a]</b> 	1 Change Pattern ParallelInsert(S, D, A, B)	9 Change Primitives AddNode(B), AddNode(AND-Split), AddNode(AND-Join), MoveEdge((Start, A), (Start, AND-Split)), MoveEdge((A, End), (AND-Join, End)), AddEdge(AND-Split, A), AddEdge(AND-Split, B), AddEdge(A, AND-Join), AddEdge(B, AND-Join)
<b>AP2 – Delete Process Fragment</b> 	1 Change Pattern Delete(S, B)	2 Change Primitives RemoveNode(B) AddEdge(A, End)  (edges are deleted automatically)
<b>AP2 – Delete Process Fragment</b> 	1 Change Pattern Delete(S, C)	3 Change Primitives RemoveNode(B), ChangeSplitToNon(XOR-Split), ChangeJoinToNon(XOR-Join)  Remark: ChangeSplitToNon and ChangeJoinToNon constitute additional change primitives as being supported by YAWL

Fig. 27. Cost of Change - Comparison

this, we base the formal description of change patterns on the behavioral semantics of the modified process schema before and after its change. One way to capture behavioral semantics is to use execution traces [64]. For this purpose, first of all, we provide some preliminary definitions.

**Definition 1 (Execution Trace)** Let  $\mathcal{PS}$  be the set of all process schemes and let  $\mathcal{A}$  be the total set of activities (or more precisely activity labels) based on which process schemes  $S \in \mathcal{PS}$  are specified (without loss of generality we assume unique labeling of activities in the given context). Let further  $\mathcal{Q}_S$

denote the set of all possible execution traces producible on process schema  $S \in \mathcal{PS}$ . A particular trace  $\sigma \in \mathcal{Q}_S$  is then defined as  $\sigma = < a_1, \dots, a_k >$  (with  $a_i \in \mathcal{A}$ ,  $i = 1, \dots, k$ ,  $k \in \mathbb{N}$ ) where the temporal order of  $a_i$  in  $\sigma$  reflects the order in which activities  $a_i$  were completed over  $S^1$ .

Furthermore, we define the following two functions:

- $\text{tracePred}(S, a, \sigma)$  is a function which returns all activities within process schema  $S$  completed before the first occurrence of activity  $a$  within trace  $\sigma$ . Formally:  $\text{tracePred}: \mathcal{S} \times \mathcal{A} \times \mathcal{Q}_S \mapsto 2^{\mathcal{A}}$  with

$$\text{tracePred}(S, a, \sigma) = \begin{cases} \emptyset & \text{if } a \notin \{\sigma(i) \mid i \leq |\sigma|\} \\ & (\sigma(i) \text{ denotes the } i^{\text{th}} \text{ item in } \sigma, \text{ cf. Fig. 29}) \\ \{a_1, \dots, a_k\} & \text{if } \sigma = < a_1, \dots, a_k, a, a_{k+1}, \dots, a_n > \\ & \wedge a_j \neq a \forall j = 1, \dots, k \end{cases}$$

- Analogously,  $\text{traceSucc}(S, a, \sigma)$  denotes a function which returns all activities within process schema  $S$  completed after the last occurrence of activity  $a$  in trace  $\sigma$ . Formally:  $\text{traceSucc}: \mathcal{S} \times \mathcal{A} \times \mathcal{Q}_S \mapsto 2^{\mathcal{A}}$  with

$$\text{traceSucc}(S, a, \sigma) = \begin{cases} \emptyset & \text{if } a \notin \{\sigma(i) \mid i \leq |\sigma|\} \\ \{a_{k+1}, \dots, a_n\} & \text{if } \sigma = < a_1, \dots, a_k, a, a_{k+1}, \dots, a_n > \\ & \wedge a_j \neq a \forall j = k + 1, \dots, n \end{cases}$$

Function  $\text{tracePred}$  ( $\text{traceSucc}$ ) determines the predecessors (successors) of the first (last) occurrence of a certain activity within an execution trace; i.e., those activities which precede (succeed) the considered activity due to a loop back are not taken into consideration. Fig. 28 shows a process schema with two loops, an example of a corresponding execution trace, and the sets resulting from the application of functions  $\text{tracePred}$  and  $\text{traceSucc}$  in different contexts.

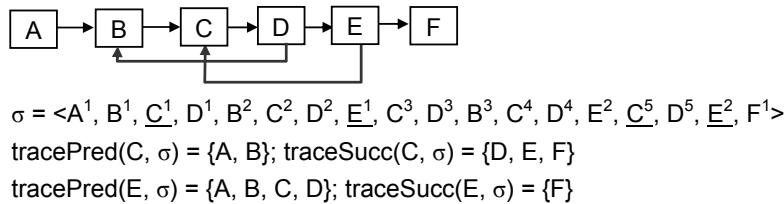


Fig. 28. Functions  $\text{tracePred}$  and  $\text{traceSucc}$  applied to execution trace

---

<sup>1</sup> A particular activity can occur multiple times within a trace due to loopbacks.

In addition to Definition 1, Fig. 29 contains useful notions which facilitate the formalization of the change patterns in the sequel.

Let  $\sigma = \langle a_1, \dots, a_n \rangle \in \mathcal{Q}_S$  be an execution trace on process schema  $S$ . Then:

---

$ \sigma $ : cardinality of $\sigma$
$\sigma(i) = a_i$ : $i^{th}$ item in trace $\sigma$
$x \in \sigma \iff \exists i \leq  \sigma  \text{ with } \sigma(i) = x$
$B \subseteq \sigma \iff \forall b \in B: b \in \sigma$
$\sigma_X^- \rightarrow$ discard all items from $\sigma$ which belong to set $X$
Example: $\sigma_{\{a_1, a_n\}}^- = \langle a_2, \dots, a_{n-1} \rangle$
$\sigma_X^+ \rightarrow$ discard all items from $\sigma$ not belonging to set $X$
example: $\sigma_{\{a_1, a_n\}}^+ = \langle a_1, a_n \rangle$

---

Fig. 29. Useful notions based on Def. 1

## 4.2 Adaptation Pattern Semantics

Based on the meta model independent notions given in Definition 1 and Table 29 we now describe the formal semantics of the different adaptation patterns (cf. Fig. 30 – Fig. 35). Note that the given formal specifications do not contain any constraints which are specific for a particular meta model (e.g., preserving the block-structure for BPEL flows or WSM nets [49] after changes). This has to be achieved separately by, for example, associating change operations with meta model-specific formal pre- and post-conditions to be met when applying them to a particular process schema. The specifications given in the following contain generally valid pre-conditions where necessary. For example, a node can only be deleted if it is present in the original process schema (cf. AP2; Delete Activity). In the following we explain the formal semantics of selected change patterns. A complete formalization is given in Fig. 30 – 35.

### 4.2.1 Discussion of Design Choices

In the following, we abstract from design choices A and B (cf. Fig. 7) since they do not affect the formal semantics of the change patterns as defined by us.

- Regarding design choice A (Pattern Scope – instance / type level), for example, adaptation pattern AP2 (Delete process fragment) could be imple-

---

**Formalization Context:** Without loss of generality we restrict the scope of our formalization to adaptation patterns being applied to (atomic) activities and assume unique labelling of activities within a process schema. Let  $\mathcal{Q}_S$  denote the set of all execution traces producible on process schema S. Let further  $\text{op}$  be an adaptation pattern transforming process schema S into S'. Finally, let  $x$  denote the activity being manipulated (e.g., inserted, deleted, moved, or replaced) due to the application of  $\text{op}$ . For pattern formalization we use the notions introduced in Definition 1 and Table 29.

---

AP1: Insert Activity	<p><i>Preliminaries:</i> <math>\text{op} = \text{Insert}(S, x, A, B) \mapsto S'</math> where A and B denote activity sets between which x shall be inserted</p> <p><i>Semantics:</i></p> <ul style="list-style-type: none"> <li>(1) Process schema S does not contain a node with label x. Process schema S contains activity sets A and B.</li> <li>(2) <math>\forall \mu \in \mathcal{Q}_{S'}: \exists \sigma \in \mathcal{Q}_S</math> with <math>\mu_{\{x\}}^- = \sigma</math> and vice versa</li> <li>(3) <math>\forall \mu \in \mathcal{Q}_{S'}</math> with <math>A \subseteq \mu</math> (i.e., all nodes of A contained in <math>\mu</math>):</li> </ul>
Considered design choices C[1,2a]: (Serial Insert, Parallel Insert)	$\{\mu_{A \cup B \cup \{x\}}^+ (i) \mid i = \nu, \dots, \nu +  A  - 1\} = A \text{ for } \nu \in \mathbb{N}$ $\implies$ $\mu_{A \cup B \cup \{x\}}^+ (\nu +  A ) = x \wedge$ $\{\mu_{A \cup B \cup \{x\}}^+ (i) \mid i = \nu +  A  + 1, \dots, \nu +  A  +  B \} = B$
Considered design choice C[2b]: (Conditional Insert)	$(3') \forall \mu \in \mathcal{Q}_{S'} \text{ with } x \in \mu:$ $\mu_{A \cup B \cup \{x\}}^+ (\nu +  A ) = x \implies$ $\{\mu_{A \cup B \cup \{x\}}^+ (i) \mid i = \nu, \dots, \nu +  A  - 1\} = A \wedge$ $\{\mu_{A \cup B \cup \{x\}}^+ (i) \mid i = \nu +  A  + 1, \dots, \nu +  A  +  B \} = B$
AP3: Move Activity	<p><i>Preliminaries:</i> <math>\text{op} = \text{Move}(S, x, A, B) \mapsto S'</math> where A and B denote the activity sets between which x is moved from its current position in schema S.</p> <p><i>Semantics:</i></p> <ul style="list-style-type: none"> <li>(1) Process schema S contains a node with label x. Process schema S contains activity sets A and B.</li> <li>(2) <math>\forall \sigma \in \mathcal{Q}_S: \exists \mu \in \mathcal{Q}_{S'}</math> with <math>\sigma_{\{x\}}^- = \mu_{\{x\}}^-</math> and vice versa</li> </ul>
Considered design choices C[1,2a] (Serial Insert, Parallel Insert):	Condition (3) of AP1 (Insert Activity)
Considered design choice C[2b] (Conditional Insert):	Condition (3') of AP1 (Insert Activity)
AP14: Copy Activity	<p><i>Preliminaries:</i> <math>\text{op} = \text{Copy}(S, x, x', A, B) \mapsto S'</math> where A and B denote the activity sets between which the copied activity x is inserted and <math>x'</math> denotes the new label for this activity (note that we assume unique labelling).</p> <p><i>Semantics:</i></p> <ul style="list-style-type: none"> <li>(1) Process schema S contains a node with label x and no node with label <math>x'</math>. Process schema S contains activity sets A and B.</li> <li>(2) <math>\forall \mu \in \mathcal{Q}_{S'}: \exists \sigma \in \mathcal{Q}_S</math> with <math>\mu_{\{x'\}}^- = \sigma</math> and vice versa</li> </ul>
Considered design choices C[1,2a] (Serial Insert, Parallel Insert):	Condition (3) of AP1 (Insert Activity)
Considered design choice C[2b] (Conditional Insert):	Condition (3') of AP1 (Insert Activity)

---

Fig. 30. Semantics of Adaptation Patterns AP1, AP3, and AP14 (Group 1)

---

**Formalization Context:** see Fig. 30

---

AP2: Delete Activity	<i>Preliminaries:</i> $\text{op} = \text{Delete}(S, x) \mapsto S'$ <i>Semantics:</i> (1) Process schema $S$ contains one node with label $x$ . (2) $\forall \mu \in \mathcal{Q}_{S'}: x \notin \mu$ (3) $\forall \mu \in \mathcal{Q}_{S'}: \exists \sigma \in \mathcal{Q}_S$ with $\mu = \sigma_{\{x\}}^- \wedge \forall \sigma \in \mathcal{Q}_S: \exists \mu \in \mathcal{Q}_{S'}$ with $\sigma_{\{x\}}^- = \mu$
----------------------	--

---

Fig. 31. Semantics of Adaptation Pattern AP2 (Group 2)

---

**Formalization Context:** see Fig. 30

---

AP4: Replace Activity	<i>Preliminaries:</i> $\text{op} = \text{Replace}(S, x, y) \mapsto S'$ <i>Semantics:</i> (1) Process schema $S$ contains a node with label $x$ , but does not contain any node labelled $y$ . (2) $\forall \sigma \in \mathcal{Q}_S: \exists \mu \in \mathcal{Q}_{S'}$ with $ \sigma  =  \mu  \wedge$ $\mu(i) = \begin{cases} \sigma(i) & \text{if } \sigma(i) \neq x \\ y & \text{if } \sigma(i) = x \end{cases}$ and vice versa <i>Alternatively:</i> (2') $\forall \sigma \in \mathcal{Q}_S: \exists \mu \in \mathcal{Q}_{S'}$ with $ \sigma  =  \mu  \wedge \sigma_{\{x\}}^- = \mu_{\{y\}}^-$ $\wedge (\sigma(k) = x \text{ for } k \implies \mu(k) = y)$ and vice versa
AP5: Swap Activities	<i>Preliminaries:</i> $\text{op} = \text{Swap}(S, x, y) \mapsto S'$ <i>Semantics:</i> (1) Process schema $S$ contains one node with label $x$ and one node with label $y$ . (2) $\forall \sigma \in \mathcal{Q}_S: \exists \mu \in \mathcal{Q}_{S'}$ with $ \sigma  =  \mu  \wedge$ $\mu(i) = \begin{cases} \sigma(i) & \text{if } \sigma(i) \notin \{x, y\} \\ x & \text{if } \sigma(i) = y \\ y & \text{if } \sigma(i) = x \end{cases}$ and vice versa <i>Alternatively:</i> (2') $\forall \sigma \in \mathcal{Q}_S: \exists \mu \in \mathcal{Q}_{S'}$ with $ \sigma  =  \mu  \wedge \sigma_{\{x,y\}}^- = \mu_{\{x,y\}}^-$ $\wedge (\sigma(k) = x \implies \mu(k) = y)$ and vice versa

---

Fig. 32. Semantics of Adaptation Patterns AP4 and AP5 (Group 3)

mented in a different way for the process type and the process instance level; e.g., replacing the activity to be deleted by a silent activity at instance level,

---

**Formalization Context:** see Fig. 30

AP8: Embed Process Fragment in Loop	<p><i>Preliminaries:</i> <math>\text{op} = \text{Embed\_in\_Loop}(S, P, \text{cond}) \mapsto S'</math> where <math>P</math> denotes the set of activities to be embedded into a loop and <math>\text{cond}</math> denotes the loop backward condition.</p> <p><i>Semantics:</i></p> <ol style="list-style-type: none"> <li>(1) The sub graph on <math>P</math> induced by <math>S</math> has to be connected and must be a hammock, i.e., have single entry and single exit node.</li> <li>(2) <math>\mathcal{Q}_S \subset \mathcal{Q}_{S'}</math></li> <li>(3) <math>\forall \mu \in \mathcal{Q}_{S'}: \text{Let } \mu' \text{ be the execution trace produced by discarding all entries of activities in } P \text{ (if existing) from } \mu \text{ except the entries of one arbitrary loop iteration over } P. \text{ Then } \mu' \in \mathcal{Q}_S \text{ holds.}</math></li> </ol>
AP10: Embed Process Fragment in Conditional Branch	<p><i>Preliminaries:</i> <math>\text{op} = \text{Embed\_in\_Cond\_Branch}(S, P, \text{cond}) \mapsto S'</math> where <math>P</math> denotes the set of activities to be embedded into a conditional branch and <math>\text{cond}</math> denotes the associated condition.</p> <p><i>Semantics:</i></p> <ol style="list-style-type: none"> <li>(1) The sub graph on <math>P</math> induced by <math>S</math> has to be connected and must be a hammock, i.e., have single entry and single exit node.</li> <li>(2) <math>\mathcal{Q}_{S'} \subseteq \mathcal{Q}_S</math></li> <li>(3) <math>\forall \sigma \in \mathcal{Q}_S: \sigma_P^- \in \mathcal{Q}_{S'} \text{ (if } \text{cond} = \text{FALSE} \text{ is possible)}</math></li> </ol>

---

Fig. 33. Semantics of Adaptation Patterns AP8 and AP10 (Group 4)

while physically deleting it at type level. Furthermore, the applicability of change patterns at instance level additionally depends on the state of the respective instances [50]. This, however, does not influence the formal semantics of pattern AP2 when defining it on basis of execution traces.

