B. Weber, M. Reichert, W. Wild, S. Rinderle-Ma (2009): Providing integrated life cycle support in process-aware information systems. World Scientific Publishing Company, Int'l Journal of Cooperative Information Systems (IJCIS) , Vol. 18, No. 1, pp. 115-165.

International Journal of Cooperative Information Systems

PROVIDING INTEGRATED LIFE CYCLE SUPPORT IN PROCESS-AWARE INFORMATION SYSTEMS

BARBARA WEBER

Department of Computer Science, University of Innsbruck,
Technikerstraße 21a, 6020 Innsbruck, Austria
barbara.weber@uibk.ac.at

MANFRED REICHERT, STEFANIE RINDERLE-MA

Dept. Databases and Information Systems, University of Ulm,
Oberer Eselsberg, 89069 Ulm, Germany
{manfred.reichert,stefanie.rinderle}@uni-ulm.de

WERNER WILD

Evolution Consulting,
Jahnstraße 26, 6020 Innsbruck, Austria
werner.wild@evolution.at

The need for more flexiblity of process-aware information systems (PAISs) has been discussed for several years and different approaches for adaptive process management have emerged. However, only few of them provide support for both changes of individual process instances and the propagation of process type changes to a collection of related process instances. Furthermore, knowledge about process changes has not yet been exploited by any of these systems. This paper presents the ProCycle approach which overcomes this practical limitation by capturing the whole process life cycle and all kinds of changes in an integrated way. Users are not only allowed to deviate from the predefined process in exceptional situations, but are also assisted in retrieving and reusing knowledge about previously performed changes in this context. If similar instance deviations occur frequently, process engineers will be supported in deriving improved process models from them. This, in turn, allows engineers to evolve the PAIS (including the knowledge about the changes) over time. Feasability of the ProCycle approach is demonstrated by a proof-of-concept prototype which combines adaptive process management technology with concepts and methods provided by case-based reasoning (CBR) technology.

Keywords: Adaptive Process, Process-Aware Information System, Process Flexibility, Learning and Evolving Systems, Exception Handling

1. Introduction

The economic success of enterprises increasingly depends on their ability to react to changes in their environment in a quick and flexible way.[1] Examples for such changes include regulatory adaptations (like the introduction of Sarbanes-Oxley[2] or Basel II[3]), market evolution, changes in customer behavior, process improvement and strategy shifts. Companies have therefore identified business agility as a competitive

advantage required for coping with business trends like increasing product and service variability, faster time-to-market[4], and increasing division of labor along the supply chain of goods and services.[5]

Nowadays needed business agility is often hindered by the lacking flexibility of contemporary enterprise information systems. Recently, a new generation of process-aware information systems (PAISs) has emerged to overcome this rigidity.[6,7] To allow for more flexibility, PAISs introduce an additional layer when compared to traditional information systems, which provides an explicit description of the *process logic*. The core of a PAIS is built on top of a process management system, which provides generic functionality for the modeling, enactment, and monitoring of (business) processes.[6] In general, process logic is represented as *process models* (also called *process schemes*) within the PAIS. Such a model represents a particular process type and consists of the activities (i.e., the tasks) to be executed, their control and data dependencies, organizational entities performing the activities, and the business objects manipulated by them. Based on a process model, new *process instances* can be created, each representing a concrete *business case*. At run-time, the PAIS is responsible for enacting these process instances and for coordinating corresponding activities as specified in the process model. In this context, *process monitoring* and *process intelligence* tools provide real-time status information about running processes, and can be used for process analysis and diagnosis.[8]

Though PAISs allow to reduce maintenance costs when compared to contemporary enterprise information systems[1,9], their development is still costly and time consuming. One major reason for this is the missing run-time flexibility of contemporary PAISs. Once a process has been implemented, its logic cannot be adapted anymore during run-time, which often freezes already modeled business processes.[10] However, to allow for the required business agility and to support business processes effectively, PAISs must be flexible at run-time and enable users to quickly adapt the supported processes to environmental changes or exceptional situations. In response to these needs adaptive PAISs have emerged to support changes at the process instance and/or process type level.[11,12,13,14,15]

Instance-specific changes affect a single process instance (i.e., one business case) and are required to handle (unexpected) exceptions.[10] For example, consider a cruciate rupture treatment process for patients, which usually includes a magnetic resonance tomography (MRT), an X-ray, and a sonography. For a particular patient the MRT may have to be skipped as he has a cardiac pacemaker. The respective change is instance-specific and must not affect any other process instances. *Process type changes*, in turn, are applied to adapt the PAIS to changes of the underlying business processes. This is done by creating a new process schema version. Due to new legal regulations, for example, it might become necessary to inform patients about alternative treatment methods before applying one of them. This newly introduced duty requires the modification of the process schema (e.g., inserting an activity `Inform Patient`). For ongoing treatments it must then be decided whether corresponding problems shall be completed according to the old process schema or

be *migrated* to the new process schema version. In the former case only newly admitted patients are informed about potential risks. By contrast, when using PAISs supporting instance migration (e.g., ADEPT2[16] and WASA2[17]) patients with an already ongoing treatment process can be informed as well.

1.1. *Problem Statement*

The need for more flexible PAISs has been recognized for several years. Existing approaches support instance-specific changes as well as the evolution of the process schema at the type level.

Regarding instance changes, same or similar exceptions can occur more than once, making the reuse of existing exception handling procedures desirable.[13,18] The definition of instance changes from scratch, however, requires user experience, especially when *secondary* adaptations become necessary. For example, when deleting a particular activity in a process schema, data-dependent activities might have to be deleted as well, requiring complex user interactions.[10] Therefore, a PAIS should not only allow for deviations from the predefined process, but also support users in reusing knowledge about previous deviations (incl. the process adaptations associated with them). For example, the knowledge that an MRT could not be performed for a patient with cardiac pacemaker is highly relevant when treating other patients with the same or similar problem.

To be able to reuse knowledge about similar deviations, respective process changes must be annotated with contextual information (e.g., information about the reasons for the deviation) and be memorized by the PAIS. This contextual information is needed to ensure that only knowledge about those changes is presented to the user, which are relevant in the current situation. For example, an MRT must not be skipped for patients in general, but only for those with cardiac pacemaker.

Another challenge concerns process evolution. When similar exceptions occur frequently, this often indicates a gap between the modeled and the real-world business processes. This misalignment can stem from errors in the design of a process model or be the result of changing requirements. Therefore, adaptive PAISs should continuously monitor deviations between a predefined process model and actual process enactment in order to detect discrepancies between modeled and observed process behavior. If a particular deviation occurs frequently, the process engineer will have to be informed and the PAIS will have to support her to evolve the process model as needed. Assume, for example, that the activity performing an MRT is skipped frequently. Then it can be beneficial to include this situation directly in the process model to reduce cost of change later. In addition, not only process models evolve over time, but also the knowledge about process changes. In particular, if frequent instance changes trigger a process evolution and are consequently pulled up to the process schema, knowledge about these past instance deviations becomes obsolete. The challenge thereby is to determine, which changes are still relevant and therefore should be kept in the knowledge base and which ones are outdated and

therefore can be dropped.

The high practical relevance of this problem has been confirmed by the application projects performed with ADEPT1[10] in domains like healthcare[19], e-Negotiation[20], and transportation.[21] All these projects have shown that similar exceptions often occur in practice and therefore have to be repeatetly handled. In addition, all these projects have revealed that certain exceptions, which occured frequently could be avoided through a redesign of the process schema.

1.2. *Contribution*

In this paper we present our ProCycle approach, which supports all phases of the process life cycle in an integrated way. It allows for the modeling, the execution and the monitoring of business processes in a flexible way, and enables process changes at the instance as well as the type level. In particular, ProCycle facilitates instance changes during run-time by supporting the retention and reuse of process deviations. Further, it allows for the automated derivation of process type changes to evolve business processes over time and to reduce cost of change. Regarding process type changes ProCycle does not only support the co-existence of process instances of different schema versions, but also enables the migration of ongoing process instances to a new schema version. In addition to process evolution, ProCycle provides system support for evolving knowledge about changes over time. This allows to support the entire process lifecycle seamlessly. To demonstrate the feasability of the ProCycle approach a powerful proof-of-concept prototype, based on the ADEPT2[10,16,22] and CBRFlow[18] technologies, has been implemented. However, most of the concepts described in this paper could be realized on top of other adaptive PAISs (e.g., WASA2[17] or METEOR[23]) as well. ADEPT2 offers full functionality with respect to the modeling, analysis, execution, and monitoring of business processes.[10,14,16,24] In addition, it provides support for adaptive processes at both process instance and process type level. CBRFlow on the other hand supports the reuse of instance changes. Our proof-of-concept implementation builds upon this functionality and extends it such that integrated process life cycle support can be achieved.

Note that the focus of this paper is on control-flow changes. Changes of the resource perspective (e.g., actor assignments) or data perspective are out of the scope of this paper. However, the proposed approach, in principle, is not limited to control-flow changes and could be extended to cover other perspectives as well.

This paper provides a significant extension of our previous work[25,26], which made several simplifications. We provide an in-depth description of all phases of the process life cycle. For example, in our previous work retrieval of similar deviations was an entirely manual process, which solely considered free text for describing the reasons of a deviation.[25] By contrast, this paper partially automates the retrieval process by considering structured information about the process context (e.g., reasons for a deviation and status of the process instance to be modified) as well. Further, if similar deviations can be retrieved, but cannot be reused directly

(e.g., if the process instance has progressed too far such that the change cannot be applied directly), user support for adapting the respective change definition to the current situation will be enabled. This paper further provides substantial extensions regarding process learning and the derivation of process type changes. In particular, we discuss strategies for deriving process type changes and suggest methods for evolving not only process schemes, but also the knowledge about instance deviations.

Section 2 provides background information needed for understanding this paper. Section 3 explains how ProCycle supports retention and reuse of instance changes. In Section 4 we show how quality of the knowledge on instance changes can be ensured. Section 5 focuses on issues relevant for process learning and evolution. Section 6 describes the implementation of ProCycle. Finally, Section 7 discusses related work and Section 8 concludes with a summary and an outlook.

2. Backgrounds

2.1. *Basic Concepts*

A process-aware information system (PAIS) is a specific type of information system which enables process support and allows for separating process logic and application code. At *build-time*, process logic has to be explicitly defined based on the constructs provided by a *process meta model*. At *run-time* the PAIS orchestrates the processes according to the defined logic. Workflow systems like Staffware, Websphere Process Server, Ultimus, ADEPT2[10], and YAWL[27] as well as Case-Handling Systems (e.g., Flower[6,28]) are PAIS-enabling technologies.

For each business process to be supported (e.g., handling a customer request or processing an insurance claim), a *process type* T represented by a *process schema* S has to be defined. For a 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 *activities* or *control connectors* (e.g., XOR-Split, AND-Join) – and a set of *control edges* between them. Control edges specify precedence relations between the different nodes. In addition, a process schema comprises a set of *data elements* and a set of *data edges*. A data edge links an activity with a data element and represents a read or write access of this activity to the respective data element.

Fig. 1a shows a simplified version of the control-flow perspective of a cruciate rupture treatment process. The respective process schema consists of ten activities and six control connectors: Activity `Patient Admission` is followed by activity `Anamnesis & Clinical Examination` in the flow of control, whereas activities `X-ray`, `MRT` and `Sonography` can be processed in parallel (i.e., in arbitrary order). Activities `Initial Treatment & Operation Planning` and `Operative Treatment` will be conditionally executed if the preceding medical examinations show that the patient is suffering from a cruciate rupture and non-operative treat-

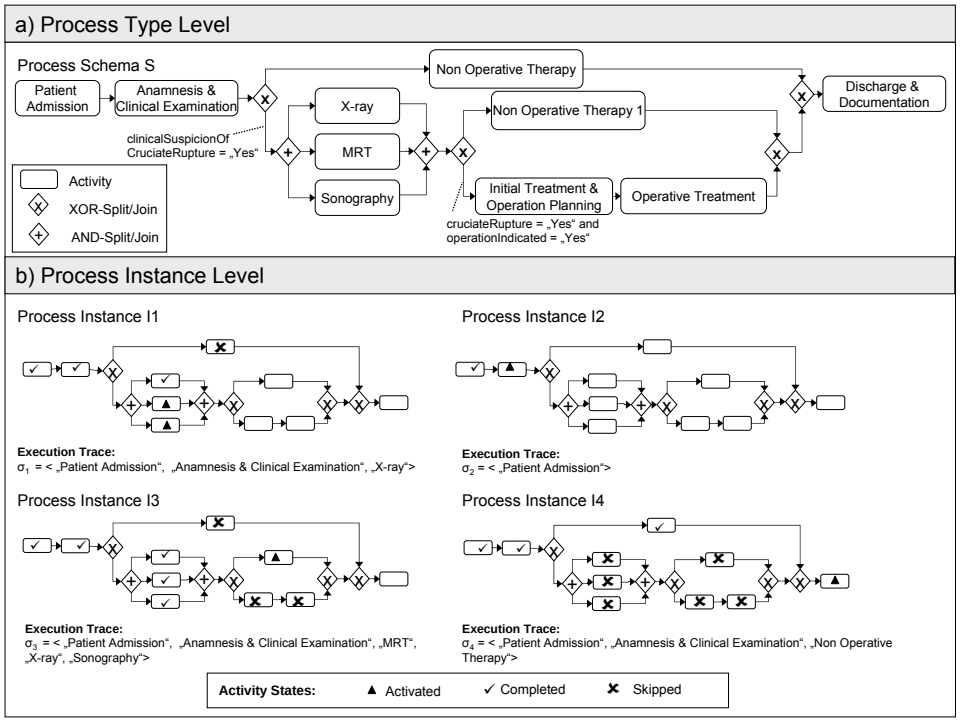

Fig. 1. Core Concepts (Control-Flow Perspective)

ment does not constitute a viable option.

Based on process schema S depicted in Fig. 1a, at run-time corresponding *process instances* can be created and executed (cf. Fig. 1b). Regarding process instance I_1 in Fig. 1b, for example, activities `Patient Admission`, `Anamnesis & Clinical Examination` and `X-ray` are completed, activity `Non Operative Therapy` is skipped, and `MRT` and `Sonography` are concurrently activated. Completion events of activities of an instance are recorded in the instance *trace*.

Definition 2.1. (Execution Trace) *Let $Inst_S$ be the set of all process instances derived from process schema S. The trace σ of a process instance $I \in Inst_S$ is given by $\sigma = <t_1, \ldots, t_k>$ where the temporal order of t_i in σ reflects the order in which activities t_i were completed over S. The set of all traces for all instances $I \in Inst_S$ is denoted as execution log.*

The traces for instances I_1 to I_4 only contain completed activities and are illustrated in Fig. 1b. Generally, a large number of instances being in different states might run on a particular process schema.

2.2. *Process Change*

Adaptive PAISs are characterized by their ability to correctly and efficiently deal with *process changes*. Before discussing different levels of change, we give a definition on the topology of change.

Definition 2.2. (Change in Process Schemas) *Let $\mathcal{P}$ be the set of all process schemas and let S, $S' \in \mathcal{P}$. Let further $\Delta = <op_1, ..., op_n>$ denote a process change, which applies change operations op_i, i=1,...,n sequentially.*

(1) S [$\Delta>$ if and only if change Δ is applicable to S
(2) S [$\Delta> S'$ if and only if Δ is applicable to S (i.e., S [$\Delta>$) and S' is the process schema resulting from the application of Δ to S (also: $S' = S + \Delta$)
(3) S [$\Delta> S'$ if and only if there are process schemes $S_1, S_2, ..., S_{n+1} \in \mathcal{P}$ with $S = S_1, S' = S_{n+1}$ and for $1 \leq i \leq n$: S_i [$\Delta_i> S_{i+1}$ with $\Delta_i = (op_i)$

In general, changes can be triggered and performed at two levels – the process type and the process instance level.

