Improving Exception Handling by Discovering Change Dependencies in Adaptive Process Management Systems

Barbara Weber[1,⋆], Werner Wild[2], Markus Lauer[3], and Manfred Reichert[4]

[1] Quality Engineering Research Group, University of Innsbruck, Austria
Barbara.Weber@uibk.ac.at
[2] Evolution Consulting, Innsbruck, Austria
werner.wild@evolution.at
[3] Dept. Databases and Information Systems, University of Ulm, Germany
markus.lauer@uni-ulm.de
[4] Information Systems Group, University of Twente, The Netherlands
m.u.reichert@utwente.nl

Abstract. Process-aware information systems should enable the flexible alignment of business processes to new requirements by supporting deviations from the predefined process model at runtime. To facilitate such dynamic process changes we have adopted techniques from case-based reasoning (CBR). In particular, our existing approach allows to capture the semantics of ad-hoc changes, to support their memorization, and to enable their reuse in upcoming exceptional situations. To further improve change reuse this paper presents an approach for discovering dependencies between ad-hoc modifications from change history. Based on this information better user assistance can be provided when dynamic process changes have to be made.

1 Introduction

Due to frequent changes in its business environment an enterprise must be able to flexibly and continuously align its information systems (IS) and its business processes. Enterprise IS therefore must provide for flexible process support while still enforcing some degree of control [1]. In particular, there is an essential requirement for maintaining a close "fit" between real-world business processes and the workflows as supported by the IS, their current generation is known as Process-Aware Information Systems (PAIS) [2].

Recently, significant efforts have been undertaken to make PAIS more flexible and several approaches for *adaptive* process management have emerged [1,3,4]. The underlying idea is to enable (dynamic) changes of different process aspects (e.g., control flow, organizational, functional, and informational perspectives) and at different process levels (e.g., instance and type level). In particular, authorized users must be able to deviate from the pre-defined process model as needed, i.e., ad-hoc changes (e.g., to add or shift activities) of individual process

⋆ Part of this research was funded by a grant from the Tiroler Wissenschaftsfond.

instances must be possible at runtime to deal with exceptional or changing situations. For example, during a medical treatment process the patient's current medication may have to be changed due to an allergic reaction, i.e., the process instance representing this treatment procedure must be dynamically adapted.

To facilitate exception handling we have adopted techniques from *case-based resoning* (CBR) [5,6,7]. This allows us to capture contextual knowledge about ad-hoc changes and to assist actors in reusing it. For this we apply an interactive variant of CBR (i.e., *conversational CBR* [8]), describe ad-hoc changes as *cases* and memorize them in a *case base* CB. In its simplest form a case covers a single change operation (e.g., insertion of a process activity). However, cases may contain several (semantically) related change operations as well. For example, in a medical treatment process a magnet resonance tomography (MRT) may have to be skipped for a patient with cardiac pacemaker, instead, another imaging procedure (e.g., computer tomography) might have to be applied. Our objective is to support reuse of such complex changes in similar situations to enable actors to operate at a higher semantical level and to relieve them from specifying the change from scratch each time.

Since ad-hoc changes are often applied in exceptional situations, we cannot expect that semantically related adaptations are always conducted at the same time, i.e., they are not always added as a single case to the PAIS. This happens when end users are rather inexperienced and do not think through all consequences, when changes to the same instance are performed by different actors or when these dependencies are not known when adding a case. Over time, the PAIS may end up with several inter-related cases which are frequently applied in combination with each other. By discovering such inter-case dependencies and by considering this knowledge in the context of change reuse we provide for better user assistance. When reusing a certain case the PAIS can suggest users to apply dependent cases as well. To further improve this approach, cases which always co-occur shall be merged and the CB should be refactored accordingly.

Section 2 summarizes background information needed for the understanding of our approach. In Section 3 we introduce the notion of co-occuring cases. Based on this, in Section 4 we sketch how actors can be assisted in reusing inter-dependent changes and in refactoring a CB by merging cases. Section 5 discusses related work and Section 6 concludes with a summary and outlook.

2 Supporting Change Reuse Through CBR

This section covers backgrounds needed for the understanding of this paper. First, we introduce basic notions related to process management. Second, we discuss how CBR is used in our approach for capturing the semantics of ad-hoc changes, for memorizing these changes, and for reusing them in similar situations.

