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

Barbara Weber [a,*], Manfred Reichert [b,c] Stefanie Rinderle-Ma [b]

[a] *Department of Computer Science, University of Innsbruck, Technikerstraße 21a, 6020 Innsbruck, Austria*

[b] *Institute of Databases and Information Systems, Ulm University James-Franck-Ring, 89069 Ulm, Germany*

[c] *Information Systems Group, University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands*

Abstract

Companies increasingly adopt process-aware information systems (PAISs), which offer promising perspectives for more flexible enterprise computing. The emergence of different process support paradigms and the lack of methods for comparing existing approaches enabling PAIS changes have made the selection of adequate process management technology difficult. This paper suggests a set of 18 change patterns and 7 change support features to foster the systematic comparison of existing process management technology in respect to process change support. While the proposed patterns are all based on empirical evidence from several large case studies, the suggested change support features constitute typical functionalities provided by flexible PAISs. Based on the proposed change patterns and features, we provide a detailed analysis and evaluation of selected approaches from both academia and industry. The presented work will not only facilitate the selection of technologies for realizing flexible PAISs, but can also be used as a reference for implementing flexible PAISs.

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

* Corresponding author.

Email addresses: barbara.weber@uibk.ac.at (Barbara Weber), manfred.reichert@uni-ulm.de (Manfred Reichert), stefanie.rinderle@uni-ulm.de (Stefanie Rinderle-Ma).

1. Introduction

In today's dynamic business world the economic success of an enterprise depends on its ability to react to changes in its environment in a quick and flexible way [1]. Causes for these changes can be manifold and include the introduction of new laws or changes in customers' attitudes. For these reasons companies have recognized business agility as a competitive advantage, which is fundamental for being able to cope with business trends like increasing product and service variability, faster time-to-market, and business-on-demand.

Process-aware information systems (PAISs) offer promising perspectives in this respect, and a growing interest in aligning information systems in a process-oriented way can be observed [2,3]. In contrast to data- or function-centered information systems, PAISs are characterized by a strict separation of process logic and application code. In particular, most PAISs describe process logic explicitly in terms of a process model providing the schema for process execution. Usually, the core of the process layer is built by a process management system, which provides generic functionality for modeling, executing, and monitoring processes. This allows for a separation of concerns, which is a well established principle in computer science for increasing maintainability and for reducing cost of change [4]. Changes to one layer often can be performed without affecting the other layers. For example, modifying the application service which implements a particular process step (i.e., 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 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 change has been identified as one of the critical success factors for any PAIS [5,6]. PAISs facilitate changes significantly through the above described separation of concerns. According to a recent study conducted among several Dutch companies, however, enterprises are reluctant to change PAIS implementations once they are running properly [7]. High complexity and high cost of change are mentioned as major reasons for not fully leveraging the potential of PAISs. To overcome this problem flexible PAISs are needed, enabling companies to capture real-world processes adequately without leading to a mismatch between the computerized business processes and those running in reality [8,5]. In particular, the introduction of a PAIS must not freeze existing business processes [9]. 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 [10]. In addition, PAISs must support users to deal with uncertainty by deferring decisions to run-time as required.

1.1. *Problem Statement*

The need for flexible and easily adaptable PAISs has been recognized and several competing paradigms for addressing process changes and process flexibility have been

developed. As example consider adaptive processes [11–13], case handling [14], declarative processes [15], and late binding and modeling [12,16–18]. However, there still is a lack of methods for systematically comparing the change frameworks provided by existing process support technologies. This makes it difficult for PAIS engineers [1] to assess maturity and change capabilities of those technologies, often resulting in wrong decisions and expensive misinvestments.

To make PAISs better comparable, *workflow patterns* have been introduced [19]. Respective patterns provide 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 [19,20], the data flow [21], and the resource perspective [22]. Obviously, broad support for workflow patterns allows for building more flexible PAISs. 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 [23]. The latter is to some degree addressed by exception handling patterns [24], which describe different ways for coping with the exceptions that occur during process execution (e.g., activity failures). Patterns like *Rollback* or *Redo* allow users to deal with exceptional situations by changing the state of a running process; usually, they do not affect the structure of a process. In many cases, changing the state of a running instance is not sufficient, but the process structure itself has to be adapted as well [23]. In addition, exception handling patterns cover changes at the process instance level, but are not applicable to process schema changes.

In addition to the expressiveness of the used process modeling language and the respective change framework, the features provided by the PAIS to support and implement these changes have to be considered as well. Expressiveness only allows for statements whether a particular change can be conducted or not; e.g., it provides information on whether or not process activities can be added or deleted. 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 [10]. 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 of a process graph. This does not only require a high level of expertise, but also slows down the entire change process. In addition, not all PAISs supporting dynamic process changes ensure correctness and robustness afterwards, which might lead to inconsistencies, deadlocks or other flaws [25]. Again, methods for a systematic comparison of these frameworks in respect to their ability to deal with changes would facilitate the selection of an appropriate technology for implementing the PAIS.

1.2. *Contribution*

The major contributions of this paper are threefold.

1.) We suggest a set of *change patterns* to foster the comparison of existing approaches with respect to their ability to deal with process change. The suggested patterns

[1] We use the notion of a PAIS engineer for a developer of PAISs with the focus on the analysis and design of (executable) processes; i.e., a PAIS engineer deals with programming in the large

complement existing worklow patterns and have been systematically identified by analyzing a large collection of process models from the healthcare and the automotive domain. On the one hand we provide patterns for high-level process adaptations at the process type as well as the process instance level. On the other hand, we identify patterns that can be used to defer decisions regarding the exact control flow to run-time to better deal with uncertainty.

2.) In addition to change patterns, we suggest a set of *change support features*. While the respective patterns allow for assessing the expressiveness of change frameworks, change support features ensure that changes are performed in a correct and consistent way, change traceability is enabled, and process changes become easier to accomplish for users.

3.) We provide an in-depth 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 [25]. While in [25] the proposed patterns have been only described very briefly, this paper provides an in-depth description of all identified change patterns and discusses the applied methods for patterns identification in detail. 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 PAISs. In analogy to the workflow patterns initiative [19], we expect further systems to be evaluated over time and vendors of existing PAISs are expected to extend their current PAISs towards more complete support for change patterns and change features.

The remainder of this paper is organized as follows: Section 2 summarizes background information needed for the understanding of the paper. Section 3 presents the research method employed for identifying the described patterns and features. Section 4 describes 18 change patterns sub-dividing them into adaptation patterns and patterns for changes in predefined regions. Section 5 deals with 7 crucial change support features. Taking these change patterns and change features, Section 6 evaluates different approaches from academia as well as industry. 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 separating 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. At *run-time* the PAIS then orchestrates the processes according

to the defined logic and allows for the integration of users and other resources. Workflow Management Systems (e.g., Staffware [2], ADEPT [11], WASA [13]) and Case-Handling Systems (e.g., FLOWer [2,14]) are typical technologies enabling PAISs (for a quantitative comparison see [26]).

For each business process to be supported (e.g., booking a 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/AND-Split, XOR/AND-Join) – and a set of *control edges* between them. Control edges specify precedence relations between the different nodes. 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 [27]). In this paper, we use the term *process fragment* as generalized concept covering atomic activities, complex activities (i.e., sub processes) and hammocks (i.e., sub graphs with single entry and single exit node). If a pattern is denoted as being applicable to a process fragment, it can be applied to all these objects.

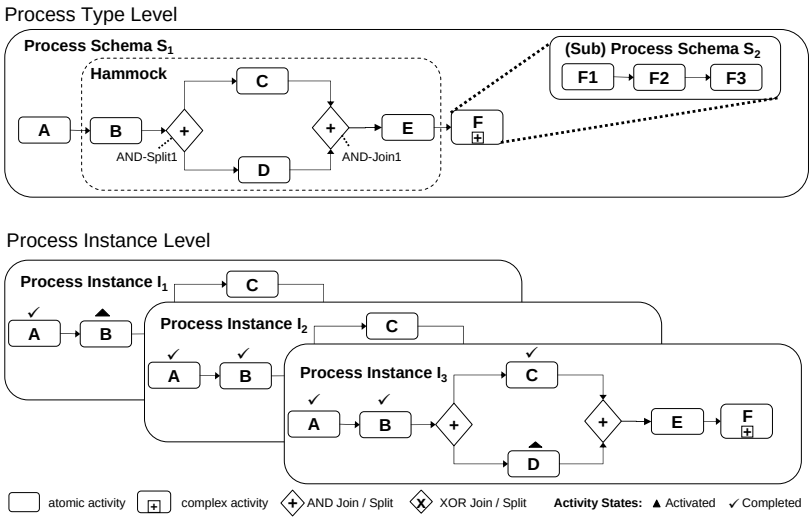


Fig. 1. Core Concepts - An Example (in BPMN Notation [28])

In Fig. 1 process schema $S1$ 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 $S2$). 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., a sub process graph with single entry and single exit nodes. 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 (e.g., process schema $S2$), 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. 1, for example, activity `A` is completed and activity `B` is activated. Generally, a large number of instances in different states might run on a particular process schema.

2.2. *Process Flexibility*

To deal with evolving processes, exceptions and uncertainty, PAISs must be flexible. This can either be achieved through structural process adaptations (cf. Fig. 2) or by allowing for loosely specified process models, which can be refined by users during run-time according to predefined criteria (cf. Fig. 2).

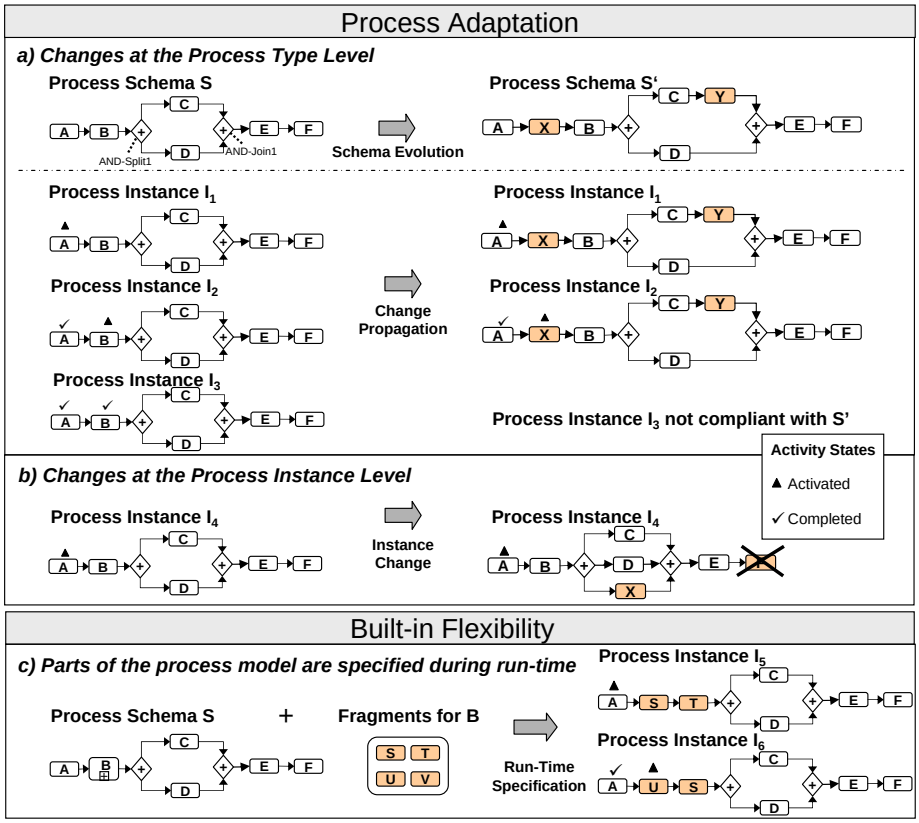


Fig. 2. Process Adaptation and Built-in Flexibility

Process Adaptation. In general, process adaptations can be triggered and performed at two levels – the process type and the process instance level (cf. Fig. 2a+b) [10].

Process schema changes at the type level (in the following denoted as *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 (of the respective type), particularly if these instances are long-running [29]. 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). For example, Fig. 2a illustrates a process type change. Schema S is transformed to S' by insterting two additional activities `X` and `Y`. Changes can be propagated to running instances as well, if these instances are *compliant* with the new schema version (i.e., their traces can be produced on S' as well) [29]. Instances I_1 and I_2 from Fig. 2a, for example, can be migrated to S', while I_3 has already progressed too far and therefore has to be completed based on original schema version S.

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 [11,13,30]. The effects of such ad-hoc changes are usually instance-specific and 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. In Fig. 2b, instance I_4 has been individually modified by inserting activity `X` and by deleting activity `F`.

Built-in 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. 2) [12,15–18]. This approach is especially useful in case of uncertainty as it allows for deferring decisions from build- to run-time, when more information becomes available. For example, when treating a cruciate rupture for a particular patient we might not know in advance which treatment 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 treatment at run-time. For example, Fig. 2c depicts a process schema with placeholder activity `B`. For placeholder activity `B` four process fragments `S`, `T`, `U` and `V` have been specified, which can be used during run-time to compose a sub process for substituting placeholder activity `B`. Instances I_5 and I_6 constitute two valid variants, which can be created based on schema S.