Changes to a process type T may become necessary to cover the evolution of real-world business processes captured by the process schema of this type.[14,17,29] Generally, process engineers can accomplish type changes of a process schema by applying a set of change operations (i.e., change primitives or change patterns) to the current schema version S of type T.[30] This results in the creation of a new schema version S' of the same type (cf. Fig. 2a). Execution of future process instances is then based on S'. In addition, for long-running process instances it is often desired to migrate them to the new schema version S' in a controlled and efficient manner.[24,30] For example, in Fig. 2a process schema S has evolved to new schema version S' by inserting 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).[24,31] For example, instances I_1 and I_2 from Fig. 2a can be migrated to S', while I_3 has already progressed too far and therefore must be completed based on original schema version S. Due to the data dependency between activities `X` and `Y`, the uncontrolled migration of I_3 to S' would not only lead to inconsistent states, but potentially introduce deadlocks or other errors.

By contrast, *changes of individual process instances* are usually performed by end users. They become necessary to react to exceptional situations.[10,18,23] In particular, the effects of such changes must be kept local, i.e., they must not affect other process instances of the same type. In Fig. 2b instance I_5 has been individually modified by inserting activity `X` and by deleting activity `F`.

Instance-specific changes applied to process instance I are denoted as *bias* Δ_I of I, i.e., $\Delta_I = <op_1, ..., op_n>$ comprises the change operations op_i $(i = 1, ..., n)$ applied to I so far and making its schema different from S. Schema $S_I := S + \Delta_I$, which results from the application of Δ_I to S, is called *instance–specific schema* of I. For example, in Fig. 2b the bias of I_5 is given by Δ_I = <`Delete(`I_5`,F)`, `Insert(`I_5`,X,AND-Split1,AND-Join1)`>.

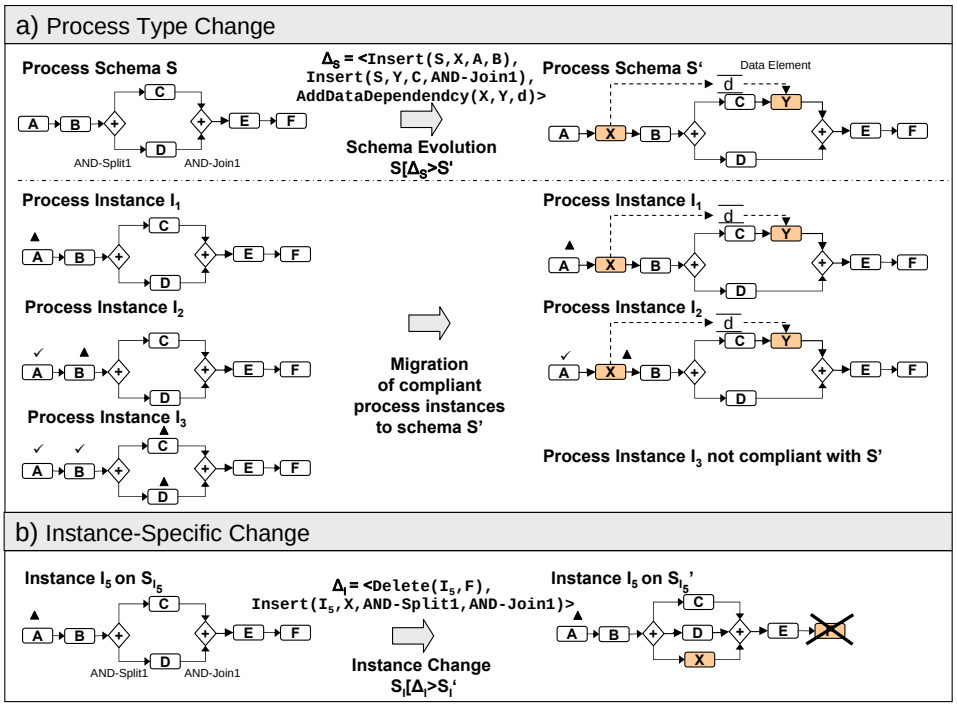

Fig. 2. Process Type and Process Instance Changes

In ProCycle, both process type and process instance changes are based on a set of high-level change operations (i.e., change patterns) with well-defined semantics.[30,32]. Table 3 presents a selection of respective change patterns, which allow for structural modifications of process schemes at a high-level of abstraction. Examples are as shown in Fig. 2b the deletion of activity `F` from process schema S (i.e., `Delete(S,F)`) or the insertion of activity `X` into schema S after node `AND-Split1` and before node `AND-Join1` (i.e., `Insert(S,X,AND-Split1,AND-Join1)`). Formal semantics of the listed patterns is described in.[32]

Note that respective changes can also be performed using change primitives like `add node`, `remove node`, `add edge` and `remove edge`. Following this approach, the realization of a particular adaptation (e.g., activity deletion or insertion) requires the application of multiple change primitives. However, due to their lack of abstraction, changes based on change primitives are more complex and error-prone. In addition, correctness of a process schema has to be explicitly checked after applying the respective set of primitives (for details see Ref. 30);

Name	Change Pattern	Short Description of Pattern
Insert Process Fragment	`Insert(S,f,A,B)`	Process fragment (*) f is inserted between activity sets A and B.
	`CondInsert(S,f,A,B,cond)`	A conditional branch with process fragment f will be inserted between activity sets A and B.
Delete Process Fragment	`Delete(S,f)`	Deletes process fragment f from schema S.
Move Process Fragment	`Move(S,f,A,B)`	Process fragment f is moved from its current position and re-inserted between activity sets A and B.
Replace Process Fragment	`Replace(S,f,g)`	Process fragment f is replaced by fragment g in schema S.
Swap Process Fragments	`Swap(S,f,g)`	Process fragments f and g are swapped.

(*) A process fragment can either be an atomic activity or a sub-graph with single entry and single exit nodes

Fig. 3. Examples of Process Change Patterns

2.2.1. *Backgrounds on Case-Based Reasoning*

Case-Based Reasoning (CBR) is a contemporary approach to problem solving and learning.[33] Problems and their solutions are described as *cases* and stored in *case–bases*. New problems are dealt with by drawing on past experiences and by adapting respective solutions to the new problem situation. Reasoning based on past experiences is a powerful and frequently applied way to solve problems by humans.[34] A physician, for example, remembers previous cases to correctly diagnose the disease of a newly admitted patient. A banker uses her experiences about previous cases in order to decide whether to grant a loan or not. Generally, a case is a contextualized piece of knowledge representing an experience[33], which typically consists of a *problem description* and the corresponding *solution*. In particular, CBR uses specific knowledge of past experiences to solve new problems. CBR also contributes to incremental and sustained learning: every time a new problem is solved, information about it and its solution is retained, and therefore immediately made available for solving future problems.[34]

Our approach described in this paper relies on *Conversational CBR* (CCBR) which constitutes an extension of the CBR paradigm. CCBR actively involves users in the case retrieval process.[35] A CCBR system can be characterized as interactive system. Via a mixed-initiative dialogue it guides users through a question-answering sequence in a case retrieval context. In contrast to traditional CBR, the user is not required to provide a complete a-priori problem specification for case retrieval. He does also not need to have a clear picture of what is relevant for problem solving. Instead, CCBR guides users, who might only have a vague idea of what they are searching for, to assess the given situation and to find relevant cases. Users can provide already known information at any time. Therefore, CCBR is especially suited to support inexperienced users in handling exceptional situations that cannot be dealt with in a fully automated way.

2.3. *Process Life Cycle Support in Adaptive PAISs*

Adaptive PAISs extend traditional PAISs with the ability to deal with process changes at different levels. This enables life cycle support as depicted in Fig. 4: At build-time an initial representation of a business process is created either by explicitly modeling the process (based on analysis results) or by discovering process models through the mining of execution logs[8] (1). At run-time new process instances can be derived from the predefined process schema (2). In general, process instances are executed according to the process type schema they were derived from, and (non-automated) activities are assigned to process participants to perform the respective activities (3). However, when exceptional situations occur during run-time, process participants may deviate from the predefined schema (4). While execution logs record information about the start and completion of activities as well as their ordering, process changes are recorded in change logs (5). The analysis of respective logs by a process engineer and process intelligence tools respectively allows discovering malfunctions or bottlenecks.[8,36] This information often results in an evolution of the process schema (6). If desired, changes can be propagated to running process instance (7).

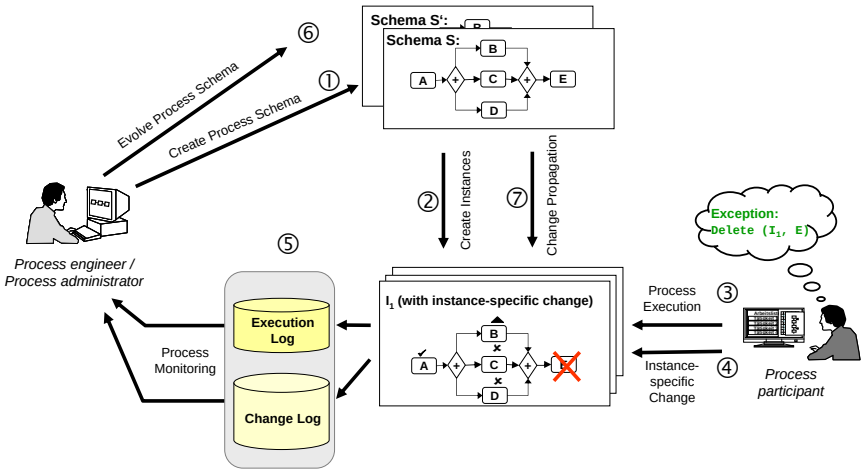

Fig. 4. Process Life Cycle Support with Adaptive PAISs

3. Retention and Reuse of Instance Deviations

To deal with unexpected exceptions adaptive PAISs allow users to define instance changes on-the-fly, e.g., to add, delete or move activities. In general, different methods for dealing with unanticipated exceptions exist. This includes structural modifications of instance-specific process schemes as well as changes of their state (e.g.,

Redo activity or Rollback process). In this paper we focus on structural instance changes. However, the ProCycle approach, in principle, is not restricted to structural adaptations and can be extended to cover other exception handling procedures like state changes as well (and ADEPT2 already provides basic support for this[10,37]).

Usually, instance changes are performed by end users requiring significant experience. This is particularly true when changes have to be defined from scratch. Change complexity will become particularly evident if multiple adaptations are necessary to handle an exception. For example, when deleting an activity during run-time, (data-)dependent activities might have to be deleted as well (e.g., when deleting activity `Write Medical Report`, the subsequent activity `Validate Medical Report` is no longer required; or when inserting a new activity its inputs and outputs have to be bound to process data elements[10]).

Adaptive PAISs like ADEPT2[10], $WASA_2$[17], METEOR[23], and CAKE2[13] support the user in defining instance changes. In particular, changes can be expressed at a high level of abstraction ensuring also their syntactical correctness. Furthermore, it is checked whether the process instance is *compliant* with the new process schema (cf. Section 2.2).[10]

Generally, user assistance going beyond syntax checks is needed to enable less experienced users to perform instance changes as well. Instead of defining every instance change from scratch existing knowledge about previously defined deviations should be exploited, especially when similar deviations occur more than once. However, pure syntactical information (i.e., information about the applied change patterns), as provided by change logs in existing adaptive PAISs[38], is not sufficient to adequately support retrieval and reuse of corresponding change definitions later. Additionally, information about the application context of the deviations is required (like the deviation is due to an allergic reaction of the patient). Later this allows the PAIS to present just those deviations to the user relevant in the given exceptional situation. Regarding a cruciate rupture treatment process, for example, the context information that the MRT was skipped for a patient with cardiac pacemaker is useful for other physicians as well, particularly when treating other patients with similar problems. Consequently, instance changes should be annotated with relevant context information to foster their retrieval and reuse later on. Section 3.1 describes how process changes and their context can be captured. In Section 3.2 we show how knowledge about instance changes can be retrieved, adapted and reused when similar exceptional situations occur.

3.1. *Memorizing Process Instance Deviations*

ProCycle uses case-based reasoning (CBR) for memorizing process instance deviations. Thereby, changes of individual process instances are annotated with relevant context information (e.g., reasons for the deviation), which allows users to retrieve and reuse this information when similar exceptional situations occur.

3.1.1. *Representing Process Instance Deviations as Cases*

We describe step-by-step how ProCycle represents process instance deviations as cases to allow for their later retrieval, adaptation and reuse. In addition, we give insights into our motivation and the criteria used for choosing this particular solution. In its simplest form, in ProCycle, a case c represents a process instance deviation consisting of a textual problem description and a solution part. The textual problem description pd_c briefly describes the exceptional situation that made the deviation necessary. The solution part Δ_c, in turn, comprises the change operations which were applied to cope with the exception. In addition, for each case c its reuse frequency $freq_c$ is maintained.

Definition 3.1. (Case) *A case c is a tuple (pd_c, Δ_c, $freq_c$) where*

- *pd_c is a textual problem description*[a]
- *$\Delta_c = <op_1, \ldots, op_k>, k \in \mathbb{N}$ is the solution part of the case comprising a list of change operations, which were sequentially applied to one or more process instances in the given problem context*
- *$freq_c$ is the reuse frequency of the case*

Example 3.1. (Case Representation) For our running example, Fig. 5 illustrates how an instance change deleting a process activity (i.e., `Delete(`I_1`, MRT)`) can be represented as case and be annotated with context. The problem description pd_{c_1} of this case c_1 briefly describes the reason for skipping the MRT in free text and the solution part sol_{c_1} contains the change operation needed to remove the MRT from the process instance schema (i.e., `Delete(`S_I`, MRT)`). In addition, $freq_{c_1} = 1$ expresses that case c_1 has been applied exactly once. To foster reuse and to allow for the application of c_1 to other process instances as well, the parameter representing the instance schema to be modified, is generalized. For this, parameter I_1 of operation `Delete(`I_1`, MRT)` is replaced by a placeholder for an instance-specific schema S_I. Thus a case description absracts from a particular process instance.

ProCycle stores instance changes related to a process schema version S as cases in the case-base cb_S associated with S.

Definition 3.2. (Case–Base) *Let $\mathcal{C}$ be the set of all cases representing instance deviations in a PAIS. Then: A case–base $cb_S \subseteq \mathcal{C}$ corresponds to the set of all cases associated with S (i.e., cases which were applied to at least one instance running on S).*

This simple representation with unstructured problem description (cf. Def. 3.1) only allows for limited user guidance during case retrieval. It is thus up to the user to formulate suitable queries. This works well for experienced users having a clear idea of what they are searching for and therefore knowing which search terms to

[a]The problem description might be free text or be based on a domain-specific ontology.

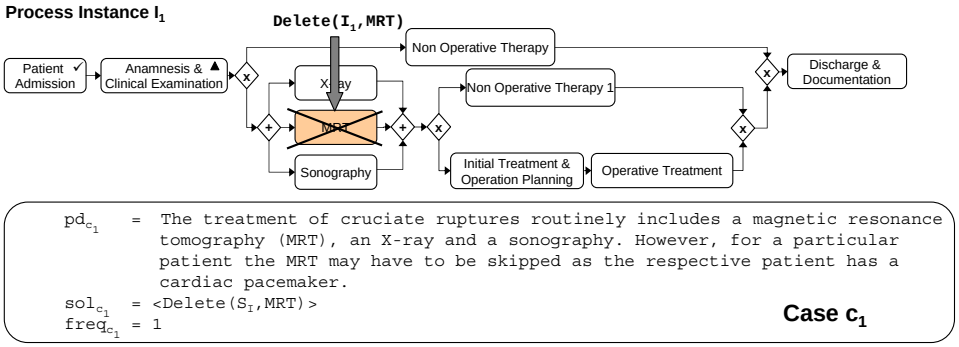

Fig. 5. Case Representation

use to achieve suitable results. Concerning inexperienced users, this lack of guidance often results in either too broadly or too narrowly formulated queries. To ease case retrieval for these users as well and to partially automate case retrieval, ProCycle additionally captures application context information about instance deviations (e.g., the reasons for deviating from the original process schema version). This context information can be used during case retrieval to filter out those cases not relevant in the current exceptional situation. Regarding our running example, a case skipping the MRT for a patient should be filtered out for all patients without cardiac pacemaker.

3.1.2. *Capturing Application Context of Instance Deviations*

When capturing the application context of a process instance change, one major challenge is to adequately represent this context information and to gather it on-the-fly. The application context of process instance deviations usually cannot be defined in a generic way, but is domain- or application-specific. ProCycle allows process engineers to define an (application-specific) context model. The respective context model defines what application context information might be relevant for reusing deviations and thus should be collected. For example, patient information like age, allergies and anamnesis should be part of a context model in the medical domain. By contrast, a context model for the logistic industry might contain information about customers, location of trucks, weather, and road conditions.