- Regarding design choice B (Is the change pattern applied to an atomic activity, sub process, or hammock?), we assume that sub processes as well as hammocks<sup>2</sup> can be encapsulated within a complex activity. Then the formal semantics defined for the application of adaptation patterns to activities can be easily transferred to design choices B[2] and B[3] as well (cf. Fig. 7). Thus, in the following, we refer to activities instead of process fragments.

In summary, design choices A and B are common for all adaptation patterns, but we can abstract from them in most cases when defining a formal pattern semantics. Design choices relevant in the context of particular patterns, however, do influence the formal semantics of these patterns. Therefore, we take respective design choices into account when formalizing patterns.

For patterns AP6 (Extract Sub Process), AP7 (Inline Sub Process), and AP8 (Embed Process Fragment in Loop), design choice B is of importance since AP6 and AP7 are applied to sub processes and AP8 to process fragments respectively.

---

<sup>2</sup> A hammock refers to a sub graph with single "entry" and single "exit" node.

---

**Formalization Context:** see Fig. 30

AP9: Parallelize Activities	<p><i>Preliminaries:</i> <math>\text{op} = \text{Parallelize}(S, P) \mapsto S'</math> where <math>P</math> denotes the set of activities to be parallelized.</p> <p><i>Semantics:</i></p> <ul style="list-style-type: none"> <li>(1) Within schema <math>S</math>, the sub graph induced by <math>P</math> constitutes a sequence with single entry and single exit node.</li> <li>(2) <math>\forall \sigma \in Q_S : \exists \mu \in Q_{S'} \text{ with } \sigma = \mu</math> (i.e., <math>Q_S \subset Q'_{S'}</math>)</li> <li>(3) <math>\forall p, p' \in P : \exists \mu_1, \mu_2 \in Q_{S'}</math> with  <math display="block">(p \in \text{tracePred}(S', p', \mu_1) \wedge p' \in \text{tracePred}(S', p, \mu_2))</math> <math display="block">(\text{assuming that the sequence defined by } P \text{ can be enabled in } S)</math> </li> </ul>
AP11: Add Control Dependency	<p><i>Preliminaries:</i> <math>\text{op} = \text{Add\_Ctrl\_Dependeny}(S, x, y) \mapsto S'</math></p> <p><i>Semantics:</i></p> <ul style="list-style-type: none"> <li>(1) Process schema <math>S</math> contains one node with label <math>x</math> and one node with label <math>y</math> and there is no control dependency between <math>x</math> and <math>y</math>.</li> <li>(2) <math>\forall \mu \in Q_{S'} : \exists \sigma \in Q_S \text{ with } \mu = \sigma</math> (i.e., <math>Q_{S'} \subset Q_S</math>)</li> <li>(3) <math>\forall \mu \in Q_{S'} \text{ with } \{x, y\} \subseteq \mu : x \in \text{tracePred}(S', y, \mu)</math>  <math display="block">(\text{i.e., } x \text{ always precedes } y \text{ in } S')</math> </li> </ul>
AP12: Remove Control Dependency	<p><i>Preliminaries:</i> <math>\text{op} = \text{Remove\_Ctrl\_Dependeny}(S, x, y) \mapsto S'</math></p> <p><i>Semantics:</i></p> <ul style="list-style-type: none"> <li>(1) Schema <math>S</math> contains one node with label <math>x</math> and one node with label <math>y</math> and there exists a control dependency <math>x \rightarrow y</math>.</li> <li>(2) <math>\forall \sigma \in Q_S : \exists \mu \in Q_{S'} \text{ with } \sigma = \mu</math> (i.e., <math>Q_S \subset Q'_{S'}</math>)</li> <li>(3) <math>\forall \mu \in Q_{S'} \text{ with } \{x, y\} \subseteq \mu :</math>  <math display="block">(x \in \text{tracePred}(S', y, \mu) \vee y \in \text{tracePred}(S', x, \mu))</math> <math display="block">(\text{i.e., } x \text{ and } y \text{ can be executed in parallel in } S')</math> </li> </ul>
AP13: Update Condition	<p><i>Preliminaries:</i> <math>\text{op} = \text{Update\_Ctrl\_Dependeny}(S, x, y, \text{newCond}) \mapsto S'</math> where <math>\text{newCond}</math> denotes the (transition) condition of control edge <math>x \rightarrow y</math> in <math>S'</math> after update.</p> <p><i>Semantics:</i></p> <p>The semantics of <math>\text{op}</math> depends on the relation between the old condition <math>\text{oldCond}</math> (on <math>S</math>) and the updated one <math>\text{newCond}</math> (on <math>S'</math>):</p> <ul style="list-style-type: none"> <li>(1) <math>\text{oldCond} \implies \text{newCond}</math>: <math>\forall \mu \in Q_{S'} \text{ for which transition condition newCond evaluates to TRUE: } \exists \sigma \in Q_S \text{ with } \mu = \sigma</math></li> <li>(2) <math>\text{newCond} \implies \text{oldCond}</math>: <math>\forall \sigma \in Q_S \text{ for which transition condition oldCond evaluates to TRUE: } \exists \mu \in Q_{S'} \text{ with } \mu = \sigma</math></li> <li>(3) Otherwise, for all traces <math>\sigma \in Q_S</math> there exists a trace <math>\mu \in Q_{S'}</math> for which the following holds: If we produce projections for <math>\sigma</math> and <math>\mu</math> by discarding all entries which belong to the conditional branch with the updated condition, these projections are equal.</li> </ul>

---

Fig. 34. Semantics of Adaptation Patterns AP9, AP11, AP12, and AP13 (Group 5)

#### 4.2.2 Formal Semantics of Adaptation Patterns

In the following we explain the formal semantics of the 14 adaptation patterns AP1 – AP14 as presented in Fig. 30 – 35. For this, we group adaptation patterns with similar or related formalization. As example, take the first group consisting of adaptation patterns AP1 (Insert Activity), AP3 (Move Activity),

---

**Formalization Context:** see Fig. 30

AP6: Extract Sub Process	<p><i>Preliminaries:</i> <math>\text{op} = \text{Extract}(S, P, x) \mapsto S'</math> where <math>P</math> denotes the set of activities to be extracted and <math>x</math> denotes the label of the activity which substitutes the sub graph induced by <math>P</math> on <math>S</math>.</p> <p><i>Semantics:</i></p> <ol style="list-style-type: none"> <li>(1) The sub graph on <math>S</math> induced by <math>P</math> has to be connected and must be a hammock, i.e., have single entry and single exit node.</li> <li>(2) <math>\forall \sigma \in \mathcal{Q}_S: \exists \mu \in \mathcal{Q}_{S'} \text{ with } \mu_{\{x\}}^- = \sigma_P^- \wedge \forall \mu \in \mathcal{Q}_{S'}: \exists \sigma \in \mathcal{Q}_S: \sigma_P^- = \mu_{\{x\}}^-</math></li> <li>(3) Let <math>z</math> denote the single exit node of the sub graph induced by <math>P</math>. Then: <math>\forall \sigma \in \mathcal{Q}_S \text{ with } \sigma_{P \setminus \{z\}}^-(k) = z: \exists \mu \in \mathcal{Q}_{S'} \text{ with } \mu(k) = x</math></li> <li>(4) Let <math>\mathcal{P}</math> denote the set of all execution traces over the sub graph induced by <math>P</math> and let further <math>\pi \in \mathcal{P}</math>. Then:  <math>\forall \mu \in \mathcal{Q}_{S'} \text{ with } \mu(\nu_i) = x \ (i = 1, \dots, n, \nu_i \in \mathbb{N}):</math>  <math>\exists \sigma \in \mathcal{Q}_S \text{ with }</math> <math display="block">\sigma(k) = \begin{cases} = \mu(k) &amp; k = 1, \dots, \nu_1 - 1, \\ = \mu(k - j *  \pi  + j) &amp; k = \nu_j +  \pi , \dots, \nu_{j+1} - 1 \wedge \\ &amp; k = \nu_n +  \pi , \dots,  \mu  + n * ( \pi  - 1) \\ = \pi(l) &amp; k = \nu_i + l - 1 \end{cases}</math> <p>where <math>j = 1, \dots, n-1; l = 1, \dots,  \pi </math></p> </li> </ol>
--------------------------	---

---

AP7: Inline Sub Process	<p><i>Preliminaries:</i> <math>\text{op} = \text{Inline}(S, x, P) \mapsto S'</math> where <math>P</math> denotes the set of activities to be inlined into <math>S'</math> and <math>x</math> denotes the label of the activity which substitutes the sub graph induced by <math>P</math> on <math>S</math>.</p> <p><i>Semantics:</i></p> <ol style="list-style-type: none"> <li>(1) The sub graph on <math>S</math> induced by <math>P</math> has to be connected and must be a hammock, i.e., have single entry and single exit node.</li> <li>(2) <math>\forall \sigma \in \mathcal{Q}_S: \exists \mu \in \mathcal{Q}_{S'} \text{ with } \mu_{\{P\}}^- = \sigma_x^- \wedge \forall \mu \in \mathcal{Q}_{S'}: \exists \sigma \in \mathcal{Q}_S: \mu_{\{x\}}^- = \sigma_P^-</math></li> <li>(3) Let <math>z</math> denote the single exit node of the sub graph induced by <math>P</math>. Then: <math>\forall \mu \in \mathcal{Q}_{S'} \text{ with } \mu_{P \setminus \{z\}}^-(k) = z: \exists \sigma \in \mathcal{Q}_S \text{ with } \sigma(k) = x</math></li> <li>(4) Let <math>\mathcal{P}</math> denote the set of all execution traces over the sub graph induced by <math>P</math> and let further <math>\pi \in \mathcal{P}</math>. Then:  <math>\forall \sigma \in \mathcal{Q}_S \text{ with } \sigma(\nu_i) = x \ (i = 1, \dots, n, \nu_i \in \mathbb{N}):</math>  <math>\exists \mu \in \mathcal{Q}_{S'} \text{ which can be built according to the construction of } \sigma \text{ in condition (4) for pattern AP6 (Extract Sub Process).}</math> </li> </ol>
-------------------------	--

---

Fig. 35. Semantics of Adaptation Patterns AP6 and AP7 (Group 6)

and AP14 (Copy Activity) which are all (more or less) based on the insertion of an activity at a certain position (activity insertion for AP1, activity deletion and insertion for AP3, and activity insertion with re-labelling for AP14).

*Group 1: Patterns AP1 (Insert Activity), AP3 (Move Activity), and AP14 (Copy Activity):* The basic adaptation pattern for this group is AP1:  $\text{op} =$

$\text{Insert}(S, x, A, B) \mapsto S'$  where A and B denote activity sets between which x shall be inserted. Then, formal pattern semantics is as follows (see Fig. 30):

- (1) Process schema  $S$  does not contain a node with label  $x$ . Further,  $S$  contains activity sets  $A$  and  $B$ .
- (2)  $\forall \mu \in \mathcal{Q}_{S'}: \exists \sigma \in \mathcal{Q}_S$  with  $\mu_{\{x\}}^- = \sigma$  and vice versa

Considered design choices C[1,2a] (Serial Insert, Parallel Insert):

$$\begin{aligned}
 (3) \forall \mu \in \mathcal{Q}_{S'} \text{ with } A \subseteq \mu & (\text{i.e., all nodes of } A \text{ contained in } \mu): \\
 \{\mu_{A \cup B \cup \{x\}}^+(i) \mid i = \nu, \dots, \nu + |A| - 1\} = A \text{ for } \nu \in \mathbb{N} \\
 \implies \mu_{A \cup B \cup \{x\}}^+(\nu + |A|) = x \wedge \\
 \{\mu_{A \cup B \cup \{x\}}^+(i) \mid i = \nu + |A| + 1, \dots, \nu + |A| + |B|\} = B
 \end{aligned}$$

When formalizing the semantics of an adaptation pattern, first of all, we present necessary preconditions for the application of the particular pattern (as far as these constraints are meta model independent). Then we describe the effects resulting from the application of the pattern. To stay independent of a certain meta model, the latter is accomplished based on execution traces; i.e., we describe the relation between traces producible on schema  $S$  and on schema  $S'$  (resulting from the application of the change pattern to  $S$ ).

Regarding pattern AP1, (1) formalizes generic pre-conditions for inserting an activity; e.g., the activity to be inserted must not yet be present in  $S$ . (2) defines the relation between execution traces on old schema  $S$  and new schema  $S'$ . For each execution trace  $\sigma$  on  $S$  there exists a corresponding execution trace  $\mu$  on  $S'$  for which  $\mu_{\{x\}}^- = \sigma$  holds (i.e., when discarding newly inserted activity  $x$  from  $\mu$ , this trace equals  $\sigma$  (and vice versa)). This expresses the close relation between traces producible on  $S$  and  $S'$ . It further indicates that execution traces on  $S'$  may additionally contain  $x$  whereas those on  $S$  do not. Regarding the position of newly inserted activity  $x$ , for design choice C[1] (Serial Insert) (cf. Fig. 8), predecessor set  $A$  and successor set  $B$  between which the new activity is inserted contain exactly one activity. As can be seen from Fig. 30, we present different formalizations for design choices C[1,2a] (Serial and Parallel Insert) and C[2b] (Conditional Insert).

For design choice C[1,2a] (Serial and Parallel Insert) the following conditions for newly inserted activity  $x$  in traces  $\mu$  on  $S'$  hold: If all nodes of predecessor set  $A$  are contained in a trace  $\mu$  on  $S'$ , then also the entries of  $x$  and  $B$  are present in  $\mu$ . When projecting  $\mu$  onto the entries of activity set  $A \cup B \cup \{x\}$  (i.e.,  $\mu_{A \cup B \cup \{x\}}^+$ ), then the entry of  $x$  is positioned directly after all entries of  $A$  and the entries of  $B$  directly succeed the entry of  $x$ . This condition has to be modified in case of a conditional insert, since the presence of entries of  $A$  in  $\mu$  on  $S'$  does not imply the presence of  $x$ . Thus, the respective modification of condition (3) turns out as follows:

Considered design choice C[2b] (Conditional Insert):

(3')  $\forall \mu \in \mathcal{Q}_{S'} \text{ with } x \in \mu:$

$$\mu_{A \cup B \cup \{x\}}^+(\nu + |A|) = x \\ \implies$$

$$\{\mu_{A \cup B \cup \{x\}}^+(i) \mid i = \nu, \dots, \nu + |A| - 1\} = A \wedge \\ \{\mu_{A \cup B \cup \{x\}}^+(i) \mid i = \nu + |A| + 1, \dots, \nu + |A| + |B|\} = B$$

This condition implies that if  $x$  is present in  $\mu$  on  $S'$  then also the entries of predecessor set  $A$  and successor set  $B$  are contained in  $\mu$  on  $S'$ . Furthermore, based on the projection of  $\mu$  onto the entries of activity set  $A \cup B \cup \{x\}$  (i.e.,  $\mu_{A \cup B \cup \{x\}}^+$ ), the entries of  $A$  are positioned directly before  $x$  and the entries of  $B$  are directly succeeding the entry of  $x$ .