3. Research Methodology

Goal of this paper is to complement existing workflow patterns with a set of change patterns and change support features suitable to assess a PAIS's ability to effectively deal with process changes. Respective patterns and features should not only allow PAIS engineers to assess the expressiveness of a PAIS's change framework, but also ensure that changes can be performed in a correct, consistent and efficient way. In the following the

research methodology employed for identifying the change patterns (cf. Section 3.1) and change support features (cf. Section 3.2) is discussed.

3.1. *Pattern Identification*

We describe the selection criteria for our change patterns, the data sources they are based on, and the procedure we applied for pattern identification.

- **Selection Criteria.** On the one hand, this paper considers patterns to support process users to efficiently deal with exceptions and to cope with the evolving nature of business processes (i.e., supporting structural *process adaptations*). On the other hand, it covers patterns for supporting users to better deal with uncertainty by deferring decisions to run-time (i.e., to allow for *Built-in Flexibility*). The focus of this paper is on changes of the control-flow perspective. The extension towards other process aspects (e.g., data flow or resources) constitutes complementary work and is outside the scope of this paper.
- **Sources of Data and Data Collection.** As sources for our patterns we consider the results of case studies we previously performed in the healthcare domain [5] and the automotive domain [31].

  One of our major data sources is a large reengineering project which we conducted at a Women's Hospital. As part of this project all core processes of the hospital were analyzed and documented [32–36]. For each of these processes an *as-is* process model (either described with ARIS Toolset or the Bonapart process modeling tool) exists. In addition, suggestions for process optimizations were textually described and, in case of structural process adaptations, additionally modeled as *to-be* processes. Finally, in the context of our process analyses interviews with 7 physicians and 4 nurses were conducted to document typical exceptional situations (i.e., deviations from the standard procedure). In total we consider 98 process models from the healthcare domain covering both administrative processes (e.g, patient admission or ordering drugs) and medical treatment processes (e.g., in-patient chemotherapy and ovarian cancer surgery) (an overview of the analyzed models is given in Fig. 3). While the considered administrative processes are quite simple, the medical treatment processes all comprise numerous subproceses and have several hierarchical levels (i.e., up to five levels of hierarchy and more than a hundred activities including all subprocesses).

  As our second major data source we use process models from the automotive domain. In particular, we consider a case study on electronic change management (ECM) [37] and process models described in [38]. As part of the ECM project (standardized) process models for two phases of electronic change management (i.e., Electronic Change Request (ECR) and Electronic Change Order (ECO)) were created. The ECR and ECO process models were iteratively modeled (either as Event-Driven Process Chains or as UML Activity Diagrams) resulting in multiple process model versions. The final models related to ECR have been published by the German Association of the Automotive Industry (VDA) as quasi-standard and are described in [37]. The process models described in [38] include processes on car repair and maintenance in garages, in-house change management and product planning. Again numerous process model variants exist. With several hundred activities the product planning process is the

most complex process we consider. In total, our material from the automotive domain consists of 59 process models (for an overview cf. Fig. 3).

Healthcare Domain	
Administrative Processes	8 process models
In-Patient Chemotherapy	13 process models
Ovarian Cancer Surgery	21 process models
Ambulatory Chemotherapy	8 process models
Endoscopic Surgery in a Day Hospital	21 process models
Laboratory Test	8 process models
Radiologic Test	9 process models
Total:	**98 process models**
Automotive Domain	
Electronic Change Request	11 process models
Electronic Change Management	16 process models
Car Repair and Maintenance in Garages	20 process models
In-house Change Management	1 process model
Product Planning	1 process model
Total:	**59 process models**

Fig. 3. Analyzed Process Models

- **Pattern Identification Procedure.** To ground our patterns on a solid basis we first create a list of candidate patterns. For generating this initial list we conduct a detailed literature review and rely on our experience with PAIS-enabling technologies. Next we thoroughly analyze the above mentioned material to find empirical evidence for our candidate patterns. In particular, we compare the available *as-is* and *to-be* processes from the healthcare case study or the different model versions from the automotive domain. Further, we analyze the described suggestions for optimizations and exceptional situations. We then map the identified structural process adaptations as well as exceptional situations to our candidate patterns and - if necessary - extend the candidate list of patterns (for two examples see below).

  For example, in the context of an in-patient chemotherapy activities `Inform Patient` and `Make Appointment` are performed sequentially in the *as-is* model, while the *to-be* process model arranges them in parallel, i.e., activities `Inform Patient` and `Make Appointment` have been parallelized. This example is one observation supporting pattern AP9 (*Parallelize Process Fragment*) as introduced in Section 4.1. As another example consider the patient admission process. Usually activity `Clinical Admission` has to be preceeded by activity `Administrative Admission`. However, in case of an emergency the administrative admission needs to be postponed (e.g., the administrative admission is moved in parallel to the clinical admission). This scenario is an example providing evidence for pattern AP3 (*Move Process Fragment*). In total, we analyzed more than a hundred exceptional situations in which structural model adaptations become necessary. Evidence for pattern AP6 *Extract Process Fragment* and AP7 *Inline Process Fragment*, for example, can be found in the product planning process (automotive domain), which comprises several hundred activities. To generate aggregated views for process visualization or to dissolve these aggregations, process fragments need to be dynamically extracted to sub processes or be inlined into the parent process [38].

As a pattern is defined as a reusable solution to a commonly occuring problem we require each of our change patterns to be observed at least three times in different models of our process samples. Therefore, only those patterns, for which enough empirical evidence exists, are included in the final list of patterns, which is presented in Section 4.

3.2. *Identification of Change Support Features*

In the following we describe the selection criteria for change support features, the data sources they are based on and the used procedure for change feature identification.

- **Selection Criteria.** While change patterns provide the expressive power to perform process adaptations or to defer decisions to run-time, change support features constitute typical functionalities offered by PAIS-enabling technologies to ensure that process changes become applicable in practice. Thereby, change support features address typical quality dimensions like correctness and consistency, efficiency, traceability, security and usability.
- **Sources of Data and Data Collection.** As data sources for our change patterns several flexible PAIS from both academia and industry are considered. As commercial systems we consider the workflow management system Staffware and the case-handling system FLOWer, which are both among the most widely used systems in their area. As academic approaches we consider popular flexible PAIS such as ADEPT2 [11,39], CAKE2 [30,40], CBRFlow [41,42], HOON [17], MOVE [18], Pockets of Flexibility (PoF) [16,43,44], WASA2 [13], WIDE [45,46], Worklets/Exlets [12,47,48], and YAWL [49].
- **Feature Identification Procedure.** To identify typical change support features we analyze the above mentioned PAIS-enabling technologies in respect to their functionality supporting process changes. To be included in our final list of change support features (cf. Section 3.2) we again require at least three independent observations, i.e., each change support feature needs to be supported by at least three PAIS-enabling technologies.

4. Change Patterns

In this section we describe 18 characteristic patterns relevant for *control flow changes* in PAISs (cf. Fig. 5). All these change patterns constitute solutions for realizing commonly occuring changes in PAISs. We divide the change patterns into two major categories: *adaptation patterns* and *patterns for changes in predefined regions*. Thereby, adaptation patterns support structural process adaptations, whereas patterns for changes in predefined regions allow for built-in flexibility (cf. Section 2.2).

Adaptation Patterns allow users to structurally modify 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 to delete a single control flow edge). Although process adaptations can be performed based on low-level change primitives as well, these primitives are not considered as real change patterns due to their lack of abstraction. Like design patterns in software engineering, change patterns aim at reducing complexity [50] by raising the level of abstraction for

expressing changes. 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 or for coping with the evolving nature of business processes.

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 [11]. 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. 4 gives a comparison of these two major pattern categories.

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

Fig. 4. Adaptation Patterns vs. Patterns for Changes in Predefined Regions

Fig. 5 gives an overview of the 18 patterns, which are 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. 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*.

To obtain unambiguous pattern descriptions and to ground pattern implementation as well as pattern-based analysis of PAISs on a solid basis, we have provided a formal semantics for change patterns in [51]. This formalization is independent of the underlying process meta model and is based on the behavioral semantics of the modified process schema before and after its change.

4.1. *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

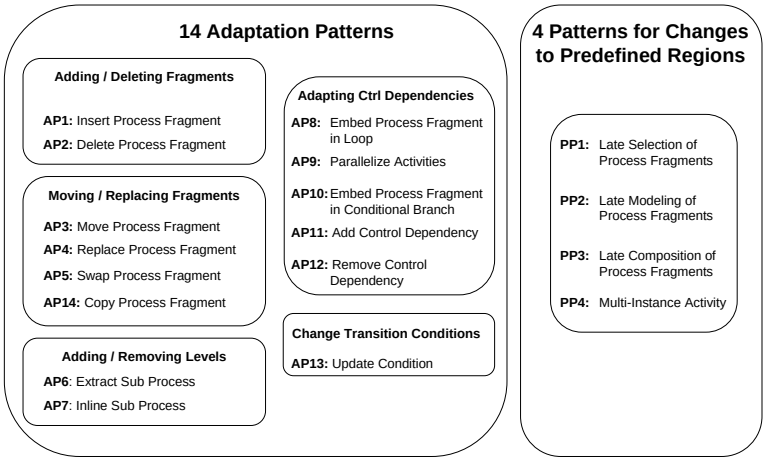


Fig. 5. Overview of Identified Change Patterns

schema S'. For this, 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 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. Specifying structural adaptations 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.

On the other hand, structural adaptations can be based on high-level change operations (e.g., to insert a 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 more 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 the PAIS to guarantee soundness when applying the respective operations [11,45]. 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 [10,52]). For these reasons we only consider high-level operations as adaptation patterns; more precisely, an adaptation pattern comprises exactly one high-level operation. Furthermore, its application to a given process schema will preserve soundness of this schema if certain pre-conditions are met.

In total, 14 out of the 18 identified change patterns can be classified as adaptation patterns (see left-hand side of Fig. 5). In the following all 14 adaptation patterns are described in detail. 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). Patterns AP6 and AP7 allow for 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. Patterns AP8–AP12 support adaptations 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, AP13 (update condition) allows for changing transition conditions.

Fig. 6 describes two general design choices, which are valid for all 14 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 description of the respective patterns (cf. Fig. 7 to Fig. 13). The design choices listed in Fig. 6 are shortly described in the following. Each adaptation pattern can be applied at the process type and/or process instance level (Design Choice A)(cf. Fig. 2a+b). If an adaptation pattern is supported at the process type level, the graphical editor of the PAIS allows users to edit a process schema at built-time using the respective pattern. If no pattern support is provided, process schema changes have to be conducted at a low level of abstraction using change primitives [51]. If a respective pattern is, in turn, supported at the process instance level, run-time changes of single instances can be accomplished. In addition, adaptation patterns can operate on an atomic activity, an encapsulated sub process or a hammock (Design Choice B) (cf. Fig. 6).

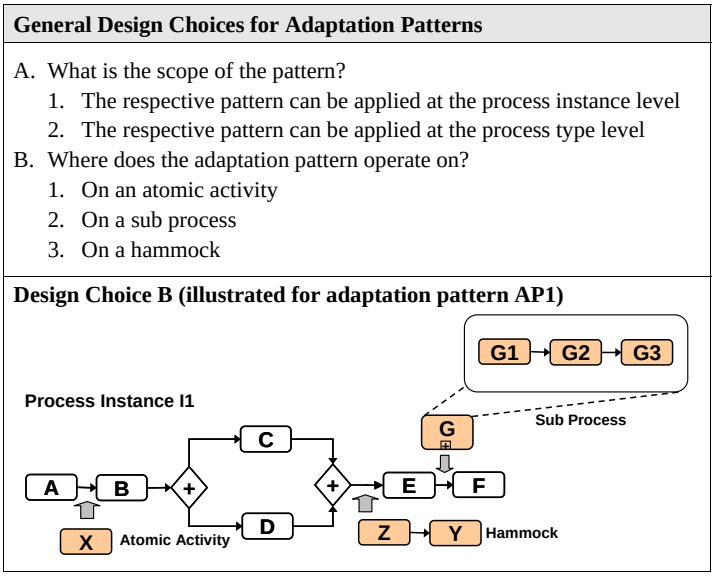


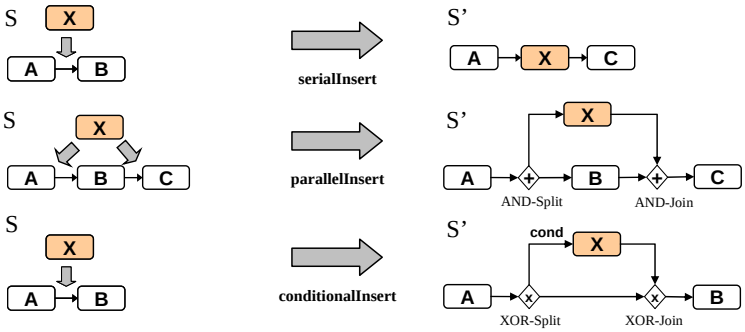
Fig. 6. General Design Choices for Adaptation Patterns

Adaptation Pattern AP1: Insert Process Fragment. The *Insert Process Fragment* pattern (cf. Fig. 7) can be used to add process fragments to a process schema. In addition to the general design choices described in Fig. 6, 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 (cf. Fig. 7). There are systems which only allow users

to serially insert a process fragment between two directly succeeding activities [53]. By contrast, other systems follow a more general approach, allowing the user to insert new fragments between two sets of activities meeting certain constraints [11]. Special cases of the latter variant include the insertion of a process fragment in parallel to another one (*parallel insert*) or the association of the newly added fragment with an execution condition (*conditional insert*).