Definition 3.3. (Application Context Model) *An application context model Ctxt=(DataObjSet, AttrSet, Attrs) consists of a set of data objects DataObjSet, a set of context attributes AttrSet, and a function Attrs: DataObjSet* $\mapsto$ *AttrSet. The latter assigns to each data object* $d \in DataObjSet$ *a set of context attributes* $Attrs(d) \subseteq AttrSet$*. Each context attribute* $attr = (aName, aType, min, max) \in Attrs$ *has a name aName, a data type aType, a minimal occurrence min, and a maximum occurrence max. The data type of a context attribute either is atomic*

(e.g., Boolean, Integer, String or Time) or complex (i.e., refers to already defined data objects).

Example 3.2. (Application Context Model) Figure 6 shows a simplified example of an application context model in the medical domain. It consists of the data objects `Diagnosis`, `Patient`, `ProblemList`, and `Therapy`. Attributes `problemList`, `diagnoses` and `therapies` are examples for context attributes with complex type as they are composed of other data objects; `age`, `weight` and `hasPaceMaker` constitute attributes with simple type. Attributes with a minimum occurrence greater than one constitute collections (e.g., `therapies` and `diagnoses`).

Object	Type	Min	Max
⊞ Diagnosis			
⊟ Patient			
age	Integer	1	1
problemList	ProblemList	1	1
weight	Integer	1	1
⊟ ProblemList			
diagnoses	Diagnosis	0	1000
hasPacemaker	Boolean	0	1
therapies	Therapy	0	1000
⊞ Therapy			

Fig. 6. Sample Application Context Model

Regarding a particular instance deviation, usually only a subset of the data objects and attributes from the application context model is relevant to describe the reasons for this deviation. It is therefore inefficient and also not necessary to memorize the full context model in the case–base.

Example 3.3. (Relevant Application Context) The application context model from Fig. 6 describes a patient record including the complete patient history. The decision to skip the MRT during the cruciate rupture treatment of a particular patient, however, only has become necessary because the respective patient has a cardiac pacemaker.

To effectively support users in reusing similar cases and to avoid storage of irrelevant context information, knowledge about the exact reasons of a deviation is required. Automated discovery of relevant application context attributes using data mining techniques (i.e., the analysis of historical data using statistics) is not applicable in the given scenario. To statistically identify correlations between deviations and context attributes, a large set of deviations is needed and deviations have to occur frequently. As ProCycle wants to support users from the first deviation onwards, it cannot rely on a long learning process. In addition, user support should be offered for rarely occurring deviations as well.

To overcome this problem, whenever an exception occurs, ProCycle encourages users to state which parts of the context model are relevant for their decision to deviate from the predefined process schema. Note that in many domains (e.g.,

healthcare[5,39] or automotive engineering[4]) users have to give reasons when deviating from the standard procedure anyway (e.g., to fulfill the legal obligation for documentation in the medical domain[39]). Following this approach the reasons for the change can be immediately captured. This allows for the reuse of changes already after the first case has been added to the system. In addition, memorizing the whole process context as part of the case is not required, but the application context of the deviation can be narrowed to the relevant context attributes. Complementary to our approach a snapshot of the entire context model (or a predefined part of it) could be automatically stored in the change log of the adaptive PAIS whenever a process instance deviation is performed.

ProCycle adopts and extends the conversational case-based reasoning approach of Ref. 40, and allows users to describe the reasons for a deviation in an intuitive way as question-answer pairs $qaSet_c$. Thereby each question-answer pair (q, a) characterizes one particular condition making the deviation necessary. Question-answer pairs are also used during case retrieval to assist users in finding similar cases (cf. Section 3.2). The question q of such a pair (q, a) corresponds to free text and the answer a is mapped to an answer expression, which can be automatically evaluated by the PAIS. ProCycle internally uses OCL (Object Constraint Language) as language for representing answer expressions.[41] Note that end-users are not expected to enter these expressions by hand, but are assisted in describing the reasons for the deviation (cf. Section 4.1). For example, domain-specific user interfaces can be built on top of the OCL representation. As an example consider the medical domain.[5] Instead of requiring a phyisician to formulate the answer expression a form can be presented to him with a list of symptoms: he then just has to select those symptoms relevant for the deviation.

Example 3.4. (Question-Answer Pair) Consider Example 3.1 where a physician decides to skip the `MRT` activity for a particular patient and assume that the decision to perform this deviation is triggered by the fact that the treated patient has a cardiac pacemaker. To record the reasons for this instance deviation the physician enters the question `Does the patient have a cardiac pacemaker?` as free text as well as an answer expression which either evaluates to `true` or `false`[b]. For example, `Patient.problemList.hasPacemaker = 'Yes'` constitutes such answer expression. It uses the `hasPacemaker` attribute of data object `Patient` from the application context model shown in Fig. 6.

To include context information in the representation of a case, Definition 3.1 has to be extended by adding question-answer pairs.

Definition 3.4. (Case - Extended With Question-Answer Pairs) *A case c is a tuple (pd_c, Δ_c, $qaSet_c$, $freq_c$) where*

[b]The entering of question-answer pairs can be interactively supported by a respective user interface (cf. Section 4).

- pd_c, Δ_c *and* $freq_c$ *are defined as in Def. 3.1*
- $qaSet_c = \{(q_1, a_1), \ldots, (q_n, a_n)\}$, $n \in \mathbb{N}$ *denotes a set of question-answer pairs depicting the application context of the case (with* $q_i \in \mathcal{Q}$ *and* $a_i \in \mathcal{A}$ *where* $\mathcal{Q}/\mathcal{A}$ *denotes the set of all possible quesions/answers.*

Example 3.5. (Representing Deviations as Cases) Fig. 7 shows how case c_1 from Fig. 5 can be enriched with application context information. In addition to the textual problem description pd_c and the solution part sol_c, case c_1 contains a question-answer pair describing the application context of the applied change.

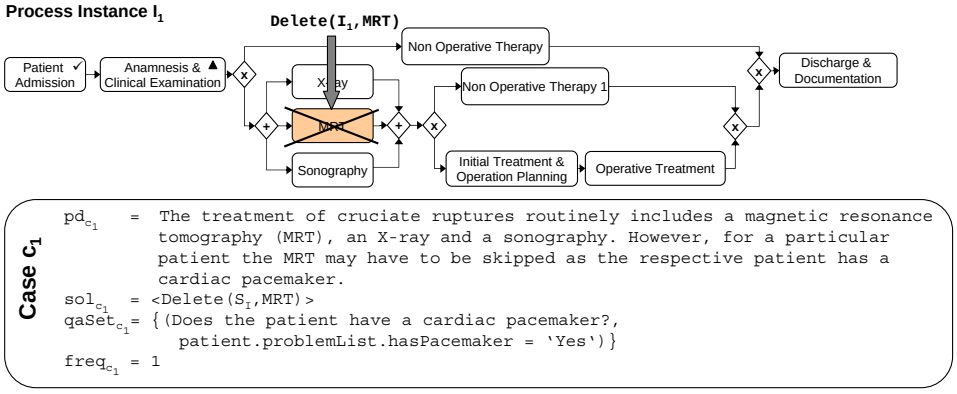

Fig. 7. Annotating Instance-Specific Changes with Application Context

3.1.3. *Going beyond Predefined Context Attributes*

So far, we are able to annotate instance deviations with application context information and to make statements about which context attributes are relevant depending on the concrete deviation. In addition to the context attributes specified in the context model there might be other criteria not (yet) known to the system, but nevertheless contributing to the decision to deviate from the predefined process. In such a situation the user should be able to record these criteria as reasons for the deviation.

Example 3.6. (Additional Application Context) Assume that the planned cruciate ligament operation for patient Miller has to be postponed – an emergency surgery has to be performed for patient Smith and therefore Miller has to be transferred back to the ward in order to wait for his intervention. Under such circumstances it is unlikely that the reasons for the deviation can be captured automatically. However, reuse of such cases is desirable as well and should therefore be supported.

In such scenarios, formal answer expressions are not applicable as the reasons for the deviation are outside the scope of the pre-specified context model. By contrast,

they must be manually entered by users. Instead of mapping answers to logical expressions on attributes from the application context model, free text can be used for both the questions and the answers, thus allowing for the specification of arbitrary reasons (even if they are not (yet) captured in the PAIS). Such free text answer expressions can be further utilized to evolve the application context model and to enable automation of the retrieval of previous instance deviations (cf. Section 4.3). In particular, if cases with free text answers are frequently reused, extending the application context model often will make sense.

3.2. *Retrieving and Adapting Similar Process Instance Deviations*

Once instance changes and their *application context* (i.e., reasons for the process change) are stored by the PAIS, they can be retrieved, adapted and reused when similar exceptional situations occur and thus similar deviations become necessary. To effectively support users in handling exceptions only such cases should be presented to them which are relevant to resolve the given exception. For this purpose the cases which occurred in an application context similar to the current exceptional situation must be determined. In addition to the application context the current state of the process instance for which a change should be performed (denoted as *control context*) has to be taken into account as well since it restricts the applicability of the adaptations defined by a particular case.[10,12] We first describe how similarity in respect to *application context* is calculated in ProCycle. We then show how the instance status (i.e., *control context*) is considered to retrieve relecant cases. Finally, we illustrate how knowledge about previous instance change can be reused.

3.2.1. *Application Context Similarity*

Application context similarity measures how well the question-answer pairs of a stored case c (cf. Def. 3.4) match with query qu representing the application context of the problem situation the user is confronted with. ProCycle uses conversational case-based reasoning to assist users in formulating this query in an interactive and incremental way. Fig. 12 gives an overview of the case retrieval process as supported by ProCycle. When an exceptional situation occurs during execution of a process instance I, case retrieval is initiated (1). ProCycle then creates a list of questions and possible answers to guide the user in formulating the query capturing the exceptional situation (2). As next step, structured answer expressions are automatically evaluated (3) and the answered questions are added to the query (4). Similarity is then calculated and cases are ranked by decreasing similarity (5). The list of ranked cases as well as the list of questions with possible answers is displayed to users (6), who then can answer any of the not yet answered questions (7). This leads to a query expansion and a re-ranking of the cases (5+6).

In the following the retrieval process is described step-by-step using an example case–base $cb_S = \{c_1, c_2\}$ (cf. Fig. 9a). After having initiated case retrieval an empty query qu is created. To guide the user in incrementally expanding qu for cb_S,

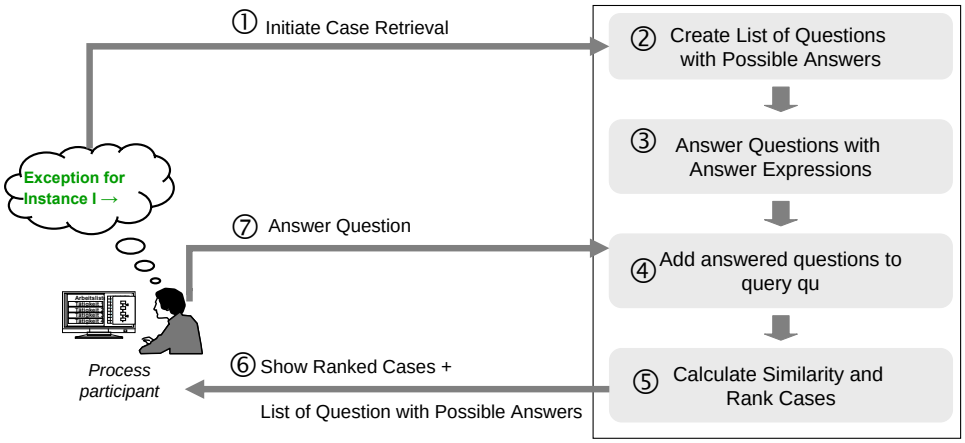

Fig. 8. Retrieving Similar Cases in ProCycle

ProCycle calculates the set of possible questions $qSet(cb_S)$. In addition, for each question q the set of all possible answers $aSet(q)$ is determined. This also includes an entry `OTHERANSWER`, which can be selected if none of the answer options applies. Fig. 9b shows the list of questions with possible answers, as it can be created for cb_S. In total, this list comprises three questions. For example, question `'Does the patient have fluid in the knee?'` has two possible answers: `'A significant amount'` or `OTHERANSWER` (if the first answer `'A significant amount'` does not apply).

a)

$qaSet_{c_1}$ = {(Does the patient have a cardiac pacemaker?, Patient.problemList.hasPacemaker = 'Yes')}

Case c_1

$qaSet_{c_2}$ = {(Does the patient have fluid in the knee?, 'A significant amount'), (Does the patient have an acute effusion of the knee?, 'Yes')}

Case c_2

b)

List of Questions with Possible Answers	
Question	**Possible Answers**
Does the patient have a cardiac pacemaker?	{Patient.problemList.hasPacemaker = ‚Yes', OTHERANSWER}
Does the patient have fluid in the knee?	{‚A significant amount', OTHERANSWER}
Does the patient have an acute effusion of the knee?	{‚Yes', OTHERANSWER}

c)

Query qu	
Question	**Given Answer**
Does the patient have a cardiac pacemaker?	OTHERANSWER

d)

List of Retrieved Cases for Query qu	
Case	**Appl. Context Similarity**
c2	50%
c1	0%

e)

Query qu‘	
Question	**Given Answer**
Does the patient have a cardiac pacemaker?	OTHERANSWER
Does the patient have fluid in the knee?	‚A significant amount‘

f)

List of Retrieved Cases for Query qu‘	
Case	**Appl. Context Similarity**
c2	75%
c1	0%

Fig. 9. List of Questions

As a next step ProCyle automatically evaluates all questions with associated answer expression. It then adds the derived question-answer pairs (q_i, a_i) to query

qu. For example, as illustrated in Fig. 9c question q_1 = `'Does the patient have a cardiac pacemaker?'` has been evaluated to false (i.e., a_1 = `OTHERANSWER`) resulting in the intermediate query $qu = \{(q_1, a_1)\}$. After this step, query qu is used to determine cases $c \in cb_S$ whose application context is most similar to the current one represented by qu. To calculate application context similarity between query qu and case c, the question-answer pairs in $qaSet_c$ and qu are matched.

Definition 3.5. (Application Context Similarity) *Let $c \in cb_S$ be a case and qu be a query representing the present application context. Then: The application context similarity $sim(qu, c)$ between qu and c corresponds to the number of questions with same answers $same(qu, qaSet_c)$ minus the number of questions with different answers diff(qu, $qaSet_c$) divided by the total number of questions in the case. To obtain similarity values between 0 and 1 we further normalize $sim(qu, c)$). Formally,*

$$sim(qu, c) = \frac{1}{2} * (\frac{same(qu, qaSet_c) - diff(qu, qaSet_c)}{|qaSet_c|} + 1)$$

Taking query $qu = \{(q_1, a_1)\}$, for c_1 we obtain $sim(c_1, qu) = 0$ as the question-answer pair in case c_1 does not match with qu (cf. Fig. 9d). Regarding c_2 we obtain $sim(c_2, qu) = 0.5$, because none of the two questions of c_2 has been answered.

In summary, ProCycle displays the list of ranked cases as well as the list of questions with possible answers to users, who then can answer any of the not yet considered questions to further refine the query. Whenever an additional question is answered or an answer is modified similarity is recalculated. Cases are displayed to the user ordered by decreasing similarity (cf. Fig. 9d+f).

Consider the above example and assume that the user additionally answers question q_2 = `'Does the patient have fluid in the knee?'` with a_2 = `'A significant amount'`. This results in new query $qu' = \{(q_1, a_1), (q_2, a_2)\}$ (cf. Fig. 9e). Considering query qu' for c_1 we still obtain $sim(c_1, qu) = 0$. However, for case c_2 similarity $sim(c_2, qu)$ increases to 0.75 (cf. Fig. 9f).

3.2.2. *Control Context*

In addition to application context, ProCycle considers the state of a process instance when retrieving similar cases. Generally, the state of a process instance restricts the applicability of the adaptations defined by the solution part of a particular case.[10,12]

Example 3.7. (Applicability of a Case Depending on Control Context) Consider case c_1 in Fig. 7, which skips activity `MRT` in a treatment process. Further, consider further process instances I_1, I_2 and I_3 from Fig. 1b as well as their execution traces. While c_1 is applicable to I_1 and I_2, it is not relevant for I_3. Regarding I_3 the `MRT` activity is completed and can therefore be not skipped anymore.[24]

To determine whether a case c is applicable to a given process instance I or not we introduce the notion of *case compliance*.

