Data Flow Correctness in Adaptive Workflow Systems

Stefanie Rinderle-Ma
Institute of Databases and Information Systems,
Ulm University, Germany,
stefanie.rinderle@uni-ulm.de

Abstract: Enterprises must be able to quickly adapt their business processes to react to changes in their environment. Needed business agility is often hindered by the lacking flexibility of contemporary workflow systems. In response to this inflexibility, adaptive workflow systems have emerged, which enable the dynamic adaptation of running workflows. One of the most important challenges in this context is to avoid inconsistencies and errors. So far, approaches providing respective correctness criteria for dynamic workflow change have mainly focused on control flow correctness (e.g., avoidance of deadlocks). However, little attention has been paid to data flow correctness even though this is crucial for any application of dynamic workflow change in practice. Specifically, missing or inconsistent input data of workflow activities, for example, can lead to blocking or breakdown of the underlying workflow system. This paper deals with fundamental challenges related to data flow correctness. We revisit and discuss data flow correctness at different phases of the workflow life cycle (i.e., buildtime and runtime), and show how data flow correctness can be ensured in an efficient way when dynamically changing a workflow.

1 Introduction

The adequate support of their business processes is crucial for any enterprise to stay competetive at the market nowadays; i.e., business processes should be supported throughout their life cycle. Specifically, processes must be modeled, implemented, and enacted (e.g., in a workflow system (WfMS)). Furthermore, running workflows might be subject to change for many domains (e.g., clinical or automotive domain). In exceptional situations, users want to deviate from the modeled workflow schema (e.g., a sudden breakdown of a patient) [RD98]. It is also possible that the workflow schema itself has to be adapted at the type level in order to comply, for example, to optimizations or new regulations. Thus, for any WfMS, it is crucial to enable *dynamic workflow change*; i.e., it must be possible to, for example, insert new workflow activities or to delete them from the workflow during runtime. In this context, one of the most important challenges is to ensure that dynamic workflow changes are applied *correctly* [RRD04a]. Specifically, dynamic workflow changes must not lead to any inconsistencies or errors in the sequel. Hence, adequate criteria must be found to enable the WfMS to guarantee correctness of workflow changes.

So far, many approaches [vB02, CCPP98, RRD04b, Wes00] have proposed correctness criteria for dynamic workflow changes (for a detailed discussion see [RRD04a]). However, these criteria focus on *control flow correctness* after modifying the workflow schema (e.g., after deletion or insertion of activities). Less attention has been paid to *data flow correctness*. However, the latter is particularly crucial for the practical implementation of workflows. Consider, for example, an activity being deleted which is writing a mandatory data element; i.e., a data element which is read by a subsequent activity and the associated application program. This would lead to a breakdown of the running system when the activity reading the data element is started.

In order to tackle the challenge of data flow correctness, basically, we have to distinguish between (1) control flow changes which also affect the data flow and (2) changes of the data flow itself; i.e., inserting new data elements and data links between activities and data elements. Where for case (1), as we will show, some of the existing correctness criteria (e.g., *compliance* [CCPP98, RRD04b]), automatically ensure data flow correctness, case (2) has not been addressed sufficiently so far.

Contribution of this paper is to provide a comprehensive discussion on data flow correctness. For case (1), we show how equipping changes with formal pre- and post-conditions ensures data flow correctness. Afterwards, for case (2) we discuss the general data consistency problem, i.e., the problem of inconsistent read accesses after data flow changes. We show how compliance can be extended based on augmented execution traces to ensure data consistency after data flow changes. Finally, we provide precise conditions to efficiently ensure data flow correctness. The efficiency of these compliance conditions is substantiated by means of an example.