Pattern AP1: INSERT Process Fragment

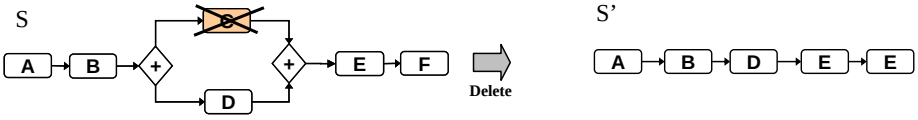
Description	A process fragment X is added to a process schema S.
Example	For a particular patient an allergy test has to be added to his treatment process due to a drug incompatibility.
Problem	In a real world process a task has to be accomplished which has not been modeled in the process schema so far.
Design Choices (in addition to those described in Fig. 6)	C. How is the new process fragment X embedded in the process schema? 1. X is inserted between two directly succeeding activities (`serial insert`) 2. X is inserted between two activity sets (`insert between node sets`) a) without additional condition (`parallel insert`) b) with additional condition (`conditional insert`)



Implementation	This adaptation pattern can be realized by transforming the high level insertion operation into a sequence of low level change primitives (e.g., add node, add edge).

Pattern AP2: DELETE Process Fragment

Description	A process fragment is deleted from a process schema S.
Example	For a particular patient a planned computer tomography must not be performed in the context of her treatment process as she has a cardiac pacemaker, i.e., the computer tomography activity is deleted.
Problem	In a real world process a planned task has to be skipped or deleted.



Implementation	This adaptation pattern can be realized by transforming the high level delete operation into a sequence of low level change primitives (e.g., remove node, remove edge). Implementation options include the deletion of the fragment from the process schema, the replacement of the fragment by a silent activity or the embedding of the fragment in a conditional branch which is never executed.

Fig. 7. *Insert (AP1)* and *Delete Proces Fragment (AP2)* Patterns

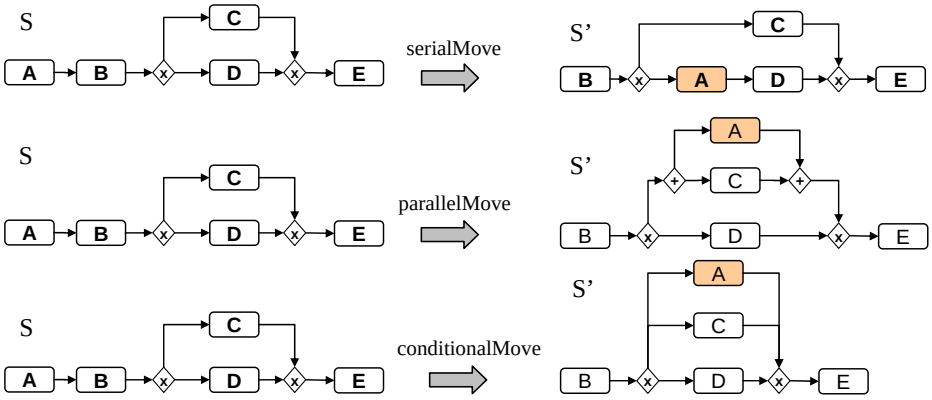
Adaptation Pattern AP2: Delete Process Fragment. The *Delete Process Frag-*

ment pattern can be used to remove a process fragment (cf. Fig 7). No additional design choices are needed for this pattern. There exist alternative ways in which this pattern can be implemented. The first implementation option is to 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, which is then never executed (i.e., the fragment remains part of the schema, but will not be executed) [46].

Adaptation Pattern AP3: Move Process Fragment. The *Move Process Fragment* pattern (cf. Fig. 8) 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. Although 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 AP3: MOVE Process Fragment

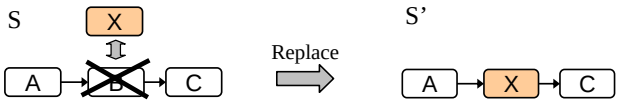
Description	A process fragment is moved from its current position in process schema S to another position within the same schema.
Example	Usually employees may only book a flight after it has been approved by the manager. Exceptionally, for a particular process the booking of a flight shall be done in parallel to the approval activity; consequently the *book flight* activity has to be moved from its current position in the process to a position parallel to the approval activity.
Problem	Predefined ordering constraints cannot be completely satisfied for a set of activities.
Design Choices (in addition to those described in Fig. 6)	C. How is the shifted process fragment X re-embedded in S? 1. X is re-inserted between two directly succeeding activities (`serial move`) 2. X is re-inserted between two activity sets (`move between node sets`) a) without additional condition (`parallel move`) b) with additional condition (`conditional move`)



Implementation	This adaptation pattern can be implemented based on patterns AP1 and AP2 (insert / delete process fragment) or be based on change primitives (e.g., add/remove node, add/remove edge).
Related Patterns	*Swap* adaptation pattern (AP5)

Fig. 8. *Move Process Fragment (AP3)* Pattern

Adaptation Pattern AP4: Replace Process Fragment. This pattern enables the replacement of a process fragment by another one (cf. Fig. 9). Like AP3, pattern AP4 can be implemented based on patterns AP1 and AP2 (*Insert/Delete Process Fragment*) or be directly based on change primitives.

Pattern AP4: REPLACE Process Fragment	
Description	A process fragment is replaced by another process fragment in process schema S.
Example	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 *computer tomography* activity, the *X-ray* activity shall be performed.
Problem	A process fragment is no longer adequate, but can be replaced by another one.
	
Implementation	This adaptation pattern can be implemented based on patterns AP1 and AP2 (insert / delete process fragment) or based on change primitives.

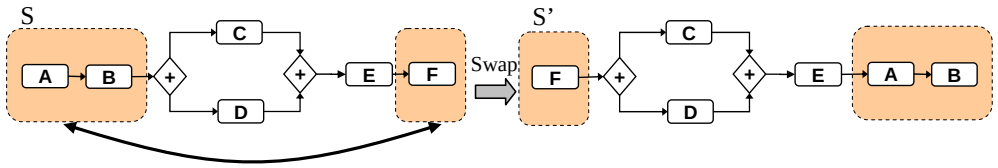
Pattern AP5: SWAP Process Fragment	
Description	Two existing process fragments are swapped in process schema S.
Example	Regarding a particular delivery process the order in which requested goods are about to be delivered to two customers has to be swapped.
Problem	The predefined ordering of two existing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

Fig. 9. *Replace (AP4)* and *Swap Process Fragment (AP5)* Patterns

Adaptation Pattern AP5: Swap Process Fragments. The *Swap Process Fragment* pattern (cf. Fig. 9) 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)*, on the combined use of patterns AP1 and AP2 (*Insert/Delete Process Fragment*), or on 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. 10). This pattern can be used to add a hierarchical level to simplify a process schema or to hide information

from process participants. If no direct support for pattern AP6 is provided a possible workaround will look as follows: The new schema representing the extracted sub process has to be created manually. Next, 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 [54].

Adaptation Pattern AP7: Inline Sub Process. As opposed to AP6 (Extract Process Fragment), 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. 10). This can be useful in case a process schema is divided into too many hierarchical levels or for improving its structure. If no direct support for AP7 is provided a couple of manual steps will be required as workaround. First the fragment representing the sub process has to be copied to the parent process schema. In a next step the activity invoking 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 AP8 an existing process fragment can be embedded in a loop to allow for its repeated execution (cf. Fig. 11). AP8 can be realized based on patterns AP1 (*Insert Process Fragment*), AP11 (*Add Control Dependency*), and AP12 (*Remove Control Dependency*). However, with AP8 the number of operations needed for accomplishing such a change can be reduced [51].

Adaptation Pattern AP9: Parallelize Process Fragments. AP9 enables the parallelization of process fragments which were confined to be executed in sequence (cf. Fig. 11). If no direct support for AP9 is provided, it can be simulated by combining adaptation patterns AP11 and AP12 (*Add / Remove Control Dependency*) or by using adaptation pattern AP3 (*Move Process Fragment*).

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. 11). AP10 can be implemented based on patterns AP1 (*Insert Process Fragment*), AP11, AP12 (*Add / Remove Control Dependency*) and AP13 (*Update Condition*).

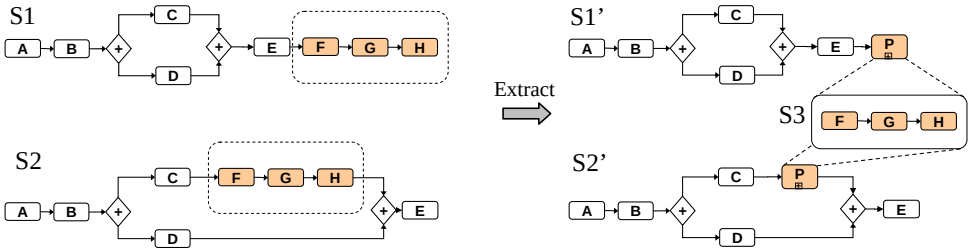
Adaptation Pattern AP11: Add Control Dependency. When applying pattern AP11 a control edge (e.g., for synchronizing the execution order of two parallel activities) is added to the given process schema (cf. Fig. 12). As opposed to the low-level change primitive *add edge*, the added control dependency must 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 can be associated with attributes (e.g., transition conditions) when applying AP11.

Adaptation Pattern AP12: Remove Control Dependency. Using this pattern a control dependency and its attributes can be removed from a process schema (cf. Fig. 12). Similar considerations as for AP11 can be made.

Adaptation Pattern AP13: Update Condition. This pattern allows users to update transition conditions in a process schema (cf. Fig. 12). Usually, an implementation of this pattern has to ensure that the new transition condition is correct in the context

Pattern AP6: EXTRACT Process Fragment to Sub Process

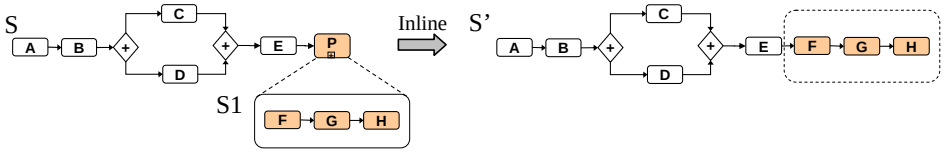
Description	From a given process schema S a process fragment is extracted and replaced by a corresponding sub process.
Example	A dynamically evolving engineering process has become too large. To reduce complexity the process owner extracts activities related to the engineering of a particular component and encapsulates them in a separate sub process.
Problem	*Large process schema.* If a process schema becomes too large, this pattern will allow for its hierarchical (re-)structuring. This simplifies maintenance, increases comprehensibility, and fosters the reuse of process fragments. *Duplication across process schemes.* A particular process fragment appears in multiple process schemes. If the respective fragment has to be changed, this change will have to be conducted repetitively for all these schemes. This, in turn, can lead to inconsistencies. By encapsulating the fragment in one sub process, maintenance costs can be reduced (see figure below).



Implementation	For implementing 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.
Related Patterns	*Inline Sub Process* (AP7)

Pattern AP7: INLINE Sub Process

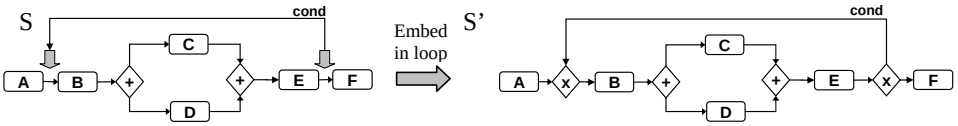
Description	A sub process to which one or more process schemes refer to is dissolved and the corresponding sub process graph is directly embedded in the parent schemes.
Example	The top level of a hierarchically structured engineering process only gives a rough overview of the product development process. Therefore, the chief engineer decides to lift selected sub processes up to the top level.
Problem	*Too many hierarchies in a process schema*: If a process schema consists of too many hierarchy levels the inline sub process pattern can be used to flatten the hierarchy. *Badly structured sub processes*: If sub processes are badly structured the inline pattern can be used to embed them into one big process schema, before extracting better structured sub-processes (based on AP6).



Implementation	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.
Related Patterns	*Extract Process Fragment to Sub Process* (AP6)

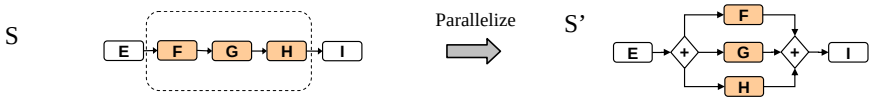
Fig. 10. *Extract (AP6)* and *Inline Sub Process (AP7)* Patterns

Pattern AP8: Embed Process Fragment in Loop	
Description	Adds a loop construct to a process schema which surrounds an existing process fragment.
Example	In 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.
Problem	A process fragment is actually executed at most once, but needs to be executed recurrently based on some condition.