2.1 Basic Notions

For each supported business process a corresponding *process type* T exists. It can be related with one or more process schemes representing different versions of the process. Each *process schema S* is described in a graph-like fashion, and comprises a set of *activities* and *control connectors* between them. Based on a schema S new *process instances* can be created and executed accordingly. For example, in instance I_1 in Fig. 1 activities `A` and `C` are completed whereas activity `B` is activated (i.e., its work items are offered to users in their worklists).

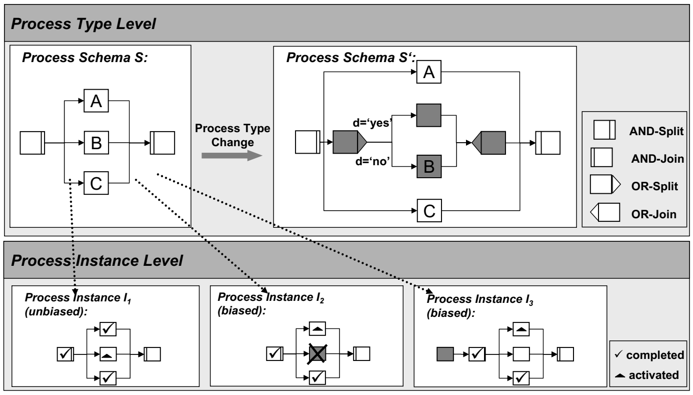


Fig. 1. Process Type and Process Instance

To deal with exceptional situations at the instance level, users must deviate from the pre-modeled schema (e.g., by deleting activities) [1,5,9]. *Ad-hoc changes* are instance-specific and do not affect the *execution schema* of any other running process instances. In Fig. 1, instance I_2 has undergone an individual modification (i.e., the dynamic deletion of activity `B`). Thus the execution schema of I_2 deviates from its original process schema S. Individually modified process instances are called *biased*, unchanged ones are denoted as *unbiased*.

2.2 Capturing Semantics of Ad-Hoc Changes with CBR

Our approach uses case-based reasoning (CBR) techniques to capture the semantics of an ad-hoc change, to memorize it and to support its reuse in similar situations (for details see [6,7]). Case-based reasoning (CBR) is a contemporary approach to problem solving and learning [10]. New problems are dealt with by drawing on past experiences – described in cases and stored in case bases – and by adapting their solutions to the new problem situation. For representing a concrete ad-hoc change we use the concept of a *case*, which captures the context of, and the reasons for the respective deviation (cf. Fig. 2). More precisely, a case contains a textual problem description *pd* which briefly summarizes the exceptional situation that led to the ad-hoc deviation. The reasons for the change are described in question-answer (QA) pairs $\{q_1a_1, \ldots, q_na_n\}$ each of which denotes one particular condition (for details see below). The solution part *sol* (i.e., the action list) of a case contains the concrete change operations applied.

The ad-hoc changes covered by a particular case c can be reused, i.e., they can be re-applied to other instances. QA pairs are used to retrieve cases handling similar problems. If an adequate case is found its solution part can be applied to the given process instance. All instances to which case c has been applied to are kept in its instance set $instanceSet_c$. If no similar cases can be found the user adds a new case with the respective change information to the system.

Definition 1 (Case). *A case c is a tuple (pd_c, $qaSet_c$, sol_c, $instanceSet_c$) where*

- *pd_c is a textual problem description*
- *$qaSet_c = \{q_1a_1, \ldots, q_na_n\}$ denotes a set of question-answer pairs*
- *$sol_c = \{ op_j \mid op_j = (opType_j, s_j, paramList_j), j = 1, \ldots, k\}$ is the solution part of the case denoting a list of change operations (i.e., the changes that have been applied to one or more process instances) [1]*
- *$instanceSet_c$ is the set of process instances to which case c has been applied*

The question of a QA pair is usually entered as free text, however, to reduce duplicates it can alternatively be selected from a list of already existing questions. The answer can either be free text or a structured answer expression (cf. Fig 2 (a)). Answer expressions allow us to use contextual knowledge already kept in the PAIS (e.g., due to legal requirements), thus avoiding redundant data entry. Questions with answer expressions can be evaluated automatically by retrieving values for their context attributes from existing data in the system (e.g., the medical problems of a patient as stored in his electronic patient record), i.e., they do not have to be answered by users, thus preventing errors and saving time. Free text answers are used when no suitable context attributes are defined within the system or the user is not trained to write answer expressions. For example, the second QA pair in Fig. 2 (a) contains an answer expression using the context attribute *Patient.age*. It can therefore be evaluated automatically. By contrast, the answer in the first QA pair is free text provided by the user.