In Sect. 2, we introduce general considerations on data flow correctness at buildtime and runtime. Afterwards, correctness issues for dynamic control flow changes and their impact on the data flow are discussed in Sect. 3. In Sect. 4 we present the data consistency problem and show how to verify data consistent compliance in Sect. 5. Related work is discussed in Sect. 6. We close with a summary and outlook in Sect. 7

2 Ensuring Data Flow Correctness at Build- and Runtime

2.1 Buildtime Issues

For each business process to be supported (e.g., handling a customer request or processing an insurance claim) a workflow type T represented by a workflow schema S has to be defined. For a particular type several workflow schemata may exist, representing the different versions and evolution of this type over time.

Let S = (C, D) be a workflow schema where C denotes the control flow schema and D denotes the data flow schema. Since we focus on data flow aspects in this paper, we abstract from a detailed definition of the control flow schema of S. Simplified, a control flow schema consists of a set of activities N and a set of control edges CtrlE (i.e., C = (N, CtrlE)). C can comprise patterns such as sequence, parallelism, alternative branchings, and loops; i.e. N contains workflow activities as well as structuring nodes (e.g., split nodes). For details, we refer to workflow meta models such as BPEL, Petri Nets, or WSM Nets including the correctness constraints set out by the particular meta model (e.g., deadlockfree control flow schema). The data flow schema D can be defined as tuple D = (DE, DataE) where DE a set of workflow data elements and DataE $\subseteq$ N $\times$ DE $\times$ $\{$read, write$\}$ is a set of read/write data links between activities and data elements.

As an example consider Fig. 1a. Control flow schema C consists of activity set N = $\{$A, B, C, D$\}$ and control edge set CtrlE = $\{$(A,B), (B,C), (C, D)$\}$. Data flow schema D contains data element set DE = $\{$d$\}$ and data edge set DataE = $\{$(A,d,write), (B,d,write), (C,d,read)$\}$. The essence of a correct data flow in WfMSs is that if an activity X is reading a data element d in a mandatory way, then d has to be supplied by another activity Y before the read access of X independently of the chosen execution path[1]. Assume now that activity C is reading data element d in a mandatory way. Specifically, d is input for an application source invoked by C. If d is not written when C is started, the execution of this application program is blocked and workflow execution can fail. Thus, system robustness is harmed in a severe manner.

Another correctness issue which is claimed, for example, in the ADEPT2 project, is the avoidance of lost updates in consequence of uncontrolled *blind write accesses* on a data element [RD98]. In Fig. 1a, a blind write access of activities A and B on data element d is depicted where the write access of A is instantly overwritten by B.

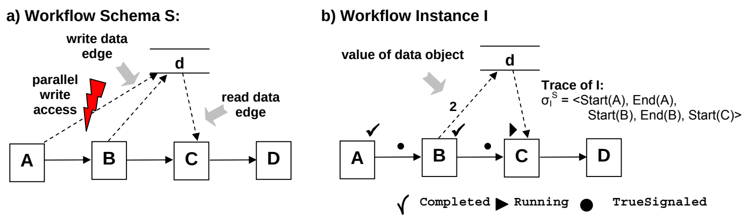


Figure 1: Basic Example

Data flow correctness can be efficiently checked at buildtime for block-structured workflow meta models such as WSM Nets (respective algorithms see [RD98]).

2.2 Runtime Issues

Based on workflow schema S, at runtime, new *workflow instances* can be created and executed. Start or completion events of activities of such instances are recorded in *traces*. WIDE, for example, only records completion events [CCPP98], whereas ADEPT2 distinguishes between start and completion events of activities [RRD04b].