Similarly, the formal semantics for AP3 (Move Activity) can be defined (cf. Fig. 30). Conditions (3) and (3') which describe the position of  $x$  in  $\mu$  (on  $S'$ ) are equal to the ones presented for AP1. However, there is a different pre-condition for AP3 (Move Activity). Here,  $x$  must be present in  $S$  in order to be moved afterwards (1). The relation between execution traces on  $S$  and  $S'$  is also different from AP1 (Insert Activity): Traces on  $S$  as well as traces on  $S'$  might contain  $x$  but at different positions. Therefore we claim that the projections of these traces (i.e., the traces where  $x$  will be discarded if present) have to be equal.

Finally, pattern AP14 (Copy Activity) is also related to AP1 (Insert Activity). Again the position of copied (and re-labelled) activity  $x'$  can be formalized by conditions (3) and (3') as for AP1 (Insert Activity). Similar to AP3 (Move Activity) the activity to be copied must be present in  $S$  (1). Additionally, labels of the activity to be copied and the copied activity must be different from each other and no activity with new label must be already contained in  $S$  (2). Copying an activity  $x$  (with new label  $x'$ ) can be seen as inserting  $x'$  at the respective position. Therefore, the relation between traces on  $S$  and  $S'$  can be defined as for AP1, but based on  $x'$ ; i.e., when discarding copied activity  $x'$  from  $\mu$  (on  $S'$ ), then there exists an equal trace  $\sigma$  on  $S$ .

*Group 2: Pattern AP2 (Delete Activity):* The second group contains only one adaptation pattern since its formalization does not directly relate to any other pattern. Consider  $\text{op} = \text{Delete}(S, x) \mapsto S'$  with formal semantics:

- (1) Process schema  $S$  contains one node with label  $x$ .
- (2)  $\forall \mu \in \mathcal{Q}_{S'}: x \notin \mu$
- (3)  $\forall \mu \in \mathcal{Q}_{S'}: \exists \sigma \in \mathcal{Q}_S \text{ with } \mu = \sigma_{\{x\}}^- \wedge$   
 $\forall \sigma \in \mathcal{Q}_S: \exists \mu \in \mathcal{Q}_{S'} \text{ with } \sigma_{\{x\}}^- = \mu$

When deleting activities, first of all, the process schema has to contain a node

with label x. As an effect resulting from the application of this pattern, all execution traces  $\mu$  on  $S'$  must not contain x. Finally, for all traces  $\sigma$  on S we can find an equal trace on  $S'$  if x is discarded from  $\sigma$  (and vice versa). Note that (2) and (3) explain best the effects of the Delete pattern.

*Group 3: Patterns AP4 (Replace Activity) and AP5 (Swap Activities):* Here we illustrate AP5 (Swap Activities) since it "contains" the formalization for AP4 (Replace Activity). Reason is that a swap between activities x and y can be (logically) seen as replacing x by y and y by x. The formal semantics of  $\text{op} = \text{Swap}(S, x, y) \mapsto S'$  turns out to:

(1) Process schema  $S$  contains one node with label x and one with label y.

(2)  $\forall \sigma \in \mathcal{Q}_S: \exists \mu \in \mathcal{Q}_{S'} \text{ with } |\sigma| = |\mu| \wedge$

$$\mu(i) = \begin{cases} \sigma(i) & \text{if } \sigma(i) \notin \{x, y\} \\ x & \text{if } \sigma(i) = y \\ y & \text{if } \sigma(i) = x \end{cases}$$

and vice versa

Alternatively we can formulate (2) as follows:

(2')  $\forall \sigma \in \mathcal{Q}_S: \exists \mu \in \mathcal{Q}_{S'}: \text{with } |\sigma| = |\mu| \wedge \sigma_{\{x,y\}}^- = \mu_{\{x,y\}}^-$   
 $\wedge (\sigma(k) = x \implies \mu(k) = y)$

and vice versa

To be swapped, activities x and y have to be both contained in process schema S (1). The relation between trace  $\sigma$  on S and corresponding trace  $\mu$  on  $S'$  can be formalized in two ways. In both cases, for all traces  $\sigma$  on S, there exists a corresponding trace  $\mu$  on  $S'$  for which the cardinality of  $\sigma$  and  $\mu$  are equal. Regarding the positions of the swapped activities x and y, we can explicitly state that for all traces  $\sigma$  on S, a trace  $\mu$  on  $S'$  can be found such that all entries of  $\mu$  are equal to entries of  $\sigma$  except at positions of x and y where the entries are swapped; i.e., at the position of x in  $\sigma$ ,  $\mu$  contains y and vice versa (2). Alternatively, for all traces  $\sigma$  on S there exists a corresponding trace  $\mu$  on  $S'$  for which the projections of  $\sigma$  and  $\mu$  resulting from discarding x and y are equal (2'). Furthermore,  $\mu$  contains the entry of y at the position of x in  $\sigma$  and vice versa.

*Group 4: Patterns AP8 (Embed Process Fragment in Loop) and AP10 (Embed Process Fragment in Conditional Branch):* As a first characteristics, we formalize the relation between  $\mathcal{Q}_S$  and  $\mathcal{Q}_{S'}$  for  $\text{op} = \text{Embed\_in\_Loop}(S, P, \text{cond}) \mapsto S'$  (AP8). If the number of loop iterations is finite (i.e., the set of execution traces on a process schema containing loops is finite too),  $\mathcal{Q}_S \subset \mathcal{Q}_{S'}$

holds (1). Reason is that for all traces  $\sigma$  on  $S$  a trace  $\mu$  on  $S'$  can be found with  $\sigma = \mu$  but not vice versa (due to the possibly iterative execution of the new loop). To find a more specific characterization of the relation between  $\mathcal{Q}_S$  and  $\mathcal{Q}_{S'}$ , for all traces  $\mu$  on  $S'$  we construct a certain trace projection  $\mu'$  as follows: All entries of activities in  $P$  (if existing) are discarded from  $\mu$  except the entries of one arbitrary loop iteration; i.e.,  $\mu$  is projected onto a "loop-free" version of itself. Obviously, the resulting trace  $\mu'$  is a trace on  $S$  (i.e.,  $\mu' \in \mathcal{Q}_S$ ).

For pattern  $op = \text{Embed\_in\_Cond\_Branch}(S, P, \text{cond}) \mapsto S'$  (AP10), first of all,  $\mathcal{Q}_{S'} \subseteq \mathcal{Q}_S$  holds(1). Reason is that due to the newly inserted conditional branch only a subset of traces might be generated on  $S'$  when compared to  $S$ . Furthermore, the relation between traces on  $S$  and traces on  $S'$  can be defined more precisely; i.e., for all traces  $\sigma$  on  $S$ , its projection resulting from discarding all entries of  $P$  from  $\sigma$  is contained in the set of traces on  $S'$  (2).

*Group 5: Patterns AP9 (Parallelize Activities), AP11 (Add Control Dependency), AP12 (Remove Control Dependency), and AP13 (Update Condition):* Patterns AP9, AP11, and AP12 are possibly changing the execution orders of activities in  $S$ .

We describe AP9:  $op = \text{Parallelize}(S, P) \mapsto S'$  in detail, since AP12 (Remove Control Dependency) can be seen as a special case of AP9 and AP11 (Add Control Dependency) is the reverse operation to AP12. AP13 (Update Condition) is explained at the end of this pattern group.

The formal semantics of AP9 follows as:

- (1) Within schema  $S$ , the sub graph induced by  $P$  constitutes a sequence with single entry and single exit node.
- (2)  $\forall \sigma \in \mathcal{Q}_S: \exists \mu \in \mathcal{Q}_{S'} \text{ with } \sigma = \mu$  (i.e.,  $\mathcal{Q}_S \subset \mathcal{Q}'_S$ )
- (3)  $\forall p, p' \in P: \exists \mu_1, \mu_2 \in \mathcal{Q}_{S'}$  with  
 $(p \in \text{tracePred}(S', p', \mu_1) \wedge p' \in \text{tracePred}(S', p, \mu_2))$   
 (assuming that the sequence defined by  $P$  can be enabled in  $S$ )

As a prerequisite, all activities to be parallelized must be ordered in sequence (1). As a basic characterization of pattern AP9, the set of traces on  $S$  is a subset of the set of traces on  $S'$  (2) since traces on  $S'$  might contain entries reflecting a sequential order of  $P$ , too, but also any other execution order regarding activities from  $P$  (3). More precisely, every pair of activities contained in trace  $\mu$  on  $S'$  is ordered in parallel in the new schema. For pattern AP12 (Remove Control Dependency), the formal semantics of AP9 can be specialized by formalizing it for 2 activities.

For AP11 (Add Control Dependency), the conditions of AP12 hold in reverse direction, i.e., the execution order is made stricter on  $S'$  such that  $\mathcal{Q}_{S'}$  becomes

a subset of  $\mathcal{Q}_S$ . For  $\text{op} = \text{Update\_Ctrl\_Dependeny}(S, x, y, u\_\text{cond}) \mapsto S'$  (AP13) we can define the following formal semantics:

- (1)  $\text{oldCond} \implies \text{newCond}$ :  $\forall \mu \in \mathcal{Q}_{S'}$  for which transition condition  $\text{newCond}$  evaluates to  $\text{TRUE}$ :  $\exists \sigma \in \mathcal{Q}_S$  with  $\mu = \sigma$
- (2)  $\text{newCond} \implies \text{oldCond}$ :  $\forall \sigma \in \mathcal{Q}_S$  for which transition condition  $\text{oldCond}$  evaluates to  $\text{TRUE}$ :  $\exists \mu \in \mathcal{Q}_{S'}$  with  $\mu = \sigma$
- (3) Otherwise, for all traces  $\sigma \in \mathcal{Q}_S$  there exists a trace  $\mu \in \mathcal{Q}_{S'}$  for which the following holds: If we produce projections for  $\sigma$  and  $\mu$  by discarding all entries which belong to the conditional branch with the updated condition, these projections are equal.

More precisely, we can derive a statement about the relation of traces between  $S$  and  $S'$  if we know the relation between old and updated condition ((1) or (2)). For (3), the projections of  $\sigma$  and  $\mu$  can be easily accomplished based on, for example, block-structured process meta models.

*Group 6: Patterns AP6 (Extract Sub Process) and AP7 (Inline Sub Process):* These adaptation patterns are the counterpart of each other. Thus, in the following we illustrate AP6 (Extract Sub Process), the formal semantics of AP7 (Inline Sub Process) follows directly. Consider  $\text{op} = \text{Extract}(S, P, x) \mapsto S$  (AP6). The particular challenge of formalizing AP6 lies in the connection between traces  $\mu$  on  $S'$  and traces  $\sigma$  on  $S$  where the entries of activities of  $P$  might have to be inlined "instead of" the entry of  $x$  (if activities of  $P$  are executed). This is especially difficult in connection with loops (possibly multiple execution of  $x$ ). We present the formal semantics of AP6 in the following and explain it afterwards:

(1) The sub graph on  $S$  induced by  $P$  has to be connected and must be a hammock, i.e., have single entry and single exit node.

$$(2) \forall \sigma \in \mathcal{Q}_S: \exists \mu \in \mathcal{Q}_{S'} \text{ with } \mu_{\{x\}}^- = \sigma_P^- \wedge \forall \mu \in \mathcal{Q}_{S'}: \exists \sigma \in \mathcal{Q}_S: \sigma_P^- = \mu_{\{x\}}^-$$

(3) Let  $z$  denote the single exit node of the sub graph induced by  $P$ .

$$\text{Then: } \forall \sigma \in \mathcal{Q}_S \text{ with } \sigma_{P \setminus \{z\}}^-(k) = z: \exists \mu \in \mathcal{Q}_{S'} \text{ with } \mu(k) = x$$

(4) Let  $\mathcal{P}$  denote the set of all execution traces over the sub graph induced by  $P$  and let further  $\pi \in \mathcal{P}$ . Then:

$$\begin{aligned} \forall \mu \in \mathcal{Q}_{S'} \text{ with } \mu(\nu_i) = x \ (i = 1, \dots, n, \nu_i \in \mathbb{N}): \\ \exists \sigma \in \mathcal{Q}_S \text{ with} \end{aligned}$$

$$\sigma(k) = \begin{cases} = \mu(k) & k = 1, \dots, \nu_1 - 1, \\ = \mu(k - j * |\pi| + j) & k = \nu_j + |\pi|, \dots, \nu_{j+1} - 1 \wedge \\ & k = \nu_n + |\pi|, \dots, |\mu| + n * (|\pi| - 1) \\ = \pi(l) & k = \nu_i + l - 1 \end{cases}$$

where  $j = 1, \dots, n-1; l = 1, \dots, |\pi|$

First, the relation between  $\sigma$  on  $S$  and  $\mu$  on  $S'$  can be formalized as follows: For all  $\sigma$  on  $S$  we can find a trace  $\mu$  on  $S'$  for which the projections resulting from discarding all entries of sub process  $P$  from  $\sigma$  and discarding the entry of  $x$  from  $\mu$  are equal (1). The interesting question is how to determine the position of  $x$  on  $S'$  when extracting  $P$  from  $S$ . For this, we "contract" trace  $\sigma$  on  $S$  by discarding all entries of activities in  $P$  except the one of single exit entry  $z$  (2). Then we can find a trace  $\mu$  on  $S'$  for which the position of the "contracted" trace  $\sigma$  determines the position of  $x$  in  $\mu$ . The other direction (i.e., how to determine the positions of activities in  $P$  on  $S$ ) which is very important in the context of AP7 (Inline Sub Process) as well, is more challenging. We solve this by constructing a trace  $\sigma$  on  $S$  for all  $\mu$  on  $S'$ . First the position(s) of  $x$  in  $\mu$  is (are) determined  $(\nu_1, \dots, \nu_n)$ . This might be more than one position in the context of loops (i.e., if  $x$  is embedded within a loop construct and possibly executed several times). Then the activities of  $P$  are inserted at this (these) position(s) within  $\sigma$  (3). The remaining part of  $\sigma$  can be constructed using the entries of  $\mu$ , only the positions have to be shifted accordingly.

#### 4.3 Semantics of Patterns for Predefind Changes

To formalize the semantics of predefind change patterns PP1 (Late Selection of Process Fragments) and PP2 (Late Modeling of Fragments) similar considerations can be made as for adaptation pattern AP7 (Inline Sub Process). Pattern  $op = \text{Inline}(S, x, P) \mapsto S'$  changes process schema  $S$  into process schema  $S'$  by inlining the sub process induced by activity set  $P$ , i.e., the sub process to be inlined is given in advance (cf. Fig. 35). For pattern PP1 (Late Selection of Process Fragments) and PP2 (Late Modeling of Fragments) also a sub process is to be inlined. Contrary to AP7, for PP1 a sub process has to be chosen from a repository first and for PP2 the sub process has to be specified before inlining. However, at the moment the sub process is either chosen or specified, the formal semantics of PP1 and PP2 can be defined as for AP7.

