Published in Enterprise Information Systems VII, part 3, 149-156, 2005
which should be used for any reference to this work

PRESERVING THE CONTEXT OF INTERRUPTED BUSINESS PROCESS ACTIVITIES

Sarita Bassil[1], Stefanie Rinderle[2], Rudolf Keller[3], Peter Kropf[4], and Manfred Reichert[5]

[1]*DIRO, University of Montreal, C.P. 6128, succ. Centre-ville, Montreal, Quebec, H3C 3J7, Canada*
bassil@iro.umontreal.ca
[2]*DBIS, Faculty of Computer Science, University of Ulm, Germany, rinderle@informatik.uni-ulm.de*
[3]*Zühlke Engineering AG, Schlieren, Switzerland, ruk@zuehlke.com*
[4]*Institute of Computer Science, University of Neuchatel, Switzerland, peter.kropf@unine.ch*
[5]*Information Systems Group, University of Twente, The Netherlands, m.u.reichert@cs.utwente.nl*

Keywords: Information systems, business processes, flexibility, data analysis, B2B and B2C applications.

Abstract: The capability to safely interrupt business process activities is an important requirement for advanced process-aware information systems. Indeed, exceptions stemming from the application environment often appear while one or more application-related process activities are running. Safely interrupting an activity consists of preserving its context, i.e., saving the data associated with this activity. This is important since possible solutions for an exceptional situation are often based on the current data context of the interrupted activity. In this paper, a data classification scheme based on data relevance and on data update frequency is proposed and discussed with respect to two different real-world applications. Taking into account this classification, a correctness criterion for interrupting running activities while preserving their context is proposed and analyzed.

1 INTRODUCTION

To stay competitive in the market, companies must be able to rapidly react to changing situations and to align their business processes accordingly (Reichert et al., 2003). In particular, e–business needs a powerful infrastructure to isolate process logic from application code (Gartner Group, 1999), and to define, control, and monitor business processes. *Process–Aware Information Systems (PAIS)* offer a promising perspective in this context (v.d. Aalst and van Hee, 2002). They aim to connect *activities*, i.e., pieces of work to perform a task, in order to achieve a common goal (Workflow Management Coalition, 1999).

However, today's companies need to maintain a satisfying level of agility. It appears that agile PAIS are the ones that provide, among other things, an appropriate and a competent way to cope with changing situations and unexpected events. This, in turn, is of particular importance for adequately supporting long-running, distributed business processes.

From this perspective, transportation companies for instance must adopt solutions where a close follow-up of activities is possible such that a customer request is well satisfied. An example of a transportation activity is "move vehicle V from origin location O to destination location D". A close follow-up of this activity can be achieved using GPS (Global Positioning System) which enables to continuously calculate and provide the position of a vehicle in movement.

Moreover, the occurrence of unexpected problems during transportation cannot be avoided. Indeed, there is ample evidence that fleet management at the operational level (e.g., scheduling of transportation activities) is highly dynamic in the sense that ongoing transportation activity sequences require a high degree of adaptation to deal with unexpected problems (Bassil et al., 2003). As an example, technical problems of vehicles, traffic jams or forced rerouting may appear at any time while V is on the road between O and D. This usually leads to the interruption of the "move V from O to D" activity. In such a situation, a dynamic adaptation of an already planned flow of activities for the satisfaction of a customer request is needed. This adaptation should take into account the current context of the interrupted activity. The new transportation solution may propose to send a new vehicle V' to the current position of V or to change the already planned route leading to D. In both cases, the current position of V should be available such that an appropriate new solution can be proposed.

In this paper, we focus on interrupted (business) process activities that require context preservation. In most cases, activity interruption is triggered by the appearance of unexpected events coming from the application environment (i.e., *semantic failures*). Preserving the context of an interrupted activity consists of saving data, which are produced by or associated with this activity. This must be done at the right time, e.g., as soon as the data become available or relevant.

At this point, it is important to have a closer look at the granularity of work unit descriptions. Usually, a business process consists of a set of activities each of them dealing with a logical task (e.g., preparing a patient for a surgery). In addition, such a process activity can be further subdivided into *atomic steps* corresponding to basic working units (e.g., measuring weight/temperature of a patient as atomic steps of activity "prepare patient") or to data provision services. Basic working units are either directly coded within application programs or worked on manually by people. Distinguishing between activities and atomic steps is useful for the following reasons: Atomic steps are not managed within worklists like activities are. This contributes to better system performance since the costs for managing and updating worklists decrease. Furthermore, this approach offers more flexibility to users (if desired) since they can choose the order in which they want to work on atomic steps. The distinction between activities and atomic steps finally leads to the following basic considerations.

