A FRAMEWORK FOR SEMANTIC RECOVERY STRATEGIES IN CASE OF PROCESS ACTIVITY FAILURES

Stefanie Rinderle[1], Sarita Bassil[2] and Manfred Reichert[3]

[1]*DBIS, Dept. DBIS, University of Ulm, Germany,*
[2]*Dept. Computer and Information Sciences, Holy Spirit University of Kaslik Lebanon,*
[3]*Information Systems Group, University of Twente, The Netherlands,*

Keywords: Process Recovery, Exception Handling, Automated Process Change.

Abstract: During automated process execution semantic activity failures may frequently occur, e.g., when a vehicle transporting a container has a breakdown. So far there are no applicable solutions to overcome such exceptional situations. Often the only possibility is to cancel and roll back respective process instances what is not always possible and more often not desired. In this paper we contribute towards the system–assisted support of finding forward recovery solutions. Our framework is based on the facility to (automatically) perform dynamic changes of single process instances in order to deal with the exceptional situation. We identify and formalize factors which influence the kind of applicable recovery solutions. Then we show how to derive possible recovery solutions and how to evaluate their quality with respect to different constraints. All statements are illustrated by well–studied cases from different domains.

1 INTRODUCTION

Exceptions in computerized processes are common (Strong and Miller, 1995). From a process perspective, critical exceptions often occur during activity execution (Elmagarmid, 1992). As an example, take the container transportation process from Fig. 2. Technical problems of vehicles or traffic jams may appear at any time while vehicle V is on the road between its origin location O and its destination location D. In such exceptional cases, the execution of activity `move vehicle` has to be stopped and an alternative solution needs to be figured out. The process-aware information system (PAIS) (v.d. Aalst and van Hee, 2002) should adequately support users by providing the context in which the exception has occured and by proposing recovery strategies to be applied.

If an activity fails during process instance execution one possibility to deal with this (semantic) failure is to roll back the instance execution or parts of it. The classical transaction concept has appeared to be inadequate in this context since long–term locks are not applicable in practice. Therefore more appropriate approaches have been figured out, e.g., nested transactions and (semantic) compensation as extensions to classical rollback and ACID transactions (Elmagarmid, 1992). One of the first approaches for the semantic rollback of processes was offered by Sagas (Garcia-Molina and Salem, 1987): in case of activity failure the effects of the failed activity are undone (i.e., the respective transaction is rolled back) whereas completed activities are compensated in backward order. More sophisticated approaches use scopes of control to express a changing compensation behavior depending on the process state (Leymann, 1995; Leymann and Roller, 2000).

In practice it is not always possible to roll back a process, e.g., there may be no compensation for an already accomplished surgery in a medical treatment process or the time for compensation may be exceeded, e.g., we cannot compensate a flight reservation only few hours before the trip. In this perspective, if no backward recovery is possible, a forward recovery solution applying dynamic process instance changes is often favorable. In the literature, so far no sufficient strategies for forward recovery have been defined. In most cases forward recovery comprises dynamic modifications of the process instances, e.g. forward or backward jumps in the flow of control (Reichert et al., 2003; Mourao and Antunes, 2004). Except the rule–based approach of AGENTWORK (Müller, 2004; Müller et al., 2004) it has not been investigated in sufficient detail how to (automatically) determine reasonable ad–hoc modifications as forward recovery solutions. Note that at a certain point during process execution the set of applicable changes may be very large, but only few of them may be reasonable to deal with the exceptional situation.

Rinderle S., Bassil S. and Reichert M. (2006).
A FRAMEWORK FOR SEMANTIC RECOVERY STRATEGIES IN CASE OF PROCESS ACTIVITY FAILURES.
In *Proceedings of the Eighth International Conference on Enterprise Information Systems - DISI*, pages 136-143
DOI: 10.5220/0002453201360143

In this paper, we contribute to better support users in finding adequate forward recovery solutions at the presence of exceptional situations. We first identify factors which may influence the right choice of a forward recovery solution. One important factor is to know the current data context of a failed activity. For example, if activity `move vehicle` fails it is crucial to know the position of the vehicle in order to figure out an alternative solution (cf. Fig. 2). We may also think of an emergency case occuring while a physician is filling in a form (e.g., to enter medical history data of her patient) and forcing her to interrupt her task. In such a case the data already filled in by the physician should not be lost, i.e., the data context of the activity should be kept in order to resume it later on. Other influence factors deal with a process as a whole. If an exception has to be handled, it is a must to know how far process execution has been progressed or what the major goals of this process are. As an example, the major goal of a medical treatment process could be to successfully operate a patient, and the completion of a prerequisite activity (e.g., prepare patient) could be defined as a milestone. If an exceptional situation occurs during the execution of this activity (e.g., detection of too high blood pressure) the subsequent surgery could not be launched.