Definition 3.6. (Case Compliance) *Let $c = (pd_c, \Delta_c, qaSet_c, freq_c)$ be a case with solution part $\Delta_c = <op_1, .., op_n>$, $n \in \mathbb{N}$. Then: Process instance I with schema S_I and execution trace σ_I is denoted as case compliant with c iff*

(1) S_I [$\Delta_c>$ S_I', i.e., Δ_c can be correctly applied to instance schema S_I and the application of Δ_c to S_I results in instance schema $S_I' = S_I + \Delta_c$ (where change operations are applied sequentially according to their order in Δ_c).
(2) Execution trace σ_I (cmp. Def. 2.1) is producible on S_I' as well.[c]

Example 3.8. (Case Compliance) Consider case c_1 from Fig. 7 with solution part Δ_{c_1} = <Delete(S_I, MRT)>. Let S_I be the schema according to which instances I_1, I_2 and I_3 are executed (cf. Fig. 10a) and assume that Δ_{c_1} is applied to S_I, resulting in new schema version $S_{I'}$ (cf. Fig. 10b). When considering the execution traces of instances I_1, I_2 and I_3 (cf. Fig. 10c), I_1 and I_2 are case compliant with c_1 as their traces are producible on $S_{I'}$. Instance I_3, in turn, does not comply with case c_1 since its trace σ_3 cannot be produced on $S_{I'}$.

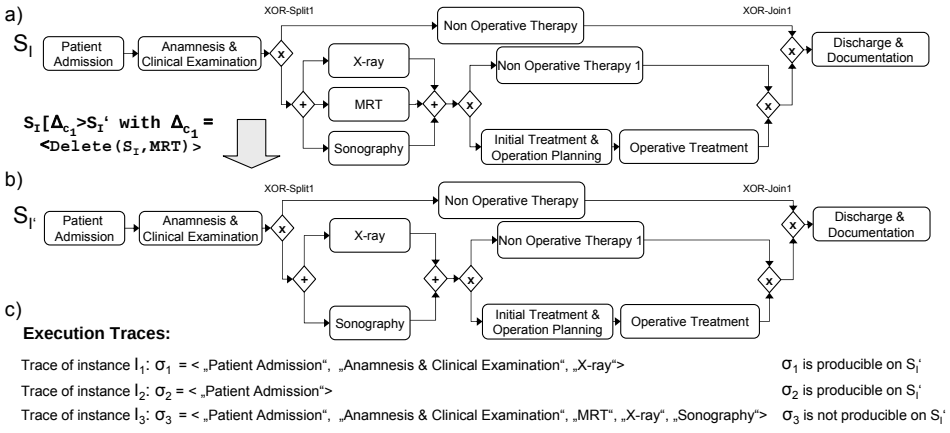

Fig. 10. Case Compliance

Case compliance is a necessary prerequisite for a case to be directly applicable to the respective process instance. It guarantees that no inconsistencies or errors are introduced when applying the case (i.e., the operations of its solution part) to this instance. As the following example shows, however, it is not sufficient to only consider case compliance when retrieving relevant cases.

[c]This notion could be relaxed in connection with loops. We omit a discussion of loop tolerant adaptations in this paper and refer to Ref. 24 instead.

Example 3.9. (Effect of Changes) Consider instance schema S_I as depicted in Fig. 11a and assume that the solution part of case c_2 shall be applied to S_I. The respective case inserts two activities `Follow-Up Examination` and `Puncture` after activity `Non Operative Therapy` and before `XOR-Join1`, resulting in schema $S_I{}'$ (cf. Fig. 11b). Further, consider the execution trace of instance I as depicted in Fig. 11c. As the events logged in this trace are re-producible on $S_I{}'$, I is case compliant with c_2. This guarantees that the application of c_2 to I does not introduce any inconsistency. However, applying c_2 to I does not have any effect on instance execution as the branch with activity `Non Operative Therapy` has been skipped for I before (and thus the newly inserted activities are skipped as well due to a deadpath elimination).

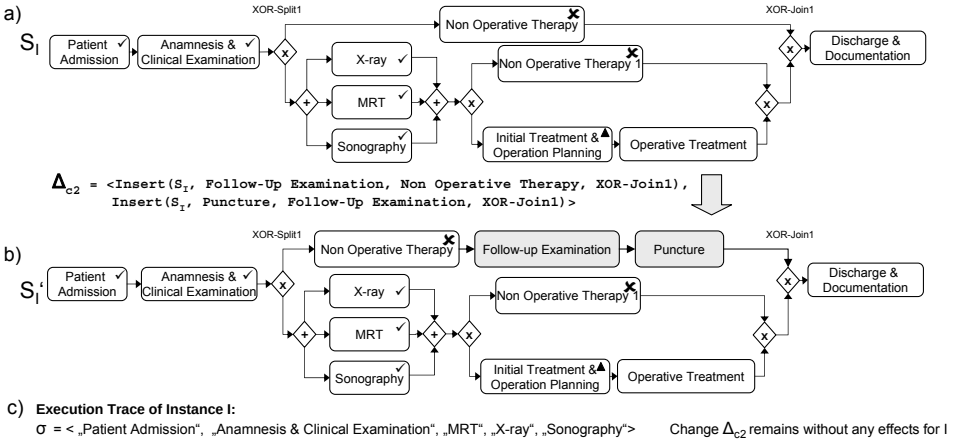

Fig. 11. Case Compliance and Changes Without Effects

Generally, changes within skipped parts of an instance schema have no effect on instance execution except the instance loops back later. In the latter case the change might affect execution of future loop iterations. To indicate whether a change has an effect on a given instance we introduce function $hasEffect(c, I)$.

Definition 3.7. (Effect of a Change) *Let $c \in cb_S$ be a case with solution part Δ_c and $I \in Inst_S$ be a process instance on S. Then:*

$$hasEffect(c,I) = \begin{cases} 1 & \textit{if } \Delta_c \textit{ has an effect on the behavior of } I \\ 0.5 & \textit{if } \Delta_c \textit{ does not have an immediate effect on the behavior of } I \textit{ but might have one when } I \textit{ loops back} \\ 0 & \textit{if } \Delta_c \textit{ does not have any effect on the behavior of } I \end{cases}$$

During case retrieval it makes sense to exclude cases, which neither would have an effect on I (i.e., $hasEffect(c, I) = 0$) nor with whom I is case compliant. However, in certain situations a case c can be adjusted such that it becomes applicable to I. We therefore introduce function $isAdjustable(c, I)$ to indicate whether case c

can be adjusted in such a way that I becomes case compliant with c and c has an effect on I.

Definition 3.8. (Case Adjustability) *Let $c \in cb_S$ be a case with solution part Δ_c and $I \in Inst_S$ be a process instance. Then:*

$$isAdjustable(c, I) = \begin{cases} 1 & \textit{if } c \textit{ can be adjusted such that } c \textit{ has an effect on } I \\ & \textit{(i.e., hasEffect(c,I)>0) and } I \textit{ becomes case compliant} \\ 0 & \textit{otherwise} \end{cases}$$

Whether a case c can be adjusted or not also highly depends on the change patterns used in Δ_c. For change patterns `Delete Process Fragment`, `Replace Process Fragment` and `Swap Process Fragment` no adjustment is possible. This applies for situations where case compliance is not met as well as for situations in which an instance change does not have any effect. By contrast, regarding patterns `Insert Process Fragment` and `Move Process Fragment` the solution part of a case can be adjusted by changing the corresponding parameterization (e.g., adjusting the position at which the process fragment shall be (re-)inserted).

Example 3.10. (Case Adjustability) Consider instance schema S_I as depicted in Fig. 11a and assume that the solution part of case c_2 shall be applied to S_I. Inserting activities `Follow-Up Examination` and `Puncture` after `Non Operative Therapy` and before `XOR-Join1` would be without any effect on I_1 in its current state. However, in the given example the parameterization of the change operations can be adjusted in such a way that c_2 becomes applicable. For example, activities `Follow-Up Examination` and `Puncture` can be inserted after activity `Operative Treatment`. We obtain $isAdjustable(c_2, I_1) = 1$.

By contrast, for case c_1 and instance I_3 from Fig. 10c no adjustment is possible. The `MRT` activity has already been completed for I_3 and cannot be skipped anymore, i.e., c_1 cannot be adjusted in such a way that I_3 becomes case compliant with c_1 (i.e., $isAdjustable(c_2, I_3) = 0$).

3.2.3. *Retrieving Similar Cases*

To determine those cases most relevant in a given exceptional situation all cases are ranked by decreasing application context similarity and the top n ranked cases are displayed to the user (cf. Fig. 12). To better support the user during case reuse the list of similar cases is further enriched with information regarding the control context of the process instance I to be modified. Cases which cannot be applied directly (i.e., cases not compliant with I or having no effect), but which can be adjusted are highlighted (together with the specific change operations that need to be adjusted). Finally, cases which cannot be adjusted are excluded from the list of similar cases.

Example 3.11. (Case Retrieval) Consider case-base $cb_S = \{c_1, c_2\}$, process instance I, and query qu as depicted in Fig. 12. For case c_1 we obtain $sim(c_1, qu) =$

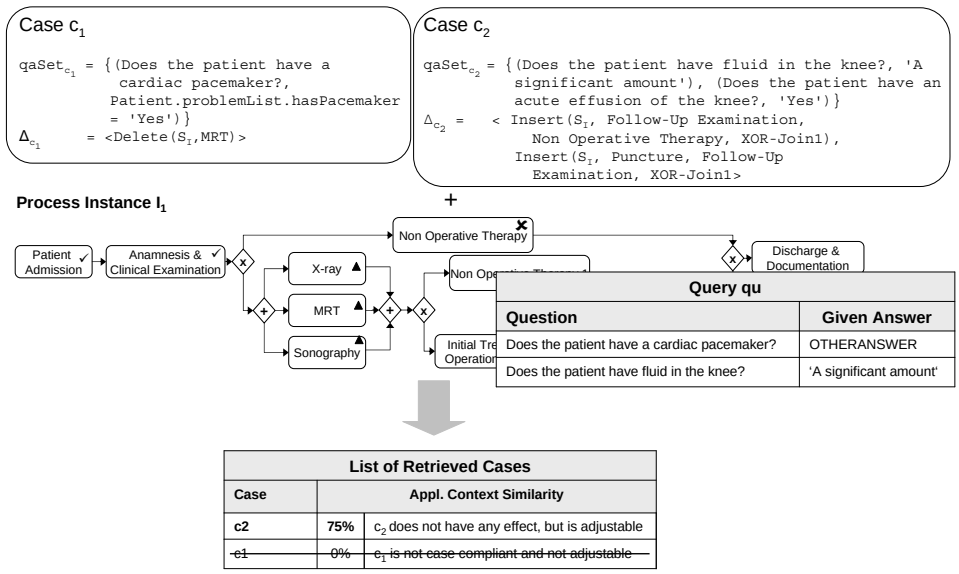

Fig. 12. Similar Cases Displayed to the User

0. Furthermore, instance I is not case compliant with c_1. As c_1 cannot be adjusted, it is eliminated from the list of similar cases. For case c_2 we obtain $sim(c_2, qu) = 0.75$. As c_2 does not have any effect on I and needs to be adjusted, the respective case is highlighted for the user.

3.2.4. *Case Reuse*

If instance I is case compliant with case c, the user may want to reuse c directly. The actions specified in the solution part of the case are then forwarded to and carried out by the PAIS. For this, in each change operation the placeholder for the instance-specific schema S_I needs to be replaced by process instance I (cf. Section 3.1.2). As the reuse of a case necessitates case compliance, correctness and consistency of the instance after change application can be ensured by the PAIS.[10] Finally, case reuse leads to an increase of the case reuse counter $freq_c$ by 1.

Example 3.12. (Case Reuse) Assume that case c_2 from Fig. 9 shall be applied to instance I. As I is case compliant with c_2, no case adjustment is needed and c_2 can be applied directly.

If the instance to be modified is not compliant with c or this case does not have any effect on I it can be adjusted prior to its reuse. For this, a copy of the case is created and modified. Afterwards the newly created case is added to the case–base.

4. Ensuring Quality of Cases

So far, we have described how instance changes can be represented, retrieved and reused in ProCycle. To effectively provide support for reusing cases as well as for

deriving process (model) optimizations based on them, the quality of the data maintained in the case-base is crucial. Poor PAIS performance, like long dialogues and inaccurate case data, can limit user acceptance significantly.[26] In ProCycle, cases are added by end users and not by experienced process engineers. In addition, case-bases evolve over time. New cases may be added to it when exceptional situations occur, which have never been dealt with before. To ensure accuracy and to improve overall performance, maintenance is becoming crucial when the case-base grows. Fig. 13 summarizes characteristic quality problems that arise in this context and shows how ProCycle deals with them.

Quality Issue	Support by ProCycle
Q1: Inexperienced users have difficulties in entering cases	Provide user assistance when entering cases
Q2: Case of poor quality are added to the case-base	Evaluate performance of cases
Q3: Free text answer expressions require user interaction	Support refactoring for question-answer pairs
Q4: Inter-case dependencies between cases exist	Discover inter-case dependencies and merge cases
Q5: Case-base contains similar cases which just differ regarding the parameterization of their change operations (i.e., the position)	Generalize cases

Fig. 13. Quality Issues and their Support in ProCycle

Section 4.1 describes how users are assisted when entering cases. To avoid reuse of low quality cases, case performance is evaluated (cf. Section 4.2). Section 4.3 introduces refactoring strategies supported by ProCycle to foster full automation of the case retrieval process. The discovery of inter-dependent cases is described in Section 4.4; Section 4.5 deals with their merging. Finally, Section 4.6 proposes generalization of cases with similar solution part.

4.1. *User Assistance When Entering Cases*

To assist users in defining question-answer pairs and to ensure data quality (*Quality Issue Q1*) user guidance is provided (e.g., by avoiding spelling errors or the entering of redundant data).

In the current version of ProCycle, question-answer pairs can be entered either by selecting the question from a list of previously defined questions (i.e., reusing questions from existing cases) or, if there is no suitable question in the system, by defining a new one and by giving the appropriate answer. Through displaying already existing question-answer pairs users are encouraged to reuse them to avoid redundancies. In addition, it is recommended that engineers review the case-base on a regular basis and refactor it if needed (e.g., by combining two question-answer pairs with the same semantics to a single pair).

To further improve reuse of question-answer pairs ontologies can be integrated. In many domains (e.g., healthcare) domain-specific vocabularies (e.g., ICD-9 for diagnoses[42]) exist and are widely used. We omit a discussion here since ontology

issues are currently outside the scope of ProCycle.

The entering of answer expressions is guided by the system. This ensures that users can only enter syntactically correct answers. In particular, user assistance for selecting context attributes is provided. In addition, depending on the chosen context attribute suitable operators are proposed to users.

4.2. *Evaluating Case Performance*

The accuracy of cases is crucial for overall PAIS performance, and consequently for the trust users have in the PAIS (*Quality Issue Q2*). When cases are added by end users, evaluation mechanisms are needed to ensure case-base quality. Similar to Cheetham and Price[43], we propose to augment the conversational case-based reasoning (CCBR) component with the ability to determine the confidence in the accuracy of individual solutions.[26] ProCycle uses the concept of *reputation* to indicate how successfully an instance change, represented by a case, was reused in the past; i.e., how much it contributed to the performance of the case-base, thus indicating the degree of confidence regarding the accuracy of the respective case.

To be able to evaluate case performance for each case c ProCycle maintains a history $hist_c$. Whenever c is applied to a process instance an entry e is added to the case history. Each history entry e refers to the process instance to which the case was applied. In addition, it contains a feedback score $fScore_e$, which either can be highly positive (2), positive (1), neutral (0), negative (-1), or highly negative (-2), reflecting the performance of the respective case. Optionally, a comment com_e can be added to provide feedback on the performance of c.