It is very important in this context to distinguish between a *continuous* and a *discrete data update by activities*. The "move V from O to D" activity introduced above is an example of an activity continuously updating the "V current position" data element by a GPS system. An example of an activity discretely updating data is even more obvious in process-oriented applications. We may think about the activity "fill in a form" with many sections, each one asking for information (i.e., data) related to a specific topic. The information becomes relevant, and therefore may be kept in the system, only after the completion of a specific section. Filling in a section could be seen as working on a particular *atomic step*.

We highlight the fact that a process activity may apply both updating kinds: it may discretely update a particular data element d_1 and continuously update another data element d_2. Moreover, data elements may be discretely updated by a specific activity n_1 and be continuously updated by another activity n_2. As an example, activity "monitor patient" in a medical treatment process, may ask to measure twice a day the "patient temperature" and to continuously control the "patient heart electric signals". On the other hand, the "patient temperature" may be continuously controlled in case of high fever within activity "monitor patient" while it may be measured twice a day after operation within activity "aftercare".

Data continuously or discretely updated by activities may be only relevant for the specifically studied application (e.g., the vehicle "current position" in Fig. 3) or they may be relevant for process execution as well; in the latter case, these data are consumed by process activities and therefore have to be supplied by preceding activities. At the occurence of exceptional situations, it may appear that mandatory process relevant data will not be available at the time an activity is invoked. Depending on the application context and the kind of data, it may be possible to provide the missing data by *data provision services* which are to be executed before the task associated with the respective activity is handled.

We distinguish between *exclusive application data* and *process relevant data*. Note that exclusive application data may become process relevant when a failure occurs. In the transportation application, an example of process relevant data would be the "container temperature" (continuously) measured during a "move V from O to D" activity and relevant for a "Report to customer" activity within the same process. Reporting on the container temperature would inform the customer whether the transported goods (e.g., foods) were or were not continuously preserved under the appropriate temperature. The "V current position" is an example of exclusive application data since it is relevant for the application, in particular for the optimisation module of the application (Bassil et al., 2004), but not for the business process management system. If, however, a road traffic problem occurs, the "current position" of V may become relevant for the process as well; i.e., the origin location O' of a newly proposed activity "move V from O' to D" changing the already planned route leading to D, would correspond to "current position" of V.

Figure 1 shows a data classification scheme in the context of business processes. This classification puts the frequency of updating activity data and the relevance of these data into relation. Within these two dimensions, we respectively differentiate between:

- continuously and discretely updated data, and
- exclusive application and process relevant data.

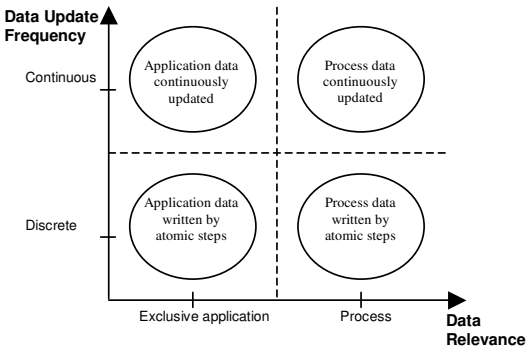

Figure 1: Data Classification Scheme

Taking into account this classification, and knowing that exceptions stemming from the application environment cannot be avoided and generally appear during activity performance, it would be a challenge not to loose available data already produced by the activity that will be inevitably interrupted or deleted. In

order to formally specify the correctness criterion for interrupting running activities while preserving their context, formal definitions of requisite foundation for this specification are indispensable.

The remainder of this paper is organized as follows: In Section 2 we define such foundation; we also discuss two application processes (a medical process and a transportation process) with respect to the provided definitions. Then, Section 3 introduces a general correctness criterion ensuring a safe interruption of a running activity. Section 4 discusses related work and Section 5 concludes the paper.