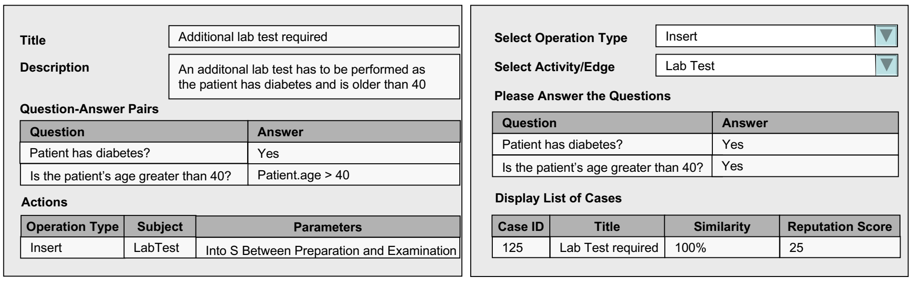
To be able to reason about the changes applied to a particular process instance I we introduce $caseList_I$ as the list of all cases which have been applied to I, in their application order. If an instance I is *biased* its $caseList_I$ is not empty. All cases applied to process instances created from schema version S are stored in a case base CB_S associated with S.

Definition 2 (Case Base). *A case base CB_S is a tuple (S, $\{c_1, \ldots, c_m\}$) where*

- *S denotes the schema version to which the case base is related*
- *$\{c_1, \ldots, c_m\}$ denotes a set of cases (cf. Def. 1)*

When deviations from the pre-defined process schema become necessary the user initiates a case retrieval dialogue (cf. Fig 2 (b)). The system then assists her in finding already stored similar cases (i.e., change scenarios in our context) by presenting a set of questions to be answered in any number and any order. Questions

[1] An operation $op_j := (opType_j, s_j, paramList_j)$ (j = 1, ..., k) consists of operation type opType$_j$, subject s_j of the change, and parameter list paramList$_j$.



Title	Additional lab test required
Description	An additonal lab test has to be performed as the patient has diabetes and is older than 40

Question-Answer Pairs

Question	Answer
Patient has diabetes?	Yes
Is the patient's age greater than 40?	Patient.age > 40

Actions

Operation Type	Subject	Parameters
Insert	LabTest	Into S Between Preparation and Examination

(a)

Select Operation Type	Insert
Select Activity/Edge	Lab Test

Please Answer the Questions

Question	Answer
Patient has diabetes?	Yes
Is the patient's age greater than 40?	Yes

Display List of Cases

Case ID	Title	Similarity	Reputation Score
125	Lab Test required	100%	25

(b)

Fig. 2. CCBR Dialogs - Adding a New Case (a) and Retrieving Similar Cases (b)

with an answer expression are automatically evaluated by retrieving the values of the respective context attributes from the PAIS without user intervention. Based on this the system then searches for similar cases by calculating the similarity for each case in the case base CB_S. Similarity is calculated by dividing the number of correctly answered questions minus the number of incorrectly answered questions by the total number of questions in the case. It then displays the top n ranked cases (ordered by decreasing similarity) as well as related information (e.g., reputation scores). The user then has several options:

1. The user can directly answer any of the remaining unanswered questions (in arbitrary order), similarity is then recalculated and the n most similar cases are displayed to the user.
2. The user can apply a filter to the case base (e.g., by only considering cases whose solution part contains a particular change operation). All cases not matching the filter criteria are removed from the displayed list of cases.
3. The user can decide to review displayed cases in detail.
4. The user can select one of the displayed cases for reuse. The actions specified in the case's solution part are then forwarded to and executed by the PAIS. The instance set of the selected case is adjusted accordingly.

3 Inter-case Dependencies

This section first gives a typical example for inter-case dependencies and then introduces the formal notation to be used in this paper.