For PP3 (Late Composition of Process Fragments), its formal semantics depends on knowledge of the process designer's decision on which process fragments to chose and how to compose them. If this knowledge is at hand, traces  $\mu$  on  $S'$  could be constructed by composing the execution traces on the particular fragments in the given order. However, this might become difficult in connection with parallelism.

Finally, for PP4 (Multi Instance Activity), the formal semantics can be defined if the number of instantiations is known. Then traces  $\mu$  on  $S'$  contain  $k$  entries of multi instance activity  $x$  at the position where  $x$  would be executed on  $S$ .

In this section we have formally defined the semantics for the adaptation patterns and (informally) for the patterns for changes in predefined regions. To stay independent of a certain meta model we have based our formalization on execution traces. However, with additional knowledge of meta model properties, some of the formalizations will become much easier. We have also showed, for which patterns runtime information is necessary in order to specify their formal semantics.

## 5 Change Support Features

So far, we have introduced a set of change patterns, which can be used to accomplish changes at the process type or process instance level. However, simply looking at the supported patterns and counting their number 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. 36). Relevant change support features include *process schema evolution, version control and instance migration, change correctness, change traceability, access control, change reuse, and concurrency control*. As illustrated in Fig. 36 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 (T), while change reuse is particularly useful at the instance level (I) [52].

Change Support Features			
Change Support Feature	Scope*	Change Support Feature	Scope
<b>F1: Schema Evolution, Version Control and Instance Migration</b>	T	<b>F3: Correctness of Changes</b>	I + T
		<b>F4: Traceability and Analysis</b>	I + T
No version control – Old schema is overwritten		1. Traceability of changes	
1. Running instances are canceled		2. Annotation of changes	
2. Running instances remain in the system		3. Change Mining	
Version control		<b>F5: Access Control for Changes</b>	I+T
3. Co-existence of old/new instances, no instance migration		1. Changes in general can be restricted to authorized users	
4. Uncontrolled migration of all process instances		2. Application of single change patterns can be restricted	
5. Controlled migration of compliant process instances		3. Authorizations can depend on the object to be changed	
<b>F2: Support for Instance-Specific Changes</b>	I	<b>F6: Change Reuse</b>	I
1. Unplanned Changes		<b>F7: Change Concurrency Control</b>	I+T
a. Temporary		1. Uncontrolled concurrent changes	
b. Permanent		2. Concurrent changes are prohibited	
2. Preplanned Changes		3. Concurrent changes of structure and state of a particular process instance controlled by PAIS	
a. Temporary		4. Concurrent changes at the type and the instance level	
b. Permanent			

\* The scope of a particular change feature can either be the process type level (T) and/or the process instance level(I)

Fig. 36. Change Support Features

## 5.1 Schema Evolution, Version Control and Instance Migration

To support changes at the process type level, version control for process schemes should be supported (cf. Fig. 36). In case of long-running processes, in addition, controlled migration of already running instances from the old to the new process schema version, might be required. In this subsection we describe different options existing in this context (cf. Fig. 37-39).

If a PAIS provides no version control, either the process designer will have to manually create a copy of the process schema (to be changed) or this schema will be overwritten (cf. Fig. 37). In the latter case, running process instances can either be withdrawn from the run-time environment or, as illustrated in Fig. 37, they remain associated with the modified schema. Depending on the execution state of the instances and on how changes are propagated to instances progressed too far, this missing version control can lead to inconsistent states and, worst case, to deadlocks or other run-time errors [49]. As illustrated in Fig. 37 schema  $S$  has been modified by inserting activities X and Y. Regarding, instance  $I1$  this change is uncritical as it has not yet entered the change region. However, instance  $I2$  would be in an inconsistent state afterwards as instance schema and execution history do not match [49].

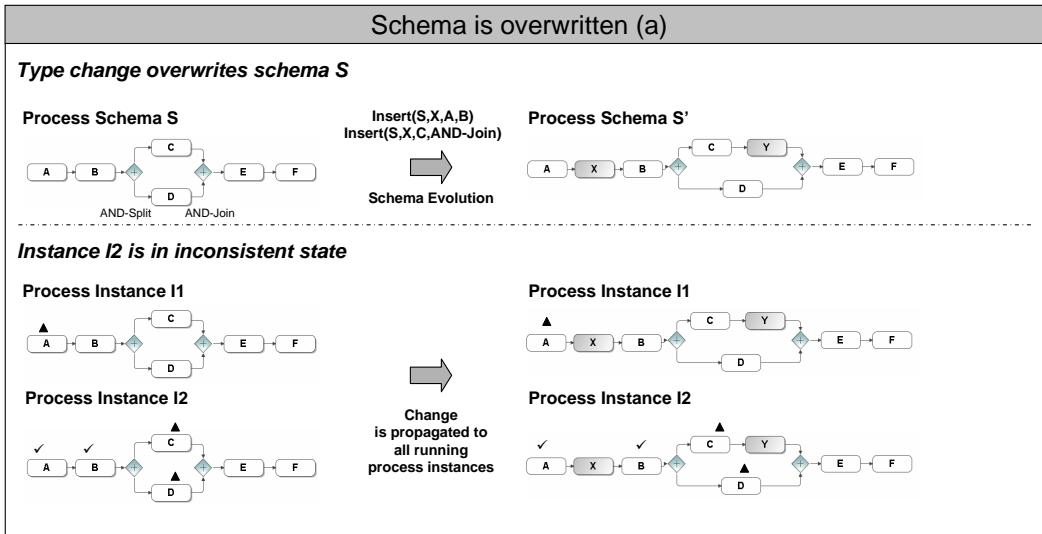


Fig. 37. Missing Version Control

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 based on the new schema version [8,50]. This approach leads to the co-existence of process instances belonging to different schema versions (cf. Fig. 38). Alternatively, a controlled migration of a (selected) collection of already running process instances to the new process schema version is supported (cf. Fig. 39).

The first option is shown in Fig. 38 where the already running instances  $I_1$ ,  $I_2$  and  $I_3$  remain associated with schema  $S$ , while new instances ( $I_4$ - $I_5$ ) are created from schema  $S'$  (co-existence of process instances of different schema versions). By contrast, Fig. 39 illustrates the controlled migration of selected process instances. Only those instances ( $I_1$  and  $I_2$ ) are migrated to the new process schema version which are *compliant*<sup>3</sup> with  $S'$ . Thereby, instance  $I_1$  can be migrated without any state adaptations, whereas for instance  $I_2$  activity X has to be enabled instead of activity B. Instance  $I_3$  remains running according to  $S$ . If instance migration is accomplished in an uncontrolled manner (as it is not restricted to *compliant* process instances) inconsistencies or errors will result. 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 6).

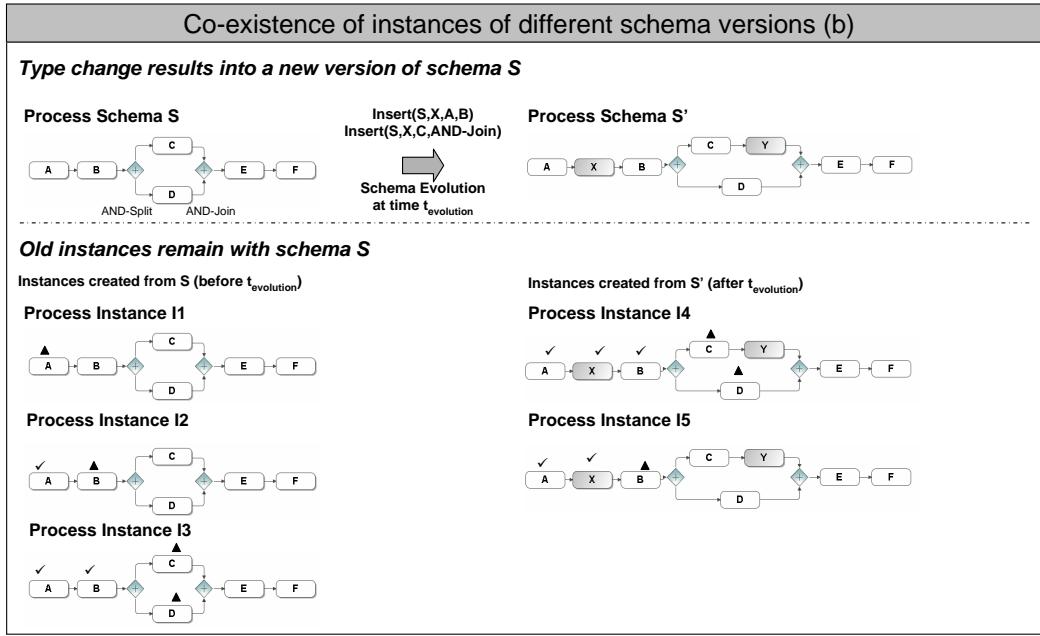


Fig. 38. Version Control - Co-existence of different schema versions

## 5.2 Other Change Support Features

**Support for Instance-Specific Changes (Change Feature F2).** To deal with exceptions PAIS must support unplanned changes at the process instance level either through high-level changes in the form of patterns (cf. Section 3) or through low-level primitives. To deal with uncertainty, PAIS must allow keeping parts of the model unspecified during build-time and deferring the concretisation of the respective part to run-time. The effects resulting from

<sup>3</sup> 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 [49]).

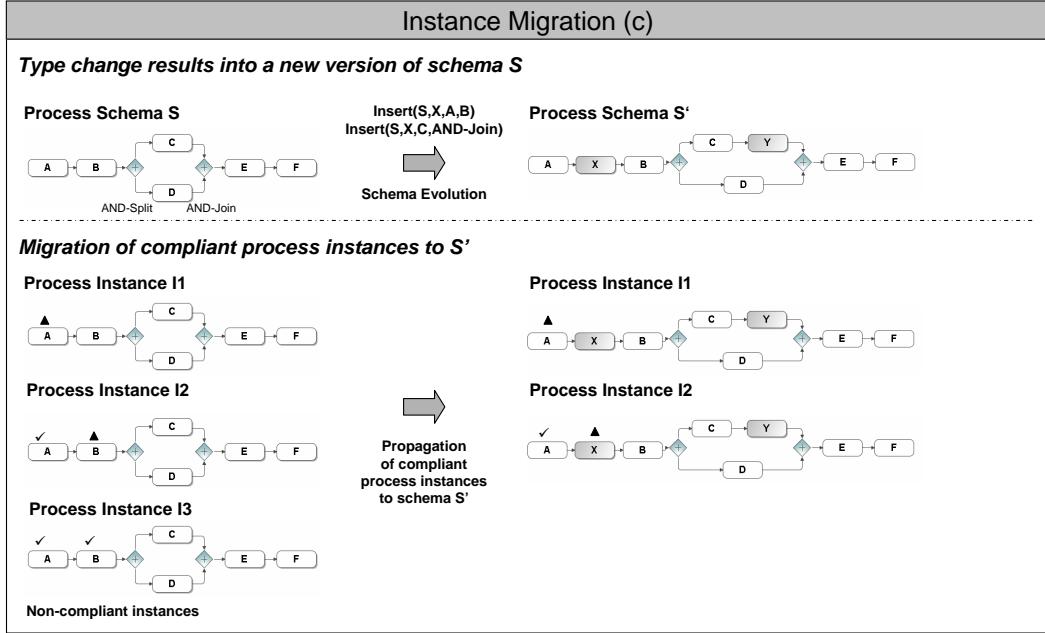


Fig. 39. Version Control - Change Propagation

instance-specific changes 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., the current iteration of a loop) (cf. Fig. 40).

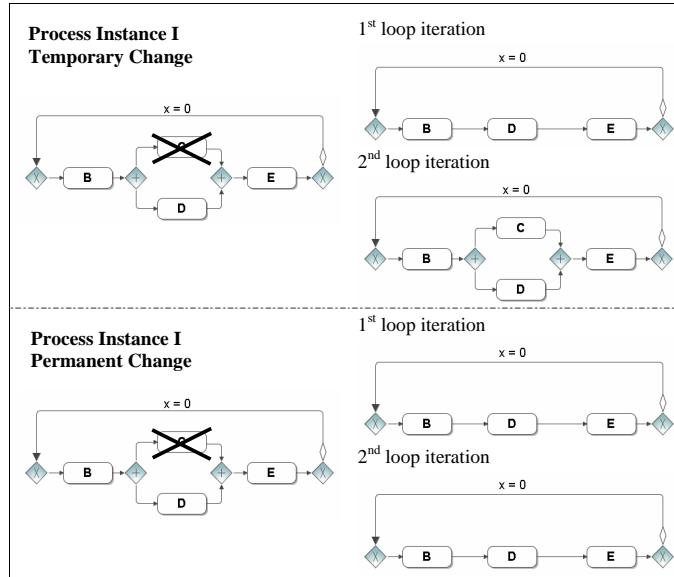


Fig. 40. Permanent and Temporary Changes

**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 criteria [49] have been intro-

duced to formally ensure that process instances can only be updated to a new schema if they are compliant with it [78,8,50]. Depending on the used process meta model, in addition, (formal) constraints of the respective formalism (e.g., concerning the structuring of the the process schemes) have to be taken into account as well when applying process changes to a particular process schema.

**Traceability and Analysis of Changes (Change Feature F4).** For adaptation patterns the applied changes have to be stored in a change log as change patterns and/or change primitives [51]. While both options allow for traceability, change analysis and change mining [20] become easier when the change log contains high-level information about the changes as well. Regarding patterns for changes in predefined regions, 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 the context of the changes [52]. Finally, change mining allows for the analysis of changes, for example, to support continuous process improvement [20].

**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 [72,14]. 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 [72] allow defining 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., a process schema or a process instance).

**Change Reuse (Change Feature F6).** In the context of unplanned instance-specific changes "similar" deviations (i.e., combinations of one or more adaptation patterns) can occur more than once [32,76,77,33]. As it requires significant user experience to define changes from scratch, change reuse should be supported. To support this changes they have to be annotated with contextual information (e.g., about the reasons for the deviation) and be memorized. 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 [52,33,75]. Regarding patterns for changes in predefined regions, reuse can be supported by making historical cases available to the user and by saving frequently re-occurring instances as templates [31].

**Change Concurrency Control (Change Feature F7).** A PAIS which supports instance-specific process adaptations (cf. Feature F2) should be able to cope with concurrent changes as well. Concurrent changes of a particular process instance may include both adaptations of its structure and its state.

For example, two users might want to apply different ad-hoc changes to a particular process instance at the same time. If this is done in an uncontrolled manner, severe errors or inconsistencies (e.g., deadlock-causing cycles) can occur (Option 1). Or the execution of a process instance proceeds (i.e., the state of the respective instance changes) while an ad-hoc change is concurrently applied to this instance. Here we have to ensure that the state change does not violate state constraints required for the correct application of the ad-hoc change (or at least the ad-hoc change has to be prohibited in such cases).

For these reasons, change concurrency control becomes an indispensable feature of any flexible PAIS. The easiest way to avoid respective conflicts is to prohibit concurrent changes at all (Option 2). This can be achieved, for example, by holding exclusive locks on a process instance when changing its structure and/or its state (e.g., an instance must then not proceed while applying an ad-hoc change to it). Though this approach is easy to implement, it is usually too strict due to the long-term locks required (e.g., when a change is defined interactively by a user). A more flexible approach allows for concurrent changes of the structure or state of a process instance, and further ensures that this does not lead to errors or inconsistencies afterwards (Option 3). Both, pessimistic and optimistic techniques can be applied in this context to control such concurrent instance changes and to ensure their correctness.