2 FORMAL FRAMEWORK

To be able to precisely define the different kinds of data and update frequencies we need a formal process meta model. In this paper, we use the established formalism of Well–Structured Marking Nets (WSM Nets) (Rinderle et al., 2004b) and extend it for our purposes. Informally, a WSM Net is a serial–parallel, attributed process graph describing control and data flow of a business process. More precisely, different node and edge types are provided for modeling control structures like sequences, branchings, and loops. A simple example is depicted in Fig. 2. Here, the upper two lanes show the control and data flow of a (simplified) medical treatment process. For example, activities "admit patient", "inform patient", and "prepare patient" are arranged in sequence whereas activities "monitor" and "operate" are executed in parallel. "Weight" and "temperature" are examples of process relevant data elements involved in a data flow between the activities "prepare patient" and "operate".

As motivated in the introduction an activity can be subdivided into a set of atomic steps. Going back to Fig. 2, the lower two lanes show the atomic steps assigned to the process activities as well as the data flow between these steps. For example, the atomic steps "measure weight", "measure temperature", and "wash patient" are assigned to activity "prepare patient". "Provide weight" is an example of a data provision service assigned to activity "operate" as atomic step. If an exceptional situation (e.g., failure at the "measure weight" atomic step level) occurs this data provision service will be invoked in order to supply input data element "weight" of the activity "operate" (and particularly of its atomic step "anesthetize"). We define a partial order relation on the set of atomic steps (incl. data provision services) assigned to a certain activity. The precedence relation depicts a micro control flow between elements of this set. Note that, by contrast, a macro control flow is defined between activities. We set up this relation by assigning numeric labels to atomic steps, e.g., an atomic step with numeric label "1" is considered as a predecessor of all atomic steps with numeric label "2" or greater. By default, all atomic steps have number "1", i.e., they can be worked on in parallel. In this case, the actor which works on the respective activity is considered as being the expert in choosing the best order. Data provision services have number "0" since they must be executed before other atomic steps assigned to the same activity, in order to properly supply these atomic steps with the required input data.

So far WSM Nets have not considered splitting activities into atomic steps. Therefore we extend the formal definition from (Rinderle et al., 2004b) by including this additional level of granularity. In the following, S describes a process schema.

Definition 1 (Extended WSM Net) *A tuple S = (N, D, NT, CtrlE, DataE, ST, P, Asn, Aso, $DataE_{extended}$) is called an extended WSM Net if the following holds:*

- *N is a set of activities and D is a set of process data elements*
- *NT: $N \mapsto$ {StartFlow, EndFlow, Activity, AndSplit, AndJoin, XorSplit, XorJoin, StartLoop, EndLoop} To each activity NT assigns a respective node type.*
- *CtrlE $\subset N \times N$ is a precedence relation setting out the order between activities.*
- *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})*
- *ST is the total set of atomic steps defined for all activities of the process (with P $\subseteq$ ST describing the set of data provision services)*
- *Asn: ST $\mapsto$ N assigns to each atomic step a respective activity.*
- *Aso: ST $\mapsto \mathbb{N}$ assigns to each atomic step a number indicating in which order the atomic steps of a certain activity are to be executed. By default: If $s \in P$, Aso(s) = 0 holds; otherwise, Aso(s) = 1.*
- *$DataE_{extended} \subseteq$ ST $\times$ D $\times$ STAccessMode is a set of data links between atomic steps and data elements (with STAccessMode = {read, write})*

As can be seen in the example from Fig. 2, there are atomic steps which produce data (e.g., "measure weight") and others which do not write any data element (e.g., "wash patient"). In order to express this fact, we logically extend the set $DataE$ to set $DataE_{extended}$ which comprises all read/write data links between atomic steps and data elements. In particular, an intra-activity data dependency may be defined such that intermediate results of an activity execution can be passed between subsequent atomic steps st_1 and st_2 with $Asn(st_1) = Asn(st_2)$; i.e., $\exists(st_1, d, write),\ (st_2, d, read) \in DataE_{extended}$. As an example (Fig. 2), consider the intra-activity data flow from "anesthetize" to "operate" via data element "sensory perception degree". In fact, the atomic