3.1 Motivating Example

Fig. 3 shows the cruciate rupture treatment process for a particular patient. This treatment process had to be modified due to an unanticipated situation. To confirm the suspicion of a cruciate rupture usually an x-ray as well as a magnet resonance tomography (MRT) are performed. However, as this patient has a cardiac pacemaker the radiologist decided to skip the MRT. To still get a reliable diagnosis, the attending physician ordered a computer tomography (CT) instead. Though these two changes are related to each other they were added to the system by different actors at different times. Case c_1 was added by the radiologist and resulted in the deletion of the MRT activity. Some time later the attending physician added case c_{17} to insert the CT activity.

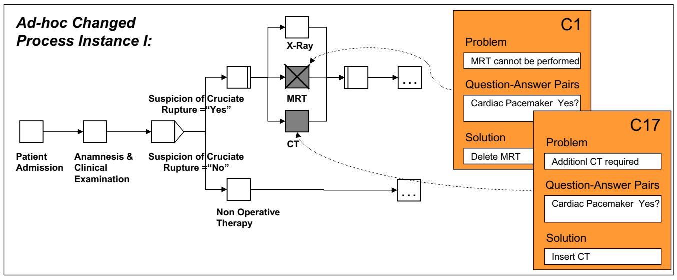


Fig. 3. Application Example

Cases applied to the same instance may be independent of, or dependent on each other. In our example the deletion of the MRT activity caused the insertion of the CT activity, i.e., the additional CT compensates the missing MRT. When such inter-case dependencies exist, the reuse of a particular case might necessitate further changes. In our example, reusing case c_1 may require the application of case c_{17} as well since an alternative imaging procedure is needed. Discovering such inter-case dependencies is crucial to better assist users when they need to make complex changes to the system.

3.2 Co-occuring Cases

Let CB_S be a case base and let c_1 and c_2 be two cases in CB_S. A metrics about inter-case dependencies is the *conditional co-occurence rate* $CoRate(c_2|c_1)$. For a set of process instances this metrics denotes the relative frequency of the application of case c_2 on condition that case c_1 has been applied too.

Definition 3 (Conditional Co-Occurrence Rate). *Let S be a process schema with case base CB_S and let c_1, $c_2 \in CB_S$ be two cases. The conditional co-occurence rate $CoRate(c_2|c_1)$ denotes the relative frequency of case c_2 on condition that case c_1 has already been applied. Formally:*

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

When a user wants to reuse a case $c \in CB_S$ we present her all other cases $c_k \in CB_S$ with $CoRate(c_k|c)$ exceeding threshold $thres \leq 1$. Section 4 describes how we assist actors in reusing inter-dependent changes. In this context cases with *strong co-occurence* are of particular interest.

Definition 4 (Strong Co-Occurence of Cases). *Let S be a process schema with case base CB_S. Let further $c_1, c_2 \in CB_S$. Then:*

1. *If $CoRate(c_2|c_1) = 1$ holds we denote case c_2 as strongly co-occurrent with case c_1 (i.e., c_2 must only have been applied when c_1 has been applied too).*
2. *If $CoRate(c_2|c_1) = CoRate(c_1|c_2) = 1$ holds we denote cases c_2 and c_1 as being strongly co-occurent with each other (i.e., c_2 always occurs when c_1 has been applied and vice versa).*

If c_2 is strongly co-occurrent with c_1 (cf. Def. 4.1), obviously, $instanceSet_{c_1} \subseteq instanceSet_{c_2}$ must hold. Consequently, we obtain $CoRate(c_2|c_1) = 1$ and $CoRate(c_1|c_2) = \frac{|instanceSet_{c_1}|}{|instanceSet_{c_2}|}$. As a special scenario consider two cases c_1 and c_2 which are strongly co-occurent with each other (cf. Def. 4.2). Trivially, we then obtain $|instanceSet_{c_1}| = |instanceSet_{c_2}|$. If cases c_1 and c_2 are strongly co-occurent with each other and the total number of co-occurrences exceeds threshold $minOccur \in \mathbb{N}$, the process engineer is notified about the option to merge these inter-dependent cases (cf. Section 4.2).

4 Discovering and Utilizing Knowledge About Inter-case Dependencies

To discover co-occurent changes we analyze a process schema's CB. We utilize the obtained knowledge to assist actors in reusing complex changes (cf. Section 4.1) and to support process engineers in refactoring CBs (cf. Section 4.2).