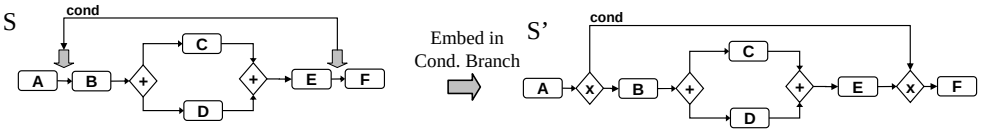
Implementation	This adaptation pattern can be implemented based on Patterns AP1 (insert process fragment), AP11, and AP12 (add / remove control dependency). Alternatively, implementation can be based on change primitives.
Related Patterns	*Embed Process Fragment in Conditional Branch* (AP10)

Pattern AP9: PARALLELIZE Process Fragments	
Description	Process fragments which have been confined to be executed in sequence so far are parallelized in a process schema S.
Example	For a production process, tasks which have been processed in sequence so far, shall now be executed in parallel.
Problem	Ordering constraints predefined for a set of process fragments turn out to be too strict and therefore have to be removed.



Implementation	This adaptation pattern can be implemented based on Patterns AP3 (move process fragment), AP11 and AP12 (add / remove control dependency) or be based on change primitives.
Related Patterns	*Remove Control Dependency* (AP12)

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 approving a business travel 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.

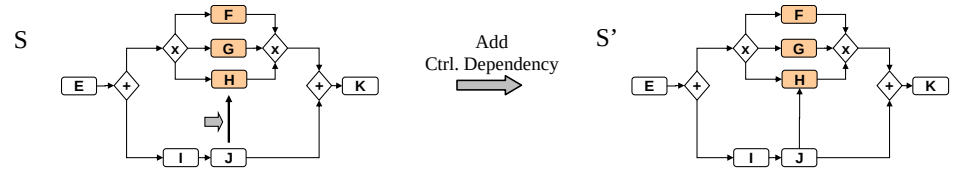


Implementation	This adaptation pattern can be implemented based on patterns AP1 (insert process fragment), AP11, and AP12 (add / remove control dependency) and AP13 (update condition) or be based on change primitives.
Related Patterns	*Embed Process Fragment in Loop* (AP8)

Fig. 11. *Embed Process Fragment in Loop* (AP8), *Parallelize Process Fragments* (AP9) and *Embed Process Fragment in Conditional Branch* (AP10) Patterns

Pattern AP11: Add Control Dependency

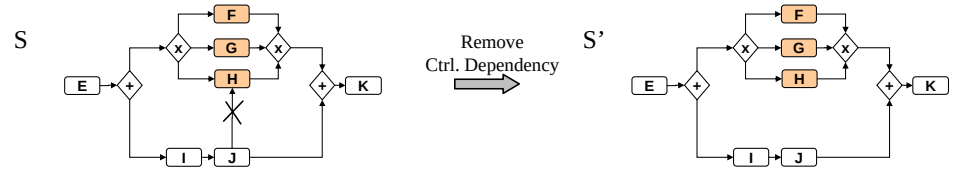
Description	An additional control edge (e.g., for synchronizing the execution order of two parallel activities) is added to process schema S.
Example	For a production process tasks which have been processed in parallel so far have now to be processed in sequence.
Problem	An additional control dependency is needed in process schema S.



Related Patterns *Remove Control Dependency* (AP12)

Pattern AP12: Remove Control Dependency

Description	A control edge is removed from process schema S.
Example	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.
Problem	An existing control dependency is not needed anymore in process schema S.



Related Patterns *Parallelize Process Fragments* (AP9), *Add Control Dependency* (AP11)

Pattern AP13: Update Condition

Description	A transition condition in the process schema is updated.
Example	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.
Problem	A transition condition has to be modified as it is no longer valid.

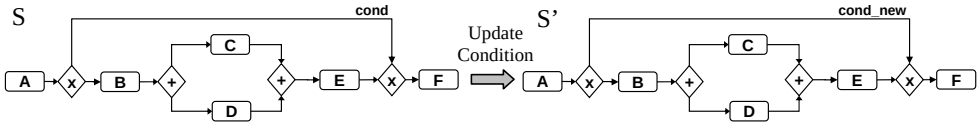


Fig. 12. *Add* (AP11) and *Remove Control Dependency* (AP12) and *Update Condition* (AP13)

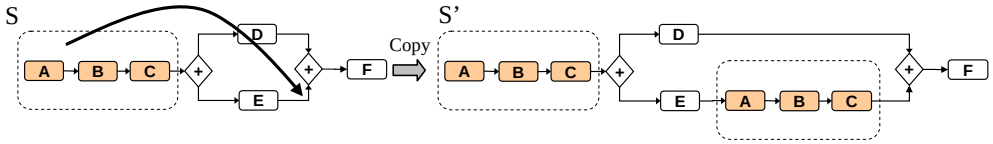
of the given process schema (e.g., process variables to which the transition condition refers must have been written before evaluating the condition). For example, it has to be ensured that all process relevant data elements, to which the transition condition refers to, are contained in the process schema.

Adaptation Pattern AP14: Copy Process Fragment. AP14 (cf. Fig. 13) allows users to copy a process fragment. In contrast to AP3 (*Move Process Fragment*) the

respective fragment is not removed from its initial position.

Pattern AP14: COPY Process Fragment

Description	A process fragment X is copied from its current position in process schema S to another position of the same schema S.
Example	In a reviewing process the papers to be reviewed are sent with the reviewing instructions 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.
Problem	A process fragment has to be executed once more.
Design Choices (in addition to those described in Fig. 6)	C. How shall the copied process fragment X be embedded in the process schema? 1. X is inserted between two directly succeeding activities (serial insert) 2. X is inserted between two activity sets (insert between node sets) a) without additional condition (parallel insert) b) with additional condition (conditional insert)



Implementation	This adaptation pattern can be implemented based on Pattern AP1 (insert process fragment) or by using change primitives.
Related Patterns	*Insert* adaptation pattern (AP1), *Move* adaptation pattern (AP5)

Fig. 13. *Copy Proces Fragment (AP14)* Pattern

4.2. *Patterns for Changes in Predefined Regions*

Patterns for changes in predefined regions allow for better dealing with uncertainty by deferring decisions regarding the exact control-flow to run-time. Instead of requiring a process model to be fully specified prior to execution, parts of the model can remain unspecified. In contrast to adaptation patterns, whose application is not restricted a priori to a particular process part, patterns for changes in predefined regions define constraints concerning the parts of a process schema that can be changed or expanded. In this category we have identified 4 patterns: *Late Selection* (PP1), *Late Modeling* (PP2) and *Late Composition of Process Fragments* (PP3) and *Multi-Instance Activity* (PP4). These four patterns differ regarding the parts that can remain unspecified resulting in a different degree of freedom during run-time.

Pattern for Predefined Change PP1: Late Selection of Process Fragments. This pattern (cf. Fig. 14) allows deferring the selection of the implementation of a particular process activity to run-time. Prior to execution only a placeholder activity has to be provided, the concrete implementation is selected during run-time either based on predefined rules or on user decisions (Design Choice A in Fig. 14). The placeholder activity can either be substituted by an atomic activity or a sub process (Design Choice B in Fig. 14). This is done before the placeholder activity is enabled or when it is enabled (Design Choice C in Fig. 14).

Pattern for Predefined Change PP2: Late Modeling of Process Fragments. This pattern (cf. Fig. 15) offers more freedom and allows for modeling selected parts of

the process schema at run-time. Prior to execution only a placeholder activity has to be provided, its implementation is modeld during run-time. Design Choice A of Fig. 15 specifies which building blocks can be used for late modeling. Building blocks can either be all process fragments from the repository, a constraint-based subset of the fragments from the repository, or newly defined activities or process fragments. Design Choice B (cf. Fig. 15) describes whether the user may apply the same modeling constructs during build-time or more restrictions apply. Late modeling can take place upon creation of the process instance, when the placeholder activity is enabled, or when a particular state in the process is reached (Design Choice C in Fig. 15). 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 in Fig. 15).

Pattern for Predefined Change PP3: Late Composition of Process Fragments. This pattern (cf. Fig. 16) 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. There is no predefined plan, but the process instance is created in an ad-hoc way by selecting from the available activities in the repository. In addition, constraints may be defined, which have to be considered while composing a process fragment (Design Choice A in Fig. 16). The *Interleaved Routing* pattern [3,20] – one of the workflow patterns [19] – 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 upon at run-time and each process fragment has to be executed exactly once. Like for pattern PP3 decisions about the exact control flow structure are deferred to run-time. The *Interleaved Routing* pattern corresponds to Design Choice A[2] of PP3 (cf. Fig. 16) as the composition has to to obey certain constraints (i.e., on how often a particular activity has to be executed).

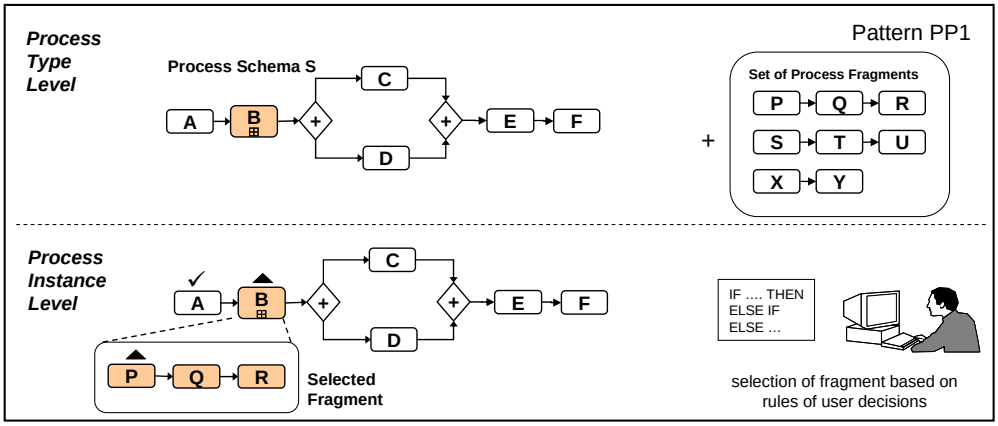
Pattern for Predefined Change PP4: Multi-Instance Activity. This pattern (cf. Fig. 16) allows for deferring the decision on how often a specific activity should be executed during run-time, while the activity itself needs to be predefined. PP4 not only constitutes a change pattern, but a workflow pattern as well [19]. It allows for the creation of multiple activity instances during run-time. The decision how many 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 a change pattern as their use does not help dealing with uncertainty. For all other types of multi-instance activities the number of instances is determined based on run-time knowledge, which is or is not 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 before executing them, this is not possible in the latter case. *Multi-Instance Activities* are considered as change patterns as their usage allows users to delay the decision on the number of instances to be created for a particular activity to run-time (cf. Fig. 16).

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

Pattern PP1: Late Selection of Process Fragments

Description	For particular activities the corresponding implementation (activity program or sub process model) can be selected during run-time. At build-time only a placeholder is provided, which is substituted by a concrete implementation during run-time.
Example	For the treatment of a particular patient one of several different sub-processes can be selected depending on the patient's disease.
Problem	There exist different implementations for an activity (including sub-processes), but for the selection of the respective implementation run-time information is required.
Design Choices	A. How is the selection process done? 1. Automatically based on predefined rules 2. Manually by an authorized user 3. Semi-automatically: options are reduced by applying some predefined rules; user can select among the remaining options B. What object can be selected? 1. Atomic activity 2. Sub process C. When does late selection take place? 1. Before the placeholder activity is enabled 2. When enabling the placeholder activity



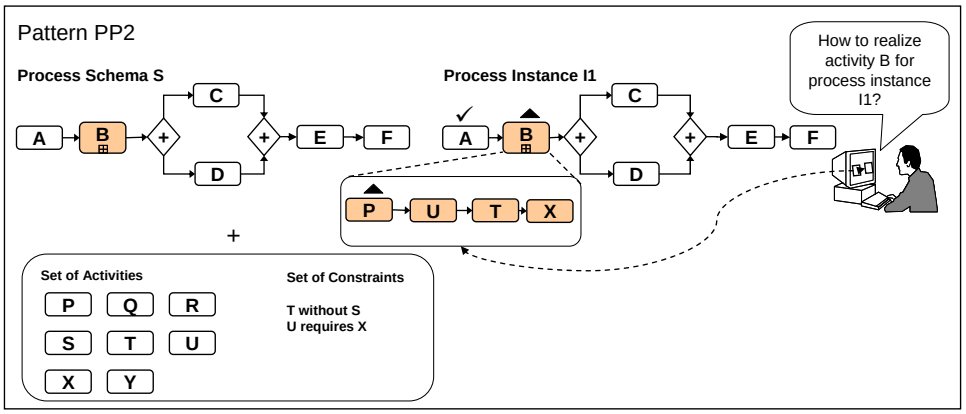
Implementation	By selecting the respective sub process or activity program, a reference to it is dynamically set and the sub-process or activity program is invoked.
Related Patterns	*Late Modeling of Process Fragments* (PP2)

Fig. 14. Late Selection of Process Fragments (PP1)

system can deal with process change. In addition, change support features must be considered to make change patterns useful in practice (cf. Fig. 17). Relevant change support features include *Schema Evolution, Version Control and Instance Migration (F1)*, *Support for Instance-Specific Changes (F2)*, *Correctness of Change (F3)*, *Traceability and Analysis of Changes (F4)*, *Access Control for Changes (F5)*, *Change Reuse (F6)*, and *Change Concurrency Control (F7)*. As illustrated in Fig. 17 the described change support features are not equally important for both process type level and process instance level changes. Version control, for example, is primarily relevant for changes at the type level (T), while change reuse is particularly useful at the instance level (I) [41].