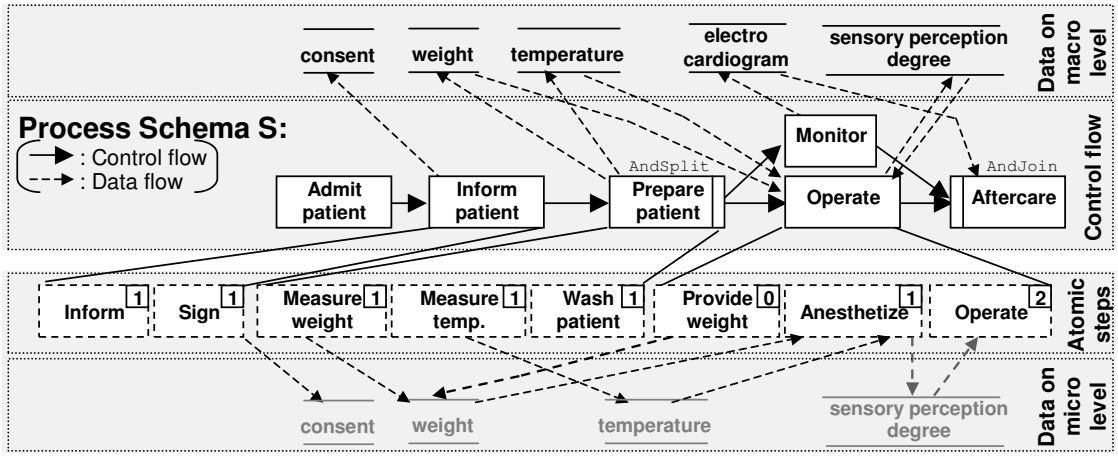

Figure 2: Medical Treatment Process

step "operate" needs this data element to decide when to begin surgery.

Based on Def. 1, *process instances* can be created and executed. As discussed in (Rinderle et al., 2004b), during runtime a process instance references the WSM Net it was created from. Its current execution state is reflected by model–inherent activity markings. An activity which can be worked on is thus labeled $Activated$. As soon as activity execution is started the marking changes to $Running$. Finally, a finished activity is marked as $Completed$ and an activity, which belongs to a non-selected, alternative execution branch, is marked as $Skipped$.

Definition 2 (Process Instance on Extended WSM Net) *A process instance I on an extended WSM Net S is defined by a tuple (S, $M^S_{extended}$, Val^S) where:*

- *S = (N, D, NT, CtrlE, ...) denotes the extended WSM Net I was derived from*
- *$M^S_{extended}$ = (NS^S, STS^S) describes activity and atomic step markings of I:*
  $NS^S : N \mapsto$ {NotActivated, Activated, Running, Completed, Skipped}
  $STS^S : ST \mapsto$ {NotActivated, Activated, Running, Completed, Skipped}
- *Val^S denotes a function on D. It reflects for each data element d ∈ D either its current value or the value Undefined (if d has not been written yet).*

Markings of activities and atomic steps are correlated. When an activity becomes activated, related atomic steps (with lowest number) become activated as well. The atomic steps will then be carried out according to the defined micro control flow. As soon as one of them is executed, both the state of this atomic step and of its corresponding activity change to $Running$. An activity is marked as $Completed$ after completion of all corresponding atomic steps. Finally, if an activity is skipped during process execution, all related atomic steps will be skipped as well.

As motivated in the introduction, it is important to distinguish between data elements only relevant in context of application and data elements relevant for process progress as well. We can see whether a data element is relevant for the process if there is an activity reading this data element.

Definition 3 (Data Relevance) *Let S be an extended WSM Net, let w ∈ {write, continuous-write} and r ∈ {read, continuous-read}. Then we denote d ∈ D as*

- *an exclusive application data element if*
  $\exists (n, d, w) \in DataE \Longrightarrow \nexists (m, d, r) \in DataE$
- *a process relevant data element if*
  $\exists (n, d, w) \in DataE \Longrightarrow$
  $\exists\, m \in Succ^*(S, n) \cup \{n\}$: $(m, d, r) \in DataE$

$Succ^*(S, n)$ denotes all direct and indirect successors of activity n.

The Data Relevance dimension captures both data elements that are produced by the process, but are only consumed by the application, and data elements that are produced and consumed by the process. In our medical treatment process (cf. Fig. 2), data elements "weight" and "temperature" taken during the "prepare patient" activity are examples of process relevant data elements. They are of utmost importance for carrying out the subsequent "operate" activity (e.g., to calculate the quantity of anesthesia that has to be administered to the patient). By contrast, "consent" is an exclusively application relevant data element. As explained in Section 1, when a failure occurs, an exclusive application data element may become relevant for the process as well. A patient who already consented upon a surgery accepts the risks, and the "consent" data element may thus be