4.1 Assisting Actors in Reusing Dependent Cases

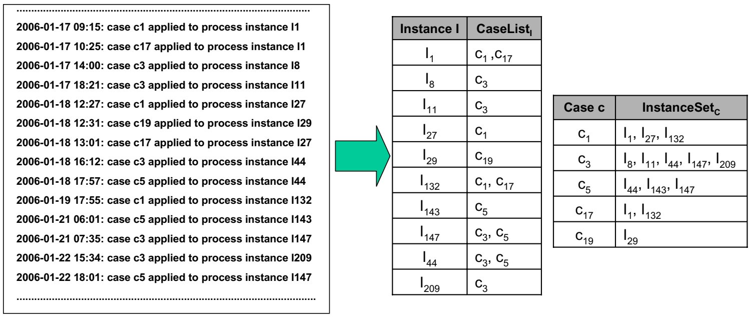
When a case $c \in CB_S$ is reused (i.e., c is applied to a process instance I) the system displays all cases $cse \in CB_S$ for optional reuse[2] which co-occur with c and for which the co-occurence rate $CoRate(cse|c)$ exceeds a given threshold. This is accomplished by Algorithm 1. First, Algorithm 1 adds case c to the list of cases ($caseList_I$) which have already been applied to instance I (line 3). For each case cse (except for already applied cases to instance I), Algorithm 1 then determines the conditional co-occurence rate $CoRate(cse|c)$. This is done by determining the total number of co-occurences between cse and c over all instances of process schema S and by dividing it by the total number of occurences for case c (line 6). Finally, only those cases c_k are displayed (for potential reuse) whose co-occurence rate $CoRate(cse|c)$ exceeds the given threshold *thres*.

For example, consider the scenario depicted in Fig. 4. Assume that the changes represented by case c_5 shall be applied to process instance I_{132}. According to Algorithm 1 the system adds case c_5 to $CaseList_{I132}$ and then determines the conditional co-occurence rate $CoRate(cse|c_5)$ for each case $cse \in CB_s \setminus CaseList_{I132}$ (i.e., $\{c_3, c_{19}\}$) related to any instance from $InstanceSet_{c_5} = \{I_{44}, I_{143}, I_{147}\}$. We

[2] Cases which have already been applied to process instance I are not shown.

Algorithm 1. Display_CoOccurent_Cases

```
 1: Input: Case $c$; ProcessInstance $I$; float thres;
 2:
 3: add case c to $CaseList_I$;
 4: Integer TotalOccurenceOfC := $|InstanceSet_c|$;
 5: **for all** $cse \in CB_S \setminus CaseList_I$ **do**
 6:    $CoOccurenceRate_{cse} := \frac{|instanceSet_c \cap instanceSet_{cse}|}{TotalOccurenceOfC}$;
 7:    **if** $CoOccurenceRate_{cse} \geq$ thres **then**
 8:       DISPLAY(cse)
 9:    **end if**
10: **end for**
```



```
..........................................................................................
2006-01-17 09:15: case c1 applied to process instance I1
2006-01-17 10:25: case c17 applied to process instance I1
2006-01-17 14:00: case c3 applied to process instance I8
2006-01-17 18:21: case c3 applied to process instance I11
2006-01-18 12:27: case c1 applied to process instance I27
2006-01-18 12:31: case c19 applied to process instance I29
2006-01-18 13:01: case c17 applied to process instance I27
2006-01-18 16:12: case c3 applied to process instance I44
2006-01-18 17:57: case c5 applied to process instance I44
2006-01-19 17:55: case c1 applied to process instance I132
2006-01-21 06:01: case c5 applied to process instance I143
2006-01-21 07:35: case c3 applied to process instance I147
2006-01-22 15:34: case c3 applied to process instance I209
2006-01-22 18:01: case c5 applied to process instance I147
..........................................................................................
```

Instance I	$CaseList_I$
I_1	c_1, c_{17}
I_8	c_3
I_{11}	c_3
I_{27}	c_1
I_{29}	c_{19}
I_{132}	c_1, c_{17}
I_{143}	c_5
I_{147}	c_3, c_5
I_{44}	c_3, c_5
I_{209}	c_3