Finally, we have to deal with "concurrent" changes at the process type and the process instance level. For example, assume that an instance-specific change is applied to process instance  $I$ , which was originally created from process schema  $S$ . Assume further that later process schema  $S$  evolves to  $S'$  due to a change at the process type level. Then, the challenging question is whether the process type change can be propagated to  $I$  as well. Though  $I$  has undergone an instance-specific change this should not mean that it must not migrate to the new schema version  $S'$  (particularly not if  $I$  is long-running). Note that respective considerations only have to be made for systems supporting both changes at the process type and the process instance level (Option 4).

## 6 Change Patterns and Change Support in Practice

In this section we evaluate approaches from both academia and industry regarding their support for change patterns and change features (cf. Fig. 41-43). For *academic approaches* the evaluation is mainly based on a comprehensive literature study. In cases where it was unclear whether a particular pattern or feature is supported, the respective research groups were additionally contacted. This has provided us with valuable insights into the implementation of change patterns and change features in respective approaches. In detail, the evaluated approaches (in alphabetical order) are ADEPT2 [43,48,45,42,18],

CAKE2 [32,33], CBRFlow [52,77,75], HOON [23], MOVE [21], Pockets of Flexibility (PoF) [59,31,58], WASA2 [78], WIDE [8,9], Worklets/Exlets [2,1,3], and YAWL [12]. As CBRFlow and ADEPT2 have been integrated within the ProCycle project [77] both systems have been evaluated together. The Worklets/Exlets approach has been evaluated together with YAWL as it has been integrated as a service for YAWL to foster its flexibility.

In respect to *commercial systems* only such systems have been considered for which we have both hands on experience and a running system installed in our labs. This has allowed us to test the change patterns and change features. We consider the case-handling system Flower [69] and the workflow management system Staffware [15] as commercial systems for the present evaluation. Evaluation results are presented in Fig. 41-43. These figures are only meant as a summary of our evaluation results. An in-depth description of each of the evaluated approaches can be found in Appendix A.

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 ADEPT2). Otherwise, a table entry describes the exact pattern variants as supported by listing available design choices. For instance, consider the evaluation of the ADEPT2 system for adaptation pattern AP6 in Fig. 41. The string ”A[1,2], B[3]” indicates that ADEPT2 supports the extraction of process fragments (AP6) with Design Choices A and B. As described in Fig. 7, ADEPT2 supports the respective pattern at the process type and the process instance level (Design Choice A). In addition, the pattern can be applied to hammocks (Design Choice B). As another example take change pattern PP1 (Late Selection of Process Fragments) as supported in the Worklet/Exlet approach [2,1]. In Fig. 43 the string ”A[1,2], B[1,2], C[2]” indicates that this change pattern (cf. Fig. 22) is supported by the Worklet/Exlet approach with Design Choices A, B and C. Further, it indicates the exact options available for every design choice. For example, for design choice A, Options 1 and 2 are supported. Taking the description from Fig. 22 this means that the selection of the activity implementation can be done automatically (based on predefined rules) or manually by a user.

If 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). Finally, partial support is labeled with “o” (e.g., the Worklet/Exlet approach supports change feature F3 partially).

## 6.1 Adaptation Patterns Support

Fig. 41 and 42 show, which change primitives and adaptation patterns are supported by the evaluated systems. Table 41 focuses on structural changes at the process type level, i.e., on changes which can be performed in the process editor of the respective system when defining or adapting a process structure. Table 42, in turn, provides the evaluation results considering the use of change patterns or change primitives at the process instance level. For a detailed description of the evaluated approaches we refer to Appendix A.

Generally, an adaptation pattern will be only considered as being provided, if the respective system supports the pattern directly, i.e., based on a single high-level change operation instead of a set of change primitives. As adaptation patterns can always be "simulated" by means of a set of basic change primitives (e.g., CAKE2 or WASA2), missing support for adaptation patterns does not necessarily mean that no changes can be performed. However, 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 (see also Section 3.3). Further, 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 an adaptation pattern, if it supports this pattern directly (i.e., it offers a single change operation for realizing the respective adaptation pattern). For instance, providing support for patterns AP1 (Insert Process Fragment) and AP2 (Delete Process Fragment) allows implementing 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 adaptation pattern AP3. Moreover, this leads to less meaningful change logs. In addition, to qualify as an adaptation pattern its application must not be restricted to predefined regions in the process.

Table 41 shows that all evaluated systems allow for process type modifications. Thereby, most systems only provide support for change primitives, i.e., they allow modifying an existing process schema by adding or deleting nodes and edges. An additional primitive which allows users to move edges is provided by CAKE2 and YAWL. The only systems offering adaptation patterns support at the process type level are ADEPT2 and WIDE. Table 42 shows that process instance modifications are supported by rather few systems. CAKE2 and WASA2 allow for structural run-time adaptations at the instance-level through change primitives (i.e., by adding or removing nodes and edges respectively). ADEPT2 provides support for a wide range of adaptation patterns at the process instance level. Both the Worklet/Exlet approach and Flower

support a limited spectrum of ad-hoc changes: the Worklet/Exlet approach allows for the replacement of activities (AP4), whereas Flower allows for the deletion of activities (AP2).

Change Pattern Support at the Process Type Level									
Primitive / Pattern	Academic					Commercial			
	ADEPT2 / CBRFlow	CAKE2	HOON	MOVE	PoF	WASA2	WIDE	YAWL <sup>+</sup> / Worklets / Exlets	Flower
Process Adaptation									
Change Primitives									
PR1 – Add Node	-	+	+	+	+	+	+	+	+
PR2 – Remove Node	-	+	+	+	+	+	+	+	+
PR3 – Add Edge	-	+	+	+	+	+	+	+	+
PR4 – Remove Edge	-	+	+	+	+	+	+	+	+
PR5 – Move Edge	-	+	-	-	-	-	-	+	-
Adaptation Patterns									
AP1 – Insert Fragment	A[1,2], B[1,2,3], C[1,2]	-	-	-	-	-	-	-	-
AP2 – Delete Fragment	A[1,2], B[1,2,3]	-	-	-	-	-	-	-	-
AP3 - Move Fragment	A[1,2], B[1,2,3], C[1,2]	-	-	-	-	-	-	-	-
AP4 – Replace Fragment	-	-	-	-	-	-	-	-	-
AP5 – Swap Fragment	-	-	-	-	-	-	-	-	-
AP6 – Extract Fragment	A[1,2], B[3]	-	-	-	-	-	-	-	-
AP7 – Inline Fragment	A[1,2], B[2]	-	-	-	-	-	-	-	-
AP8 – Embed Fragment in Loop	A[1,2], B[1,2,3]	-	-	-	-	-	-	-	-
AP9 – Parallelize Activities	A[1,2], B[1,2,3]	-	-	-	-	-	-	-	-
AP10 - Embed Fragment in Conditional Branch	-	-	-	-	-	-	A[2]	-	-
AP11 – Add Control Dependency	A[1,2]	-	-	-	-	-	-	-	-
AP12 – Remove Control Dependencies	A[1,2]	-	-	-	-	-	-	-	-
AP13 – Update Condition	A[1,2]	-	-	-	-	-	A[2]	-	-
AP14 – Copy Fragment	-	-	-	-	-	-	-	-	-

Fig. 41. Adaptation Patterns Support at the Process Type Level

Change Pattern Support at the Process Instance Level								
Primitive / Pattern	Academic				Commercial			
	ADEPT2 / CBREFlow	CAKE2	HOON	MOVE	PoF	WASA2	WIDE	YAWL + Worklets / Events
Process Adaptation								
Change Primitives								
PR1 – Add Node	–	+	–	–	–	+	–	–
PR2 – Remove Node	–	+	–	–	–	+	–	–
PR3 – Add Edge	–	+	–	–	–	+	–	–
PR4 – Remove Edge	–	+	–	–	–	+	–	–
PR5 – Move Edge	–	+	–	–	–	–	–	–
Adaptation Patterns								
AP1 – Insert Fragment	A[1..2], B[1..2,3], C[1..2]	–	–	–	–	–	–	–
AP2 – Delete Fragment	A[1..2], B[1..2,3]	–	–	–	–	–	–	–
AP3 – Move Fragment	A[1..2], B[1..2,3], C[1..2]	–	–	–	–	–	–	–
AP4 – Replace Fragment	–	–	–	–	–	–	–	–
AP5 – Swap Fragment	–	–	–	–	–	–	–	–
AP6 – Extract Fragment	A[1..2], B[3]	–	–	–	–	–	–	–
AP7 – Inline Fragment	A[1..2], B[2]	–	–	–	–	–	–	–
AP8 – Embed Fragment in Loop	A[1..2], B[1..2,3]	–	–	–	–	–	–	–
AP9 – Parallelize Activities	A[1..2], B[1..2,3]	–	–	–	–	–	–	–
AP10 - Embed Fragment in Conditional Branch	–	–	–	–	–	–	–	–
AP11 – Add Control Dependency	A[1..2]	–	–	–	–	–	–	–
AP12 – Remove Control Dependencies	A[1..2]	–	–	–	–	–	–	–
AP13 – Update Condition	A[1..2]	–	–	–	–	–	–	–
AP14 – Copy Fragment	–	–	–	–	–	–	–	–

Fig. 42. Adaptation Patterns Support at the Process Instance Level

## 6.2 Support for Patterns for Changes in Predefined Regions

Table 43 shows how patterns for changes in predefined regions are supported by the evaluated approaches.

Pattern PP1 (*Late Selection of Process Fragment*) is supported by 3 distinct systems (HOON, Worklets/Exlets and Staffware). Similar support is offered by CAKE2, MOVE and PoF, which provide support for pattern PP2 (*Late Modeling of Process Fragment*). While MOVE and CAKE2 offer the end user the whole expressiveness of the modeling environment, PoF facilitates model construction by introducing modeling constraints. Validation ensures that the lately modeled process fragment is compliant with the constraints [59]. Pattern PP3 is not supported by any of the evaluated systems. Nevertheless, the *Late Composition of Process Fragment* is listed as a distinct pattern as it constitutes a typical strategy for dealing with changes, which we have identified in our case studies [27,35,80,60]. Finally, the *Multi-Instance Activity* pattern PP4 can be found in WIDE, YAWL, Flower and Staffware.

Pattern PP1 (*Late Selection of Process Fragment*) as well as pattern PP2 (*Late Modeling of Process Fragment*) allow for the realization of parts of the functionality of adaptation pattern AP1 through workarounds (e.g., HOON, MOVE, PoF). Generally, a placeholder activity can be positioned between two fragments or parallel to an existing one in the process schema. By substituting this placeholder activity during run-time with a concrete (sub) process fragment, in principle, a partially pre-planned serial and a parallel insertion becomes possible (cmp. Design Choice D[1,2] of pattern AP1). However, the insertion is restricted to the placeholder activity. Furthermore, these approaches do not allow for structural (ad-hoc) changes of a process fragment once it has been instantiated, unless this fragment itself contains placeholder activities.

### 6.3 Change Support Features in Practice

Table 43 shows the evaluation results regarding the change support features described in Section 5. As the evaluation shows, Feature F1 (Schema Evolution, Version Control and Instance Migration) is only partially supported. Only half of the evaluated systems provide any versioning support at all (i.e., ADEPT2, WASA2, WIDE, YAWL, Flower and Staffware). As missing versioning support requires users to overwrite an existing schema in case of process type level changes or to save the modified schema with a new name, practical applicability is limited. Flower, allows for the overwriting of a process schema in addition to co-existence of process instances. In case ongoing instances are not removed from the system, applying this feature can lead to undesired behaviour. In connection with process schema evolution the controlled propagation of changes to ongoing instances is only considered in ADEPT2, WASA2 and WIDE. Further, Staffware offers a feature for propagating changes to all ongoing instances. As instance migration cannot be restricted to compliant instances, the usage of this feature might lead to inconsistencies or deadlocks having the same effect than overwriting a process schema.

Feature F2 (Support for Instance-Specific Changes) is provided by most approaches in some form. However, unplanned changes through adaptation patterns are only supported by ADEPT2, Flower and the Worklet/Exlet approach. While ADEPT2 has realized most adaptation patterns, Flower restricts ad-hoc modifications to the deletion of activities. Further, the Worklets / Exlet approach only supports replacement of activities. In addition, CAKE2 and WASA2 allow for unplanned changes using change primitives. In addition, preplanned changes are supported by CAKE2, HOON, MOVE, PoF, YAWL / Worklets / Exlets, Flower and Staffware. For a detailed description of the change primitives and the change patterns supported by the respective approaches we refer to Fig. 42.

While feature F3 (Correctness of Changes) is supported quite well by most of the evaluated academic approaches, the Worklet/Exlet approach only provides partial support for it. Both commercial systems Staffware and Flower do not provide formal correctness criteria for schema evolution and instance migration, which can lead to inconsistencies and deadlocks under certain circumstances. This especially holds for the overwriting of process schemes in Flower and the instance propagation in Staffware.

Feature F4 (Traceability and Analysis of Changes) is supported by all evaluated systems. However, most of them only provide simple execution and/or change logs and do not enhance these logs with further information like the context of a change and the reasons for it. Change annotations are only available in ADEPT2/CBRFlow and CAKE2. First approaches towards change mining are supported by ADEPT2 for which a plugin for the process mining tool ProM [40] has been developed.

Feature F5 (Access Control for Changes) is supported by most approaches through a simple role concept. Several systems additionally allow for a more fine-grained definition of access rights. For instance, ADEPT2/CBRFlow [72], HOON, PoF, the Worklets/Exlets approach, Flower [69] and Staffware allow for specifying distinct authorizations for each pattern. In addition, all these approaches also allow for object dependent authorizations as well. This feature further supported by MOVE and WIDE.

Support for Feature F6 (Change Reuse) is only provided in ADEPT2 / CBR-Flow, CAKE2, PoF and the Worklets/Exlets approach. In ADEPT2/CBRFlow case-based reasoning (CBR) techniques are used for retrieving instance-specific changes which have occurred previously in a similar context [75,77]. CAKE2 also uses CBR for change retrieval and considers structural as well as contextual information [32]. The retrieval component of the PoF approach is primarily based on structural information [30]. Finally, the Worklets/Exlet approach supports the reuse of Worklets through selection rules [2].

Most evaluated systems provide support for Feature F7 (Change Concurrency Control) and control change concurrency by the engine. For approaches, which only provide support for changes to predefined regions, concurrency control can be easily achieved as changes are local to the placeholder activities (e.g., HOON, MOVE, PoF, Worklets/Exlets and Staffware). Therefore, changes to different placeholder activities can be performed concurrently. In the context of structural process adaptations concurrency control becomes more complicated. ADEPT2 and CAKE2 allow for concurrent changes and control concurrency by the PAIS [43,32]. In contrast, WASA2 prohibits concurrent changes and requires the entire process instance to be locked. Similarly, Flower does not allow users to work on the same case simultaneously and therefore prohibits concurrent changes as well [46]. Concurrency of process type and process instance changes is only addressed by ADEPT2.

#### *6.4 Summary of Evaluation Results*

Our pattern-based evaluation of selected approaches shows that no single system exists which supports all change patterns and change features in an integrated way (cf. Fig. 41-43). In particular, none of the approaches provides a holistic change framework considering both adaptation patterns and patterns for changes in predefined regions at both the process type and the instance level. ADEPT2 and WIDE score well in respect to adaptation patterns, but lack support for changes to predefined regions. WASA provides good support for ad-hoc changes using change primitives, but does not consider changes to predefined regions and high-level change operations. CAKE2 supports structural changes at the instance level and changes to predefined regions, but does not consider process type changes and only supports change primitives.