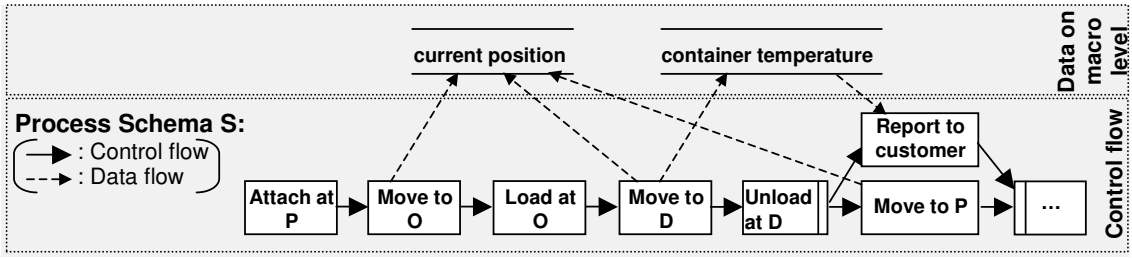

Figure 3: Container Transportation Process

used in subsequent activities dealing with respective problems. Turning now to the container transportation process, "current position" is an exclusive application data element whereas "container temperature" is a process relevant data element (cf. Fig. 3).

We now define the notion of data update frequency. Based on this notion we will be able to define a criterion for safely interrupting running activities while preserving their context. Intuitively, for a discrete data update by atomic steps there are certain periods of time between the single updates, whereas for continuous data updates by activities the *time slices* between the single updates converge to 0. For defining the *time slices* between data updates, we need the function $stp : ST \mapsto \mathbb{R} \cup \{Undefined\}$ which maps each atomic step of ST either to a specific point in time or to $Undefined$. In detail:

$$stp(st) := \begin{cases} t_{st} \;\; if \;\; \exists (st, d, write) \in DataE_{extended} \\ Undefined \; otherwise \end{cases}$$

$$whereby \; t_{st} := \begin{cases} completion \; time \; of \; st \\ \infty \; by \; default \end{cases}$$

Note that the infinite default value we assign to t_{st} is updated as soon as st is completed. Hence, the real completion time of st is assigned to t_{st}.

Definition 4 (Data Update Frequency) *Let S be an extended WSM Net, let $w \in \{write, continuous\text{-}write\} \subset NAccessMode$, and let $d \in D$, $n \in N$ with $(n, d, w) \in DataE$. Let further ST_n^d be the set of atomic steps associated with activity n and writing data element d; i.e., $ST_n^d := \{st \mid asn(st) = n, \exists (st, d, write) \in DataE_{extended}\}$. Then we denote (d, n) as:*

- *A discrete data update of d by n if*
  $\exists (n, d, write) \in DataE$
  In terms of atomic steps:
  $\forall st \in ST_n^d$: *stp(st)* = $t_{st} \neq$ *Undefined*
- *A continuous data update of d by n if*
  $\exists (n, d, continuous\text{-}write) \in DataE$
  In terms of atomic steps: $ST_n^d = \emptyset$

In case an activity n continuously updates a data element d no atomic steps writing d are dissociated, i.e., there are no atomic steps associated with n that write d; e.g., take the absence of atomic steps writing the "current position", the "container temperature", and the "electro cardiogram" in Figures 2 and 3. These data elements are examples of data continuously updated respectively by a GPS system, a thermometer, and a cardiograph instrument.

On the other hand, the set of atomic steps discretely writing a data element may be limited to only one atomic step. The "consent", the "weight", and the "temperature" are written once respectively by the "sign", the "measure weight" and the "measure temperature" atomic steps (cf. Fig. 3).

Fig. 4 summarizes the classification of the data involved in the medical treatment and in the container transportation process, taking into account the general data classification scheme presented in Fig. 1.

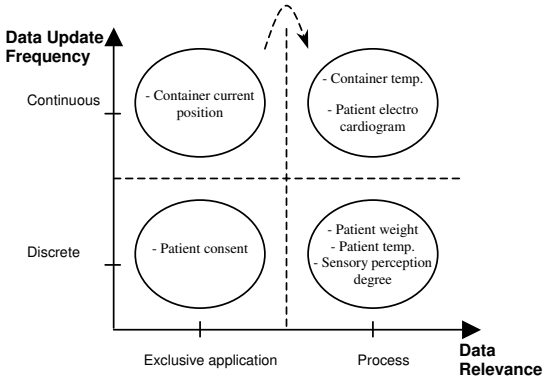

Figure 4: Data Classification for the Medical Treatment / Container Transportation Processes

3 CORRECTNESS CRITERION

In order to correctly deal with exceptional situations, it is crucial to know those points in time when running activities can be *safely* interrupted. A running activity is safely interrupted means that the con-

text of this activity is kept (1) such that all input data of subsequent activities are correctly supplied, or (2) in order to find possible solutions for exceptional situations. We denote these certain points in time as *safe points* of the respective activities.