The influence factors identified in this paper stem from analyzing different process scenarios from the logistics area, the medical domain, and the automotive sector. We formalize these influence factors in order to draw precise conclusions afterwards and follow a two–step approach for suggesting forward recovery strategies to the user. Within the first step possible dynamic changes of the concerned process instance are figured out depending on the identified influence factors. In order to improve the quality of the suggested recovery solutions, in a second step we check whether the ad–hoc changes are reasonably applicable, i.e., whether they preserve certain constraints or not (e.g., a given time schedule).

Sect. 2 provides necessary background information. In Sect. 3 we formalize factors influencing the choice of adequate recovery solutions in case of activity failures. Methods for deriving forward recovery solutions and evaluating their quality afterwards are provided in Sect. 4. In Sect. 5 we discuss related work and we close with a summary in Sect. 6.

2 FUNDAMENTALS

To provide a formal foundation for our further considerations we first present basic definitions. We use Activity Nets (Leymann and Altenhuber, 1994) as process meta model, but our considerations can be transferred to other process meta models as well. We extend the defintion of Activity Nets by subdividing process activities (e.g., `Prepare patient` for a surgery, cf. Fig. 1) into smaller work units which we call *atomic steps* (e.g., measuring weight/blood pressure of a patient as atomic steps of activity `Prepare patient`). We use this distinction for further considerations but it also reflects that work units may be modeled at different levels of granularity.

Definition 1 (Process Type Schema) *A tuple S = (N, D, CtrlE, DataE, EC, ST, Asn, Aso, $DataE_{extended}$) is called a process type schema with:*

- *N is a set of activities and D is a set of process data elements*
- *CtrlE $\subset N \times N$ is a precedence relation (notation: $n_{src} \rightarrow n_{dest} \equiv (n_{src}, n_{dest}) \in$ CtrlE)*
- *DataE $\subseteq N \times D \times$ NAccessMode is a set of data links between activities and data elements (with NAccessMode = {read, write, continuous-read, continuous-write})*
- *EC: CtrE $\mapsto$ Conds(D) assigns to each control edge an optional transition condition where Conds(D) denotes the set of transition conditions on data elements of D*
- *ST is the total set of atomic steps defined for all activities of the process; we define the following functions on ST:*
  - *Asn: ST $\mapsto$ N assigns each atomic step to an activity.*
  - *Aso: ST $\mapsto \mathbb{N}$ assigns to each atomic step a number indicating its relative position with respect to other atomic steps of a certain activity.*
- *$DataE_{extended} \subseteq$ ST $\times$ D $\times$ {read, write} is a set of data links between atomic steps and data elements*

Let us analyze the important issue of process data (flow) in detail. A first step is to distinguish between data flow on macro and micro level (cf. Fig. 1), i.e., data flowing between process activities (macro level) and between the atomic steps of one activity (micro level). We extend this discussion later on. At this level, we introduce the notion of process instances.

Definition 2 (Process Instance) *A process instance I is defined by a tuple (S, M^S, Val^S) where:*

- *S = (N, D, CtrlE, . . .) denotes the process type schema I was derived from*
- *M^S = (NS^S, ES^S) describes activity/atomic step and edge markings of I:*
  $NS^S : (N \cup ST) \mapsto$ {`NotAct, Act, Run, Comp, Skipped`}
  $ES^S : CtrlE \mapsto$ {`Not_Signaled, True_Signaled, False_Signaled`}
- *Val^S denotes a function on D, formally: Val^S: D $\mapsto$ $Dom_D \cup$ {`Undef`}. It reflects for each data element d $\in$ D either its current value from domain Dom_D or the value `Undef` (if d has not been written yet).*

$\mathcal{I}_S$ denotes the set of all instances running according to S.

In Def. 2 we abstain from using an execution log for process instances. In ADEPT, for example, such an execution log captures start and end events as well as the data values written by the processed activities