Definition 4.1. (Case History) *Let cb_S be a case-base and let further $c \in cb_S$ be a case. Then: Case history $hist_c = <e_1, .., e_n>$ with $n = freq_c$ comprises a list of history entries e_i for case c in the order they were created. A history entry e_i is a tuple $e_i = (I, fScore_{e_i}, com_{e_i})$ where*

- *I denotes the process instance to which c was applied*
- *$fScore_{e_i} \in \{-2, -1, 0, 1, 2\}$ corresponds to the feedback score*
- *com_{e_i} is a free text comment about the performance of case c*

The overall reputation $rScore_c$ of case c corresponds to the average feedback score of the history entries from $hist_c$. While a high reputation score of a case is an indicator for its semantical correctness, negative feedback probably results from problems after performing the respective instance change. Negative feedback leads to immediate notification of the process engineer who may deactivate the case to prevent its further reuse. The case itself remains in the system to allow for learning from failures and to ensure traceability. Definition 4.2 extends Definition 3.4 with case history $hist_c$ and reputation score $rScore_c$.

Definition 4.2. (Case With History) *A case c is a tuple (pd_c, Δ_c, $qaSet_c$, $freq_c$, $hist_c$,$rScore_c$) where*

- pd_c, Δ_c, $qaSet_c$ *and* $freq_c$ *are defined as in Def. 3.1*
- $hist_c = <e_1, .., e_n>$ *is the history associated with c (cf. Def. 4.1)*
- $rScore_c = \frac{\sum_{i=1}^{n} fScore_{e_i}}{|hist_c|}$ *denotes the reputation score. It is calculated as the average feedback score in* $hist_c$.

4.3. *Refactoring Question-Answer Pairs*

Question-answer pairs describe the reasons making a deviation at the process instance level necessary. A question constitutes free text, an answer can either be a structured expression or free text. In general, answer expressions should be used instead of free text to increase problem solving efficiency. While answer expressions can be automatically evaluated by the system (i.e., answer values are automatically calculated from existing data), free text answers have to be provided by the user (*Quality Issue Q3*). The ability to also give free text answers is crucial for the success of the conversational case-based reasoning component since it is not always possible to use structured answer expressions. In the following we describe three scenarios where free text question-answer pairs are entered into the system. In addition, maintenance policies for refactoring free text answers to formal answer expressions are sketched (cf. Fig. 14).

Problem	Maintenance Policy
Scenario 1: End user is not knowledgeable enough to specify formal answer expressions	Notify process engineer when answer frequency exceeds a predefined threshold and support him in refactoring the free text answer to a formal answer expression.
Scenario 2: End user is unaware of the application context	Notify process engineer when answer frequency exceeds a predefined threshold and support him in refactoring the free text answer to a formal answer expression.
Scenario 3: Needed context attributes are not available within the system	Notify process engineer when answer frequency exceeds a predefined threshold and support him in refactoring the free text answer to a formal answer expression and to extend the context model.

Fig. 14. Maintenance Policies for Question-Answer Pairs

Scenario 1 (Inexperienced User): The end user applies ProCycle to handle an exception, but is not knowledgeable enough to specify formal answer expressions. As the exception has to be resolved quickly the user enters free text question-answer pairs to capture the reasons for the deviation, and he applies the case immediately. To increase problem solving efficiency the question-answer pair will be refactored to a formal answer expression later on, if feasible. Thus, a notification is sent to the process engineer to accomplish this refactoring when a particular question-answer pair is frequently applied by users in their queries.

Scenario 2 (Unawareness of Application Context): In this scenario the end user is unaware of the application context and cannot find suitable context attributes for specifying answer expressions even though they are available in the system. Therefore, the user enters free text to capture the reasons for the deviation. In this scenario, the process engineer is not informed immediately, but only when the re-

spective question-answer pair has been answered frequently enough, thus exceeding a predefined threshold value. The process engineer can then refactor the free text to an equivalent answer expression, which will be used during case retrieval instead of the free text.

Scenario 3 (Missing Context Attributes): In certain situations no suitable context attributes are available within the system to describe the concrete instance deviation. In this scenario, the user must specify the question-answer pair using free text. Like Scenario 2 the process engineer is informed when the question-answer pair has been answered frequently enough. He can then decide whether to extend the application context (cf. Section 3.1.2) by adding the required context attribute to the context model.

4.4. *Discovering Inter-dependent Cases*

ProCycle does not only provide support for refactoring question-answer pairs, but also for maintaining cases. In particular, it supports the discovery of correlations between cases based on respective case histories (*Quality Issue Q4*).

Example 4.1. (Inter-Dependent Changes) Assume that for a particular treatment process the MRT has to be skipped for a patient with cardiac pacemaker and another imaging technique is applied instead, i.e., one activity is deleted and another one inserted.

Since ad-hoc changes are applied in exceptional situations, we cannot expect that such semantically related adaptations are always conducted at the same time; i.e., they might be not added as single case to the PAIS. This will particularly happen if end users are inexperienced and do not consider all consequences of a change. Further, if change dependencies are not known when adding a case, semantically related adaptations will be stored in different cases as well.

To allow for better user assistance, ProCycle collects information about inter-case dependencies. For each case c the set of dependent cases $interDep_c$ is provided. Whenever a user selects a case c to be applied to a process instance I and I has not yet been changed (i.e., $|\Delta_I| = 0$), ProCycle presents her all cases from $interDep_c$. The user can then decide to additionally apply any of these cases, if they are relevant for her current situation. If instance I has already been changed (i.e., $|\Delta_I| > 0$), ProCycle does not only present the cases from $interDep_c$ to the user, but also those cases c_i not contained in $interDep_c$, but previously been applied to I (i.e, $c_i \in \Delta_I$). The user can then tag any of the cases $c_i \in \Delta_I$ as inter-dependent case. As a consequence c_i is added to $interDep_c$.

4.5. *Merging Inter-Dependent Cases*

Two inter-dependent cases c_1 and c_2 can be merged into a single case c if they always co-occur. For a given set of process instances the co-occurrence rate $CoRate(c_2|c_1)$ denotes the relative frequency of the application of case c_2 on condition that case

c_1 has been applied as well. Consequently, two cases c_1 and c_2 are merged if $CoRate(c_1|c_2) = 1$ and $CoRate(c_2|c_1) = 1$ hold. In this situation a new case is added to the case-base and the original cases c_1 and c_2 are deactivated[d], i.e., a refactoring of the case-base takes place. Problem descriptions as well as question-answer pairs related to the two cases are manually merged by the process engineer, who has the domain-specific knowledge needed for this; the corresponding solution parts, in turn, can be automatically merged. In addition, different optimizations for purging the resulting operation sets in case of redundancies can be applied.[44] Finally, as both cases c_1 and c_2 always co-occur we can set $freq_c$ and $hist_c$ accordingly.

Definition 4.3. (Conditional Co-Occurrence Rate) *Let c_1, $c_2 \in cb_S$ be two cases and let $Inst_c$ denote the set of all process instances $I \in Inst_S$ to which c was applied. Then: The conditional co-occurence rate $CoRate(c_2|c_1)$ denotes the relative frequency of c_2 on condition that c_1 has been applied as well. Formally:*

$$CoRate(c_2|c_1) = \frac{|Inst_{c_2} \cap Inst_{c_1}|}{|Inst_{c_1}|}$$

4.6. *Generalizing Cases*

As described in Section 3.2 cases are not always directly applicable, but might have to be adjusted. Over time this can lead to similar cases with slightly different solution parts, i.e., same list of change operations, but different parameterization (*Quality Issue Q5*). To speed up case retrieval and to ensure maintainability of the case–base such cases need to be generalized. In ProCycle, the process engineer is notified when an existing case c is adjusted and memorized as new case c'. He then can decide to create a generalized version c_g. When generalizing two cases c and c' the solution part of the generalized case Δ_{c_g} needs to cover both the changes defined by c and c'. In addition, the question-answer pairs of both cases need to be merged. The reuse frequency of the generalized case is calculated as the sum of $freq_c$ and $freq_{c'}$. Finally, the history $hist_{c_g}$ is determined as the union of $hist_c$ and $hist_{c'}$. To ensure traceability cases c and c' are not deleted, but can be deactivated by the process engineer.

Example 4.2. (Generalizing Cases) Consider schema S_I and the solution parts of cases c_2 and c_3 as depicted in Fig. 15. Both cases insert activities `Follow-Up Examination` and `Puncture`, but at different positions. The process engineer creates a generalized case c_g, which covers the changes defined by c_2 as well as by c_3. For example, c_g allows to insert `Follow-Up Examination` and `Puncture` after `Anamnesis & Clinical Examination` and before `Discharge & Documentation`. Case c_g covers the changes of c_2 and c_3, but also allows for some additional behaviour by

[d]For traceability reasons respective cases are not deleted, but only deactivated.

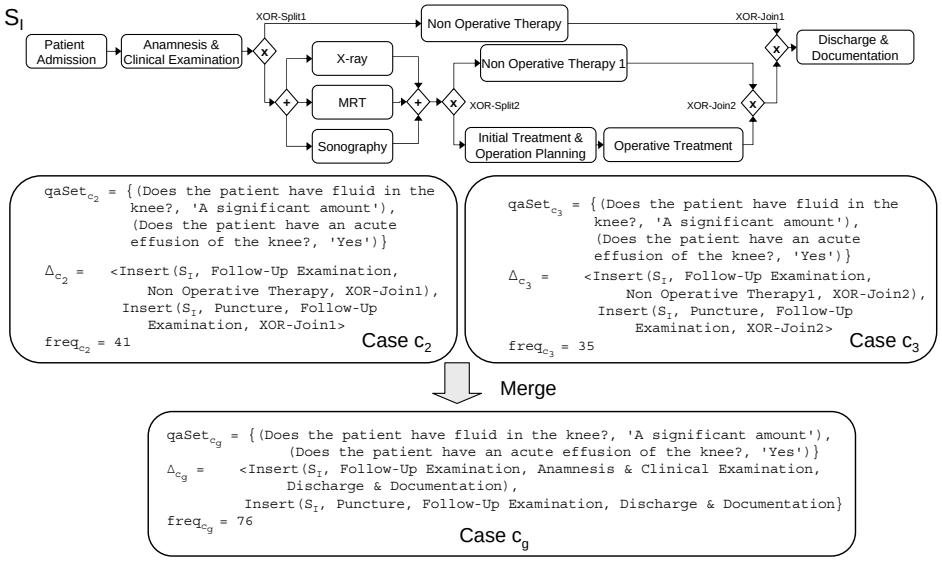

Fig. 15. Generalizing Similar Cases

inserting the activities in parallel to a larger schema region when compared to c_2 and c_3. The question-answer pairs of c_g are the same than for c_2 and c_3. As reuse frequency we obtain $freq_{c_g} = 76$.

5. Process Learning and Process Evolution

In ProCycle, the knowledge about previous instance changes is not only used for assisting users in exceptional situations, but also for process improvement. When same or similar process instance changes occur frequently, this indicates a gap between the used process schema and the real-world business process. This misalignment often stems from errors in the design of the process schema or can be the result of changing requirements. To discover such discrepancies, ProCycle continuously monitors deviations between process schema and actual instance enactment. When same or similar deviations occur frequently the process engineer is notified about the potential need to evolve the process schema. He can then perform a process type change by pulling up instance changes to the type level and migrating already running instances to the new schema version.

5.1. *Discovery of Process Type Changes*

To derive suggestions for process type changes from a collection of process instances and their associated changes, we need to monitor case reuse. Let S be a process schema and $Inst_S = \{I_1, \ldots, I_n\}$, $n \in \mathbb{N}$ be the set of process instances created

from S. Let further cb_S be the set of all cases c applied to at least one instance $I \in Inst_S$.

A naive solution would be to suggest a process type change exactly when for a particular case $c \in cb_S$ its reuse frequency $freq_c$ exceeds the predefined threshold value thr:

$$\frac{freq_c}{|Inst_S|} \geq thr$$

As detailed in the following solely monitoring the reuse frequency for each case $c \in cb_S$ is not sufficient. For example, cb_S might contain distinct cases with same solution parts, but different question-answer pairs, i.e., the same instance change was applied in different context. As another example consider cases, which do not have the same, but similar solution parts, i.e., overlapping change operations or same change operations with different parameterization. For example, in Fig. 16 cases c_2 and c_3 comprise the same list of change operations, which just differ in their parameterization. To better support process engineers in deriving process type changes ProCyle does not only monitor case reuse, but also the reuse frequency of single change operations. Thereby, change operations, which just differ in their parameterization are combined to abstract change operations *abstOp*. An abstract change operation only consists of an operation type and the activity effected by the change, while the insertion or move positions are neglected.

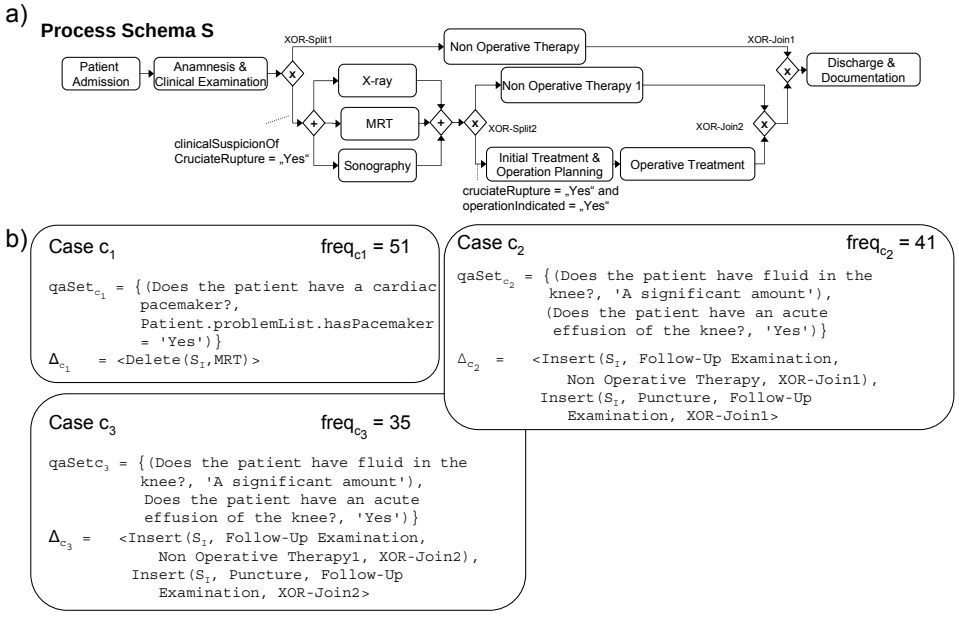

Fig. 16. Discovery of Process Type Changes

Example 5.1. (Abstract Change Operations) Consider schema S and related case-base $cb_S = \{c_1, c_2, c_3\}$ as depicted in Fig. 16. As abstract change operations applied on instances from S we obtain $abstOP_1$ = `Delete(`S_I`,MRT)`, $abstOP_2$ = `Insert(`S_I`,Follow-Up Examination)`, and $abstOp_3$ = `Insert(`S_I`,Puncture)`.

ProCycle suggests a process type change when the reuse frequency of an abstract change operation exceeds the predefined threshold value thr:

$$\frac{freq_{abstOp}}{|Inst_S|} \geq thr$$

Example 5.2. (Reuse Frequency of Change Operations) Consider process schema S and $cb_S = \{c_1, c_2, c_3\}$ as depicted in Fig. 16. We obtain $freq_{abstOp_1} = 0.051$, $freq_{abstOp_2} = 0.076$, and $freq_{abstOp_3} = 0.076$. With threshold $thr = 0.075$ this suggests to pull $abstOp_2$ and $abstOp_3$ up to the type level.

To inform the process engineer about the potential need of a process type change a notification with information about the abstract change operation *abstOp* exceeding the threshold value is sent to him. In addition, for each abstract change operation *abstOp* all cases $c \in cb_S$ with a matching change operation in their solution part are listed. The process engineer can then decide whether and how to perform the process type change (i.e., to create a new schema version S' by applying a list of change operations Δ_S to S). In most situations the change operations as defined in the solution part of respective cases cannot be directly applied to the process schema as the change operations have been only performed in a particular context. Examining the question-answer pairs additionally enables the process engineer to gain valuable insights into the context of a previously applied change operation as each question-answer pair represents a condition under which the corresponding case was applied. For example, activities `Follow-up Examination` and `Puncture` from Fig. 16 should not be performed for all future process instances, but only if certain conditions hold (e.g., there is fluid in the knee and an acute effusion of the knee exists). Therefore, the solution part of the case is not directly pulled up to the process type level, but the process engineer inserts the corresponding activity only conditionally (cf. Fig. 17). In addition, pulling changes up to the type level often necessitates the generalization of the change operations through parameter adjustments (cf. Sect. 4.6). In particular, this can happen if for an abstract change operation *abstOp* several matching cases exist.