The challenging question is how to determine the safe point of an activity. In order to adequately answer this question, our distinction between continuous and discrete data update is helpful. As the following definitions show, it is possible to precisely determine 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 are correctly supplied.

Definition 5 (Safe Interrupt Point for a Discrete Data Update) *Let (d, n) (n ∈ N, d ∈ D) be a discrete data update of d by n, and let ST_n^d be the set of atomic steps associated with n and writing d. Let further B := {stp(st), st ∈ ST_n^d | ∄ p ∈ P: Asn(p) = n and (p, d, write) ∈ $DataE_{extended}$}. Then the safe interrupt point t_{safe}^d of (d, n) corresponds to the maximum point in time any atomic step writes d (on condition that d cannot be provided by a data provision service). Formally:*

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

Informally, the safe interrupt point for a discrete data update by atomic steps is that maximum point in time when the last write access to the respective data element has taken place.

Definition 6 (Safe Interrupt Point for a Continuous Data Update) *Let (d, n) (n ∈ N, d ∈ D) be a continuous data update of d by n with a start updating time t_1 and a finish updating time t_k. 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 := Undefined holds.*

Intuitively, for continuous data updates there is no "natural" safe interrupt point. Therefore, we offer the possibility to define a safe interrupt point by the user. An example usage for such a user-defined safe interrupt point would be the "waiting time" in order to get the right container temperature after attaching it to the vehicle that shall power the refrigeration system within the container.

In order to determine the safe point of an activity, we have to consider that there might be several safe interrupt points. One example is the activity "prepare patient" which has two safe interrupt points belonging to data elements "weight" and "temperature" (Fig. 2).

Definition 7 (Activity Safe Point) *Let {d_1, ..., d_k} be the set of data elements (continously) written by activity n ∈ N (i.e., ∃ (n, d_i, w) ∈ DataE, i = 1, ..., k, w ∈ {write, continuous−write}). Let further $t_{safe}^{d_1}, \ldots, t_{safe}^{d_k}$ be the related safe interrupt points. Then we denote t_{safe} = max{$t_{safe}^{d_1}$, ..., $t_{safe}^{d_k}$} as the safe point of n (if $t_{safe}^{d_i}$ = Undefined ∀ i = 1, ..., k, t_{safe} is set to Undefined as well). Thereby, t_{safe} corresponds to the time when n can be safely interrupted keeping its context. An activity n can be safely interrupted if all input data of subsequent activities of n are provided.*

Using the notion of activity safe point we can state a criterion based on which it is possible to decide whether a running activity can be safely interrupted or not.

Criterion 1 (Interrupting a Running Activity by Keeping its Context) *Let S be an extended WSM Net, let I be an instance on S, and let w ∈ {write, continuous-write} ⊂ NAccessMode. A node n ∈ N with NS^S(n) = Running and safe point t_{safe} can be safely interrupted at $t_{interrupt}$ if one of the following conditions holds:*

- *∄(n, d, w) ∈ DataE*
- *$t_{safe} <= t_{interrupt}$ or t_{safe} = Undefined*
- *∀(n, d, w) ∈ DataE, $t_{interrupt} < t_{safe}^d$: d is an exclusive application data element*

A running activity can be safely interrupted from a process perspective if it either writes no data or if it solely writes exclusive application data. If a running activity writes process relevant data it can be safely interrupted if it has an undefined safe point or its safe point has been already transgressed. Finally, if exclusive application data become process relevant (e.g., if an exception handling process makes use of the full context of the interrupted activity), the last condition of Criterion 1 may not be applicable.

In order to illustrate the defined correctness criterion, we consider the container transportation process. Based on process schema S provided in Fig. 3, instance I_S in Fig. 5 has been started. Taking into account a defined transportation network, each of the activities' locations in I_S is captured by a coordinate (x, y). E.g., the origin and the destination locations in activity "move vehicle V from Montréal to Québec" would respectively correspond to the coordinates (1.5, 3.5) and (13, 8) within the transportation network. Suppose that a road traffic problem occurs at time $t_{interrupt} = t_{begin}^n + 75minutes$ (elapsed time since departure) while V is on the road between Montréal and Québec. At this time, suppose that the GPS system is indicating (7, 5.5) for the current position of V. To avoid the traffic problem, an optimisation module may propose a new transportation solution that consists of changing the already planned route leading to Québec. The new