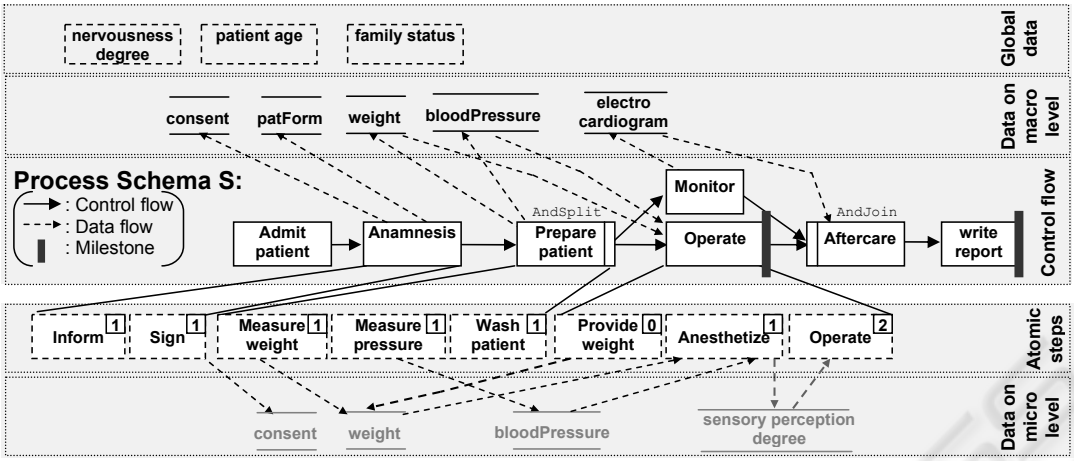

Figure 1: Medical Treatment Example.

(Rinderle et al., 2004). This constitutes important information in the context of backward recovery.

We give further definitions in order to be able to formalize the influence factors afterwards. A first important definition is that of *milestones*. Reaching a milestone indicates that the process execution has reached a certain stage, e.g., in the engineering domain a certain development stage. The notion of milestones can be also used for defining process goals, e.g., the surgery of the patient and the documentation of the medical treatment process (cf. Fig. 1). We define a milestone as a subset of activites of a process type schema with their associated activity markings.

Definition 3 (Milestone) *Let S = (N, ...) be a process type schema. Then ms = {(n, State) | n ∈ N' ⊆ N, State ∈* `{Act, Run, Comp}`*} denotes a milestone for S. The set of milestones defined on S is denoted by* MS_S.

When using data flow information for deriving recovery solutions it is necessary to distinguish between different kinds of data. For this purpose, a data classification schema has been introduced (Bassil et al., 2005), which puts the frequency of updating activity data and the relevance of these data into relation. In the latter dimension, a distinction is made between data elements only relevant in the context of the application and data elements relevant for process progress as well. A data element produced by any activity is relevant for the process if there is a subsequent activity reading this data element. If this is not the case we call this data element application–relevant. Furthermore, we refine this definition by also considering data elements that are not linked to any specific activity but to the overall process or to the application itself. We denote such data elements as "global data". An example is `nervousness degree` as depicted for the medical treatment process in Fig. 1.

The other dimension in the data classification schema is built by the data update frequency, i.e., how often a certain data element is updated during activity execution. A distinction is done between a discrete and a continuous data update of a data element d by an activity n. Examples for continuously updated data elements are `current position` and `container temperature` for the transportation process in Fig. 2. By contrast, in Fig. 1, `consent` is an example for a discretely updated data element. This classification only applies when a write or a continuous-write data link exists between d and n.

Informally, for a discrete data update there are certain time spans between the single updates, whereas for continuous data updates the time slices between the single updates converge to 0. The following function specifies time slices between data updates: $stp : ST \mapsto \mathbb{R}^+ \cup \{\texttt{Undef}\}$ where stp maps each atomic step either to a specific point in time or to `Undef`. As introduced in (Bassil et al., 2005) we further distinguish between a *discrete data update* of data element d by activtiy n (if $\exists(n, d, write) \in DataE$) and a *continuous data update* of d by n (if $\exists(n, d,$ continuous-write$) \in DataE$).

In order to correctly deal with exceptional situations, it is crucial to know the points in time when the data context of an activity becomes preserved (i.e., it is semantically meaningful and it is made persistent). In order to adequately answer this question, it is important to consider the distinction between continuous and discrete data updates. The following definition precisely determines the particular safe interrupt points for discrete and continuous data updates, i.e., those points in time when the respective data are updated such that subsequent activities reading these data can be correctly supplied with input data.

Definition 4 (Safe Interrupt Point for Data Update) *Let S be a process type schema, let (d, n) (n ∈ N,*

d ∈ D) be a data update, and let ST_n^d be the set of atomic steps associated with n and writing d; i.e., ST_n^d := {st | Asn(st) = n}.

- *Discrete Data Update: Let B := {stp(st) | st ∈ ST_n^d}. Then the safe interrupt point t_{safe}^d of (d, n) corresponds to the maximum point in time any atomic step writes d. Formally:*

$$t_{safe}^d := \begin{cases} max(B) & : \ B \neq \emptyset \\ \texttt{Undef} & : \ otherwise \end{cases}$$

- *Continuous Data Update: Let t_1 be the start updating time and t_k the finish updating time of d by n. Then the safe interrupt point t_{safe}^d of (d, n) ($t_1 < t_{safe}^d < t_k$) corresponds to the time when d becomes relevant for subsequent activities. This time is fixed by the user. If no safe interrupt point is fixed by the user t_{safe}^d =* `Undef` *holds.*