Pattern PP2: Late Modeling of Process Fragments

Description	Parts of the process schema have not been defined at build-time, but are modeled separately 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 respective process fragment can be executed.
Example	The exact treatment process of a particular patient is composed out of existing process fragments at run-time.
Problem	Not all parts of the process schema can be completely specified at build time.
Design Choices	A. What are the basic building blocks for late modeling? 1. All process fragments from the repository can be chosen. 2. A constraint-based subset of the process fragments from the repository can be chosen. 3. New activities or process fragments can be defined. B. What is the degree of freedom regarding late modeling? 1. Same modeling constructs and change patterns can be applied as for modeling at the process type level. Which of the adaptation patterns are supported within the placeholder activity is determined by the expressiveness of the modeling language. 2. More restrictions apply than for modeling at the process type level. C. When does late modeling take place? 1. When a new process instance is created. 2. When the placeholder activity is instantiated. 3. When a particular state in the process (preceding the instantiation of the placeholder activity) is reached. D. Does the modeling start from scratch? 1. Starts with an empty template. 2. Starts with a predefined template which can be adapted.

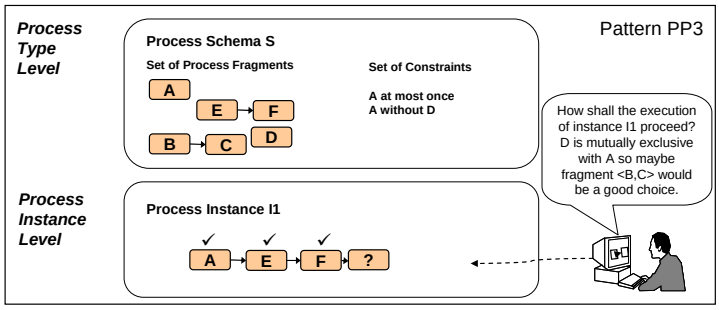


Implementation	After having modeled the placeholder activity with the process editor during run-time, the fragment is stored in the repository and deployed afterwards. Then, the process fragment is dynamically invoked as a sub process. The assignment of the respective process fragment to the placeholder activity is done through late binding.
Related Patterns	*Late Selection of Process Fragments* (PP1)

Fig. 15. Late Modeling of Process Fragments(PP2)

Pattern PP3: Late Composition of Process Fragments

Description	At build-time a set of process fragments is defined from which the schema of a concrete process instance can be composed during run time. This can be achieved by dynamically selecting fragments and by specifying the control dependencies between them on the fly.
Example	Different kinds of medical examinations are accomplished in a hospital. The exact examinations to be applied to a particular patient and the order in which they are performed are defined for each patient individually depending on his/her medical problems.
Problem	There exist several variants of how process fragments can be composed. To reduce the number of process variants to be specified by the process engineer during build time, process instances are dynamically composed from a given set of fragments.
Design Choices	A. What are the basic building blocks for late modeling? 1. All process fragments from the repository can be chosen. 2. A constraint-based subset of the process fragments from the repository can be chosen. 3. New activities or process fragments can be defined.



Pattern PP4: Multi Instance Activity

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

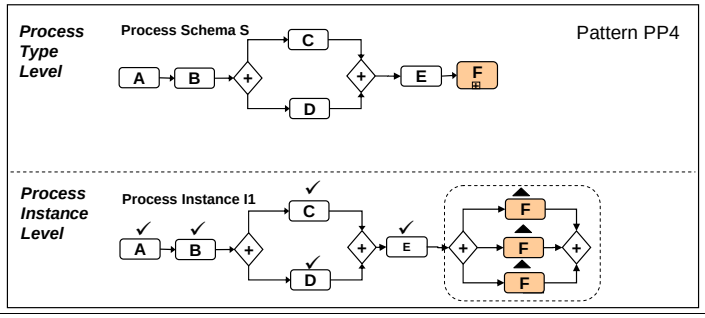


Fig. 16. Late Composition of Fragments (PP3) and Multi-Instance Activity (PP4)

5.1. *Schema Evolution, Version Control and Instance Migration*

To support changes at the process type level, version control for process schemes is needed (cf. Fig. 17). In case of long-running processes, in addition controlled migration

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

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

Fig. 17. Change Support Features

of already running instances from the old to the new process schema version is often required when conducting a schema change at the process type level. In this subsection we describe different options existing in this context (cf. Fig. 18).

If no version control is provided, either the process designer will have to manually create a copy of the process schema to be changed or the original schema will be overwritten (cf. Fig. 18a). In the latter case, running instances can either be withdrawn from the run-time environment (Design Choice F1[1]) or, as shown in Fig. 18a, they remain associated with the modified schema (Design Choice F1[2]). Depending on current instance execution state and on how changes are propagated to instances progressed too far, missing version control can lead to inconsistent states and in the worst case to deadlocks or severe other run-time errors [10]. As shown in Fig. 18a schema S has been modified by adding activities X and Y. Regarding instance *I1* this change is uncritical as I_1 has not yet entered the changed region. However, instance $I2$ would be in an inconsistent state afterwards as instance schema and execution history do not match [10].

By contrast, if a PAIS provides explicit version control three 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 [45,29]. This approach leads to the co-existence of process instances belonging to different schema versions (cf. Fig. 18b). Alternatively, for a (selected) collection of already running process instances a controlled migration to the new process schema version is supported (cf. Fig. 18c).

Design Choice F1[3] is shown in Fig. 18b where already running instances *I1*, $I2$ and $I3$ remain associated with schema $S1$, while new instances ($I4$, $I5$) are created from schema S' (co-existence of process instances of different schema versions). By contrast, Fig. 19 illustrates the controlled migration of selected instances (Design Choice F1[5]). Only those instances (*I1* and $I2$) are migrated to the new schema version which are

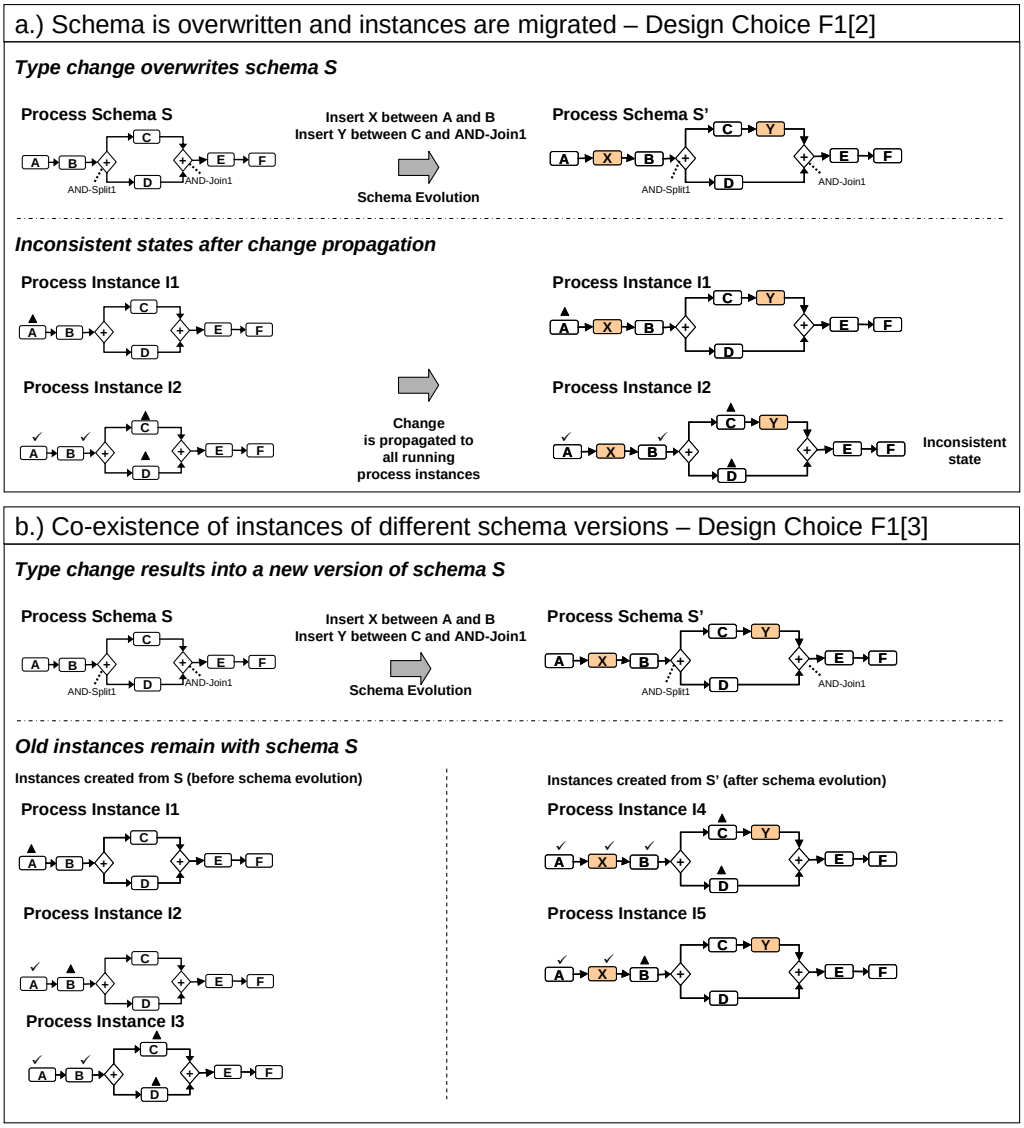


Fig. 18. Version Control - Different Options (1)

compliant[2] with S'. Thereby, $I1$ can be migrated without any state adaptations, whereas for $I2$ the newly inserted Activity `X` has to be enabled instead of Activity `B`. Instance $I3$ remains running according to S since it is non-compliant with S'. If instance migration is accomplished in an uncontrolled manner (i.e., it is not restricted to *compliant* process instances) inconsistencies or errors will result (Design Choice F1[4]). Nevertheless, we

[2] Simply speaking, 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 [10]).

treat the uncontrolled migration of process instances as a separate design choice since this functionality can be found in existing systems (cf. Section 6).

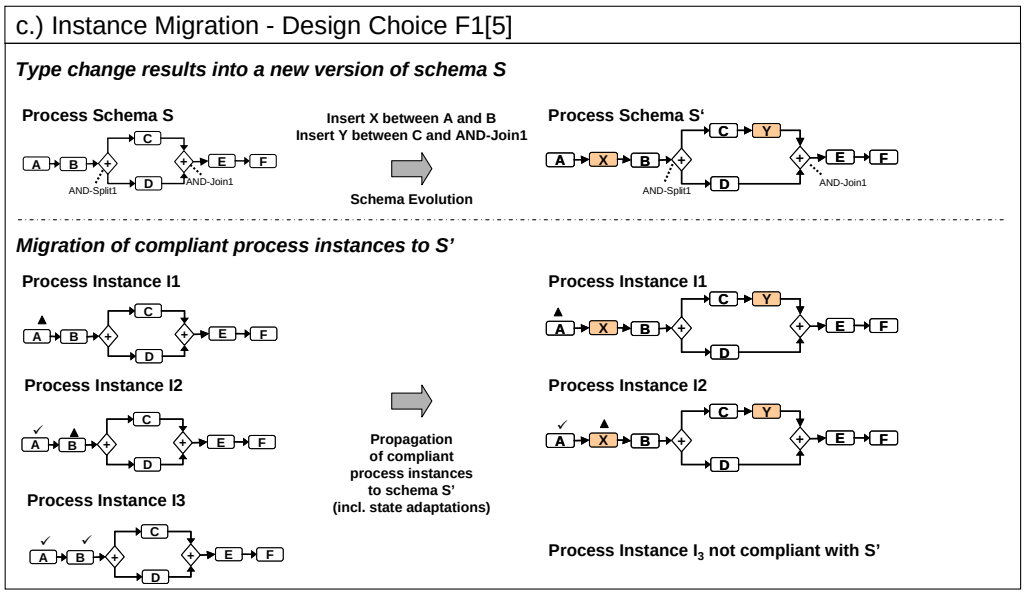


Fig. 19. Version Control - Different Options (2)

5.2. *Other Change Support Features*

To ensure that changes can be conducted in a correct and consitent way, that traceability is ensured, and change reuse is provided additional change support features are needed.