Example 5.3. (Deriving a Process Type Change) Consider process schema S and cases $\{c_2, c_3\}$ as depicted in Fig. 17. Based on this information the process engineer decides to insert activities `Follow-up Examination` and `Puncture` at the type level resulting in a new schema version S'. Considering the question-answer pairs of c_2 and c_3 he decides to insert the respective activities only on condition that the patient has a significant amount of fluid in the knee and an acute effusion of the knee. In addition, the process engineer generalizes the insert posi-

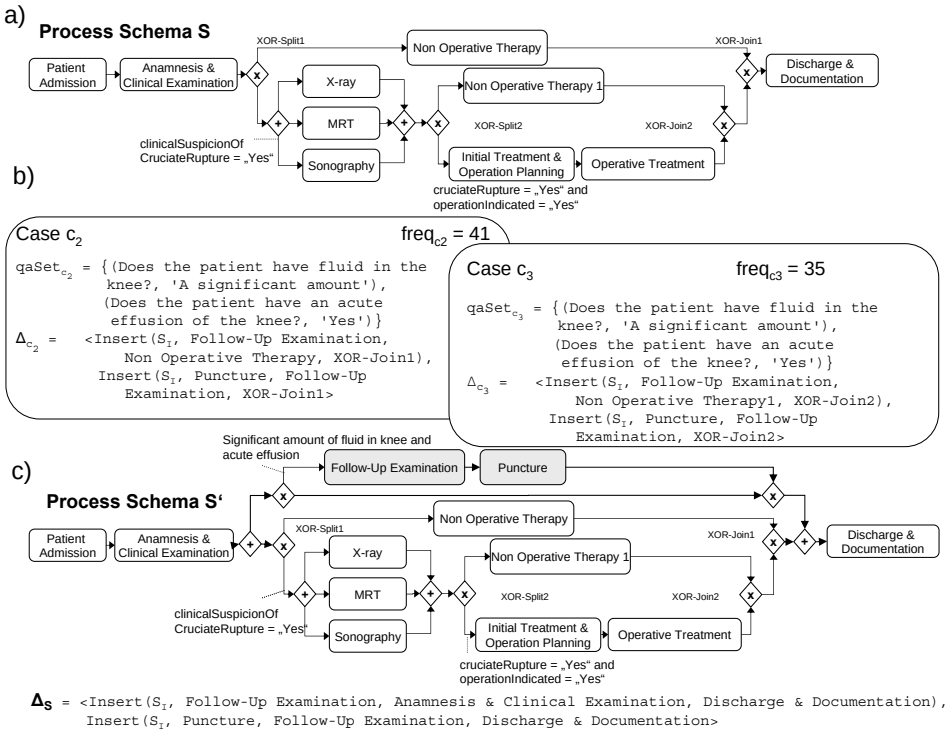

Fig. 17. Deriving Process Type Changes

tion of activities `Follow-up Examination` and `Puncture`. He decides to insert the two activities after `Anamnesis & Clinical Examination` and before `Discharge & Documentation`. The list of change operations the process engineer applies to transform S into S' is denotes as Δ_S.

5.2. *Process Schema Evolution and Process Instance Migration*

Assume that the process engineer decides to perform a process type change, i.e., to transform process schema S into a new version S' by applying a list of change operations Δ_S. For example, schema S in Fig. 17 has been modified by inserting activities `Follow-up Examination` and `Puncture` after `Anamnesis & Clinical Examination` and before `Discharge & Documentation` on condition that a significant amount of fluid is in the knee and an acute effusion of the knee exists. When a process type change takes place ongoing process instances can either be completed based on original schema version S or be migrated to the new schema version S'.[30] The latter will be only possible if the respective instances have not progressed too far. Migration to the new schema version then is accompanied by state adaptations of the *compliant instances*.

When migrating process instances to a new type schema version it has to be considered whether respective instances are still running according to their original schema (*unbiased* instances) or have already been individually modified due to exceptional situations (*biased* instances). The deviation of a process instance I from its original schema S is thereby denoted as bias Δ_I. Fig. 18 depicts four process instances derived from process schema S as depicted in Fig. 17a. Instance I_1 is still running according to the original schema and is therefore denoted as *unbiased* (i.e., $|\Delta_{I_1}| = 0$). By contrast, instances I_2, I_3 and I_4 were individually modified before, i.e., they constitute examples of *biased* instances. For biased instances it has to be further considered whether the instance-specific change Δ_I overlaps with the process type change Δ_S (e.g., I_3 and I_4) or whether these changes are disjoint (e.g., I_2 inf Fig. 18). Obviously, instances for which Δ_I and Δ_S overlap require a different migration policy than instances for which Δ_I and Δ_S are disjoint (see Ref. 45 for details). Fig. 19 summarizes the different migration strategies supported by ProCycle depending on the degree of overlap between Δ_S and Δ_I.

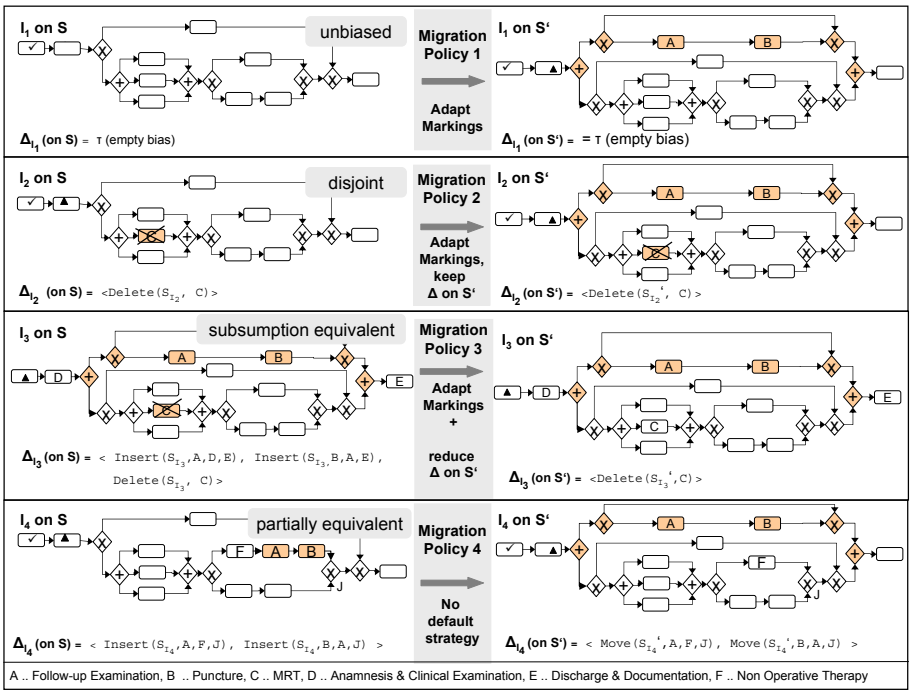

Fig. 18. Instance Migrations after the Process Type Change Depicted in Fig. 17 (Instance-specific modification are grey shaded)

For unbiased process instances, state–related *compliance* with the new schema version has to be checked[24] (cf. Def. 2.2). Compliant instances can be migrated

to the new schema version S' (i.e., S [Δ_S> S') by re-linking them to S' and by adapting their state markings accordingly (e.g., instance I_1 from Fig. 18).

Process instances for which Δ_I and Δ_S are *disjoint* can migrate to the new schema version S' on condition that they are compliant regarding their state as well as their structure.[46] Regarding Fig. 18, for example, Δ_{I_2} and Δ_S are disjoint; i.e., they affect different activities and regions of the process schema. In addition, I_2 is compliant with S'. Therefore, I_2 can be migrated to the new schema version S' by adapting its markings and by keeping its bias Δ_{I_2} unchanged on S', i.e., (S + Δ_{I_2}) [Δ_S> (S' + Δ_{I_2}) (cf. Fig. 19).

Instances which have 'anticipated' the process change completely (i.e., Δ_I and Δ_S are *equivalent*) can automatically migrate to new schema S' if they are compliant with it. In this scenario, bias Δ_I becomes empty after migration as the effects of the instance change are entirely covered by S'.

An instance where Δ_I *subsumes* Δ_S can automatically migrate to S' if it is compliant with S'.[46] In Fig. 18, for example, Δ_{I_3} captures all effects of Δ_S on S, but has additional ones. In our example, both Δ_I and Δ_S conditionally insert activities `A` and `B`. However, Δ_I additionally deletes activity `C`. As I_3 is compliant with S' it can be migrated to S' by adapting instance markings. In addition, bias Δ_I is re-calculated, i.e., only those changes from Δ_I not covered by Δ_S are kept.

For process instances, which have partially anticipated the process type change (resulting in an overlap of process type and instance changes), no default migration policy can be provided and user interaction is required. The migration policy to be applied to such instances depends on the particular *degree of overlap* between process and instance change. To determine an appropriate migration policy it is not sufficient to only check the parameterization of change operations. In addition, the order in which the operations are applied may have an effect on change comparison (e.g., when inserting activities in a sequence). We therefore compare the effects of change operations. This is accomplished by a hybrid approach, which compares change operations on the one side, but also the resulting process schema on the other side (for details see Refs. 45 and 47). Fig. 18 shows an example of *partially equivalent* changes. Both Δ_S and Δ_{I_4} insert the same activities, but at different insert positions. To migrate instance I_4 to S', in addition to adapting markings, bias Δ_{I_4} needs to be re-calculated on S'. This is accomplished by replacing the insert operations by move operations.

5.3. *Case–Base Evolution*

ProCycle does not only support the evolution of business processes and the migration of running instances to a new schema version, but additionally provides system support for evolving the knowledge about process changes. This is crucial for any approach on learning processes. Given a case-base cb_S and a process type change Δ_S with S[Δ_S> S', parts of cb_S might become obsolete on S', because a subset of the knowledge encoded in the cases of cb_S is captured by S' afterwards. This will

Let S be a process schema and let Δ_S be a type change transforming S into S' (i.e., S' = S + Δ_S). Let further I be an instance created from S with schema S_I and bias Δ_I (i.e., S_I = S + Δ_I).		
Degree of overlap between Δ_S and Δ_I	**Description**	**Migration Policy**
Category 1 Δ_S and Δ_I are disjoint	Δ_S and Δ_I have different effects on S, i.e., Δ_S and Δ_I concern different schema regions and different activities. Δ_S and Δ_I are commutative.	Migrate I to S' Δ_I':= Δ_I S_I':= S' + Δ_I'
Category 2 Δ_S and Δ_I are equivalent	Δ_S and Δ_I have the same effects on S, i.e., changes Δ_S and Δ_I effect the same activities, the same schema regions, and apply the same change operations in same order.	Migrate I to S' Δ_I':= τ (i.e., bias of I on S' becomes empty) S_I':= S'
Category 3 Δ_S subsumes Δ_I	Δ_S subsumes the effects of Δ_I; i.e., Δ_I captures the effects of Δ_S, but has additional ones as well. Example: • Δ_S is a sub-list of Δ_I	Migrate I to S' Δ_I':=Δ_I – Δ_S S_I':= S' + Δ_I'
Category 4 Δ_I subsumes Δ_S	Δ_I subsumes the effects of Δ_S; i.e., Δ_S captures the effects of Δ_I, but has additional ones as well. Example: • Δ_I is a sub-list of Δ_S	Migrate I to S' Δ_I':= τ (i.e., bias of I on S' becomes empty) S_I':= S'
Category 5 Δ_S and Δ_I are partially equivalent	Δ_S and Δ_I are overlapping, i.e., they have (partially) overlapping effects. Examples: • Δ_S and Δ_I insert different activities at the same position of S • Δ_S and Δ_I insert the same activities at different positions of S • Δ_I moves an activity which is deleted by Δ_S (i.e., different operations, same activity) • Δ_I inserts an activity, which is only conditionally inserted by Δ_S	No "standard" migration policies can be provided.

Fig. 19. Degrees of Overlap Between Changes and Related Migration Policies

particularly apply if the type change is triggered by the PAIS itself when case reuse exceeds a given threshold (cf. Section 5.1). The challenge is to decide which cases of cb_S shall be transferred to $cb_{S'}$, which cases have to be adapted before being migrated to $cb_{S'}$, and which ones are already covered by the new schema version S' and can therefore be dropped. Note that this has to be done independently from instance migrations as decribed in the previous section. Furthermore, we have to consider that same or similar changes might have been applied in different application context.

To determine suitable migration policies for a case $c \in cb_S$ the relationship between type change Δ_S and solution part Δ_c (representing an instance change) has to be analyzed. Depending on the degree of overlap between Δ_S and Δ_c three major migration policies exist (cf. Fig. 20).

Cases for which Δ_S and Δ_c are *disjoint* can be automatically added to $cb_{S'}$ without need for adaptation since Δ_c does not contain any change already reflected by S'. Regarding Fig. 21, for example, Δ_{c_1} and Δ_S are disjoint, i.e., they affect

Let S be a process schema and Δ_S be a type change transforming S into S' (i.e., S' = S + Δ_S). Let $cb_{S'}$ be the set of cases applied to instances on S and $cb_{S'}$ be the case-base associated with the new schema version S'. Let further be c a case with solution part Δ_c.		
Degree of overlap between Δ_S and Δ_c	**Description**	**Migration Policy**
Δ_S and Δ_c are disjoint	As Δ_c is not yet covered by Δ_S case c is still relevant	Keep case c unchanged and migrate it to $cb_{S'}$
Δ_c and Δ_S are equivalent or Δ_c subsumes Δ_S	All changes in Δ_c are covered by Δ_S. However, Δ_S and Δ_c might have been applied in different application context, i.e., be performed due to different reasons. Consequently, it has to be analyzed whether the set of question-answer pairs $qaSet_c$ is reflected by Δ_S as well.	Drop c if question-answer pairs are reflected by Δ_S
		Migrate c to $cb_{S'}$ if question-answer pairs are not reflected by Δ_S
		Migrate c to $cb_{S'}$ and manually adapt set $qaSet_c$ if question-answer pairs are only partially reflected by Δ_S
Δ_S subsumes Δ_c or Δ_S and Δ_c are partially equivalent	As Δ_c has additional effects than Δ_S parts of change Δ_c are still relevant.	Migrate c to $cb_{S'}$ Adapt solution part Δ_c as well as $qaSet_c$ if question-answer pairs are already partially reflected

Fig. 20. Evolution Policies for Cases

different regions of the process schema. Therefore c_1 is automatically added to $cb_{S'}$.

If Δ_c and Δ_S are *equivalent* or Δ_c *subsumes* Δ_S, migrating c to S' will not provide any additional information regarding the solution part. However, the question–answer pairs $qaSet_c$ of c will differ if Δ_S and Δ_c have been performed under different conditions (i.e., different application context). Consequently, question answer pairs $qaSet_c$ have to be analyzed in order to decide on the migration of c. Case c will not be migrated to S' if all question-answer pairs are reflected by the conditions inserted by Δ_S at type level. Case c will be added to $cb_{S'}$ without adaptation if none of the question-answer pairs is reflected by Δ_S. If the question-answer pairs are partially reflected by Δ_S, $qaSet_c$ will have to be adapted. The solution part of case c_2 in Fig. 21 is subsumption equivalent with Δ_S. All changes of Δ_{c_2} are covered by Δ_S. In addition, the question-answer pairs of c_2 are reflected by the process type change as well, as activities `Follow-up Examination` and `Puncture` are inserted under the same conditions. Therefore, case c_2 is not migrated to $cb_{S'}$.

If Δ_c *subsumes* Δ_S, or Δ_S and Δ_c are *partially equivalent*, case adaptation will become necessary before adding the respective case to $cb_{S'}$. In this situation Δ_c has additional effects when compared to Δ_S, and consequently parts of Δ_c are still relevant. Therefore, case c must be added to $cb_{S'}$ after adapting Δ_c. Question-answer pairs might also have to be adapted if they are already partially reflected by Δ_S. Case c_3 and Δ_S as depicted in Fig. 21 are partially equivalent. Consequently, c_3 is migrated after adapting its solution part Δ_{c_3}. The question-answer pairs can remain unchanged.