Informally, the safe interrupt point for a discrete data update by atomic steps is the maximum point in time when the last write access to the respective data element has taken place. Since there is no "natural" safe interrupt point for continuous data updates (e.g., the GPS continuously updating the vehicle position) we offer the possibility to define a safe interrupt point by the user. Based on the safe interrupt points for data updates the safe point of an activity can be defined as follows:

Definition 5 (Activity Safe Point) *Let S = (N, D, NT, ...) be a process type schema and let I = (S, M^S, Val^S) ∈ $\mathcal{I}_S$ be a process instance. Let further {d_1, ..., d_k} be the set of data elements (continuously) written by activity n ∈ N (i.e., ∃ (n, d_i, w) ∈ DataE, i = 1, ..., k, w ∈ {write, continuous-write}) and let $t_{safe}^{d_1}, \ldots, t_{safe}^{d_k}$ be the related safe interrupt points. Then we denote the activity safe point of n with t_{safe}^n. Formally:*

$$t_{safe}^n = \begin{cases} \texttt{Undef} & t_{safe}^{d_i} = \texttt{Undef} \ \forall i = 1, .., k \\ max\{t_{safe}^{d_1}, \ldots, t_{safe}^{d_k}\} & otherwise \end{cases}$$

3 ON FORMALIZING INFLUENCE FACTORS

In this section we formalize factors that may influence the selection of recovery solutions coping with process activity failures. In order to elaborate these factors we have systematically studied real–world business processes from different application environments (Dadam et al., 2000; Bassil et al., 2004).

Influence Factor 1 (Activity Context) If a running activity is interrupted an important question is whether we can preserve the (data) context of this activity or not. To be able to do so we distinguish between different kinds of data, e.g., data written in a continuous or in a discrete way.

Example 1 *If a* `move vehicle` *activity is interrupted due to a technical problem, it is to be ensured that the current position of the vehicle is known in order to, for example, send a replacement vehicle. This can be achieved using a GPS system which continously writes data element* `position` *(cf. Fig. 2).*

In Def. 5 for an activity n we have specified the concept of its safe point t_{safe}^n. Intuitively, the data context of a failed activity n could be preserved (i.e., all relevant output data of n have been already written) either if n was interrupted after having reached its safe point t_{safe}^n or if n has no write access to any data element. (The latter is expressed by t_{safe}^n = `Undef`.) If the data context of an activity n is preserved all input data of subsequent activities, that are supposed to read data elements written by n, are actually provided by n. This influences the set of applicable recovery solutions. Formally:

Definition 6 (Activity Context aCP) *Let S = (N, D, ...) be a process type schema and let I = (S, M^S, Val^S) ∈ $\mathcal{I}_S$ be a process instance. Assume that activity n ∈ N (with safe point t_{safe}^n) is interrupted at time t. Then aCP indicates whether the data context of n can be preserved or not. Formally: aCP: $\mathcal{I} \times \mathcal{N} \times \mathbb{R} \mapsto$ {*`T, F`*}*

$$aCP(I, n, t) := \begin{cases} \texttt{T} & t \geq t_{safe}^n \vee t_{safe}^n = \texttt{Undef} \\ \texttt{F} & otherwise \end{cases}$$

Knowing whether the context of an interrupted activity n is preserved or not influences the choice of the adequate recovery solution to a great extent. If the activity context is not preserved the respective recovery solution must at least deal with the successors of n which are not correctly supplied with input data.

Influence Factor 2 (Process Data Context) The process data context of a process instance comprises all data elements and their current values provided so far. Evaluating the process data context yields useful information when looking for an adequate recovery strategy as the following example shows.

Example 2 *Assume that the* `Prepare patient` *activity is supposed to measure, among other things, the blood pressure (cf. Fig. 1). If it is too high the subsequent surgery cannot be carried out, i.e., activity* `Prepare Patient` *will be interrupted. One recovery solution would be to skip the surgery. However, taking into account the process data context, in particular the global data element* `nervousness degree` *shows that the patient is extremely nervous at the moment. Therefore another recovery solution would be to "wait till the patient relaxes".*

Definition 7 (Process Data Context pDC) *Let S = (N, D, NT, ...) be a process type schema and let I = (S,*

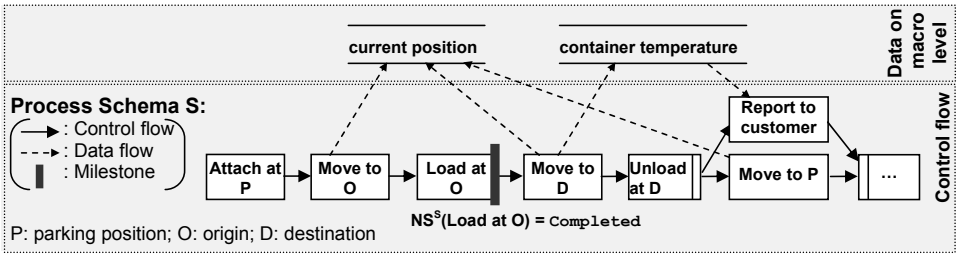

Figure 2: Container Transportation Example.