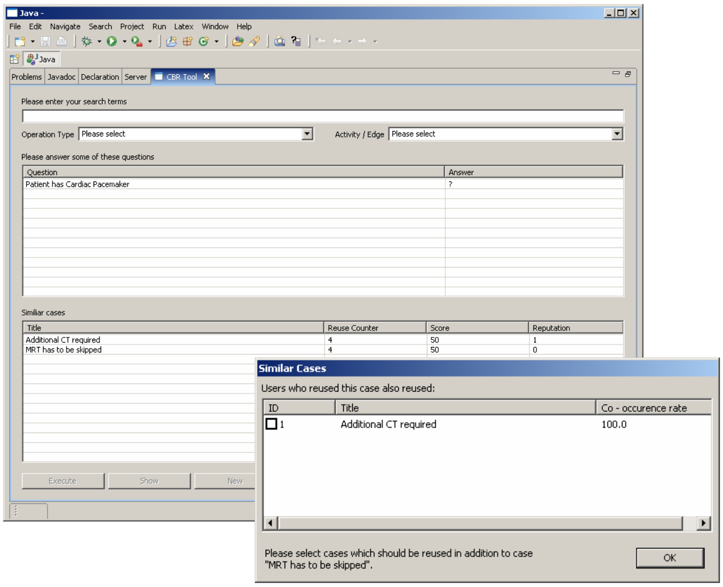
Case c	$InstanceSet_C$
c_1	I_1, I_{27}, I_{132}
c_3	$I_8, I_{11}, I_{44}, I_{147}, I_{209}$
c_5	I_{44}, I_{143}, I_{147}
c_{17}	I_1, I_{132}
c_{19}	I_{29}

Fig. 4. Log File

obtain $CoRate(c_3|c_5) = \frac{2}{3}$ and $CoRate(c_{19}|c_5) = \frac{0}{3}$. For example, if we have chosen $thres = 0.6$ case c_3 will be displayed to the users (for optional reuse) when applying c_5 to instance I_{132}.

When reusing a case a wizard opens and all dependent cases are displayed to the user (cf. Fig. 5). For each dependent case its identifier, title and co-occurrance rate are shown. The co-occurance rate reflects the confidence of the system that the dependent case should be applied too in this particular situation. The user can then optionally reuse any of the displayed cases.

At first glance the described approach seems to be easy to implement. However, when reusing a case and applying its changes it must be guaranteed that the respective process instance still meets certain correctness and consistency constraints. In particular, the pre-conditions for applying the change operations captured by a case must be met when reusing it. As an illustrative example consider the medical treatment process depicted in Fig. 6. Assume that the respective patient complains about pains in his knee. The attending physician therefore orders an additional examination. This change is represented by case c_{22} which captures the insertion of activity `Follow-up examination` between activities `Non Operative Therapy` and `Documentation and Discharge`. Assume further that during the follow-up examination it is found that the patient suffers from a contusion and the physician therefore decides that a puncture has


Java -
File Edit Navigate Search Project Run Latex Window Help
Java
Problems Javadoc Declaration Server CBR Tool
Execute Show New
OK

Please enter your search terms

Operation Type: Please select   Activity / Edge: Please select

Please answer some of these questions

Question	Answer
Patient has Cardiac Pacemaker	?

Similiar cases

Title	Reuse Counter	Score	Reputation
Additional CT required	4	50	1
MRT has to be skipped	4	50	0

Similar Cases

Users who reused this case also reused:

ID	Title	Co - occurence rate
☐ 1	Additional CT required	100.0

Please select cases which should be reused in addition to case "MRT has to be skipped".

Fig. 5. Presenting Dependent Cases to the User

to be performed as well. This change is captured by case c_{35} (subsequent insertion of activity `Puncture` between activities `Follow-up examination` and `Documentation and Discharge`). Note that the definition of the latter change depends on the presence of activity `Follow-up examination` which was introduced by case c_{22}. Such dependencies, in turn, could result in parameterization problems or inconcistencies when a user solely wants to reuse case c_{35}. Generally, the change framework we use allows us to efficiently detect such undesired situations [11]. In our example, a user who wants to reuse case c_{35} has two options. Either she can apply case c_{22} as well or she may adapt the case by modifying the parameterization of the respective change (e.g., by replacing the position parameter `Follow-up examination` with `Non Operative Therapy`).