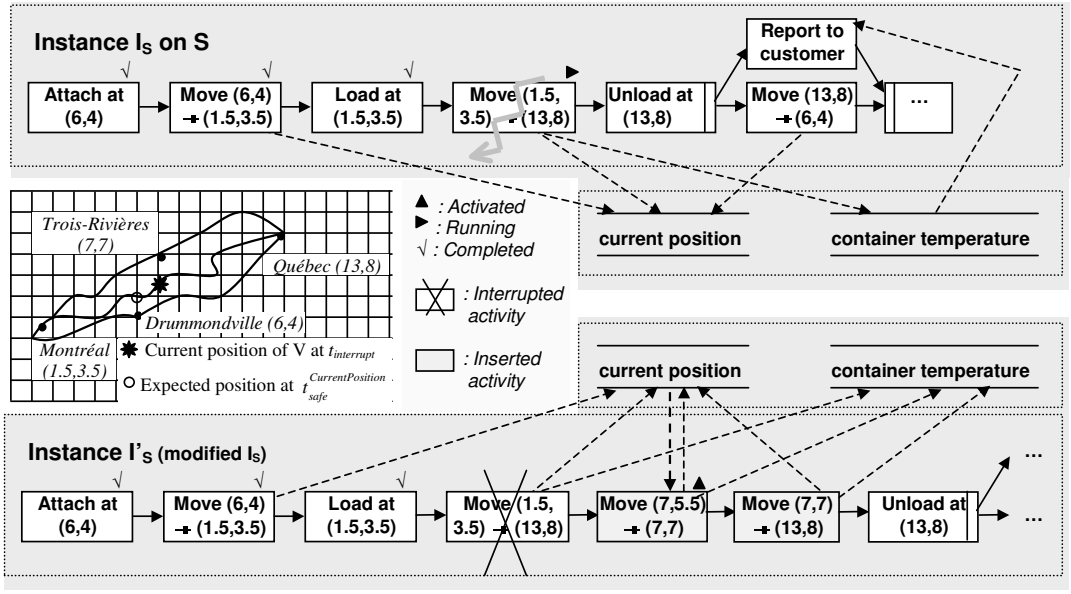

Figure 5: Container Transportation Scenario

route includes a detour via another location, that is Trois-Rivières located at position (7, 7). However, this new solution is only possible if V is close enough to Trois-Rivières, which means that the current position of V is beyond (6, 5). This correponds to $t_{safe}^{"CurrentPosition"} = t_{begin}^{n} + 60minutes$. In addition, suppose that the right container temperature is reached $15minutes$ after finishing loading the container and hence after the departure from the origin location, i.e., $t_{safe}^{"ContainerTemperature"} = t_{begin}^{n} + 15minutes$. Taking into account Def. 7, the safe point of activity "move vehicle V from Montréal to Québec" corresponds to $\max\{t_{safe}^{"CurrentPosition"}, t_{safe}^{"ContainerTemperature"}\} < t_{interrupt}$. Hence, this activity can be safely interrupted. The exclusive application data element "current position" was used to generate the new solution shown in Fig. 5. Following the road traffic problem, this data element becomes process relevant as well: it is given as input to the inserted activity "move vehicle V from current location to Trois-Rivières". Note that in this specific example, the "container temperature" data element is not relevant for the definition of the safe point, and hence it could be fixed to $Undefined$.

4 DISCUSSION

In this paper a "divide and conquer" approach is adopted: An activity is divided into atomic steps so that the interruption of this activity becomes possible by preserving its context.

In (Sadiq et al., 2001; Mangan and Sadiq, 2002) "pockets of flexibility" are defined. So called "containers" comprise different activities and constraints posed on these activities (e.g., activity B always before activity C). These containers can be inserted into certain regions within the process. If process execution reaches such a container the assigned user can choose the order of working on the offered activities by obeying the imposed constraints. This idea can be compared to our approach of subdividing activities into atomic steps and posing an order relation on them if necessary. However, both approaches use a different level of granularity and focus on different aims. The approach presented by (Sadiq et al., 2001) provides more flexibility regarding process modeling whereas our approach uses atomic steps for being able to preserve the data context in case of unexpected events during runtime.