$M^S, Val^S) \in \mathcal{I}_S$ *be a process instance. Then pDC determines all data element values written by I so far. Formally:*

pDC: $\mathcal{I}_S \mapsto (D \times (Dom_D \cup \{\texttt{Undef}\}))^{\mathcal{P}}$ *with*

$pDC(I) := \{(d, Val^S(d)) \mid d \in D\}$

Influence Factor 3 (State Context) The applicability of recovery strategies also depends on how far process execution has progressed. This can be expressed, for example, by defining milestones (cf. Def. 3). If a certain milestone is passed we may have to apply different recovery solutions than before.

Definition 8 (State Context statCtxt) *Let S be a process type schema, let* MS_S *be the set of milestones defined on S, and let I = (S,* $M^S = (NS^S, ES^S), Val^S) \in \mathcal{I}_S$ *be a process instance running on S. Then statCtxt determines all milestones reached by the execution of I so far. Formally: statCtxt:* $\mathcal{I} \mapsto (MS_S)^{\mathcal{P}}$

statCtxt(I) := $\{ms = \{(n_1, State_1), ..., (n_k, State_k)\} \in MS_S \mid (NS^S(n_i) = State_i) \vee (State_i \in \{\texttt{Act, Run, Comp}\} \Longrightarrow NS^S(n_i) \in \{\texttt{Run, Comp, Skipped}\})\}$*(♣)*

The second part of condition ♣ captures the following situations: if the activity marking associated with a milestone activity n equals `Act` or `Run` the milestone is reached if $NS^S(n)$ equals `Act, Run`, or `Comp`. If a milestone is defined based on an activity within an alternative branching the milestone may be never reached. In this paper we handle this case by agreeing on that a milestone is also reached when the respective activity is skipped. Of course, other conventions are conceivable as well.

Example 3 *Consider the transportation process depicted in Fig. 2. Milestone* ms_{trans} *=* $\{$`(Load at O, Comp)`$\}$ *has been defined for this process. If an instance I has not yet reached this state, i.e.,* NS^S`(Load at O)` $\neq$ `Comp` *and an exception occurs we can still load the goods to another container/vehicle. By contrast, if the goods have been already loaded, i.e.,* NS^S`(Load at O)` = `Comp` *we have to additionally insert an* `Unload` *activity.*

Influence Factor 4 (Process Goal) The choice of an adequate forward recovery solution strongly depends on the goal of the respective process. Thereby several goals with different priorities may exist. In practice, process goals often correspond to reaching a certain milestone, e.g., a development stage within an automotive engineering process. Therefore, in this paper, we define process goals as the milestones not yet reached by instance execution so far. Milestones which refer to activities from alternative branches and may therefore be not reached are not considered as process goals in this context (cf. Def. 8). There may be other possibilities to define process goals, e.g., based on the availability of process data. We will extend our considerations in this direction in the future.

Definition 9 (Process Goal Activities prGoalAct) *Let S = (N, ...) be a process type schema, let* MS_S *be the set of milestones defined for S, and let I = (S,* $M^S = (NS^S, ES^S), Val^S) \in \mathcal{I}_S$ *be a process instance running on S. Let further statCtxt(I) be the state context of I (cf. Def. 8). We define process goals as those milestones which have not been reached during execution of I so far. prGoalAct determines the activities associated with these process goals:*

prGoalAct: $\mathcal{I}_S \mapsto (N)^{\mathcal{P}}$

prGoalAct(I) := $\{n \in N \mid \exists\, ms \in (MS_S \setminus statCtxt(I))$ *with* $\exists\, (n, State) \in ms\}$

Example 4 *Consider the medical treatment process depicted in Fig. 1. The milestones for this process are* ms^1_{med} *=* $\{$`(Operate, Comp)`$\}$ *and* ms^2_{med} *=* $\{$`(write report, Comp)`$\}$*. Assume that the execution of process instance I fails during* `Prepare Patient`*. Then both milestones have not been reached so far and therefore process goal activities are* `Prepare Patient` *and* `write report` *for instance I in the current situation.*

There may be further influence factors depending on the particular application. However, based on the factors mentioned above different forward recovery solutions can be already suggested.

4 FORWARD RECOVERY STRATEGIES

In this section we provide a first contribution towards the (automatic) derivation of forward recovery solutions taking into consideration the described influence factors. We proceed in two steps: Firstly we derive propositions for recovery solutions. Secondly, we evaluate them with respect to certain criteria.

4.1 Semantic Recovery Solutions

As motivated in the introduction, in many cases forward recovery can be based on dynamic changes of the concerned process instance. Table 1 presents a selection of respective change operations that can be used in this context.