Definition 1 (Trace) *Let $\mathcal{PS}$ be the set of all workflow schemata and let $\mathcal{A}$ be the set of activities (or more precisely activity labels) based on which workflow schemata $S \in \mathcal{PS}$ are specified (without*

[1]This is particularly important in the context of alternative branchings

loss of generality we assume unique labeling of activities). Let further $\mathcal{Q}_S$ denote the set of all possible traces producible on workflow schema $S \in \mathcal{PS}$. A particular trace $\sigma_I^S \in \mathcal{Q}_S$ of instance I on S is defined as σ_I^S = $< e_1, \ldots, e_k >$ (with e_i $\in$$\{$Start(a), End(a)$\}$, $a \in \mathcal{A}$, $i = 1, \ldots, k,\ k \in \mathbb{N}$) where the order of e_i in σ_I^S reflects the order in which activities were started and/or completed over S.[2]

As proposed by several approaches [Wes00, CCPP98, RRD04b], alternatively, the execution state of an instance I can be captured by marking function M$^{I^S}$ =(NS$^{I^S}$, ES$^{I^S}$) where M$^{I^S}$ reflects a compact representation of trace σ_I^S on S. For example, in ADEPT2, M$^{I^S}$ assigns to each activity n its current status $NS^{I^S}(n)$ $\in$ $\{$NotActivated, Activated, Running, Completed, Skipped$\}$ and to each control edge its current marking $ES^{I^S}(e)$ $\in$$\{$TrueSignaled, FalseSignaled$\}$. Markings are determined according to well defined rules [RD98]; markings of already passed regions and skipped branches are preserved (except loop backs). In Fig. 1b activities A and B are already completed whereas C is in state Running.

Regarding data flow, at runtime, read accesses take place when activities are started and write accesses take place when activities are completed [Wes00]. In ADEPT2, different values for a particular data element might be written (and read afterwards), for example, in consequence of different loop iterations. How data values can be logged within traces is described in Sect. 4.

3 Data Flow Correctness after Control Flow Changes

3.1 Structural Correctness

Structural workflow change can be defined as follows [RMRW08]:

Definition 2 (Workflow Change) *Let $\mathcal{PS}$ be the set of all workflow schemata and let S, $S' \in \mathcal{PS}$. Let further Δ = $<op_1, \ldots, op_n>$ denote a workflow change which applies change operations $op_i,\ i = 1, \ldots, n,\ n \in \mathbb{N}$ sequentially. Then:*

1. *$S[\Delta{>}S'$ if and only if Δ is correctly applicable to S and S' is the workflow schema resulting from the application of Δ to S (i.e., $S' \equiv S + \Delta$)*
2. *$S[\Delta{>}S'$ if and only if there are workflow schemata $S_1, S_2, \ldots, S_{n+1} \in \mathcal{PS}$ with S = S1, S' = S_{n+1} and for $1 \leq i \leq n$: $S_i[\Delta_i{>}S_{i+1}$ with Δ_i = (op_i)*

In general, we assume that change Δ is applied to a correct workflow schema S; i.e., S obeys the structural correctness constraints set out by the particular workflow meta model (e.g., bipartite graph structure for Petri Nets). This structural correctness can be achieved in two ways: (1) either Δ itself preserves correctness by formal pre-/post-conditions (e.g., in ADEPT2) or (2) Δ is applied and structural correctness of resulting schema S' is checked afterwards (e.g., by reachability analysis for Petri Nets).

[2] An entry of a particular activity can occur multiple times due to loopbacks.

Control flow changes can be also used to design a workflow schema; i.e., starting with the empty schema and building up the desired workflow schema step by step. If the applied change operations are equipped with formal pre- and post-conditions, the resulting workflow schema is "correct by construction" [DRR$^+$08].

3.2 Behavorial Correctness

Furthermore, we claim that after applying change Δ, any workflow instance on resulting schema S' must obey *behavorial correctness* (i.e., must not run into deadlocks or livelocks). Consider Fig. 1b: Assume that new activity X is inserted between activities A an B for instance I. This would result in an inconsistent marking (X in state Activated precedes B in state Completed). One prominent correctness criterion to guarantee behavorial correctness after dynamic workflow change is *compliance* [CCPP98, RRD04b].