The two kinds of data addressed by the Data Relevance dimension of our data classification scheme have already been discussed within the literature (Workflow Management Coalition, 1999; v.d. Aalst and van Hee, 2002). In (Workflow Management Coalition, 1999), a differentiation is made between application data and process relevant data. It is argued that application data may become process relevant if they are used by the workflow system to determine a state change. In this paper, we adopt the same definitions and interpretations as provided in (Workflow Management Coalition, 1999); furthermore, we judiciously highlight the fact that exclusive applica-

tion data may become process relevant when a failure occurs. In (v.d. Aalst and van Hee, 2002), a bigger variety of process data is featured: analysis data, operational management data, historical data, etc. It is stated that application data cannot be directly accessed by a workflow system but only indirectly through instance attributes and applications themselves. Hence, only the way of accessing application data from a WfMS is discussed.

The infinite completion time assigned as a default value to an atomic step st may be more precisely predicted using, for instance, the forward/backward calculation technique based on the duration of activities as proposed in (Eder and Pichler, 2002; Eder et al., 2003). This would allow estimating an activity safe point (t_{safe}) as a specific point in time (instead of infinite) even before reaching this point.

Another interesting application of the presented results arises in the context of process schema evolution (Rinderle et al., 2004a) i.e., process schema changes and their propagation to running process instances. One important challenge in this context is to find correctness criteria in order to ensure correct process instance migration after a process schema change. According to the compliance criterion (Casati et al., 1998; Rinderle et al., 2004a) it is forbidden to skip already running activities, i.e., the respective process instances are considered as being non–compliant. However, if we transfer the concepts of safe interruption of activities to the *safe deletion of activities* the number of process instances compliant with the changed process schema can be increased.

5 SUMMARY AND OUTLOOK

We have proposed a framework to correctly address the issue of safely interrupting running business process activities in case of exceptional situations; i.e., interrupting running activities by preserving their data context, which is extremely important in order to be able to provide adequate solutions in the sequel. This work was motivated by the analysis of data involved in the context of specific complex, yet representative, process-oriented applications, namely the container transportation application and the medical application. Besides modeling logical work units as process activities we have introduced another level of granularity by defining the atomic step concept. The latter is of utmost importance to build up the basis for a two-dimensional data classification scheme. On the one hand, the definition of the *data relevance* dimension, distinguishing between exclusive application data and process relevant data, is considered at its pure level within the *safely interruption* criterion conditions statement. On the other hand, we dug deeper regarding the *data update frequency* dimension by defining safe interrupt points for each of the discrete and the continuous data update by activities. This has led to the formal definition of the activity safe point considered as the backbone for the *safely interruption* criterion. Preserving this criterion, in turn, guarantees that if an activity is safely interrupted all necessary data is kept and can be used to figure out an adequate solution for the respective exceptional situation.

As future work, we aim to study extended transactional issues (e.g., semantic rollback) at both the micro flow and the macro flow level. In particular, this must be done in a way that enables flexible exception handling procedures (incl. dynamic flow changes). Respective facilities are indispensable for realizing adaptive enterprise applications.

REFERENCES

Bassil, S., Bourbeau, B., Keller, R., and Kropf, P. (2003). A dynamic approach to multi-transfer container management. In *Proc. Int'l Works. ODYSSEUS'03*, Sicily.

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

Casati, F., Ceri, S., Pernici, B., and Pozzi, G. (1998). Workflow evolution. *DKE*, 24(3):211–238.

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

Eder, J., Pichler, H., Gruber, W., and Ninaus, M. (2003). Personal schedules for workflow systems. In *Proc. Int'l Conf. BPM'03*, pages 216–231, Eindhoven.

Gartner Group (1999). Why e-business craves workflow technology. Technical Report T-09-4929.

Mangan, P. and Sadiq, S. (2002). A constraint specification approach to building flexible workflows. *Journal of Research and Practice in Inf Tech*, 35(1):21–39.

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. (2004a). Correctness criteria for dynamic changes in workflow systems – a survey. *DKE*, 50(1):9–34.

Rinderle, S., Reichert, M., and Dadam, P. (2004b). On dealing with structural conflicts between process type and instance changes. In *Proc. Int'l Conf. BPM'04*, pages 274–289, Potsdam.

Sadiq, S., Sadiq, W., and Orlowska, M. (2001). Pockets of flexibility in workflow specifications. In *Proc. ER'01 Conf.*, pages 513–526, Yokohama.

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

Workflow Management Coalition (1999). Terminology & glossary. Technical Report WFMC-TC-1011, WfMC.