Table 1: *Selection of Process Change Operations.*

Change Operation Δ Applied to Schema S	**Effects on Schema S**
interruptAct(S, X)	interrupts execution of activity X
insertAct(S, X, S, B)	inserts activity X between activity sets A, B
deleteAct(S, X)	deletes activity X from schema S
jumpTo(S, t)	jumps from currently executed activity to t
addDataEdges(S, DE)	adds set of data edges DE

When an activity is interrupted it is by far not trivial to automatically determine an adequate set of correctly parameterized change operations. (Müller, 2004) proposes a rule–based approach for applying recovery strategies at the occurence of exceptional situations. However, this does not completely consider the information we have about the affected instance, e.g., regarding its data context or reached milestones.

In the following we provide an approach based on which this set can be restricted. Our suggestions are based on the influence factors discussed before.

Suggestion 1 (aCP): Based on the activity context (cf. Def. 6) we can draw conclusions about the change operations applicable for an instance I. If $aCP(I, n) = 1$ holds the context of n is preserved. Consequently, no process relevant data will be lost when interrupting n, i.e., no data flow errors will occur. Activity n can be safely interrupted and all subsequent activities can be executed as planned. By contrast, if $aCP(I, n) = 0$ holds, the input parameters of subsequent activities may not be correctly supplied. Then recovery solutions may either bypass these activities (by applying jump operations) or insert activities providing the required data.

Suggestion 2 (prGoalAct): Regarding factor prGoalAct (cf. Def. 9) it is important to know whether process goal activities can be still executed. This, in turn, depends on whether their input data can be correctly supplied or not by preceding activities. In this context, we have to consider the successors of failed activity n (including n). From this we determine (possibly transitively) which activities provide input data for process goal actitivities. They can be correctly provided with input data if the set of activities providing this data does not contain n or if the activity context of n can be preserved, formally:

Definition 10 (Provision of Process Goal Activities) *Let S = (N, D, ...) be a process type schema and let $I \in \mathcal{I}_S$ be a process instance. Assume that activity $n \in N$ fails. Then function pVG denotes whether a goal activity $pG \in$ prGoalAct(I) is correctly provided with input data or not. Formally:*

pVG: $\mathcal{I}_S \times N \times N \mapsto \{\texttt{T}, \texttt{F}\}$

$$pVG(I, n, pG) = \begin{cases} \texttt{T} & (n \notin GPA(n, pG) \vee aCP(I, n) = 1) \\ & \vee WE(S_n, pG) = \texttt{T} \\ \texttt{F} & otherwise \end{cases}$$

where

- *GPA(n, pG) = {m ∈ N | m ∈ (succ*(S, n) ∪ {n}, ∃ (pG, d, [read|continuous-read]), (m, d, [write|continuous-write] ∈ DataE, d ∈ D} ∪ GPA(n, m)*
- *succ*(S, n): all direct and indirect successors of n in S*
- *WE(S, n) determines whether activity n of schema S is correctly supplied with input data or not (Reichert, 2000).*
- *S_n denotes the induced subgraph of S when deleting activity n and all associated data read and data write edges*

If pVG(I, n, pG) = T we know that the process goal connected with activity pG can be achieved, i.e., no change operations have to be applied for reaching this goal. Otherwise, if we obtain pVG(I, n, pG) = F for failed activity n and process goal activity pG a possible solution would be to insert one or more activities properly providing the missing input data.

Example 5 *Consider the process depicted in Fig. 1. Assume that for process instance $I_{med} \in \mathcal{I}_S$, during the execution of its activity* `Prepare Patient`*, an exception occurs (e.g., assume that the patient's blood pressure cannot be taken due to an instrumental error). Assume further that influence factors aCP and prGoalAct turn out as follows:*

- *aCP(I_{med},* `Prepare Patient`*) =* `F`
- *prGoalAct(I_{med}) =* {`Operate, Write report`}

The set of activities providing input data for `Operate` *then contains one element, namely the failed activity* `Prepare Patient` *for which the activity context cannot be preserved, i.e., pVG(I_{med},* `Prepare Patient, Operate`*) =* `F`*. A first solution to treat this exception would be to insert another activity* `TakePressure` *and to retry measuring the blood pressure in order to reach the first process goal. In this context the application of the change operations* `insertAct(S, Take Pressure, {Prepare Patient}, {Operate})` *and* `addDataEdges(S, {(TakePressure, bloodPressure, write), (Operate, bloodPressure, read)})` *can be suggested to users. If this is not possible we may want to bypass goal activity* `Operate` *and to achieve the other process goal activity* `write report` *instead. Process goal* `write report` *has no input parameters, i.e., pVG(I_{med},* `Prepare Patient, write report`*) =* `T` *what implies that* `Write report` *can be executed in any case. Therefore a possible solution would be to suggest change* {`jumpTo(S, writeReport)`}.