Support for Instance-Specific Changes (Change Feature F2). To deal with exceptions PAISs must support unplanned changes (Design Choice F2[1]) at the process instance level either through high-level changes in the form of patterns (cf. Section 4) or through low-level primitives. To deal with uncertainty, PAISs must further allow process modelers to keep parts of the model unspecified during build-time and to defer the concretisation of the respective part to run-time (Design Choice F2[2]). The effects resulting from 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. 20).

Correctness of Changes (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 [10] have been introduced to formally ensure that process instances can only be updated to a new schema if they are compliant with it [13,45,29]. In addition, depending on the used process meta model constraints of the respective formalism (e.g.,

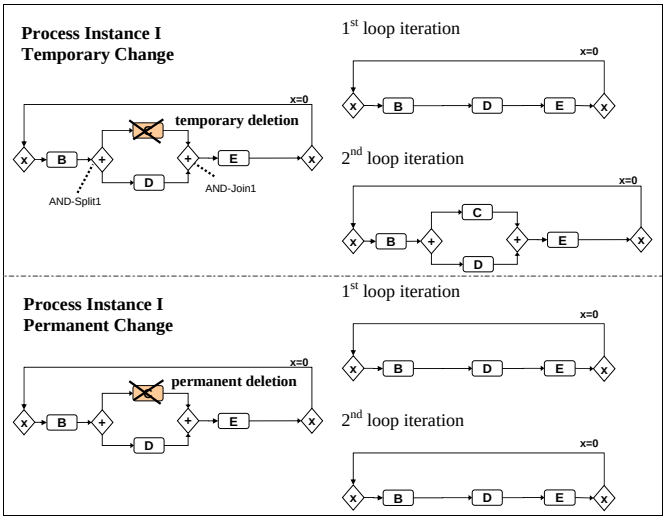


Fig. 20. Permanent and Temporary Changes

concerning the structuring of process schemes) have to be taken into account as well when applying process changes to a particular process schema. A detailed overview on correctness issues in the context of (dynamic) process changes can be found in [10].

Traceability and Analysis of Changes (Change Feature F4). For adaptation patterns, applied changes have to be stored in a change log as change patterns or primitives [55]. Regarding patterns for changes in predefined regions, an execution log is usually sufficient to enable traceability (Design Choice F4[1]). In addition, logs can be enriched with semantical information, e.g., about the reasons and the context of the changes [41,56] (Design Choice F4[2]). Finally, change mining allows for the analysis of changes and resulting schema variants, for example, to support continuous process improvement [57,58] (Design Choice F4[3]).

Access Control for Changes (Change Feature F5). The support of change patterns leads to increased process flexibility. This, in turn, imposes security issues as the PAIS becomes more vulnerable to misuse [59,60]. Therefore, the application of changes at the process type and the process instance level must be restricted to authorized users (Design Choice F5[1]). 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 [59] allow users to define restrictions at the level of single change patterns (e.g., a certain user is only allowed to insert additional activities, but not to delete existing ones) (Design Choice F5[2]). In addition, authorizations may depend on the object to be changed (e.g., the concrete process schema or a process instance) (Design Choice F5[3]).

Change Reuse (Change Feature F6). In the context of unplanned instance changes "similar" deviations (i.e., combinations of one or more adaptation patterns) can occur more than once [30,40]. As it requires significant user experience to define changes from scratch, change reuse has to be supported. For this, changes should be annotated with contextual information (e.g., about the reasons for the deviation) and be memorized (Change Feature F4[2]). This contextual information can be used for retrieving similar

problem situations. This approach ensures that only changes relevant in the current situation are presented to the user [41,40,42]. 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 [43].

Change Concurrency Control (Change Feature F7). PAISs supporting instance-specific adaptations (cf. Feature F2), should be able to cope with concurrent changes. 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, errors or inconsistencies (e.g., deadlock-causing cycles) can occur (Design Choice F7[1]). Or, the execution of an instance proceeds (i.e., the state of the instance may change) while an ad-hoc change is concurrently applied to it. 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).

The easiest way to avoid respective conflicts is to prohibit concurrent changes in general (Design Choice F7[2]). This can be achieved, for example, by holding exclusive locks on a process instance when changing its structure or state (e.g., an instance must not proceed while applying an ad-hoc change to it). Although this approach is easy to implement, it is usually too restrictive due to the long-term locks required (e.g., when a change is defined interactively by a user this might block instance execution for a while). A more flexible approach would allow for concurrent changes of the structure or state of a process instance, and further ensure that this does not lead to errors or inconsistencies afterwards (Design Choice F7[3]). Both, pessimistic and optimistic techniques can be applied in this context to control such concurrent instance changes and to ensure their correctness. Pessimistic locking is an approach where the process instance to be modified is locked until the change is accomplished. In contrast, optimistic concurrency control does not put a lock on the process instance, but checks for conflicts when commiting changes.

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. Although 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) [61]. Note that respective considerations only have to be made for systems supporting both changes at the process type and the process instance level (Design Choice F7[4]).

6. Change Patterns and Change Support Features in Practice

In the following we describe the evaluation of selected approaches from academia and industry regarding their support for process change patterns and change support features. Section 6.1 describes our evaluation methodology. Evaluation results for adaption patterns are covered in Section 6.2, while support for changes in predefined regions is discussed in Section 6.3. Section 6.4 shows how the described change features are supported by the evaluated approaches. Finally, a summary of our evaluation results is provided in Section 6.5.

6.1. *Evaluation Methodology*

This section describes the methodology employed for conducting our evaluation. In particular, we describe the evaluation goal, evaluation objects, evaluation criteria, evaluation metrics, and the evaluation procedure.

Definition of Evaluation Goal. The goal of our evaluation is to measure how well current PAISs and PAIS-enabling technology cope with process changes. Thereby the focus is on changes of the control flow perspective.

Selection of Evaluation Objects. As evaluation objects we choose PAIS-enabling technologies from both academia and industry. Our focus is on approaches which primarily model their business processes in an imperative way. Declarative approaches to process modeling are not covered by our evaluation. In terms of academic approaches our evaluation includes (in alphabetical order) ADEPT2 [11,39], CAKE2 [30,40], CBRFlow [41,42], HOON [17], MOVE [18], Pockets of Flexibility (PoF) [16,43,44], WASA2 [13], WIDE [45,46], Worklets/Exlets [12,47,48], and YAWL [49]. As CBRFlow and ADEPT2 have been integrated in the ProCycle [62,56] project both systems are evaluated together. The Worklets/Exlets approach is evaluated together with YAWL as it has been integrated as a service for YAWL to foster its flexibility [12]. All academic approaches included in our evaluation have proof-of-concept implementations. Conceptual frameworks without implementation have not been considered. As samples for commercial systems our evaluation includes the case-handling system FLOWer and the workflow system Staffware. For both we have hands-on experience [26] as well as an installation running in our labs.

Definition of Evaluation Criteria and Metrics. Evaluation criteria are the 18 change patterns and the 7 change support features as described in Sections 4 and 5. Furthermore, 5 change primitives (i.e., add/remove node add/remove edge and move edge) are added to our evaluation criteria. As changes can always be conducted at a low level of abstraction through change primitives, their inclusion as evaluation criteria is required to ensure completeness. We measure the ability of a PAIS to deal with change as the degree of support for the described evaluation criteria. For each evaluation criterion we differentiate between *no support*, *partial support*, *support* and *not applicable*. If a particular evaluation object does not support an evaluation criterion at all this criterion will be labelled with "-"; partial support will be labelled with "○". If an evaluation object provides support for a particular criterion and no design choices exist, this will be labelled with "+". Otherwise, the supported design choices will be listed. As an example take pattern PP1 (Late Selection of Process Fragments) in Fig. 14. Assume that an evaluation object supports pattern PP1 with Design Choices A, B and C. Further assume that for both Design Choice A and Design Choice B, Options 1 and 2 are supported, while Design Choice C is only supported with Option 3. This would result in String "A[1,2], B[1,2], C[2]" (e.g., Worklet/Exlet approach in Fig. 23). As another example consider pattern AP1 (Insert Process Fragment) as depicted in Fig. 7. Asumme that an evaluation object supports Design Choice A with all two options, Design Choice B with all three option, and Design Choice C with all two options. This would result in String "A[1, 2], B[1,2,3], C [1, 2]" (e.g., ADEPT2 in Fig. 21).

Analyzing the Evaluation Objects along the Evaluation Criteria. For the academic approaches we base our evaluation on a comprehensive literature study. In cases where it is unclear whether a particular pattern or feature is supported, the respective

research groups were contacted. This provided us with valuable insights into the implementation of change patterns and change support features in the respective approaches. Regarding the commercial systems support for change patterns and change features was determined based on the installations in our lab and on our hands-on experience with respective sytems [26].

A summary of our evaluation results is given in Fig. 21-23. An in-depth description of each of the evaluated approches can be found in [51]. 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. For example, FLOWer supports dynamic changes of role assignments and allows process engineers to dynamically add or delete forms [14]. As these changes do not constitute control flow changes, respective functionaliy is not considered in our evaluation.

6.2. *Support for Adaptation Patterns*

Fig. 21 and 22 show which change primitives and adaptation patterns are supported by the evaluated systems. Table 21 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 model. Table 22, in turn, provides the evaluation results considering the use of adaptation patterns or change primitives at the process instance level. For a detailed description of the evaluated approaches we refer to [51].

Generally, an adaptation pattern will only be considered as being provided if the evaluated system supports the pattern directly, i.e., based on a single high-level change operation instead of a set of change primitives. As structural process adaptations can be always realized by means of a set of basic change primitives (e.g., CAKE2, WASA2), missing support for adaptation patterns does not necessarily mean that no changes can be performed. However, the support of adaptation patterns allows introducing changes at a higher level of abstraction hiding as much complexity from the user as possible. Further, certain adaptation patterns (e.g., AP3 or AP4) can be implemented by combining basic patterns (e.g., AP1, AP2, AP10 and AP11). Again, a given approach will only qualify for supporting an adaptation pattern, if it supports this pattern directly; e.g., providing support for patterns AP1 (Insert Process Fragment) and AP2 (Delete Process Fragment) allows for the straightforward implementation of pattern AP3 (Move Process Fragment). However, moving activities by using AP1 and AP2 in combination with each other is more complicated than the direct application of AP3. Moreover, this leads to less meaningful change logs.

Table 21 shows that all evaluated systems allow for process type modifications. Thereby, most systems only provide support for change primitives, i.e., they allow for 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 support for adaptation patterns at the process type level are ADEPT2 and WIDE. Table 22 shows that process instance modifications are supported by rather few systems. CAKE2 and WASA2 allow for structural run-time adaptations at the instance-level based on change primitives (i.e., by adding or removing nodes and edges respectively). ADEPT2, in turn, provides support for a wide range of adaptation patterns at the process instance level. Both the Worklet/Exlet approach and FLOWer support a lim-

ited 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).

6.3. *Support for Patterns for Changes in Predefined Regions*

Table 23 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 whole expressiveness of the modeling environment to the end user, PoF facilitates model construction by introducing modeling constraints. Validation ensures that the lately modeled process fragment is compliant with the constraints [16]. 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 different kinds of changes which we observed in several case studies [5,31]. Finally, the *Multi-Instance Activity* pattern PP4 can be found in WIDE, YAWL, FLOWer and Staffware. In addition, it is currently under implementation in ADEPT2.

Patterns PP1 (*Late Selection of Process Fragment*) and 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 pre-planned serial or parallel insertion becomes possible (cmp. Design Choice D[1,2] of AP1 in Fig. 7). However, insertion of activities 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.

6.4. *Change Support Features in Practice*

Table 23 shows evaluation results regarding the described change support features (cf. Sect. 5). As the evaluation shows, Feature F1 (Schema Evolution, Version Control and Instance Migration) is partially supported. Only half of the evaluated systems provide versioning support at all (ADEPT2, WASA2, WIDE, YAWL, FLOWer, and Staffware). As missing versioning support requires users to overwrite an existing schema in case of process type changes or to save the modified schema with new name, practical applicability is limited. FLOWer allows for overwriting a process schema in addition to the co-existence of instances running on different schema versions. If running instances are not removed from the system, overwriting a process schema can lead to undesired behaviour. In connection with process schema evolution the controlled propagation of changes to ongoing instances is only considered by ADEPT2, WASA2 and WIDE. Furthermore, Staffware offers a feature for propagating changes to all ongoing instances. As instance migration cannot be restricted to compliant instances only, the usage of this feature might lead to inconsistencies or deadlocks, thus having the same effect as overwriting a process schema.