Definition 3 (Compliance) *Let S, $S' \in \mathcal{PS}$ be two workflow schemata. Further let I be an instance running on S with trace σ_I^S. Then: I is compliant with S' iff trace σ_I^S could have been produced by an instance on S' (i.e., all events captured by σ_I^S in the same order as set out by σ_I^S).*

Basically, compliance of workflow instance I can be ensured by replaying trace σ_I^S on S' and checking for correctness of the resulting marking $M^{S'^I}$. However, as we will show in Sect. 5.2, doing so might quickly result in performance problems due to the possibly large size of σ_I^S. Thus, in [RRD04b], we presented precise conditions based on M^{S^I} which ensure compliance much more quickly (cf. Sect. 5.2).

Table 1 exemplarily shows two change operations as realized in the ADEPT2 approach together with their structural and behavorial pre-conditions and post-conditions. If the preconditions are fulfilled, based on the post-conditions the control and data flow are adapted. For an overview on realization of such workflow change (patterns) see [WRSRM08]. For details on change realization in ADEPT2 see [RJR07]. Specifically, data flow correctness after dynamic control flow change is preserved. As example consider Fig. 2. After deleting activity B, the data flow correctness of the resulting workflow schema is violated; i.e., input parameter d of activity C is no longer supplied. Either the change would be rejected or – if possible – "healing techniques" could be applied, for example, by deleting subsequent activity C or inserting a special data supplying service [RD98].

4 The Data Consistency Problem

So far, we have focused on data flow correctness after dynamic control flow change (e.g., removing data edges of an activity to be deleted). However, changes of the data flow itself might become necessary in some scenarios (e.g., re-linking data edges in order to correct modeling errors). Table 2 summarizes generic data flow changes.

The following example illustrates the problem which might arise in the context of "pure"

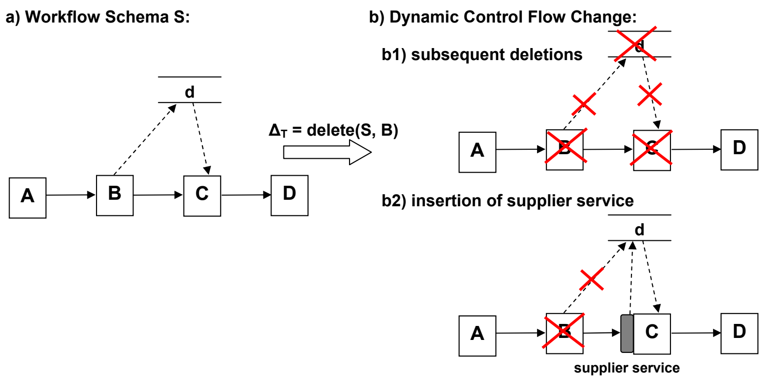


Figure 2: Basic Example

Table 1: *Selection of High-Level Change Operations Including Pre- and Post-Conditions*

sInsert(S, X, A, B) for instance I on schema S = (C, D) with C = (N, CtrlE)	
Inserts activity X between subsequent activities A and B into S	
Pre-conditions (structural)	X, A, B ∈ N ∧ (A,B)[a] ∈ CtrlE
Pre-Conditions (behavorial)	NS^{I^S}(B) ∈ {NotActivated, Activated, Skipped}
Post-Conditions (structural)	adjust control flow, adjust data flow
Post-Conditions (behavorial)	NS^{I^S}(B) = Activated $\Longrightarrow$ NS^{I^S}(B) = NotActivated ∧ NS^{I^S}(X) = Activated
delAct(S, X) for instance I on schema S = (C, D) with C = (N, CtrlE)	
Deletes activity X from S	
Pre-conditions (structural)	X ∈ N
Pre-Conditions (behavorial)	NS^{I^S}(X) ∈ {NotActivated, Activated}
Post-Conditions (structural)	adjust control flow, adjust data flow
Post-Conditions (behavorial)	NS^{I^S}(X) = Activated ∧ NS^{I^S}(succ(X)[b]) = Activated

[a] where (A,B) denotes the control edge between activities A and B
[b] succ(X) denotes all direct successors of X in S