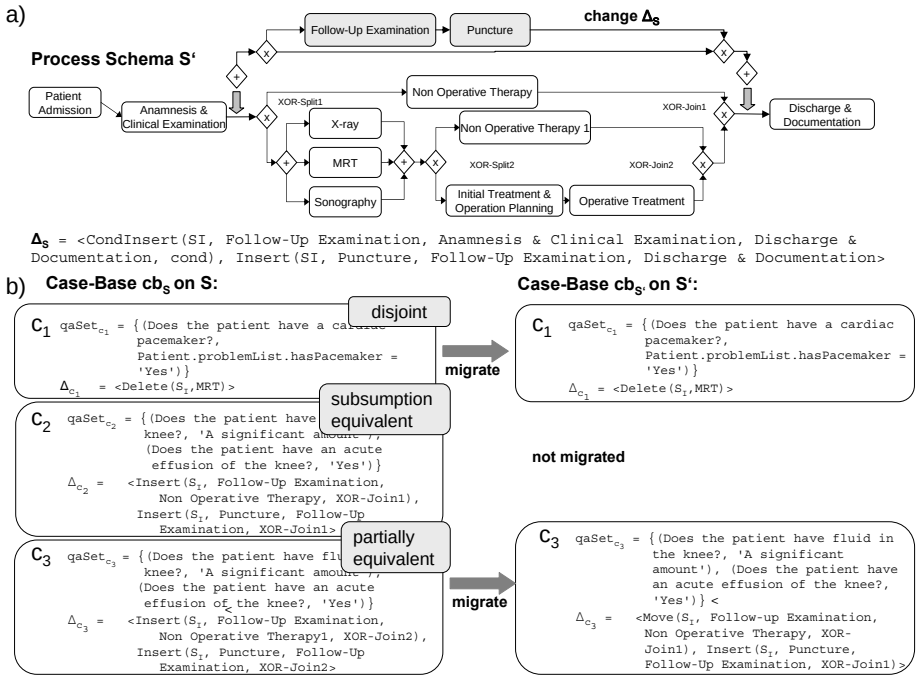

Fig. 21. Case–Base Evolution

6. Architecture and Implementation

To demonstrate the feasibility of our ProCycle approach an integrated prototype was implemented. It combines ADEPT2[10] and the conversational-case based reasoning (CCBR) component CCBR-Tool[18,48] (cf. Fig. 24). This section gives an overview of how our proof-of-concept prototype supports the process life cycle (cf. Section 2.3). Thereby, we focus on the integration of the ADEPT2 and CCBR-Tool as well as the extensions made in this context. Detailed descriptions of ADEPT2 and its architecture can be found in Ref. 49.

The prototype consists of two components ADEPT2 and CCBR-Tool. The ADEPT2 technology[16] supports the modeling, execution, and monitoring of business processes. In addition, ADEPT2 enables process changes at both the process type and the process instance level (for details see Ref. 49). Particularly, ADEPT2 provides functionality for migrating running process instances to the new schema version. In this context, CCBR-Tool allows annotating instance changes with context information. The resulting cases are memorized in a case-base and can be later reused when applying CCBR-Tool.

In the following we describe the interplay between ADEPT2 and CCBR-Tool. ADEPT2 is used for modeling, executing and monitoring processes (cf. Fig. 23). During run-time users are working with worklists generated by the ADEPT2 system.

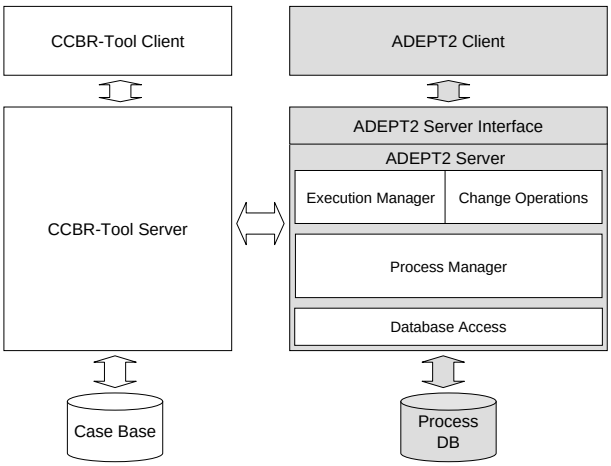

Fig. 22. Architecture of the Integrated Prototype

If for a particular process instance an instance change becomes necessary, the user will initiate the CCBR-Tool component for retrieving similar cases.

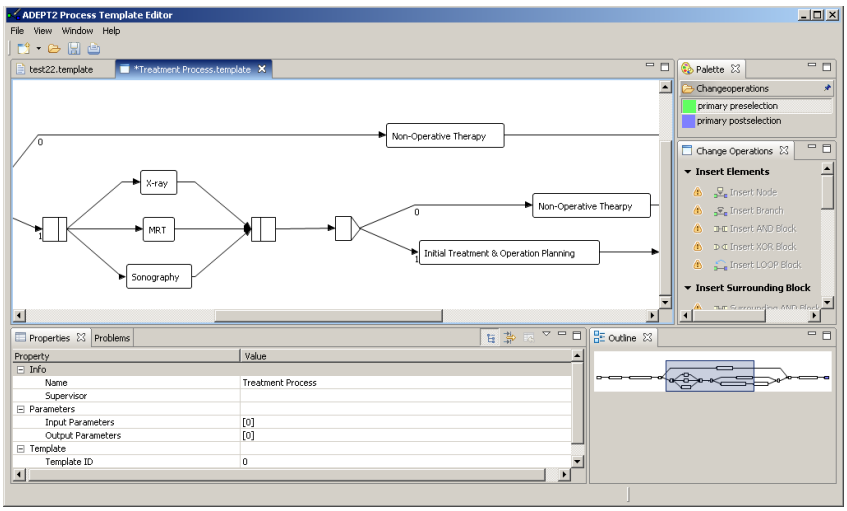

Fig. 23. Modeling the Process from Fig. 1a with the ADEPT2 Process Composer

Fig. 24 illustrates the interplay of ADEPT2 and CCBR-Tool during case retrieval. The ADEPT2 Client sends a request containing the instance to be modified to the ADEPT2 Server (1). This server then determines the schema from which this instance was derived and the case-base associated with this schema (2). It

then forwards the request to the CCBR-Tool Server and the CCBR-Tool Client (3). CCBR-Tool is then started with the respective case–base and the graphical user interface is opened (cf. Fig. 25), which assists users in finding similar cases (4). To reuse a particular case the user needs to select and execute it. Following this, the change operations from the solution part of the case are forwarded to the ADEPT2 system and applied to the respective instance (5). As a next step the reuse counter of the case is increased by one (6). To ensure case quality the user has the option to evaluate the case with a feedback form (7+8). The respective history entry is then added to the case (9).

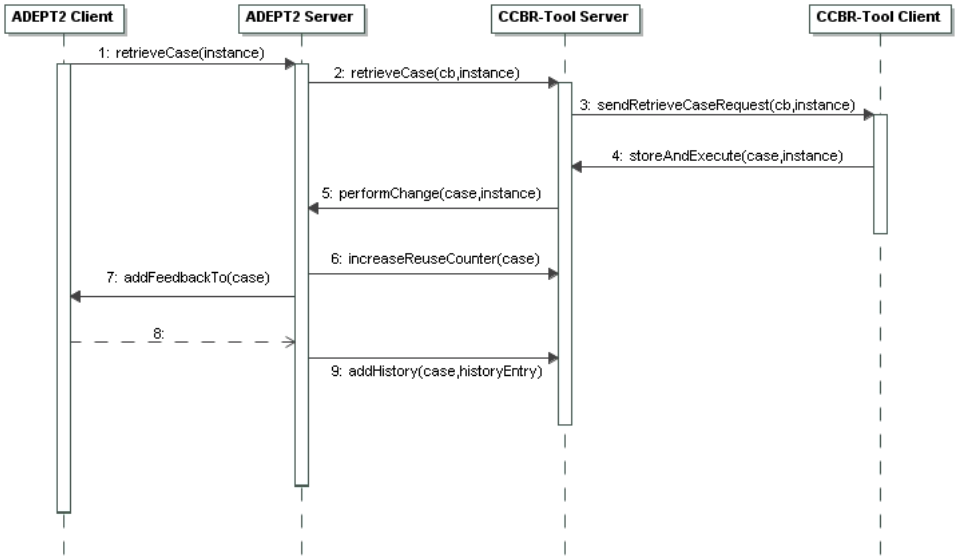

Fig. 24. Interplay of ADEPT2 and CCBR-Tool during Case Retrieval

Fig. 25 shows the graphical user interface for case retrieval. The upper part of the screen shows the list of questions, which assists the user during case retrieval. Questions can be answered in arbitrary order by selecting any of the available answer options. On the lower part of the screen the most similar cases are displayed ordered by decreasing similarity. When answering a question the list containing the most similar cases is re-calculated. Cases, which need to be adjusted before reuse, are highlighted in red, i.e., these cases cannot be directly applied to the respective instance, but need adjustment. By clicking on the `Show` button users can access case details, i.e., knowledge about problem description, question-answer pairs, change operations, and full case history. If a case cannot be directly reused, the change operations to be adjusted will be highlighted.

When adjusting a case, a copy is made, which can then be modified by the

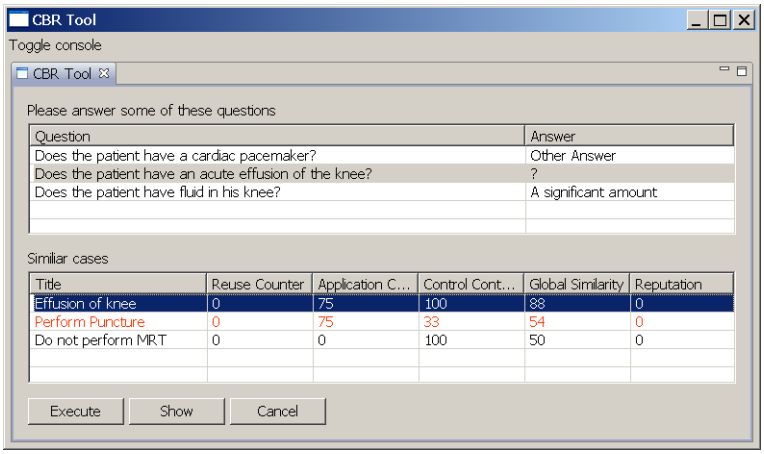

Fig. 25. Case Retrieval

user. This functionality can be used to quickly add cases to the case-base, if they are similar to already existing ones (e.g., same problem description, but different parameterization of change operations). To support later generalization, cases which are the result of an adjustment contain a reference to the case they were derived from.

If no similar case can be found the user will close the retrieval dialogue and add a new case to the case–base. He first performs the desired change using the process composer of ADEPT2. The changes are forwarded to CCBR-Tool to be annotated with context information. For this, a textual problem description as well as the reasons for the deviation in terms of question-answer pairs have to be entered. Additional administrative attributes like creation date are automatically added by the system. Finally, the case is stored in the case–base before ADEPT2 executes the change operations.

A detailed description of the implementation can be found in Ref. 50.

7. Related Work

Adaptive PAISs are related to our approach. Most existing approaches either support process instance or process type changes (e.g., Refs. 11, 12, 13 and 29). Except ADEPT2[10], WASA2[15] is the only adaptive PAIS, which has implemented both kinds of changes in an integrated manner. While ADEPT2 provides high-level change patterns, in WASA2 process changes have to be performed using change primitives (e.g., add node, delete edge).[30] ADEPT2 is the only adaptive PAIS, which allows migrating biased instances to a new process schema version.

As change definition requires extensive user experience adaptive PAISs like ADEPT2 or WASA2 support the user in the definition of instance changes by

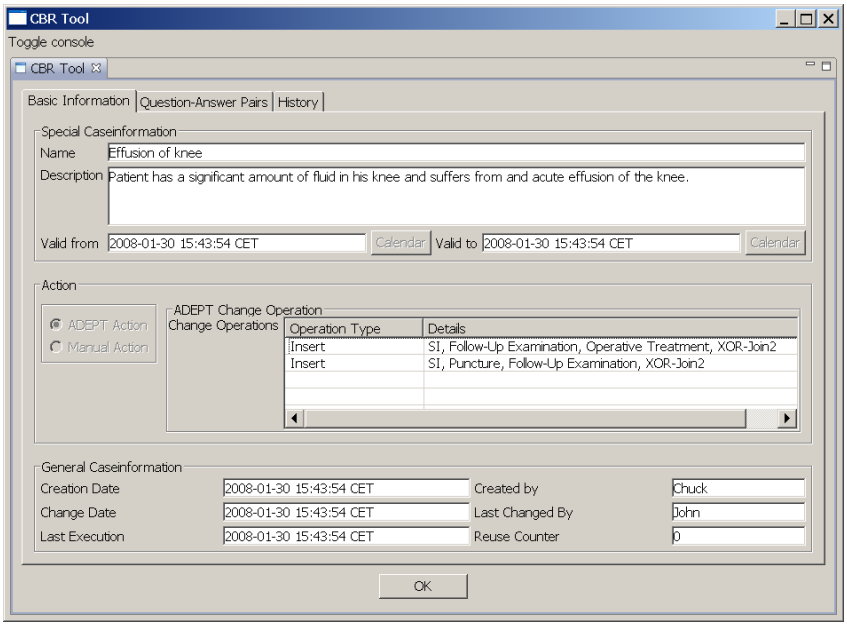

Fig. 26. Add Case

ensuring that the resulting instance schemes are syntactically correct and the instance is compliant with the new schema version.[10] In addition, CBRFlow[18] assists users in exceptional situations by allowing for the reuse of previously defined instance changes. However, no framework for integrated and seamless process lifecycle support has been provided so far.

CAKE2[13,51] is an adaptive PAIS, which is related to our approach. It was recently developed in the context of complex engineering processes in the domain of digital chip design. Focus is on instance-specific changes and late modeling. In contrast to ProCycle, CAKE2 does not support high-level change patterns. Instead structural instance changes are based on a set of change primitives.[30] Process type changes have not been addressed by CAKE2, but each process instance has its own process schema. Like ProCycle, CAKE2 provides authoring support for changes using case-based reasoning techniques. Thereby, a case consists of a problem description (i.e., the process instance before the change including its context) and a solution (i.e., the process instance after the change). Like in ProCycle the measure used for retrieving similar cases considers context information and the status of the process instance to be modified.[51] The proposed similarity measure has turned out to be appropriate for the chip design domain. However, it does not consider that activities are often inserted at different positions in the process. In such situation, the similarity measure used by CAKE2 will be too restrictive. In addition, case-base evolution and quality issues are not addressed by CAKE2.

The Pockets of Flexibility approach (PoF) is related to ProCycle as well.[52] PoF uses a combination of imperative and declarative process modeling. The process itself is modeled in an imperative way, however, so called placeholder activities can be specified in a declarative way based on constraints. During run-time the process model is concretized by modeling the sub-processes for placeholder activities. In Ref. 53 a querying component for supporting the concretization of placeholder activities is suggested. However, the feasibility of the proposed similarity measure has still to be demonstrated through a proof-of-concept implementation. A similar approach is provided by Worklets[54], which addresses flexibility through the late binding of placeholder activities. Using Worklets the need for structural process changes can be reduced, but still is an issue. For these situations the Worklets approach only provides very limited support.[30] Basic support for change reuse is provided by supporting the incremental evolution of selection rules for sub-process fragments.

Also related to our approach are declarative approaches for defining business processes (e.g., DECLARE[55] or MOBILE[56]). Instead of requiring process modelers to specify how the process shall be executed, declarative models only specify what shall be done by the process during run-time resulting in more flexibility. By using declarative approaches the need for process changes can be reduced. However, run-time modifications still can be an issue, e.g., a particular constraint might have to be violated for a particular process instance due to an unforeseen situation. Furthermore, constraints themselves may evolve over time, which raises the challenge of propagating changes to ongoing instances. In Ref. 55 these issues are addressed from a pure control-flow perspective, while data issues are not handled. In addition, it still has to be evaluated how well maintenance issues of constraint-based process models, particularly in case of large constraint sets, can be addressed. Furthermore, performance issues might arise in the presence of larger constraint sets.