Change Pattern Support at the Process Type Level

Primitive / Pattern	Academic							Commercial	
	ADEPT2 / CBRFlow	CAKE2	HOON	MOVE	PoF	WASA2	WIDE	YAWL + Worklets / Exlets	Flower	Staffware
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]	–	–	–	–	–	A[2], B[1], C[1,2]	–	–	–
AP2 – Delete Fragment	A[1, 2], B[1,2,3]	–	–	–	–	–	A[2], B[1]	–	–	–
AP3 - Move Fragment	A[1, 2], B[1,2,3], C[1,2]	–	–	–	–	–	–	–	–	–
AP4 – Replace Fragment	–	–	–	–	–	–	A[2], B[1]	–	–	–
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. 21. Adaptation Patterns Support at the Process Type Level

Change Pattern Support at the Process Instance Level												
Primitive / Pattern	**Academic**								**Commercial**			
	ADEPT2 / CBRFlow	**CAKE2**	**HOON**	**MOVE**	**PoF**	**WASA2**	**WIDE**	**YAWL + Worklets / Exlets**	**Flower**	**Staffware**
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]	–	–	–	–	–	–	–	A[2], B[1]	–		
AP3 - Move Fragment	A[1, 2], B[1,2,3], C[1,2]	–	–	–	–	–	–	–	–	–		
AP4 – Replace Fragment	–	–	–	–	–	–	–	A[1], B[1]	–	–		
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. 22. Adaptation Patterns Support at the Process Instance Level

Feature F2 (Support for Instance-Specific Changes) is provided by most approaches in some form. Ad-hoc changes based on adaptation patterns are only possible in ADEPT2, FLOWer and Worklets/Exlets. While ADEPT2 has realized most adaptation patterns, FLOWer restricts ad-hoc changes to activity deletions and Worklets/Exlets only allow for activity replacements. Furthermore, CAKE2 and WASA2 enable ad-hoc changes based on change primitives. Preplanned changes are supported by CAKE2, HOON, MOVE, PoF, Worklets/Exlets, FLOWer, and Staffware. Fig. 23 gives a detailed overview.

Feature F3 (Correctness of Changes) is realized quite well by most of the academic approaches, except Worklets/Exlets which only provide partial correctness support. Both commercial systems, Staffware and FLOWer, do not use formal correctness criteria in the context of schema evolution and instance migration, which can lead to runtime inconsistencies and errors. This especially holds for the overwriting of process schemes in Flower and the instance migrations in Staffware.

Feature F4 (Traceability and Analysis of Changes) is supported by all systems. However, most of them only provide simple execution and/or change logs and do not enhance these logs with further information (e.g., context of a change). 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 [63] has been developed [57].

Feature F5 (Access Control for Changes) is mostly supported based on a simple role concept. Several systems additionally enable more fine-grained definitions of access rights; e.g., ADEPT2/CBRFlow [59], HOON, PoF, Worklets/Exlets, FLOWer [14], and Staffware allow for specifying distinct authorizations for each pattern. In addition, all these approaches support object-dependent authorizations as well, i.e., authorizations may depend on the process schema or process instance to be modified. The latter is also supported by MOVE and WIDE.

Feature F6 (Change Reuse) is supported in ADEPT2/CBRFlow, CAKE2, PoF, and Worklets/Exlets. Both ADEPT2/CBRFlow and CAKE2 use case-based reasoning techniques for retrieving instance-specific changes which occurred previously in similar context [42,30]. The retrieval component of PoF is primarily based on structural information [64]. Finally, Worklets/Exlets support reuse of sub process fragments through selection rules [12].

Feature F7 (Change Concurrency Control) is addressed by most systems. Approaches which restrict changes to predefined regions (e.g., HOON, PoF, Worklets/Exlets and Staffware) usually allow only one user to modify a particular placeholder activity at a time. Further, changes to different placeholder activities are not conflicting and can therefore be introduced concurrently. Regarding structural adaptations, concurrency control is more complicated. ADEPT2 and CAKE2 allow for concurrent ad-hoc changes [11,30]. 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 [65]. Concurrency of process type and process instance changes is only addressed by ADEPT2 [61,66].

Primitive / Pattern	Academic							Commercial	
	ADEPT2 / CBRFlow	CAKE2	HOON	MOVE	PoF	WASA2	WIDE	YAWL + Worklets / Exlets	Flower	Staffware
Built-In Flexibility										
Patterns for Changes in Predefined Regions										
PP1 – Late Selection of Fragments	–	–	A[1,2], B[1,2], C[2]	–	–	–	–	A[1,2], B[1,2], C[2]	–	A[1,2], B[1,2], C[2]
PP2 – Late Modeling of Fragments	–	A[1], B[1], C[2,3], D[1]	–	A[1], B[1], C[3], D[1,2]	A[1,2], B[2], C[2], D[1,2]	–	–	–	–	–
PP3 – Late Composition of Fragments	–	–	–	–	–	–	–	–	–	–
PP4 – Multi-Instance Activity	–	–	–	–	–	–	+	–	+	+

Change Support Features										
Feature	**Academic**							**Commercial**		
	ADEPT2 / CBRFlow	CAKE2	HOON	MOVE	PoF	WASA2	WIDE	YAWL + Worklets / Exlets	Flower	Staffware
Change Features										
F1 – Schema Evolution, Version Control and Instance Migration	F1[3, 5]	F1[1]	F1[1]	F1[1]	F1[1]	F1[3, 5]	F1[3, 5]	F1[3]	F1[1, 2, 3]	F1[3, 4]
F2 – Support for Instance-Specific Changes	F2[1a,b]	F1[1b, 2b]	F2[2a]	F2[2a]	F2[2b]	F2[1b]	F2[2b]	F2[1a,b, 2a,b]	F2[1b, 2b]	F2[2b]
F3 – Correctness of Changes	+	+	+	+	+	+	+	°	–	–
F4 - Traceability & Analysis	F4[1, 2, 3]	F4[1, 2]	F4[1]	F4[1]	F4[1]	F4[1]	F4[1]	F4[1]	F4[1]	F4[1]
F5 – Access Control for Changes	F5[1, 2, 3]	–	F5[1, 2, 3]	F5[1, 3]	F5[1, 2, 3]	F5[1]	F5[1, 3]	F5[1, 2, 3]	F5[1, 2, 3]*	F5[1, 2, 3]
F6 - Change Reuse	+	+	–	–	+	–	–	+	–	–
F7 - Change Concurrency Control	F7[3, 4]	F7[3]	F7[3]	F7[3]	F7[3]	F7[2]	not applicable	F7[3]	F7[2]	F7[3]

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

Fig. 23. Change Features in Practice

6.5. *Summary of Evaluation Results*

Our evaluation shows that no single system exists which supports all change patterns and features in an integrated way (cf. Fig. 21-23). In particular, none of the approaches offers a holistic change framework considering both adaptation patterns and patterns for changes in predefined regions. ADEPT2 and WIDE score well in respect to adaptation patterns, but lack support for changes restricted to predefined regions. WASA2 provides good support for ad-hoc changes using change primitives, but neither considers changes to predefined regions nor high-level change operations. Finally, CAKE2 supports ad-hoc instance changes and changes to predefined regions based on primitives, but does not consider process type changes.

An integrated change framework considering both adaptation patterns and patterns for changes in predefined regions would allow for 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.

There exists a trade-off between expressiveness of a process meta model and support for structural adaptations. For example, ADEPT2 has been designed with the goal to enable the latter [11]. To allow for an efficient implementation of adaptation patterns, restrictions on the process meta model have been made. Similar restrictions in terms of expressiveness hold for other approaches supporting structural adaptations (CAKE2, WASA2 and WIDE). YAWL, in turn, provides a reference implementation for workflow patterns and therefore allows for a high degree of expressiveness [49]. Structural adaptations have not yet been addressed in YAWL and their implementation would be more difficult 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 realized for expressive process meta models as well.

As discussed, change support features are needed to make changes applicable in practice. However, 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 Christopher Alexander [67] to describe solutions to recurring problems and best practices in architectural design. Patterns also have a long tradition in computer science. Gamma et al. [50] applied the same concepts to software engineering and described 23 design patterns. In the workflow area, patterns have been introduced for analyzing the expressiveness of process meta models [19,20,68]. In this context, control flow patterns describe different constructs to specify activities and their ordering. In addition, workflow data patterns [21] provide ways for modeling the data aspect in PAISs, and workflow resource patterns [22] describe how resources can be represented in workflows. Furthermore, patterns for describing service interactions and process choreographies were introduced [69].

The introduction of workflow patterns has had significant impact on PAIS design as

well as on the evaluation of PAISs and process languages like BPEL [20], BPMN [70], EPC [20], and UML [71]. To evaluate the powerfulness of a PAIS regarding its ability to cope with change, the existing workflow patterns are important, but not sufficient. In addition, a set of patterns addressing the aspect of process change is needed. Although workflow pattern support allows for reducing the need for modifying process instances, these patterns require flexibility to be entirely built into the process model. By contrast, patterns for changes to predefined regions allow deferring decisions from build- to run-time to better cope with uncertainty. In addition, adaptation patterns will allow for structural process modifications to deal with non-anticipated exceptions and evolving needs.

Exception handling patterns [24] like *Rollback* only change the state of a process instance (i.e., its behavior), but not its schema (i.e., its structure). Thus they are well suited for dealing with expected situations. Non-anticipated situations, in turn, might require structural adaptations as well [23]. Change patterns support this by allowing to modifiy the observable behavior of a process instance and its structure. In addition, change patterns can be applied at the process type level decreasing the efforts needed for accomplishing a change. To provide a complete evaluation framework for PAIS flexibility, expected and unexpected exceptions as well as schema evolution must be considered.

Exception handling often requires combined use of several exception handling patterns resulting in rather complex routines. The Exlet approach [47,48] addresses this problem by allowing for the combination of different exception handling patterns to an exception handling process called *Exlet*. Generally, Exlets are executed in parallel to the process instance to be modified and can be reused when similar exceptions re-occur. Exlets allow for "simulating" several of the adaptation patterns. However, as Exlets are executed independently of the process instance without structurally modifying it, end users have to suspend the process instance manually if required. In contrast, change patterns hide this complexity from users by providing high-level change operations.

In [72] it has been shown that patterns can additionally be used for facilitating process modeling. It proposes 9 patterns for business functions (e.g., approval, notification) and shows that modeling efforts can be decreased when using this pattern set as building blocks. This approach speeds up process modeling, change patterns likewise allow for reducing the efforts of accomplishing process changes.

In [73], some of the proposed change patterns are used to implement refactoring techniques, which will be behaviour-preserving if certain pre- and post conditions are met. In particular, refactorings allow PAIS engineers to keep process models maintainable and understandable over time and consequently reduce costs of future process changes.

Most systems considered by our evaluation, model processes in a procedural or imperative way. An exception is provided by Pockets of Flexibility (PoF) [16], which use a combination of imperative and declarative modeling style. The process itself is modeled imperatively, but the placeholder activities are specified in a declarative way using constraints. Other declarative approaches not considered in our evaluation are MOBILE [74] and DECLARE [15,75]. Instead of requiring designers to specify how the process shall be executed, they only have to state what shall be done during process execution. With declarative approaches changes become less frequent. However, run-time adaptations still can be an issue (e.g., a constraint might have to be violated for a particular instance due to an unforeseen situation). Further, constraints themselves may evolve over time, which raises the challenge of propagating changes to ongoing instances. A promising approach

towards this direction is offered by DECLARE [15]. Another challenging issue is the maintenance and testing of constraint-based process models, particularly in case of large constraint sets.

Different frameworks exist for comparing specific aspects related to process change. In [10] the authors provide one such framework for elaborating strengths and weaknesses of adaptive PAISs along typical dynamic change problems. Main emphasis is on investigating formal properties and correctness criteria in connection with dynamic process changes. In [76] graph- and rule-based languages along dimensions along dimensions like flexibility, adaptability, complexity, and expressiveness.

Several approaches target the automatic handling of process exceptions. Some of these approaches [77–79,52] apply structural adaptations to process instances to deal with the exceptions. By using the adaptation patterns offered by existing systems. In ADEPT2, for example, respective patterns are not only accessible via a process editor, but can also be invoked via a powerful application programming interface. Several approaches have utilized ADEPT2's interface to implement agents for automated exception handling [77–79,52].

8. Summary and Outlook

We have proposed 18 change patterns and 7 change support features which – in combination – allow for assessing PAIS change frameworks. In addition, we have evaluated selected approaches and systems regarding their ability to deal with process change. 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 the process management technology which meets their flexibility requirements best (or to find that no system satisfies their requirements). Our work will make the comparison of change frameworks simpler and allows (PA)IS engineers 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 to evaluate their PAISs along these criteria and to extend them towards better support for process changes.

Our future work includes change patterns for aspects other than control flow (e.g., data or resources) and patterns for advanced change scenarios (e.g., adapting data flow when changing control flow). Further, we will evaluate additional academic and commercial systems. Currently, we are also working on a reference implementation supporting all change patterns and change support features. Based on this we will conduct a series of 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, Mathias Weske, Yanbo Han, Mirjam Minor, Daniel Schmalen, Hajo Reijers, Sheetal Tiwari, Ullrich Kreher and Peter Dadam for their valuable feedback regarding the evaluation of the described approaches. In addition, we would like to thank Shazia Shadiq, Michael Adams and Werner Wild for the many fruitful discussions, which helped us to significantly

improve the quality of this paper.

References

[1] M. Poppendieck, T. Poppendieck, Implementing Lean Software Development: From Concept to Cash, Addison-Wesley, 2006.

[2] M. Dumas, A. ter Hofstede, W. van der Aalst (Eds.), Process Aware Information Systems, Wiley Publishing, 2005.