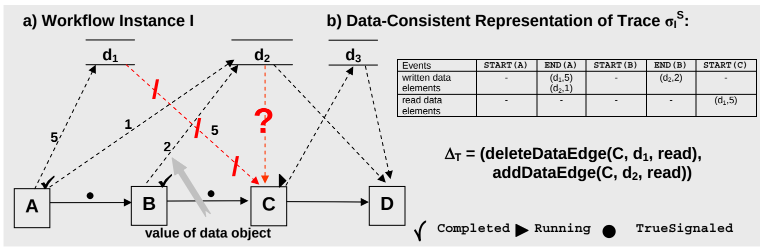
data flow changes. Consider instance I depicted in Fig. 3a. Activity C has been started and therefore has already read data value 5 of data element d_1. Assume now that due to a modeling error read data edge $(C, d_1, read)$ is deleted and new read data edge $(C, d_2, read)$ is inserted afterwards. Consequently, C should have read data value 2 of data element d_2 (instead of data value 5). This inconsistent read behavior may lead to errors, if for example the execution of this instance is aborted and therefore has to be rolled back. Using trace σ_I^S as defined in Def. 1, this erroneous case would not be detected; i.e., according to Def. 3, instance I would be classified as compliant.

How to overcome such inconsistencies after data flow changes? Apparently, if we want to avoid dirty reads, we have to somehow capture the information about read and write accesses within traces. This can be defined as follows [RMRW08]:

Definition 4 (Data-consistent Trace) *Let the assumptions be as in Def. 1. Let further $\mathcal{D}_S$*

Table 2: *Data Flow Change Operations*

Change Operation Δ Applied to Schema S	**Effects on Workflow Schema S**
addDataElements(S, dataLabels, dom, defaultVal)	adds dataLabels to D whereby domain dom and default value defaultVal are assigned to the set of new data elements $D^* := D \cup$ dataLabels
deleteDataElements(S, elementSet)	deletes set of data elements elementSet from D $D^* := D \setminus$ elementSet
addDataEdges(S, dataEdges)	adds set of data edges dataEdges to DataE $DataE^* := DataE \cup$ dataEdges
deleteDataEdges(S, dataEdges)	deletes set of data edges dataEdges from DataE $DataE^* := DataE \setminus$ dataEgdes



a) Workflow Instance I
b) Data-Consistent Representation of Trace σ_I^S:
d1
d2
d3
5
1
2
5
?
A
B
C
D
value of data object
Δ_T = (deleteDataEdge(C, d1, read), addDataEdge(C, d2, read))
✓ Completed ▶ Running ● TrueSignaled

Events	START(A)	END(A)	START(B)	END(B)	START(C)
written data elements	-	$(d_1,5)$ $(d_2,1)$	-	$(d_2,2)$	-
read data elements	-	-	-	-	$(d_1,5)$

Figure 3: Data Consistency Problem

be the set of all data elements relevant in the context of schema S. Then we denote $\sigma_I^{S^{dc}}$ as data-consistent trace representation of σ_I^S

with $\sigma_I^{S^{dc}}$ = $<e_1, \ldots, e_k>$:

$e_i \in \{START(a)^{(d_1,v_1),\ldots,(d_n,v_n)}$, $END(a)^{(d_1,v_1),\ldots,(d_m,v_m)}\}$, $a \in \mathcal{A}$[3]

where tuple (d_i, v_i) describes a read/write access of activity a on data element $d_i \in \mathcal{D}_S$ with associated value v_i ($i = 0, \ldots, k,\ k \in \mathbb{N}$) if a is started/completed.

Based on data consistent trace $\sigma_I^{S^{dc}}$, compliance can be defined as for Def. 3, i.e., checking compliance based on $\sigma_I^{S^{dc}}$ leads to correct data flow. This is illustrated by Figure 3. Assume that the data-consistent trace $\sigma_I^{S^{dc}}$ is used instead of σ_I^S. Then the intended data flow change Δ (deleting data edge $(C, d_1, read)$ and inserting data edge $(C, d_2, read)$ afterwards) cannot be correctly propagated to I since entry Start(C)$^{(d_1,5)}$ of $\sigma_I^{S^{dc}}$ cannot be reproduced on the changed schema.