Process Mining techniques like Delta Analysis[57,58,59], Conformance Testing[59] and Change Mining[38] can be used alternatively to our approach to improve the quality of business processes. Using these techniques, discrepancies between the modeled business process and the observed execution behavior can be discovered. For this, Delta Analysis compares the present business process model with the model derived from the execution log of the respective business process.[59] By contrast, Conformance Testing directly compares the modeled business process with its execution log. Though both techniques can be used to reveal malfunctions or bottlenecks, they do not provide any semantics about the reasons for the observed discrepancies. However, contextual knowledge about the reasons for these discrepancies is needed by the process engineer in order to create an improved process model.

Like ProCycle, change mining[38] addresses the question what can be learnt from the information about changes and suggests the use of adapted process mining techniques to exploit the information gathered in change logs. A mining technique is proposed, which reflects all changes applied to the instances of a particular process

type so far. The obtained change process can be used for optimizing and evolving the process schema and is therefore complementary to our approach. For supporting the user in reusing previously performed instance changes, the approach presented in Ref. 38 is not yet suitable as the context of the change is not considered.

Orthogonal to this paper are security aspects. SecServ[60] is a security service, which can be used together with ProCycle to ensure that no uncontrolled changes are performed and compliance rules are met. SecServ enforces the execution of particular activities (e.g., due to legal requirements) and ensures that only activities which are applicable in a specific context can be inserted into a process instance. For a drug procurement process in a hospital, for example, the insertion of a patient treatment step makes no sense and should thus not be allowed.

Existing research on versioning of process schemes is complementary to this paper. While several commercial workflow products use an incremental numbering system for versioning process schemes, Zhao et al. propose a version preserving directed graph.[61] The proposed approach is suitable to represent changes at both the process type and process instance level. For a discussion on issues related to the internal representation of versions we refer to Refs. 44 and 62.

8. Summary and Outlook

In this paper we have presented the ProCycle approach, which provides one of the first implementations for integrated and seamless process life cycle support. As illustrated by Fig. 27, ProCycle extends existing life cycle support as offered in adaptive PAISs (cf. Fig. 4). This includes advanced support for change reuse (see Item 4 in Fig. 27) as well as intelligent support for deriving improved process models based on knowledge about instance changes (see Item 8). In addition, ProCycle allows to evolve knowledge on instance changes over time (see Item 9). These features are fundamental for ensuring maintainability and up-to-date knowledge of the adaptive PAIS.

We elaborated the need for the described kind of change support in several case studies in which we applied the ADEPT1 technology.[19,20,21] Furthermore, ProCycle does not only provide a conceptual framework for flexible process life cycle support, but also a powerful proof-of-concept implementation, which is based on two mature research prototypes – ADEPT2 and CCBR-Tool. Our ambitious goal is to successfully transfer large parts of the described conceptual framework to practice later on. On the one hand, a commercial product is currently built around the ADEPT2 technology. On the other hand, CCBR-Tool has been integrated into the Dynamic Logic Engine (DLE)[63]. The DLE supports domain specialists in defining and executing business processes in highly dynamic environments like the logistics industry. The integration of CBR technologies into the DLE should foster continuous improvement of business processes.

Future work includes the integration of ProCycle with our research on change mining.[38,64] In the MinAdept[64,65] project, we are working on mining techniques for

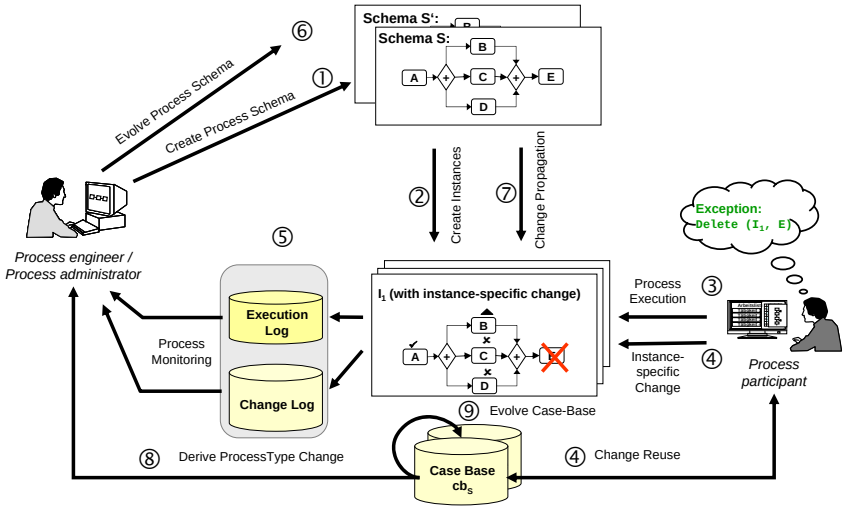

Fig. 27. Process Life Cycle Support with ProCycle

exploiting the information gathered in process change logs. Respective techniques can be used, for example, to provide more advanced support for the generalization of instance changes. One of the themes we are currently working on are algorithms for mining process variants and for merging these variants in a generic (i.e., improved) process model.[64,65] This is done in such a way that the average distance between process instance and newly derived process type schema becomes minimal; i.e., the number of high-level change operations required to derive an instance schema from the new type schema.

Acknowledgements. Parts of this work was funded by the Tiroler Wissenschaftsfond. We would like to thank Stefan Zugal and Jakob Pinggera for the implementation of the CCBR-Tool component and Ulrich Kreher, Markus Lauer and Elisabeth Marini for their help in integrating CCBR-Tool and ADEPT2. In addition, we would like to express our appreciation to Peter Dadam for his feedback and the many valuable discussions on this topic.

References

1. B. Mutschler and M. Reichert and J. Bumiller, "Unleashing the effectiveness of process-oriented information systems: Problem analysis, critical success factors, implications." *IEEE Trans. on Systems, Man and Cybernetics (Part C)* **38(1)** (2008) 280-291.
2. Sarbanes-Oxley Act of 2002, "Public Law 107-204 (116 Statute 745)", (United States Senate and House of Representatives in Congress, 2002).
3. Basel Committee on Banking Supervision, "International convergence of capital measurement and capital standards (Basel II)" (2004).
4. D. Müller and J. Herbst and M. Hammori and M. Reichert, "IT support for release

management processes in the automotive industry." *Proc. BPM'06*, Vienna (2006) 368-377.
5. R. Lenz and M. Reichert, "IT support for healthcare processes - premises, challenges, perspectives." *Data and Knowledge Engineering* **61** (2007) 39-58
6. M. Dumas and A. ter Hofstede and W. van der Aalst (eds.), "Process aware information systems." (Wiley Publishing, 2005).
7. M. Weske, "Business process management: concepts, methods, technology." (Springer 2007).
8. W. van der Aalst and H. Reijers and A. Weijters and B. van Dongen and A.A. de Medeiros and M. Song and H. Verbeek, "Business process mining: An industrial application." *Information Systems* **32** (2007) 713-732.
9. N. Kleiner, "Supporting usage-centered workflow design: Why and how?" *Proc. BPM'04*, (2004) 227-243.
10. M. Reichert and P. Dadam, "ADEPTflex - Supporting dynamic changes of workflows without losing control." *J. of Intelligent Information Systems* **10** (1998) 93-129.
11. W. van der Aalst and T. Basten, "Inheritance of workflows: An approach to tackling problems related to change." *Theoret. Comp. Science* **270** (2002) 125-203.
12. C. Ellis and K. Keddara and G. Rozenberg, "Dynamic change within workflow systems." *Proc. COOCS'95* (1995) 10-21.
13. M. Minor and A. Tartakovski and D. Schmalen and R. Bergmann, "Agile workflow technology and case-based change reuse for long-term processes." *International Journal of Intelligent Information Technologies* **1** (2008) 80-98.
14. S. Rinderle and M. Reichert and P. Dadam, "Correctness criteria for dynamic changes in workflow systems - a survey." *Data and Knowledge Engineering* **50** (2004) 9-34.
15. M. Weske, "Formal foundation and conceptual design of dynamic adaptations in a workflow management system." *Proc. HICSS-34* (2001).
16. M. Reichert and S. Rinderle and U. Kreher and P. Dadam, "Adaptive process management with ADEPT2", *Proc. ICDE'05* (2005) 1113-1114.
17. M. Weske, "Workflow management systems: Formal foundation, conceptual design, implementation aspects", University of Münster, Germany (2000) Habil Thesis.
18. B. Weber and W. Wild and R. Breu, "CBRFlow: Enabling adaptive workflow management through conversational case-based reasoning.", *Proc. ECCBR'04* (2004) 434-448
19. R. Müller, "Event-oriented dynamic adaptation of workflows." PhD thesis, University of Leipzig, Germany (2002)
20. S. Bassil and M. Benyoucef and R. Keller and P. Kropf, "Addressing dynamism in e-negotiations by workflow management systems." *Proc. Workshop on Negotiations in e-Markets - Beyond Price Discovery (DEXA'02)* (2002).
21. S. Bassil and R. Keller and P. Kropf, "A workflow-oriented system architecture for the management of container transportation." *Proc. BPM'04* (2004) 116-131.
22. M. Reichert and S. Rinderle and P. Dadam, "ADEPT workflow management system: Flexible support for enterprise-wide business processes." *Proc. BPM'03* (2003) 370-379.
23. Z. Luo and A. Sheth and K. Kochut and J. Miller, "Exception handling in workflow systems." *Applied Intelligence* **13** (2000) 125-147.
24. S. Rinderle and M. Reichert and P. Dadam, "Flexible support of team processes by adaptive workflow systems." *Distributed and Parallel Databases* **16** (2004) 91-116.
25. S. Rinderle and B. Weber and M. Reichert and W. Wild, "Integrating process learning and process evolution - a semantics based approach." *Proc. BPM'05* (2005) 252-267.
26. B. Weber and M. Reichert and W. Wild, "Case-base maintenance for CCBR-based

process evolution." *Proc. ECCBR'06* (2006) 106-120.
27. W. van der Aalst and A. ter Hofstede, "YAWL: Yet another work ow language." *Information Systems* **30** (2005) 245-275.
28. W. van der Aalst and M. Weske and D. Grünbauer, "Case handling: A new paradigm for business process support." *Data and Knowledge Engineering* **53** (2005) 129-162.
29. F. Casati and S. Ceri and B. Pernici and G. Pozzi, "Workflow evolution." *Data and Knowledge Engineering* **24** (1998) 211-238.
30. B. Weber and M. Reichert and S. Rinderle-Ma, "Change patterns and change support features - enhancing flexibility in process-aware information systems. *Data and Knoweldge Engineering* (accepted for publication).
31. S. Rinderle-Ma and M. Reichert and B. Weber, "Relaxed compliance notions in adaptive process management systems." *Proc. ER'08* (2008).
32. S. Rinderle-Ma and M. Reichert and B. Weber, "On the formal semantics of change patterns in process-aware information systems." *Proc. ER'08* (2008).
33. J.L. Kolodner, "Case-based reasoning." (Morgan Kaufmann, 1993).
34. A. Aamodt and E. Plaza,"Case-based reasoning: Foundational issues, methodological variations and system approaches." *AI Communications* **7** (1994) 39-59.
35. D.W. Aha and H. Muñoz-Avila, "Introduction: Interactive case-based reasoning." *Applied Intelligence* **14** (2001) 7-8.
36. S. Subramaniam and V. Kalogeraki and D. Gunopulos and F. Casati and M. Castellanos and U. Dayal and M. Sayal, "Improving process models by discovering decision points." *Information Systems* **32** (2007) 1037-1055.
37. M. Reichert and P. Dadam and T. Bauer, "Dealing with forward and backward jumps in workflow management systems." *Software and System Modeling* **1** (2003) 37-58.
38. C. Guenther, S. Rinderle-Ma and M. Reichert and W. van der Aalst and J. Recker, "Using process mining to learn from process changes in evolutionary systems." *Int'l Journal of Business Process Integration and Management, Special Issue on Business Process Flexibility* (accepted for publication) (2008).
39. A. Advani and M. Goldstein and Y. Shahar and M. Musen, "Developing quality indicators and auditing protocols from formal guideline models." *AMIA'03* (2003) 11-15.
40. D.W. Aha and L. Breslow and H. Muñoz-Avila, "Conversational case-based reasoning." *Applied Intelligence* **14** (2001) 9-32.
41. OMG: Object Constraint Language (OCL), www.omg.org/docs/ptc/03-10-14.pdf (2003).
42. ICD-9: International classification of diseases, http://www.cdc.gov/nchs/about/otheract/icd9/abticd9.htm (2003).
43. W. Cheetham and J. Price, "Measures of solution accuracy in case-based reasoning systems." *Proc. ECCBR'04* (2004) 106-118.
44. S. Rinderle and M. Reichert and M. Jurisch and U. Kreher, "On representing, purging, and utilizing change logs in process management systems." *Proc. BPM'06* (2006) 241-256.
45. S. Rinderle and M. Reichert and P. Dadam, "Disjoint and overlapping process changes: Challenges, solutions, applications." *Proc.On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE*, LNCS 3290 (2004) 101-120.
46. S. Rinderle and M. Reichert and P. Dadam, "On dealing with structural conflicts between process type and instance changes." *Proc. BPM'04*, (2004) 274-289:
47. S. Rinderle, "Schema evolution in process management systems." PhD thesis, University of Ulm (2004)
48. B. Weber and W. Wild, "Conversational case-based reasoning support for business

process management." *Proc. Mixed-Initiative Problem-Solving Assistant - Papers from the AAAI Fall Symposium.* (AAAI Press, 2005).
49. M. Reichert and P. Dadam and U. Kreher and M. Jurisch and K. Göser, "Architectural Design of Flexible Process Management Technology." *Proc. MKWI'08*, (2008) 328-343.
50. J. Pinggera and S. Zugal and B. Weber and W. Wild and M. Reichert, "Integrating case-based reasoning with adaptive process management." Technical report, CTIT, University of Twente, Enschede (2008).
51. M. Minor and A. Tartakovski and R. Bergmann, "Representation and structure-based similarity assessment for agile workflows." *Proc. ICCBR'07* (2007) 224-238.
52. S. Sadiq and W. Sadiq and M. Orlowska, "A framework for constraint specification and validation in flexible workflows." *Information Systems* **30** (2005) 349-378.
53. R. Lu and S. Sadiq, "On the discovery of preferred work practice through business process variants." *Proc. ER2007* (2007) 165-180.
54. M. Adams and A. ter Hofstede and D. Edmond and W. van der Aalst, "A service-oriented implementation of dynamic flexibility in workflows." *Proc. On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*, LNCS 4275 (2006).
55. M. Pesic and M. Schonenberg and N. Sidorova and W. van der Aalst, "Constraint-based workflow models: Change made easy." *Proc. On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS*, LNCS 4803 (2007) 77-94.
56. S. Jablonski and K. Stein and M. Teschke, "Experiences in workflow management for scientific computing." *Proc. DEXA '97* (1997) 56-61.
57. W. van der Aalst, "Inheritance of business processes: A journey visiting four notorious problems." *Proc. Petri Net Technology for Communication Based Systems.* (2003) 383-408.
58. V. Guth and A. Oberweis, "Delta analysis of petri net based models for business processes." *Proc. Applied Informatics.* (1997) 23-32.
59. W. van der Aalst, "Business alignment: Using process mining as a tool for delta analysis and conformance testing." *Requirements Eng. Journal* (2005) 198-211.
60. B. Weber and M. Reichert and W. Wild and S. Rinderle, "Balancing flexibility and security in adaptive process management systems." *Proc. On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE*, LNCS 3760 (2005) 59-76.
61. X. Zhao and C. Liu, "Version management in the business process change context." *Proc. BPM'07* (2007) 198-213.
62. S. Rinderle-Ma and U. Kreher and M. Lauer and P. Dadam and M. Reichert, "On representing instance changes in adaptive process management systems." *Proc. WET-ICE'06* (2006) 297-302.
63. W. Wild and R. Wirtensohn and B. Weber, "Dynamic engines - a flexible approach to the extension of legacy code and process-oriented application development." *Proc. WETICE'06* (2006) 279-284.
64. C. Li and M. Reichert and A. Wombacher, "On measuring process model similarity based on high-level change operations." *Proc. ER'08* (2008).
65. C. Li and M. Reichert and A. Wombacher, "Discovering reference process models by mining process variants." *Proc. ICWS'08* (2008).