[3] M. Weske, Business Process Management: Concepts, Methods, Technology., Springer, 2007.

[4] E. Dijkstra, A Discipline of Programming., Prentice-Hall, 1976.

[5] R. Lenz, M. Reichert, IT Support for Healthcare Processes - Premises, Challenges, Perspectives., Data and Knowledge Engineering (1) (2007) 39–58.

[6] K. Kochut, J. Arnold, A. Sheth, J. Miller, E. Kraemer, B. Arpinar, J. Cardoso, IntelliGEN: A Distributed Workflow System for Discovering Protein-Protein Interactions, Distributed Parallel Databases 13 (1) (2003) 43–72.

[7] H. Reijers, W. van der Aalst, The Effectiveness of Workflow Management Systems: Predictions and Lessons Learned., International Journal of Information Management (5) (2005) 457–71.

[8] D. Strong, S. Miller, Exceptions and Exception Handling in Computerized Information Processes., ACM Transactions on Information Systems 13 (2) (1995) 206–233.

[9] P. Dadam, M. Reichert, K. Kuhn, Clinical Workflows - The Killer Application for Process-oriented Information Systems?, in: Proc. BIS'00, 2000, pp. 36–59.

[10] S. Rinderle, M. Reichert, P. Dadam, Correctness Criteria for Dynamic Changes in Workflow Systems – A Survey., Data and Knowledge Enginnering 50 (1) (2004) 9–34.

[11] M. Reichert, P. Dadam, ADEPT$_{flex}$ – Supporting Dynamic Changes of Workflows Without Losing Control., Journal of Intelligent Information Systems 10 (2) (1998) 93–129.

[12] M. Adams, A. ter Hofstede, D. Edmond, W. van der Aalst, Worklets: A Service-Oriented Implementation of Dynamic Flexibility in Workflows., in: Proc. Coopis'06, 2006, pp. 291–308.

[13] M. Weske, Workflow Management Systems: Formal Foundation, Conceptual Design, Implementation Aspects., University of Münster, Habil Thesis (2000).

[14] 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.

[15] M. Pesic, M. Schonenberg, N. Sidorova, W. van der Aalst, Constraint-Based Workflow Models: Change Made Easy., in: CoopIS'07, 2007.

[16] S. Sadiq, W. Sadiq, M. Orlowska, A Framework for Constraint Specification and Validation in Flexible Workflows, Information Systems 30 (5) (2005) 349 – 378.

[17] Y. Han, Software Infrastructure for Configurable Workflow Systems., Ph.D. thesis, University of Berlin (1997).

[18] J. Hagemeyer, T. Hermann, K. Just, S. Rüdiger, Flexibilität bei Workflow-Management-Systemen, in: Software-Ergonomie '97, 1997, pp. 179–190.

[19] W. van der Aalst, A. ter Hofstede, B. Kiepuszewski, A. Barros, Workflow Patterns, Distributed and Parallel Databases 14 (1) (2003) 5–51.

[20] N. Russell, A. ter Hofstede, W. van der Aalst, N. Mulyar, Workflow Control-Flow Patterns: A Revised View., Tech. Rep. BPM-06-22, BPMcenter.org (2006).

[21] 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).

[22] N. Russell, A. ter Hofstede, D. Edmond, W. van der Aalst, Workflow Resource Patterns, Tech. Rep. WP 127, Eindhoven Univ. of Technology (2004).

[23] 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.

[24] N. Russell, W. van der Aalst, A. ter Hofstede, Exception Handling Patterns in Process-Aware Information Systems, in: Proc. CAiSE'06, 2006, pp. 288–302.

[25] B. Weber, S. Rinderle, M. Reichert, Change Patterns and Change Support Features in Process-Aware Information Systems., in: Proc. CAiSE'07, LNCS 4495, 2007, pp. 574–588.

[26] B. Mutschler, B. Weber, M. Reichert, Workflow management versus case handling - results from a controlled software experiment, in: Proc. SAC'08, 2008, pp. 82–89.

[27] F. Zhang, E. D'Hollander, Using Hammock Graphs to Structure Programs., IEEE Transactions on Software Engineering 30 (2004) 231–245.
[28] Business Process Modeling Notation Specification, OMG Final Adopted Specification. (2006).
[29] S. Rinderle, M. Reichert, P. Dadam, Flexible support of team processes by adaptive workflow systems., Distributed and Parallel Databases 16 (1) (2004) 91–116.
[30] 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, p. (to appear).
[31] D. Müller, J. Herbst, M. Hammori, M. Reichert, IT Support for Release Management Processes in the Automotive Industry., in: Proc. BPM'06, 2006, pp. 368–377.
[32] B. Schultheiß, J. Meyer, R. Mangold, T. Zemmler, M. Reichert, Prozessentwurf für den Ablauf einer stationären Chemotherapie (in German). (1996).
[33] B. Schultheiß, J. Meyer, R. Mangold, T. Zemmler, M. Reichert, Prozessentwurf am Beispiel eines Ablaufs aus dem OP. (in German). (1996).
[34] B. Schultheiß, J. Meyer, R. Mangold, T. Zemmler, M. Reichert, Prozessentwurf für den Ablauf einer ambulante Chemotherapie (in German). (1996).
[35] I. Konyen, M. Reichert, B. Schultheiß, S. Frank, R. Mangold, Prozessentwurf für den Bereich der minimal invasiven Chirurgie (in German). (1996).
[36] B. Schultheiß, S. Frank, M. Reichert, Prototypische Implementierung des Prozessentwurfs für den Bereich der minimal invasiven Chirurgie mit WorkParty (in German). (1996).
[37] German Association of the Automotive Industry (VDA), Engineering Change Management. Part 1: Engineering Change Request (ECR), V 1.1., Doc. No. 4965, Dec 2005 (2005).
[38] R. Bobrik, Konfigurierbare visualisierung komplexer prozessmodelle, Ph.D. thesis, Univ. of Ulm (2008).
[39] M. Reichert, S. Rinderle, U. Kreher, P. Dadam, Adaptive Process Management with ADEPT2., in: Proc. ICDE'05, 2005, pp. 1113–1114.
[40] M. Minor, A. Tartakovski, R. Bergmann, Representation and Structure-based Similarity Assessment for Agile Workflows., in: Proc. ICCBR'07, 2007, pp. 224–238.
[41] S. Rinderle, B. Weber, M. Reichert, W. Wild, Integrating Process Learning and Process Evolution - A Semantics Based Approach., in: Proc. BPM'05, LNCS 3649, 2005, pp. 252–267.
[42] B. Weber, W. Wild, R. Breu, CBRFlow: Enabling Adaptive Workflow Management Through Conversational CBR., in: Proc. ECCBR'04, LNCS 3155, 2004, pp. 434–448.
[43] R. Lu, S. Sadiq, Managing Process Variants as an Information Resource., in: Proc. BPM06, 2006, pp. 426–431.
[44] S. Sadiq, W. Sadiq, M. Orlowska, Pockets of Flexibility in Workflow Specifications., in: Proc. ER'01, 2001, pp. 513–526.
[45] F. Casati, Models, Semantics, and Formal Methods for the design of Workflows and their Exceptions., Ph.D. thesis, University of Milano (1998).
[46] F. Casati, S. Ceri, B. Pernici, G. Pozzi, Workflow Evolution, Data and Knowledge Engineering 24 (3) (1998) 211–238.
[47] M. Adams, A. ter Hofstede, D. Edmond, W. van der Aalst, Dynamic and Extensible Exception Handling for Workflows., Tech. Rep. BPM-07-03, BPMcenter.org (2007).
[48] M. Adams, A. ter Hofstede, W. van der Aalst, D. Edmond, Dynamic, Extensible and Context-Aware Exception Handling for Workflows., in: Proc. CoopIS'07, 2007.
[49] W. van der Aalst, A. ter Hofstede, YAWL: Yet Another Workflow Language., Information Systems 30 (4) (2005) 245–275.
[50] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.
[51] B. Weber, S. Rinderle, M. Reichert, Change Support in Process-Aware Information Systems - A Pattern-Based Analysis., Tech. Rep. TR-CTIT-07-76, CTIT, University of Twente, The Netherlands (2007).
[52] R. Müller, U. Greiner, E. Rahm, AgentWork: A Workflow System Supporting Rule–Based Workflow Adaptation., Data and Knowledge Engineering 51 (2) (2004) 223–256.
[53] B. Karbe, N. Ramsperger, Influence of Exception Handling on the Support of Cooperative Office Work., in: Proc. IFIP WG 8.4 Conf., 1990.
[54] R. Bobrik, M. Reichert, T. Bauer, View-Based Process Visualization, in: Proc. BPM'07, 2007, pp. 88–95.

[55] S. Rinderle, M. Reichert, M. Jurisch, U. Kreher, On Representing, Purging, and Utilizing Change Logs in Process Management Systems, in: Proc. BPM'06, 2006, pp. 241–256.
[56] J. Pinggera, S. Zugal, B. Weber, W. Wild, M. Reichert, Integrating Case-Based Reasoning with Adaptive Process Management, Tech. Rep. TR-CTIT-08-11, CTIT, University of Twente, The Netherlands (2008).
[57] C. Günther, S. Rinderle, M. Reichert, W. van der Aalst, Change Mining in Adaptive Process Management Systems, in: Proc. CoopIS'06, LNCS 4275, 2006, pp. 309–326.
[58] C. Li, M. Reichert, A. Wombacher, Issues in Process Variants Mining, Tech. Rep. TR-CTIT-08-10, CTIT, University of Twente, The Netherlands (2008).
[59] B. Weber, M. Reichert, W. Wild, S. Rinderle, Balancing Flexibility and Security in Adaptive Process Management Systems., in: Proc. CoopIS'05, LNCS 3761, 2005, pp. 59–76.
[60] D. Domingos, A. Rito-Silva, P. Veiga, Authorization and Access Control in Adaptive Workflows., in: ESORICS 2003, 2003, pp. 23–38.
[61] S. Rinderle, M. Reichert, P. Dadam, Disjoint and overlapping process changes: Challenges, solutions, applications., in: Proc. CoopIS'04, LNCS 3290, 2004, pp. 101–120.
[62] B. Weber, W. Wild, M. Reichert, P. Dadam, ProCycle - Integrierte Unterstützung des Prozesslebenszyklus., KI-Zeitung 12 (4) (2007) 9–15.
[63] Process Mining Research, www.processmining.org (2005).
[64] R. Lu, S. Sadiq, On the Discovery of Preferred Work Practice through Business Process Variants., in: Proc. ER'07, 2007.
[65] H. Reijers, J. Rigter, W. van der Aalst, The case handling case., International Journal of Cooperative Information Systems. 12 (3) (2003) 365—391.
[66] M. Reichert, S. Rinderle, P. Dadam, On the Common Support of Workflow Type and Instance Changes Under Correctness Constraints., in: Proc. CoopIS'03, LNCS 2888, 2003, pp. 407–425.
[67] C. Alexander, S. Ishikawa, M. Silverstein, A Pattern Language, Oxford University Press, New York, 1977.
[68] F. Puhlmann, M. Weske, Using the Pi-Calculus for Formalizing Workflow Patterns., in: Proc. BPM'05, 2005, pp. 153–168.
[69] A. Barros, M. Dumas, A. ter Hofstede, Service Interaction Patterns., in: Proc. BPM'05, 2005, pp. 302–318.
[70] P. Wohed, W. van der Aalst, M. Dumas, A. ter Hofstede, N. Russell, On the Suitability of BPMN for Business Process Modelling, in: Proc. BPM'06, 2006, pp. 161–176.
[71] N. Russell, W. van der Aalst, A. ter Hofstede, P. Wohed, On the Suitability of UML 2.0 Activity Diagrams for BP Modelling, in: Proc. APCCM '06, 2006, pp. 95–104.
[72] L. Thom, J. Lau, C. Iochpe, J. Mendling, Extending Business Process Modeling Tools with Workflow Pattern Reuse., in: Proc. ICEIS'07, 2007.
[73] B. Weber, M. Reichert, Refactoring Process Models in Large Process Repositories., in: Proc. CAiSE'08, 2008, pp. 124–139.
[74] S. Jablonski, K. Stein, M. Teschke, Experiences in Workflow Management for Scientific Computing, in: Proc. DEXA'97 Workshops, 1997, pp. 56–61.
[75] W. van der Aalst, M. Pesic, DecSerFlow: Towards a Truly Declarative Service Flow Language., Tech. Rep. BPM-06-21, BPMcenter.org (2006).
[76] R. Lu, S. Sadiq, A Survey on Comparative Modelling Approaches., in: Proc. BIS'07, 2007, pp. 82–104.
[77] H. Mourão, P. Antunes, Supporting Effective Unexpected Exceptions Handling in Workflow Management Systems, in: Proc. SAC'07, 2007, pp. 1242–1249.
[78] M. Golani, A. Gal, Optimizing Exception Handling in Workflows Using Process Restructuring, in: Proc. BPM'06, 2006, pp. 407–413.
[79] S. Bassil, R. Keller, P. Kropf, A Workflow-Oriented System Architecture for the Management of Container Transportation, in: Proc. BPM'04, 2004, pp. 116–131.