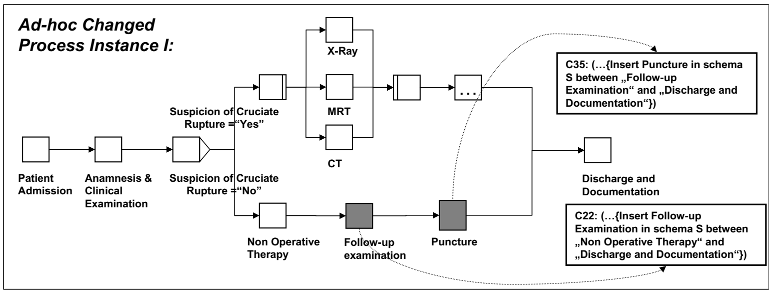

Ad-hoc Changed Process Instance I:
Patient Admission
Anamnesis & Clinical Examination
Suspicion of Cruciate Rupture ="Yes"
Suspicion of Cruciate Rupture ="No"
X-Ray
MRT
CT
...
Non Operative Therapy
Follow-up examination
Puncture
Discharge and Documentation
C35: (...{Insert Puncture in schema S between „Follow-up Examination" and „Discharge and Documentation"})
C22: (...{Insert Follow-up Examination in schema S between „Non Operative Therapy" and „Discharge and Documentation"})

Fig. 6. System Supported Conflict Resolution

4.2 Refactoring Case Bases by Merging Cases

In order to increase problem solving efficiency we can compress the case base by merging strongly co-occurent cases. For this scenario, consider the reuse of a case c and its application to a particular process instance. Whenever such an event occurs, it triggers an analysis of the case base. More precisely, it is checked whether the reused case is strongly co-occurent with other cases. If so, the knowledge engineer is notified accordingly and may then decide to merge respective cases. Note that in this scenario we can restrict the analysis (i.e., the comparison of case c with other cases) to those cases belonging to the case list $caseList_I$, as only cases within that list can be strongly co-occurent with case c. Algorithm 2 is applied to detect cases which strongly co-occur with each other (cf. Def. 4). For the sake of readability we treat this algorithm separately from Algorithm 1, however, for a practical implementation they can easily be merged.

Algorithm 2. Notify_About_Strongly_CoOccurent_Cases

```
1: Input: Case $c$; ProcessInstance $I$; int minOccur;
2:
3: Add $I$ to $instanceSet_c$;
4: for all $cse \in CaseList_I$ do
5:    if $instanceSet_{cse} = instanceSet_c$ then
6:       if $|instanceSet_c| \geq minOccur$ then
7:          NOTIFY(c, cse)
8:       end if
9:    end if
10: end for
```

Again consider the example from from Fig. 4. Assume that case c_{17} is applied to instance I_{27}. Instance I_{27} is then added to the instance set of case c_{17}. Further, Algorithm 2 (with, e.g., $minOccur = 3$) identifies case c_1 as being strongly co-occurent with case c_{17}. Consequently, the process engineer is notified that c_1 and c_{17} are strongly co-occurent which each other and thus could merge these two cases. In this situation a new case c' can be created in CB_S and the original cases are deactivated[3], i.e., a refactoring of the case base takes place. The problem descriptions and QA pairs related to the two cases have to be manually merged by the process engineer; the corresponding solution parts, in turn, can be automatically merged by unifying the change operations of the original cases in the correct order. Different optimizations for purging the resulting operation sets can be applied in this context. However, this is beyond the scope of this paper. In addition, the schema-specific case base CB_S can be regularly searched for any strongly co-occurent cases by applying Algorithm 3.

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

Algorithm 3. Scan_For_Strongly_CoOccurent_Cases

```
1: Input: Case Base $CB_S$; int minOccur;
2:
3: $CB = CB_S$
4: while $CB \neq \emptyset$ do
5:    take arbitrary case $cse$ from $CB$
6:    $CB = CB \setminus \{cse\}$
7:    for all $case \in CB$ do
8:       if $instanceSet_{case} = instanceSet_{cse}$ then
9:          if $|instanceSet_{case}| \geq minOccur$ then
10:            NOTIFY(case, cse)
11:         end if
12:      end if
13:   end for
14: end while
```

5 Related Work

Similar to our approach process mining aims at extracting process knowledge from log data. So far, focus of mining techniques has been on the extraction of process models from execution logs [12,13,14]. For example, the alpha algorithm can be used to construct a Petri net model describing the behavior observed in the log. Similarly, the Multi-Phase Mining approach can be used to construct event-driven process chains from logs. Recent approaches also use event-based data for mining model perspectives other than control flow (e.g., process performance [15]). Mature tools like the ProM framework allow constructing different types of models from real process executions. However, process mining research has not yet addressed applying minig techniques to change logs.