An integrated change framework considering both adaptation patterns and patterns for changes in predefined regions would allow addressing a much broader process spectrum and a larger variety of process flexibility scenarios. While patterns for changes in predefined regions provide support for dealing with uncertainty by providing more flexible models, adaptation patterns allow for structural changes which cannot be preplanned. In addition, they make changes more efficient, less complex, and less error-prone through providing high-level change operations.

The evaluation also shows a trade-off between expressiveness of the used process meta model and support for structural process adaptations. For instance, the adaptive process management system ADEPT2 has been designed with the goal to support structural process adaptations [43]. To allow for an efficient implementation of the respective patterns, restrictions on the process modeling language have been made. Similar restrictions in terms of expres-

Primitive / Pattern	Academic						Commercial			
	ADEPT2 / CBRFlow	CAKE2	HOON	MOVE	PoF	WASA2	WIDE	YAWL + Worklets / Extlets	Flower	Staffware
<b>In-Built Flexibility</b>										
<b>Patterns for Changes in Predefined Regions</b>										
PP1 – Late Selection of Fragments	–	–	A[1,2], B[1,2], C[2]	–	–	–	–	A[1,2], B[1,2], C[2]	–	
PP2 – Late Modeling of Fragments	–	A[1], B[1], C[2,3], D[1]	–	A[1,1], B[1], C[3], D[1,2]	A[1,2], B[2], C[2], D[1,2]	–	–	–	–	
PP3 – Late Composition of Fragments	–	–	–	–	–	–	–	–	–	
PR4 – Multi-Instance Activity	–	–	–	–	–	–	+	–	+	
<b>Change Support Features</b>										
Feature	ADEPT2 / CBRFlow	CAKE2	HOON	MOVE	PoF	WASA2	WIDE	YAWL + Worklets / Extlets	Flower	Staffware
F1 – Schema Evolution, Version Control and Instance Migration	3,5	1	1	1	1	3,5	3,5	3	1,2,3	3,4
F2 – Support for Instance-Specific Changes	1a,b	1b, 2b	2a	2b	1b	2b	1a,b, 2a,b	1b, 2b	2b	2b
F3 – Correctness of Changes	+	+	+	+	+	+	◦	–	–	–
F4 - Traceability & Analysis	1,2,3	1,2	1	1	1	1	1	1	1	1
F5 – Access Control for Changes	1,2,3	–	1,2,3	1,3	1,2,3	1	1,3	1,2,3	1,2,3*	1,2,3
F6 - Change Reuse	+	+	–	–	+	–	–	+	–	–
F7 - Change Concurrency Control	3,4	3	3	3	3	2	not applicable	3	2	3

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

Fig. 43. Change Features in Practice

siveness also hold for the other approaches supporting structural adaptations like CAKE2, WASA and WIDE. On the other hand, YAWL is a reference implementation for the workflow patterns and therefore allows for a high degree of expressiveness [12]. However, structural adaptations have not yet been

addressed in YAWL and their implementation would be significantly more difficult compared to ADEPT2 due to the higher expressiveness. However, the integration of Worklets/Exlets with YAWL has shown that patterns for changes in predefined regions can be easily integrated also for highly expressive process modeling languages.

In addition to change patterns, change features are needed to make changes applicable in practice: correctness of changes, traceability and analysis of changes, authorization, change reuse, and concurrency control. Our evaluation has shown that deficits in respect to change features exist in several systems. Especially correctness of changes is not always guaranteed.

## 7 Related Work

Patterns were first used by Alexander to describe solutions to recurring problems. In particular, he applied patterns to describe best practices in architectural design [4]. Patterns also have a long tradition in computer science. Gamma et al. applied the same concepts to software engineering and described 23 design patterns [16].

In the area of workflow management, patterns have been introduced for analyzing the expressiveness of process modeling languages [67,55]. In this context, control flow patterns describe different constructs to specify activities and their ordering. In addition, workflow data patterns [53] provide different ways for modeling the data aspect in PAIS, and workflow resource patterns [54] describe how resources can be represented and utilized in workflows. Furthermore, patterns for describing service interactions and process choreographies were introduced [5]. Finally, a formalization of workflow patterns based on pi-calculus has been provided [41].

The introduction of workflow patterns has had significant impact on the design of PAIS and has contributed to the systematic evaluation of PAIS and process modeling and execution languages like BPEL [55], BPMN [81], EPC [55] and UML [57]. However, to evaluate the powerfulness of a PAIS regarding its ability to deal with changes, the existing workflow patterns are important, but not sufficient. In addition, a set of patterns for dealing with the aspect of process change is needed. Although, support for the aforementioned workflow patterns allows reducing the need for modifying process instances during run-time, they require flexibility to be entirely in-built into the process model. By contrast, the patterns for changes to predefined regions as proposed in this paper allow deferring decisions from build- to run-time to be better able to deal with uncertainty. In addition, adaptation patterns will allow for structural modifications of the process if unanticipated exceptions occur or business

processes evolve over time.

When evaluating the ability of PAIS to deal with process change, the degree to which workflow patterns are supported provides an indication on 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 [56] 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 (i.e., its structure). These patterns are therefore well suited for dealing with expected and less complex situations. However, unanticipated situations might additionally require structural adaptations [44]. The change patterns described in this paper modify both the observable behavior of a process instance and its process structure. Therefore, they are particularly suited for coping with unanticipated exceptions. In addition, change patterns can also be applied at the process type level decreasing the efforts needed for accomplishing a particular change. To provide a complete evaluation framework regarding flexibility in PAIS expected and unexpected exceptions as well as schema evolution must be considered in an integrated way. Therefore, the existing exception handling patterns should be complemented with a set of change patterns.

In many cases exception handling requires the combined use of several exception handling patterns and therefore might result in rather complex exception handling routines. The Exlet approach [1,3] addresses this problem by allowing for the combination of different exception handling patterns to an exception handling process called Exlet. In general, Exlets are executed in parallel to the process instance to be modified and can be reused when a similar exception occurs again. Exlets allow "simulating" several of the adaptation patterns. However, as Exlets are executed independently of the process instance without structurally modifying it, end-users have to deal with suspending and resuming process instances when synchronization is required. In contrast, change patterns hide this complexity from end-users by providing high-level change operations.

The aforementioned approaches focus in their evaluations on expressiveness. Thom et al. have shown that patterns can additionally be used for facilitating process modeling [63,62]. They propose a set of 9 patterns for business functions (e.g., approval, notification) and show that by using these patterns the efforts for creating a process model can be decreased significantly [63]. Like these semantical patterns speed up process modeling, change patterns allow reducing the efforts of accomplishing process changes (cf. Section 3.3).

Most systems considered by our evaluation model business processes in a procedural or imperative way. The only exception constitutes the pockets of flexibility (PoF) approach [59], which uses a combination of imperative and declarative process modeling. The process itself is modeled in an imperative way, however, the placeholder activities are specified in a declarative way based on constraints. Other approaches relying on declarative specifications, which have not been considered in our evaluation, are MOBILE [24] and DECLARE [39,66]. For instance, DECLARE uses linear temporal logic as an underlying formalism to model business processes. Instead of requiring process modelers to specify how the process should be executed, they only have to state what should be done by the process during run-time resulting in more flexibility. By using declarative approaches changes definitely become less frequent, however, run-time modifications still can be an issue (e.g., a particular constraint might have to be violated for a particular process instance due to an unforeseen situation). Further, constraints themselves may evolve over time, which raises the challenge of propagation changes to ongoing instances. A promising approach towards this direction is offered by DECLARE [39], which aims at integrating these aspects. Another challenging issue concerns maintenance of constraint-based process models, particularly in case of large constraint sets.

There exist other frameworks focusing on the comparison of specific aspects related to process change. For instance, the work presented in [49] provides a framework for elaborating the strengths and weaknesses of adaptive PAIS (e.g., ADEPT2, WASA2, WIDE) along typical dynamic change problems. Main emphasis of this work is on investigating formal process properties (e.g., proper termination) and correctness criteria (e.g., compliance) in connection with process changes. In addition, [29] compares graph-based and rule-based languages along the dimensions of flexibility, adaptability, dynamism, complexity, and expressiveness.

There exist several approaches targeting at the automatic handling of process exceptions (e.g., activity failures or deadline expiry). Some of these approaches [34,17,6,36] also apply structural adaptations to respective process instances to deal with the exceptions. For this, they use the adaptation patterns offered by existing systems. In ADEPT2, for instance, respective patterns are not only accessible via a process editor, but can be also invoked via a powerful API (application programming interface). Several approaches have utilized this to implement advanced agents for automated exception handling [34,17,6,36].

## 8 Summary and Outlook

In this paper we proposed 18 change patterns and 7 change support features. In combination they allow assessing the power of a particular change frame-

work for PAIS. In addition, we evaluated selected approaches and systems regarding their ability to deal with process change. We believe that the introduction of change patterns complements existing workflow patterns and allows for more meaningful evaluations of existing systems and approaches, particularly if flexibility is an issue. In combination with workflow patterns the presented change framework will enable (PA)IS engineers to choose a process management technology which meets their flexibility requirements best (or to realize that no system satisfies all these requirements). Our work will make the comparison of change frameworks much simpler and allow (PA)IS engineer to easily assess whether vendors really hold what they promise in respect to process changes and process flexibility. Our evaluation shows that currently none of the evaluated systems provides a holistic change framework supporting all kind of changes in an integrated way. However, in analogy to workflow patterns we expect vendors of existing PAIS to evaluate their PAIS along these criteria and to extend their current systems towards better support for process changes.

Our future work will include the identification and the design of 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). In addition, we plan to include additional systems and approaches in our evaluation. Further, we are currently working on a reference implementation, which will provide support for all 18 change patterns and 7 change support features. Based on this reference implementation we plan to complement our work on change patterns with several experiments, e.g., to measure the efforts for changing process schemes either based on change patterns or change primitives.

**Acknowledgements.** We would like to thank Shazia Shadiq, Michael Adams, Matthias Weske, Yanbo Han, Mirjam Minor, Daniel Schmalen, Hajo Reijers, Sheetal Tiwari, Ulrich Kreher and Peter Dadam for their valuable feedback regarding the evaluation of the described 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] M. Adams, A. ter Hofstede, D. Edmond, W. M. v. d. Aalst, Dynamic and extensible exception handling for workflows: A service-oriented implementation., Tech. Rep. BPM-07-03, BPMcenter.org (2007).
- [2] M. Adams, A. ter Hofstede, D. Edmond, W. van der Aalst, A Service-Oriented Implementation of Dynamic Flexibility in Workflows., in: Proc. Coopis'06, 2006.

- [3] M. Adams, A. ter Hofstede, W. van der Aalst, D. Edmond, Dynamic, Extensible and Context-Aware Exception Handling for Workflows., in: Proc. CoopIS'07 (to appear), 2007.
- [4] C. Alexander, S. Ishikawa, M. Silverstein, A Pattern Language, Oxford University Press, New York, 1977.
- [5] A. Barros, M. Dumas, A. ter Hofstede, Service Interaction Patterns., in: Proc. BPM'05, 2005.
- [6] S. Bassil, R. K. Keller, P. G. Kropf, A workflow-oriented system architecture for the management of container transportation, in: Proc. BPM'04, 2004.
- [7] R. Bobrik, M. Reichert, T. Bauer, View-based process visualization, in: Proc. BPM'07, 2007.
- [8] F. Casati, Models, semantics, and formal methods for the design of workflows and their exceptions., Ph.D. thesis, Milano (1998).
- [9] F. Casati, S. Ceri, B. Pernici, G. Pozzi, Workflow evolution, Data and Knowledge Engineering 24 (3) (1998) 211–238.
- [10] P. Dadam, M. Reichert, K. Kuhn, Clinical workflows - the killer application for process-oriented information systems?, in: Proc. BIS'00, 2000.
- [11] W. Deiters, V. Gruhn, The funsoft net approach to software process management., Int'l Journal of Software Engineering and Knowledge Engineering 4 (2) (1994) 229–256.
- [12] W. V. der Aalst, A. ter Hofstede, YAWL: Yet Another Workflow Language., Information Systems 30 (4) (2005) 245–275.
- [13] E. Dijkstra, A Discipline of Programming., Prentice-Hall, 1976.
- [14] D. Domingos, A. Rito-Silva, P. Veiga, Authorization and access control in adaptive workflows., in: ESORICS 2003, 2003.
- [15] M. Dumas, A. ter Hofstede, W. van der Aalst (eds.), Process Aware Information Systems, Wiley Publishing, 2005.
- [16] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.
- [17] M. Golani, A. Gal, Optimizing Exception Handling in Workflows Using Process Restructuring, in: Proc. BPM'06, 2006.
- [18] K. Göser, M. Jurisch, H. Acker, U. Kreher, M. Lauer, S. Rinderle-Ma, M. Reichert, P. Dadam, Next-generation Process Management with ADEPT2., in: Demo Proc. BPM'07, 2007.
- [19] V. Gruhn, Validation and verification of software process models., Ph.D. thesis, University of Dortmund (1991).
- [20] C. Günther, S. Rinderle, M. Reichert, W. van der Aalst, Change Mining in Adaptive Process Management Systems, in: Proc. CoopIS'06, 2006.

- [21] J. Hagemeyer, T. Hermann, K. Just, S. Rüdiger, Flexibilität bei Workflow-Management-Systemen, in: Software-Ergonomie '97, 1997.
- [22] M. Hammer, S. Stanton, The Reengineering Revolution – The Handbook, Harper Collins Publ., 1995.
- [23] Y. Han, Software Infrastructure for Configurable Workflow Systems., Ph.D. thesis, Univ. of Berlin (1997).
- [24] S. Jablonski, K. Stein, M. Teschke, Experiences in Workflow Management for Scientific Computing, in: Proc. DEXA '97, 1997.
- [25] B. Karbe, N. Ramsperger, Influence of Exception Handling on the Support of Cooperative Office Work., in: Proc. IFIP WG 8.4 Conf. on Multi-User Interfaces and Applications, 1990.
- [26] K. Kochut, J. Arnold, A. Sheth, J. Miller, E. Kraemer, B. Arpinar, J. Cardoso, IntelliGEN: A Distributed Workflow System for Discovering Protein-Protein Interactions, *Distrib. Parallel Databases* 13 (1) (2003) 43–72.
- [27] R. Lenz, M. Reichert, IT Support for Healthcare Processes - Premises, Challenges, Perspectives., *Data and Knowledge Engineering* (1) (2007) 39–58.
- [28] F. Leymann, D. Roller, Production Workflow., Prentice Hall, 2000.
- [29] R. Lu, S. Sadiq., A Survey on Comparative Modelling Approaches., in: In Proc. BIS2007, 2007.
- [30] R. Lu, S. Sadiq., On the Discovery of Preferred Work Practice through Business Process Variants., in: Proc. ER2007 (to appear), 2007.
- [31] R. Lu, S. W. Sadiq, Managing process variants as an information resource., in: Proc. BPM06, 2006.
- [32] M. Minor, D. Schmalen, A. Koldehoff, R. Bergmann, Structural adaptation of workflows supported by a suspension mechanism and by case-based reasoning., in: Proc. WETICE'07, 2007.
- [33] M. Minor, A. Tartakovski, R. Bergmann, Representation and Structure-based Similarity Assessment for Agile Workflows., in: Proc. ICCBR'07, 2007.
- [34] H. Mourao, P. Antunes, Supporting effective unexpected exceptions handling in workflow management systems, in: Proc. SAC'07, ACM Press, 2007.
- [35] D. Müller, J. Herbst, M. Hammori, M. Reichert, IT support for release management processes in the automotive industry., in: Proc. BPM'06, Vienna, 2006.
- [36] R. Müller, U. Greiner, E. Rahm, AGENTWORK: A workflow system supporting rule-based workflow adaptation., *Data & Knowledge Engineering* 51 (2) (2004) 223–256.
- [37] B. Mutschler, M. Reichert, J. Bumiller, Why Process-Orientation is Scarce: An Empirical Study of Process-oriented Information Systems in the Automotive Industry., in: Proc. EDOC'06, 2006.