There are situations where the activity context may be preserved and process goals can be reached at first

sight. However, when analyzing the application context as well the process goal cannot be always correctly reached any longer:

Example 6 *Consider again the process depicted in Fig. 1. Assume that for instance $I_{med} \in \mathcal{I}_S$ during execution of activity* `Prepare Patient` *an exception occurs, e.g., the patient's blood pressure could be too high for carrying out the subsequent surgery. Assume further that influence factors aCP, pDC, and prGoalAct turn out as follows:*

- *aCP(I_{med},* `Prepare Patient`*)* = `T`
- *pDC(I_{med})* = {`..(nervousness degree, 5), (bloodPressure, (180,100))`}
- *pVG(I_{med},* `Prepare Patient, Operate`*)* = `T`

Though pVG(I_{med}, `Prepare Patient, Operate`*) =* `T` *holds the application context (value of blood pressure exceeds the limit) shows that we cannot reach this goal without exception handling. Analyzing the application context also reveals that the patient is very nervous (value of his* `nervousness degree` *is very high). One possible recovery solution would be to wait until the patient relaxes. This could be achieved, for example, by inserting activity* `waitMeasure` *between* `Prepare Patient` *and* {`Monitor, Operate`} *accompanied by newly inserted data edges* `(waitMeasure, nervousness degree, read)` *and* `(waitMeasure, bloodPressure, write)`.

Analyzing the application context factor may depend on the application semantics, e.g., the blood pressure being too high. Thus the above recovery solution can be only figured out semi–automatically. An automatic treatment would become possible if the user had specified any limit for the values related to blood pressure and nervousness degree and deposits a respective rule within the system.

Finally, we want to analyze the impact of influence factor state context on finding an adequate recovery solution. Consider the following example:

Example 7 *Consider the process depicted in Fig 2. Assume that a container was loaded (i.e., reached milestone). In case of a problem the proposed solution would be different from the situation where the container is still unloaded. Indeed, in the latter case, we may send the vehicle back to the parking position P and cancel the processing of the customer request. In the former case, it is an obligation to deliver the merchandise. Taking into account a defined transportation network, each of the vehicle positions is captured by a coordinate (x, y). Assume for instance $I_{trans1} \in \mathcal{I}_S$ that during the execution of activity* `Move to D`*, a road traffic problem occurs, i.e.,* `Move to D` *is interrupted at position (7, 5.5) measured by the GPS system. Let the influence factors be as follows:*

- *aCP(I_{trans1},* `Move(1.5, 3.5)` $\rightarrow$ `(13, 8)`*)* = 1
- *pDC(I_{trans1})* = {*(*`current position`*, (7, 5.5))*}
- *statCtxt(I_{trans1})* = {*(*`Load at O, Comp`*)*}

Evaluating the influence factors, in particular the current position of the vehicle, the traffic problem can be avoided by changing the already planned route leading to the destination location. The new solution includes a detour via another location at position (7, 7). This recovery solution can be expressed by applying the following dynamic change operations: {{`insertAct(S, Move to (7, 7), {Move to (13, 8)}, {unloadAt(13, 8)}), insertAct(S, Move to (13, 8), {Move to (7, 7)}, {UnloadAt(13, 8)})`}}

Of course, the discussion of different influence factors especially in conjunction with the definition and evaluation of rules as proposed in (Müller et al., 2004) is to be extended. Due to lack of space we abstain from further details here.

4.2 Evaluating Recovery Solutions

In the previous section we derived forward recovery solutions, i.e., sets of conceivable process instance changes, by considering certain influence factors. However, some of the suggested ad–hoc changes may not be applicable, since they may offend against other constraints (e.g., they may violate a given time schedule). Thus, the set of forward recovery solutions presented to the user should be further restricted by checking their applicability at the presence of these constraints. In this paper we exemplarily discuss time and resource constraints.

Temporal Context: The applicability of a particular forward recovery solution may depend on time constraints. For example, assume that our solution implies the insertion of a new activity. The duration to process this activity may lead to a violation of the intended process end time.

Resource Context: Specific resources (e.g., vehicles, containers, drivers) are often associated with a specific application (e.g., container transportation) (Bassil et al., 2004). Those resources enable the accomplishment of the process activities (e.g., "attach a container to a vehicle", "move a container"). Resource availability should be taken into account when checking the proposed recovery solutions.

Checking the validity of both temporal and resource constraints when applying dynamic workflow changes is a very challenging issue. However, there have been only few approaches regarding this issue so far (Sadiq et al., 2000). We have been investigating the interdependencies between processes and temporal as well as resource constraints in detail. Due to lack of space we abstain from further details here.