The necessity to support the user in exceptional situations has been addressed by adaptive process management technology [1,9,16]. ADEPT [1], for instance, supports the user in defining (syntactically) correct process changes during runtime. For example, when an activity is deleted at the process instance level, the system might suggest the deletion of data dependent activities as well. However, ADEPT does not consider semantical dependencies between changes yet.

Complementary to this paper [17] covers quality aspects relevant when applying CBR for the memorization and the reuse of ad-hoc modifications and the deriviation of process type changes. While [17] broadly deals with quality issues and aims at increasing the performance of the CBR system by increasing problem solving efficiency, CB competence and solution quality, this paper addresses how user assistance can be improved through mining inter-case dependencies.

6 Summary and Outlook

We have proposed an approach to improve exception handling in adaptive process management systems through discovering and utilizing knowledge about dependencies between ad-hoc modifications. Actors are assisted in reusing previously applied ad-hoc changes by presenting related modifications to them as well. In addition, knowledge about inter-case dependencies is used to improve the quality of the CB (i.e., the collection of retrievable and reusable ad-hoc changes) and to increase problem solving efficiency. Ongoing work includes the evaluation of our prototype in a real world scenario. Future work will investigate how the reuse of ad-hoc modifications can be further improved. For example, a particular case representing an ad-hoc modification might not directly be applicable, but may require some adaptation (e.g., the parameterization of change operations may have to be adapted). Finally, we aim to use more of the semantics captured in our approach, e.g., to be able to reason about "similarity" of changes.

References

1. Reichert, M., Dadam, P.: ADEPT$_{flex}$ – supporting dynamic changes of workflows without losing control. JIIS **10** (1998) 93–129
2. Dumas, M., ter Hofstede, A., van der Aalst, W., eds. In: Process Aware Information Systems. Wiley Publishing (2005)
3. Jørgensen, H.D.: Interactive Process Models. PhD thesis, Norwegian University of Science and Technology, Trondheim, Norway (2004)
4. Rinderle, S., Reichert, M., Dadam, P.: Correctness criteria for dynamic changes in workflow systems – a survey. Data and Knowledge Engineering **50** (2004) 9–34
5. Weber, B., Wild, W., Breu, R.: CBRFlow: Enabling adaptive workflow management through conversational case-based reasoning. In: ECCBR'04, Madrid (2004) 434–448
6. Weber, B., Rinderle, S., Wild, W., Reichert, M.: CCBR–driven business process evolution. In: ICCBR'05, Chicago (2005) 610–624
7. Rinderle, S., Weber, B., Reichert, M., Wild, W.: Integrating process learning and process evolution - a semantics based approach. In: BPM 2005. (2005) 252–267
8. Aha, D.W., Muñoz-Avila, H.: Introduction: Interactive case-based reasoning. Applied Intelligence **14** (2001) 7–8
9. Luo, Z., Sheth, A., Kochut, K., Miller, J.: Exception handling in workflow systems. Applied Intelligence **13** (2000) 125–147
10. Kolodner, J.L.: Case-Based Reasoning. Morgan Kaufmann (1993)
11. Rinderle, S.: Schema Evolution in Process Management Systems. PhD thesis, University of Ulm (2004)
12. v.d. Aalst, W., van Dongen, B., Herbst, J., Maruster, L., Schimm, G., Weijters, A.: Workflow mining: A survey of issues and approaches. Data and Knowledge Engineering **27** (2003) 237–267
13. Golani, M., Pinter, S.S.: Generating a process model from a process audit log. In: Proc. BPM'03, Eindhoven (2003) 136–151
14. van Dongen, B., van der Aalst, W.: Multi-phase process mining: Building instance graphs. In: Conceptual Modeling - ER 2004. LNCS 3288, Berlin (2004) 362–376
15. van der Aalst, W., Song, M.: Mining social networks. uncovering interaction patterns in business processes. In: Proc. BPM'04, Potsdam, Germany (2004) 244–260
16. Weske, M.: Workflow management systems: Formal foundation, conceptual design, implementation aspects. University of Münster, Germany (2000) Habil Thesis.
17. Weber, B., Reichert, M., Wild, W.: Case-base maintenance for ccbr-based process evolution. In: ECCBR'06. (2006)