- [38] Object Management Group, Business Process Modeling Notation Specification.
- [39] M. Pesic, M. Schonenberg, N. Sidorova, , W. van der Aalst, Constraint-Based Workflow Models: Change Made Easy., in: CoopIS'07 (to appear), 2007.
- [40] Process Mining Research, [www.processmining.org](http://www.processmining.org) (2005).
- [41] F. Puhlmann, M. Weske, Using the Pi-Calculus for Formalizing Workflow Patterns., in: Proc. BPM'05, 2005.
- [42] M. Reichert, Dynamic changes in workflow-management-systems., Ph.D. thesis, University of Ulm, Computer Science Faculty, (in German) (2000).
- [43] M. Reichert, P. Dadam, ADEPT<sub>flex</sub> – Supporting Dynamic Changes of Workflows Without Losing Control., JIIS 10 (2) (1998) 93–129.
- [44] M. Reichert, P. Dadam, T. Bauer, Dealing with Forward and Backward Jumps in Workflow Management Systems, Software and System Modeling 1 (2) (2003) 37–58.
- [45] M. Reichert, S. Rinderle, U. Kreher, P. Dadam, Adaptive Process Management with ADEPT2., in: Proc. ICDE'05, 2005.
- [46] H. Reijers, J. Rigter, W. van der Aalst, The case handling case., International Journal of Cooperative Information Systems. 12 (3) (2003) 365—391.
- [47] H. Reijers, W. van der Aalst, The Effectiveness of Workflow Management Systems: Predictions and Lessons Learned., International Journal of Information Management (5) (2005) 457–471.
- [48] S. Rinderle, Schema evolution in process management systems, Ph.D. thesis, University of Ulm (2004).
- [49] S. Rinderle, M. Reichert, P. Dadam, Correctness Criteria for Dynamic Changes in Workflow Systems – A Survey., Data and Knowledge Engineering 50 (1) (2004) 9–34.
- [50] S. Rinderle, M. Reichert, P. Dadam, Flexible support of team processes by adaptive workflow systems., Distributed and Parallel Databases 16 (1) (2004) 91–116.
- [51] S. Rinderle, M. Reichert, M. Jurisch, U. Kreher, On Representing, Purging, and Utilizing Change Logs in Process Management Systems., in: Proc. BPM'06, 2006.
- [52] S. Rinderle, B. Weber, M. Reichert, W. Wild, Integrating Process Learning and Process Evolution - A Semantics Based Approach., in: Proc. BPM 2005, 2005.
- [53] N. Russell, A. ter Hofstede, D. Edmond, W. van der Aalst, Workflow data patterns, Tech. Rep. FIT-TR-2004-01, Queensland Univ. of Techn. (2004).
- [54] N. Russell, A. ter Hofstede, D. Edmond, W. van der Aalst, Workflow resource patterns, Tech. Rep. WP 127, Eindhoven Univ. of Technology (2004).

- [55] N. Russell, A. ter Hofstede, W. van der Aalst, N. Mulyar, Workflow Control-Flow Patterns: A Revised View., Tech. rep., BPMcenter.org (2006).
- [56] N. Russell, W. van der Aalst, A. ter Hofstede, Exception Handling Patterns in Process-Aware Information Systems, in: Proc. CAiSE'06, 2006.
- [57] N. Russell, W. M. P. van der Aalst, A. H. M. ter Hofstede, P. Wohed, On the suitability of UML 2.0 activity diagrams for business process modelling, in: Proc. APCCM '06, 2006.
- [58] S. Sadiq, W. Sadiq, M. Orlowska, Pockets of Flexibility in Workflow Specifications., in: Proc. ER'01, 2001.
- [59] S. Sadiq, W. Sadiq, M. Orlowska, A Framework for Constraint Specification and Validation in Flexible Workflows, *Information Systems* 30 (5) (2005) 349 – 378.
- [60] P. Stoll, E-Procurement - Basics, Standards, Practice (E-Procurement - Grundlagen, Standards und Praxis), Ph.D. thesis, Diploma Thesis, University of Ulm (2005).
- [61] D. Strong, S. Miller, Exceptions and Exception Handling in Computerized Information Processes., *ACM–TOIS* 13 (2) (1995) 206–233.
- [62] L. H. Thom, A Pattern Based Approach for Business Process Modeling., Ph.D. thesis, Universidade Federal do Rio Grande do Sul (2006).
- [63] L. H. Thom, J. M. Lau, C. Iochpe, J. Mendling, Extending Business Process Modeling Tools with Workflow Pattern Reuse., in: Proc. ICEIS'07, 2007.
- [64] R. V. Glabbeek, U. Goltz, Refinement of actions and equivalence notions for concurrent systems., *Acta Informatica* 37 (4–5) (2001) 229–327.
- [65] W. Van der Aalst, The Application of Petri Nets to Workflow Management., *The Journal of Circuits, Systems and Computers*.
- [66] W. van der Aalst, M. Pesic, DecSerFlow: Towards a Truly Declarative Service Flow Language., Tech. rep., BPMcenter.org (2006).
- [67] W. Van der Aalst, A. ter Hofstede, B. Kiepuszewski, A. Barros, Workflow Patterns, *Distributed and Parallel Databases* 14 (1) (2003) 5–51.
- [68] W. Van der Aalst, K. van Hee, *Workflow Management*, MIT Press, 2002.
- [69] W. Van der Aalst, M. Weske, D. Grünbauer, Case handling: A new paradigm for business process support., *Data and Knowledge Engineering*. 53 (2) (2005) 129–162.
- [70] Wave-front BV, Flower 3 Designer's Guide.
- [71] B. Weber, M. Reichert, W. Wild, Case-Base Maintenance for CCBR-based Process Evolution, in: Proc. ECCBR'06, 2006.
- [72] B. Weber, M. Reichert, W. Wild, S. Rinderle, Balancing Flexibility and Security in Adaptive Process Management Systems., in: Proc. CoopIS'05, 2005.

- [73] B. Weber, S. Rinderle, M. Reichert, Change patterns and change support features in process-aware information systems., in: Proc. CAiSE'07, 2007.
- [74] B. Weber, S. Rinderle, W. Wild, M. Reichert, CCBR–Driven Business Process Evolution., in: Proc. ICCBR'05, Chicago, 2005.
- [75] B. Weber, W. Wild, R. Breu, CBRFlow: Enabling adaptive workflow management through conversational cbr., in: Proc. ECCBR'04, 2004.
- [76] B. Weber, W. Wild, M. Lauer, M. Reichert, Improving exception handling by discovering change dependencies in adaptive process management systems., in: Business Process Management Workshops 2006, 2006.
- [77] B. Weber, W. Wild, M. Reichert, P. Dadam, ProCycle Integrierte Unterstützung des Prozesslebenszyklus., KI-Zeitung (to appear).
- [78] M. Weske, Workflow Management Systems: Formal Foundation, Conceptual Design, Implementation Aspects., University of Münster, Germany, Habil Thesis (2000).
- [79] M. Weske, Business Process Management: Concepts, Methods, Technology., Springer, 2007.
- [80] W. Wild, R. Wirtensohn, B. Weber, Dynamic engines a flexible approach to the extension of legacy code and process-oriented application development., in: Proc. WETICE'06, 2006.
- [81] P. Wohed, W. M. P. van der Aalst, M. Dumas, A. H. M. ter Hofstede, N. Russell, On the Suitability of BPMN for Business Process Modelling, in: Proc. BPM'06, 2006.
- [82] F. Zhang, E. D'Hollander, Using Hammock Graphs to Structure Programs., IEEE Transactions on Software Engineering 30 (2004) 231–245.

## A Evaluation Details

In the following the detailed evaluation results for each of the considered approaches are discussed.

### A.1 Evaluation Details: ADEPT2 / CBRFlow

ADEPT2 is a process management system which allows for the support of dynamic change both at the process type and process instance level. In the ProCycle project [77] ADEPT2 has been integrated with the case-based reasoning component of CBRFlow [75] to foster change annotation and reuse.

**Support of Adaptation Patterns.** ADEPT2 allows for structural process changes using adaptation patterns. Generally, ADEPT2 enables such changes 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 hammocks (Design Choice B[1,2,3]).<sup>4</sup>

In detail: In terms of adaptation patterns ADEPT2 supports the insertion of process fragments (AP1) to a process schema or process instance schema respectively. Respective fragments can be added serially, conditionally, or in parallel (Design Choice C[1,2(a+b)]). 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 Process Fragment) are not directly supported by the current ADEPT2 system, but can be "simulated" 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 ADEPT2 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 AP1 and AP3. Finally, adaptation patterns AP11 (Add Control Dependency), AP12 (Remove Control Dependency), and AP13 (Update Condition) are supported. Details about ADEPT2 change operations (e.g., formal semantics, implementation, etc.) can be found in [42,48].

It is important to mention that the block-structured modeling approach used in ADEPT2 significantly facilitates the implementation and use of adaptation patterns (for details see [43]).

**Support of Patterns for Changes in Predefined Regions.** ADEPT2 provides currently no support for patterns for changes in predefined regions.

**Schema Evolution, Version Control and Instance Migration (Change Feature F1).** ADEPT2 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, ADEPT2 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 ADEPT2 process meta model and is based on a relaxed notion of trace equivalence [50]. Particularly, it considers

---

<sup>4</sup> 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.

all kinds of control flow changes and works correctly in connection with loop backs as well. To enable efficient compliance checks, for each high-level change operation (or adaptation pattern respectively) ADEPT2 provides precise and easy to check compliance conditions [50]. Finally, efficient procedures exist for correctly adapting the states of compliant process instances when migrating them to the new schema version.

**Instance-Specific Changes (Change Feature F2).** From the very beginning, ADEPT2 has supported ad-hoc changes at the process instance level [43]. These ad-hoc changes are based on the aforementioned adaptation patterns (F2[1a+b]). 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 ADEPT2 this is achieved based on well-defined pre- and post-conditions for the supported adaptation patterns. 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 is hidden from users. In general, structural changes in ADEPT2 can be applied in both a temporary or a permanent manner.

**Correctness of Changes (Change Feature F3).** One of the major design goals of the ADEPT2 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 [43,50]. 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 ADEPT2 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 captures data about the applied adaptation patterns and their parameterizations, the latter covers contextual knowledge about the changes (e.g., change reason and change performer). ADEPT2 provides powerful support for maintaining, purging, and utilizing such logs, and for annotating log entries [51]. In the ProCycle project [77], the ADEPT2 system has been integrated with the conversational case-based reasoning component of CBRFlow [75]. Among other things, this integration allows enriching change logs with contextual information [52,74,71]. In the MinADEPT project, first techniques and tools for analyzing and mining change logs have been provided [20] (F4[3]).

**Access Control (Change Feature F5).** In respect to access control ADEPT2 allows restricting changes to authorized users (F5[1]). In addition, authorizations can be defined at the level of single adaptation 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, which has been developed as part of the SecServ project, can be found in [72].

**Change Reuse (Change Feature F6).** The integration of ADEPT2 with case-based reasoning techniques 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 [52,76,71]. If no change reuse is possible, the user can specify a new ad-hoc modification using the patterns provided by ADEPT2 (F6[+]).

**Change Concurrency Control (Change Feature F7).** ADEPT2 supports concurrent changes of single process instances through optimistic concurrent change techniques (F7[3]). In addition, ADEPT2 supports the propagation of type changes to process instances, which have already been individually modified (F7[4]) [48].

## A.2 Evaluation Details: CAKE2

CAKE2 is a process management system, which has been recently developed in the context of complex engineering processes (i.e., digital chip design). The primary focus of CAKE2 is on instance-specific changes and late modeling.

**Support of Adaptation Patterns.** The CAKE2 approach [32,33] does not provide any high-level change operations and consequently does not directly support any of the adaptation patterns. Nevertheless, CAKE2 enables structural changes at the process instance level based on a set of change primitives (Design Choice A[1]). Process type changes have not been addressed by CAKE 2. All change primitives can operate on atomic activities and sub processes (Design Choice B[1,2]). In particular, CAKE2 supports change primitives PR1-PR5 (i.e., insert, delete and move single nodes and edges).

**Support of Patterns for Changes in Predefined Regions.** CAKE2 allows late modeling of process fragments. Thereby, the user can select process fragments from the repository (Design Choice A[1]) and make use of the full expressiveness of the process editor (Design Choice B[1]). The modeling can be done any time before the placeholder activity gets enabled or upon enabling (Design Choice C [2,3]). In general, the process modeling starts with an empty template (Design Choice D [1]).

**Schema Evolution, Version Control and Instance Migration (Change Feature F1).** Currently schema evolution is not supported in CAKE2 (F1[1]).

In the application context CAKE2 is currently used missing version control is not very critical. Process instances are distinct from each other, rather long running, and are not derived from a process schema. However, when applying CAKE2 in the context of repetitive processes this would constitute a problem.

**Instance-Specific Changes (Change Feature F2).** CAKE2 allows users to perform unplanned ad-hoc changes with the process editor using change primitives. Based on this all 14 adaptation patterns can be "simulated" (Change Feature F2[1b,2b]). In addition to unplanned changes, CAKE2 also provides support for preplanned changes. In general, in CAKE2 changes are conducted in a permanent manner.

**Correctness of Changes (Change Feature F3).** CAKE ensures correctness of ad-hoc changes by only allowing for changes to not yet enabled parts of a process instance (i.e., compliance of the process instance with the new schema is guaranteed by construction). Parts which have been already executed cannot be modified by ad-hoc changes (F3[+]).

**Traceability and Analysis (Change Feature F4).** Traceability is ensured through process execution logs (F4[1]). In addition, CAKE2 supports the annotation of changes as cases. Thereby, context as well as information related to the status of the process instance to be changed are recorded (F4[2]).

**Access Control (Change Feature F5).** Access control is currently not addressed by CAKE2 (F5[-]).

**Change Reuse (Change Feature F6).** CAKE2 provides a CBR component supporting the reuse of past process changes. Thereby, a case consists of a problem description (i.e., the process before a revision including its context) and a solution (i.e., the respective revision). For retrieving similar changes context information and the status of the process instance to be modified are taken into consideration (F6[+]) [33].

**Change Concurrency Control (Change Feature F7).** CAKE2 supports concurrency of instance-specific changes through pessimistic locking (F7[3]). The breakpoint mechanism provided by CAKE2 allows that only those parts of a process instance, which have to be modified are suspended, while parallel branches, not affected by the change, can proceed with their execution.

### A.3 Evaluation Details: HOON

HOON [23] is a Petri-net based tool for modeling and executing well structured workflows. In particular, HOON provides support for the late selection

of process fragments.