5 RELATED WORK

Related work can be distinguished into approaches for backward and forward recovery. Backward recovery strategies require to interrupt a running instance and to (partially) conduct a rollback. As discussed in

the introduction, advanced transactional concepts like nested transactions and semantical rollback have been elaborated (Elmagarmid, 1992). A detailed discussion of transactions applied in the domain of workflows is given in (Worah and Sheth, 1997). Approaches in this context are Sagas (Garcia-Molina and Salem, 1987) and Spheres of Compensation (Leymann, 1995; Leymann and Roller, 2000). The concept of spheres has been defined such that compensation can be applied not only on one activity but also on a group of activities (sphere, scope).

Less attention has been paid to the (automatic) derivation of forward recovery strategies. Müller et al. (Müller, 2004; Müller et al., 2004) propose to represent recovery strategies by rules. They use a combination of F-Logic and Transaction Logic in order to formalize the interaction between rules and process instances. If an exception occurs and if a respective rule is specified recovery solutions can be automatically executed. Based on rules the system behavior in the case of an exception is always hard–wired within the rule neglecting the context of the respective process instance.

Regarding the quality improvement of the suggested recovery solutions temporal and resource constraints were dicussed within the paper. There are approaches to specify time within process management systems, e.g., (Eder and Pichler, 2002; Sadiq et al., 2000). However, we will check the applicability of these approaches for our purposes.

6 SUMMARY AND OUTLOOK

In this paper we have elaborated factors which may influence the choice of an adequate forward recovery solution. These factors comprise the safe interruption of activities preserving the data context, the values of the data elements written so far, certain milestones within the process, and process goals. All of these influence factors help the system to propose reasonable forward recovery solutions to users. We have discussed further aspects and constraints which have to be checked in order to improve the quality of the proposed solutions. All results presented in this paper stem from deep analysis of application scenarios. Currently, basic concepts are implemented in an advanced prototype. In the future we will search for further factors influencing potential recovery solutions. We will also focus on the quality checking process, i.e., we want to elaborate adequate representations for temporal and resource constraints in order to formally verify the applicability of recovery solutions.

REFERENCES

Bassil, S., Keller, R., and Kropf, P. (2004). A workflow–oriented system architecture for the management of container transportation. In *BPM'04*, pages 116–131.

Bassil, S., Rinderle, S., Keller, R., Kropf, P., and Reichert, M. (2005). Preserving the context of interrupted business process activities. In *Proc. ICEIS'05*, pages 38–45.

Dadam, P., Reichert, M., and Kuhn, K. (2000). Clinical workflows - the killer application for process-oriented information systems? In *BIS'00*, pages 36–59.

Eder, J. and Pichler, H. (2002). Duration histograms for workflow systems. In *Proc. Conf. EISIC'02*, pages 25–27, Kanazawa, Japan.

Elmagarmid, A. (1992). *Database Transaction Models for Advanced Applications*. Morgan Kaufman.

Garcia-Molina, H. and Salem, K. (1987). Sagas. In *Proc. ACM SIGMOD Int'l Conf. on Management of Data*, pages 249–259, San Francisco, CA.

Leymann, F. (1995). Supporting business transactions via portal backward recovery in workflow management systems. In *BTW'95*, pages 51–70, Dresden.

Leymann, F. and Altenhuber, W. (1994). Managing business processes as an information ressource. *IBM Systems Journal*, 33(2):326–348.

Leymann, F. and Roller, D. (2000). *Production Workflow*. Prentice Hall.

Mourao, H. and Antunes, P. (2004). Exception handling through a workflow. In *CoopIS'04*, pages 37–54.

Müller, R. (2004). *Event–Oriented Dynamic Adaptation of Workflows: Model, Architecture and Implementation.* PhD thesis, University of Leipzig, Germany.

Müller, R., Greiner, U., and Rahm, E. (2004). AGENTWORK: A workflow-system supporting rule-based workflow adaptation. *DKE*, 51(2):223–256.

Reichert, M. (2000). *Dynamic Changes in Workflow-Management-Systemen.* PhD thesis, University of Ulm, Computer Science Faculty. (in German).

Reichert, M., Dadam, P., and Bauer, T. (2003). Dealing with forward and backward jumps in workflow management systems. *Int'l Journal SOSYM*, 2(1):37–58.

Rinderle, S., Reichert, M., and Dadam, P. (2004). Flexible support of team processes by adaptive workflow systems. *DPD*, 16(1):91–116.

Sadiq, S., Marjanovic, O., and Orlowska, M. (2000). Managing change and time in dynamic workflow processes. *IJCIS*, 9(1&2):93–116.

Strong, D. and Miller, S. (1995). Exceptions and exception handling in computerized information processes. *ACM–TOIS*, 13(2):206–233.

v.d. Aalst, W. and van Hee, K. (2002). *Workflow Management*. MIT Press.

Worah, D. and Sheth, A. (1997). *Advanced Transactional Models and Architectures*, pages 3–34. Kluwer Academic Publishers.