**Support of 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 of Patterns for Changes in Predefined Regions.** HOON enables process flexibility by supporting late selection of process fragments, i.e., 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 design choices, HOON is comparable with the Worklet/Exlet approach. An activity implementation can be selected in a fully automated way based on well-defined procedures and workflow run-time 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 after enabling the placeholder activity (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. In principle, the late selection support of HOON allows users 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 ADEPT2, WASA, or Staffware. The same consideration also hold for all other approaches supporting late binding or late modeling. However, when structural adaptations have to be accomplished, HOON requires the modeler to overwrite the process schema or to save the process schema as a new schema (F1[1]).

**Instance-Specific Changes (Change Feature F2).** Although HOON does not support any of the adaptation patterns modifications of running process instances become possible through late selection of process fragments (F2[2a]). In general, these preplanned changes are performed in a temporary manner. In terms of structural changes, HOON allows "simulating" part of the functionality described in AP1 through workarounds. Placeholder activities allow inserting a process fragment into a running process instance by selecting an activity implementation. Generally, a placeholder activity can be positioned between two fragments or parallel to an existing one. By substituting this placeholder activity during run-time with a concrete (sub) process fragment, in principle, a serial and a parallel insertion can be "simulated" (cmp. Design Choice C[1,2] of pattern AP1). However, the insertion is restricted to the placeholder activity, i.e., the position of the dynamically added activity in the process schema has to be predefined at build-time. In addition, 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.

**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 analysis of the respective HOON nets) (F3[+]). If a particular activity implementation is not available, alternative activity implementations will be automatically assigned, or the user will be involved to manually choose 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 are not supported.

**Access Control (Change Feature F5).** Besides the automated selection of activity implementations, HOON allows restricting 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 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.

**Change Concurrency Control (Change Feature F7).** As HOON restricts changes to placeholder activities concurrency of changes can be easily achieved. Changes to different placeholder activities can be performed concurrently (F7[3]).

#### A.4 Evaluation Details: MOVE

MOVE [11,19] is an approach comparable to HOON, which is based on high-level petri-nets. In particular, MOVE provides support for the late modeling of process fragments.

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

**Support of predefined change patterns.** Similar to the PoF approach MOVE only allows for changes in restricted and predefined process areas through late modeling (cf. Appendix 9.5). 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 for model construction can be defined. For modeling the placeholder activity the same constructs are used than for defining the process schema (Design Choice B[1]). More precisely, modeling is based on a high-level Petri Net formalism (FunSoft Nets) [11,19]. 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).** The challenges related to process type changes and process schema evolution are not addressed in MOVE (F1[1]).

**Instance-Specific Changes (Change Feature F2).** MOVE does not support any of the adaptation patterns directly. However, the functionality of Adaptation Pattern 1 (Insert Process Fragment) can be partially "simulating" (cf. Appendix 9.3). Support for preplanned changes is provided through its late modeling capabilities (F2[2a]). These changes are performed in a temporary manner.

**Correctness of Changes (Change Feature F3).** All process fragments created by late modeling constitute FunSoft process models. Due to the use of this Petri Net formalism, model correctness can be ensured based on conventional model checking techniques [19] (F3[+]).

**Traceability and Analysis (Change Feature F4).** Process instances are stored in the process repository. As no unplanned ad-hoc changes are supported this is sufficient for ensuring traceability (F4[1]). Change annotations and change mining are not addressed.

**Access Control (Change Feature F5).** MOVE allows restricting changes to particular users through assigning the placeholder activity to a specific role (F5[1]). No further restrictions can be specified in MOVE. For different placeholder activities different authorizations can apply (F5[3]).

**Change Reuse (Change Feature F6).** Change reuse is unsupported (F6[-]).

**Change Concurrency Control (Change Feature F7).** Like in HOON changes are restricted to placeholder activities. As changes are always local to the placeholder activities, different placeholder activities can be concurrently modified (F7[3]).

## A.5 Evaluation Details: Pockets of Flexibility (PoF)

The core idea of this approach is to extend traditional process schemes by so called *Pockets 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 [59,58].

**Support of Adaptation Patterns.** The provision of adaptation patterns is not in the focus of PoF. None of the described adaptation patterns is supported through high-level change operations and changes can only be applied within preplanned regions.

**Support of Predefined Change Patterns.** The PoF approach enables flexibility through change pattern PP2 (Late Modeling of Process Fragments) [58,59]. Optionally, constraints can be defined regarding the selection of activities as well as their ordering (Design Choice A[1,2]). Finally, for the late modeling of a placeholder activity 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]). This facilitates late modeling of process fragments for end users and thus increases user acceptance.

Late modeling starts when the placeholder activity 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 late modeling the newly defined process fragment is validated against the modeling constraints and then 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).** The PoF approach does not address schema evolution. Instead, the existence of placeholder activities allows users to individually specify parts of the process model or even the whole process model during run-time. Though this individualization reduces the frequency of structural changes of the process model, the need for changing the "core process" (i.e., the toplevel process) cannot be completely discarded. As the PoF approach does not provide versioning control, schema modifications require the user to overwrite the existing schema (F1[1]) or to save the modified process schema as a new schema.

**Instance-Specific Changes (Change Feature F2).** As aforementioned the PoF approach does not offer direct support for any adaptation pattern. However, like HOON and MOVE the PoF approach allows the partial realization of the functionality described by adaptation pattern AP1 (Insert

Process Fragment) using placeholder activities (cf. Appendix 9.3). Changes in preplanned regions can be performed through its late modeling support. For late modeling a process editor based on change primitives is provided, which allows for the validation of the modeled process fragment against existing modeling constraints. As PoF does not consider loops all changes are permanent (F2[2b]).

**Correctness of Changes (Change Feature F3).** The PoF approach ensures change correctness as process fragments created by late modeling are validated before they get instantiated (F3[+]).

**Traceability and Analysis (Change Feature F4).** Each process fragment resulting from late modeling is stored in the process repository as process variant [31,30]. 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 the PoF approach 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 the use of single activities within a placeholder activity. Consequently, the kind of changes that may 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 focusing on control flow [31,30].

**Change Concurrency Control (Change Feature F7).** Like in HOON changes are restricted to placeholder activities. As changes are always local to the placeholder activities, different placeholder activities can be concurrently modified (F7[3]).

#### A.6 Evaluation Details: WASA2

Like ADEPT2, WASA2 constitutes an adaptive process management system supporting both process type and process instance level changes.

**Support of Adaptation Patterns.** The design of high-level change operations was out of the scope of the WASA2 project. Thus, no direct support for adaptation patterns is provided. However, WASA2 enables structural process

changes using change primitives. In particular, change primitives PR1-PR4 (i.e., insert/delete node, insert/delete edge) are supported. By using these change primitives changes can be performed at both the process type and the process instance level (Design Choice A[1,2]). All change primitives can operate on atomic activities and sub processes due to the object-oriented approach followed by WASA2 (Design Choice B[1,2]).

**Support of Predefined Change Patterns.** No support for patterns for changes in predefined regions is provided.

**Schema Evolution, Version Control and Instance Migration (Change Feature F1).** WASA2 provides advanced support for version control of process schemes: If a schema is changed, a new schema version is created. Further, it is checked which process instances may migrate to the new schema version based on a well-defined correctness criterion (Design Choice F1[3, 5]).

**Instance-Specific Changes (Change Feature F2).** In WASA2 unplanned changes can be performed using a workflow editor and a set of change primitives (F2[1b]). All changes are applied in a permanent manner.

**Correctness of Changes (Change Feature F3).** Correctness of dynamically changed process instances is ensured by the mentioned correctness criterion (F3[+]). Based on it both control and data flow correctness are guaranteed.

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

**Access Control (Change Feature F5).** Changes can be constraint by role-based access control in WASA2. For example, only users with role *process administrator* are authorized to conduct process type changes (F5[1]).

**Change Reuse (Change Feature F6).** Change reuse is not supported in WASA2 (F6[-]).

**Change Concurrency Control (Change Feature F7).** WASA2 provides change concurrency control by prohibiting concurrent changes (F7[2]). In case of an instance-specific change the entire process instance is locked.

## A.7 Evaluation Details: WIDE

WIDE [8,9] is one of the first workflow management systems supporting schema evolution.

**Support of Adaptation Patterns.** Generally, WIDE enables the application of adaptation patterns only at the process type level (Design Choice A[2]). Thereby, the respective pattern operates on atomic activities (Design Choice B[1]). In detail, WIDE supports the insertion of process fragments (AP1) which can be added in a serial and conditional manner (Design choice C[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 adaptation patterns 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 [9]. Obviously, non-supported adaptation patterns (e.g., AP3 and AP5) could be easily implemented in WIDE by the combined use of existing patterns. For a more detailed description of the supported adaptation patterns see [8,9].

**Support of Predefined Change Patterns.** In terms of predefined change patterns WIDE provides support for multi-instance activities (PP4) [8]. 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: When a process schema is changed, a new process schema version is created (Design Choice F1[3]). Further, it is checked which instances can migrate to the new version according to the so called *compliance criterion* (Design Choice F1[5]).

**Instance-Specific Changes (Change Feature F2).** WIDE does not address ad-hoc changes of process instances. Pre-planned changes are supported through the *Multi Instance Activity* pattern. All changes pattern are supported in a permanent way (F2[2b]).

**Correctness of Changes (Change Feature F3).** Correctness and consistency of (compliant) process instances are 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 [9]).

**Traceability and Analysis (Change Feature F4).** The existence of process schema versions ensures traceability of process type changes (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[-]).

**Change Concurrency Control (Change Feature F7).** As WIDE does not support any instance-specific changes, except for the *Multi Instance Activity* pattern, this feature is no applicable.

#### A.8 Evaluation Details: YAWL/Worklets/Exlets

YAWL is an open source workflow engine implementing workflow patterns. To increase flexibility the Worklet/Exlet approach has been implemented as a service for YAWL. Essentially, a Worklet is a process fragment that acts as a late-bound sub process for an enabled activity [2]. Exlets are exception handling processes, which are invoked if specific events occur [1,3].

**Support of 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) [2]. 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]). Other adaptation patterns are not supported directly, as no high-level change operations are provided. However, certain unplanned ad-hoc changes can be realized by using change primitives (see Change Feature F2).

**Support of 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 [2]. 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 will be executed as "ordinary" task. In general, 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, con-

sisting 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 supports version control of process schemes. Thereby, ongoing 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 late selection reduces the need for structurally changing the toplevel process (i.e., the "parent" schema).

**Instance-Specific Changes (Change Feature F2).** This approach provides support for unplanned as well as preplanned changes. As the selection of Worklets depends on rules and Exlets are triggered by events changes can be performed in a temporary and permanent manner (F2[1a+b,2a+b]).

Using Worklets parts of the functionality covered by adaptation pattern AP1 can be "simulated" through its late selection concept. Thereby, similar considerations hold than for HOON, MOVE and the PoF approach (cf. Appendix 9.3, Appendix 9.4 and Appendix 9.5, Change Feature F2).

Finally, the Exlet extension provides powerful exception handling mechanisms which allow coping with both expected and unexpected exceptions [1,3]. Using Exlets further adaptation pattern can be "simulated" with workarounds. The approach allows invoking exception handling processes at the occurrence of particular events. The exception handling processes are thereby executed as parallel threads of control. For example, to realize a serial insert of activity X before activity B and after activity A 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 resumed. However, through the lack of high-level change operations changes can become very complex and might therefore affect usability of the PAIS.

**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 will be ensured through the inbuilt verification and validation feature of the YAWL Process Editor. In case Exlets are used in addition, proper termination of the process cannot always be guaranteed. 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).

**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 rule creation (F4[2]). Change mining is 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-enabled activity (F5[2]); i.e., for each worklet-enabled activity a repertoire (i.e., a collection) of Worklets is maintained (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 (F6[+]).

**Change Concurrency Control (Change Feature F7).** Like in HOON and MOVE changes are restricted to placeholder activities. As changes are always local to the placeholder activities, different placeholder activities can be concurrently modified (F7[3]).

#### A.9 Evaluation Details: Flower

Flower is a process management system based on the case-handling paradigm. Our evaluation of the Flower case handling system is based on Version 3.1 of the software. Note that this evaluation only considers flexibility in respect to control-flow aspects. Especially in the case of Flower this only provides a partial picture of what the system can offer in terms of flexibility (e.g., dynamic changes of role assignments or dynamically adding or deleting forms).

**Support of Adaptation Patterns.** Flower provides explicit support for the *Delete* adaptation pattern (AP2) through 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]).

**Support of Predefined Change Patterns.** Pattern PP4 (*Multi-Instance Activity*) is supported through the concept of *dynamic subplans*. Like in Staffware (cf. Appendix 9.10) and WIDE (cf. Appendix 9.7), the number of activity instances can either be fixed during build-time or be defined based on process 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 allows for the co-existence of process instances of different schema versions (F1[3]). In addition, Flower allows users with respective authorizations 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]). However, according to the Flower Designer’s Guide [70] overwriting a schema is not permitted for control-flow changes (e.g., adding or deleting activities or changing the ordering of activities). The only exception is the changing of guards. As Flower does not prohibit users from overwriting the process schema after having changed the control flow, this can potentially lead to inconsistencies. In summary, both options – removing running instances or overwriting process schemes in an uncontrolled manner – do not provide a satisfactory solution.

**Instance-specific Changes (Change Feature F2).** Flower provides support for unplanned changes through skipping of activities. 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. In addition, changes to predefined regions are supported through the *Multi-Instance Activity* pattern (F2[1b,2b]). Changes are thereby performed in a permanent manner.

**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 (F4[1]).

**Traceability and Analysis (Change Feature F4).** Traceability is ensured in Flower as the completed instances are maintained in the system (F5[1]).

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

**Change Reuse (Change Feature F6).** No change reuse is supported in Flower (F6[-]).

**Change Concurrency Control (Change Feature F7).** In Flower concurrent changes are not possible as users are not allowed to work simultaneously on the same case (F7[2]).

#### A.10 Evaluation Details: Staffware

Staffware is a widely used process management system. The evaluation of Staffware is based on Version 10 of Staffware.

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

**Support of Predefined Change Patterns** Regarding predefined change patterns Staffware provides support for pattern PP1 (*Late Selection of Process Fragments*) and pattern PP4 (*Multi-Instance Activity*), which have been recently introduced. The Late Selection of Process Fragments is supported through the *Graft Activity* which is a feature of the Staffware Process Orchestrator as well. The selection of the activity implementation can either be automated or be done manually 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 provided by the Staffware Process Orchestrator. The number of activity instances can either be fixed during build-time or be defined based on process 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[3]). 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 (F1[4]).

**Instance-specific Changes (Change Feature F2).** Staffware provides support for unplanned changes, whereas no structural changes to the process instance schema can be performed (F2[2b]). However, parts of adaptation patterns AP1 can be realized through workarounds like in HOON, MOVE, or PoF (cf. Sections 9.3). 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 surprises during run-time cannot be avoided. When testing the respective instance migration feature of Staffware in our lab, several process instances ended up in deadlocks.

**Traceability and Analysis (Change Feature F4).** As no ad-hoc changes are supported an audit trail is sufficient for traceability of process instances. Additionally, version management of process schemes allows for traceability of process type changes as well (F4[1]).

**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 Staffware (F6[-]).

**Change Concurrency Control (Change Feature F7).** Like in HOON, MOVE and PoF changes are restricted to placeholder activities. As changes are always local to the placeholder activities, different placeholder activities can be concurrently modified (F7[3]).