[3]Generally, read accesses on process data take place when an adtivitiy is started and write accesses take place when an activity is completex [Wes00, RRD04b]. However, if application data is involved, there might be read and write accesses – even in a continuous manner – during the execution of an activity [BRKK05]. We will investigate such kind of application data accesses and their correctness when changing the process in future work.

5 Compliance Conditions for Data-Flow Changes

5.1 Compliance Conditions for Data Flow Change

So far, there is no approach dealing with correctness issues in the context of data flow changes. One possibility would be to replay data-consistent traces on changed workflow schemata and check for consistent markings and consistent data accesses afterwards. However, doing so might be complex as we will show in Sect. 5.2. Thus, in Theorem 1, compliance conditions for data flow changes (cf. Tab. 2) are presented based on which compliance is automatically preserved.

Theorem 1 (Compliance Conditions For Data Flow Changes) *Let S = (C, D) be a correct workflow type schema (e.g., represented by a WSM Net) and I be a workflow instance on S with data-consistent trace* $\sigma_I^{S^{dc}}$ *and with marking* $M^{S^I} = (NS^{S^I}, ES^{S^I})$. *Assume further that change* Δ *transforms S into a correct workflow schema S' = (C', D').*

(a) Δ *inserts a data element d into S; i.e.,* Δ *= addDataElements(S,* $\{d\}$*, . . .). Then:*
I is compliant with S'.

(b) Δ *deletes a data element d from S; i.e.,* Δ *= deleteDataElements(S,* $\{d\}$*, . . .). Then:*
I is compliant with S' $\Leftrightarrow$
No read access on d by an activity with state Running or Completed[4]

(c) Δ *inserts or deletes a read edge (d, n, read); i.e.,* $\Delta \in \{$*addDataEdges(S,* $\{$*(d, n, read)*$\}$*), deleteDataEdges(S,* $\{$*(d, n, read)*$\}\}$*. Then:*
I is compliant with S' $\Leftrightarrow$ *NS(n)* $\in \{$*NotActivated, Activated, Skipped*$\}$

(d) Δ *inserts or deletes a write edge (d, n, write); i.e.,* $\Delta \in \{$*addDataEdges(S,* $\{$*(d, n, write)*$\}$*), deleteDataEdges(S,* $\{$*(d, n, write)*$\}\}$*. Then:*
I is compliant with S' $\Leftrightarrow$ *NS(n)* $\neq$ *Completed*

Due to lack of space we omit a formal proof. To explain how Theorem 1 works we come back to the example depicted in Fig. 3. The depicted data flow change cannot be applied to instance I according to Theorem 1 since activity C is Running and therefore has already read data element d_1. Consequently re-linking the data access of C to d_2 would be prohibited what complies to the desired behavior in this case.

As already mentioned, data flow adaptations also become necessary in conjunction with control flow changes. In this case, the conditions of Theorem 1 are already met if the behavorial pre-conditions of the according change are fulfilled.

5.2 Efficient Compliance Checks

Basically, replaying traces as necessary for checking compliance in a traditional way, can be very expensive for workflows with a multitude of activity instances; i.e., even for work-

[4]The deletion of data in connection with write accesses is sufficiently backed up by the structural correctness conditions of the associated delete data element operation (for details see [Rei00]).

flow instances without loops replaying traces results in a complexity of $O(n)$ per instance (where n denotes the number of activities). If workflow instances contain loops which run through a possibly high number of interations, trace size might easily explode and increase complexity in the sequel. Consider, for example, a workflow schema consisting of 20 activities where 10 activities are situated within a loop construct. Assume further that the loop runs through 5 iterations in average. A single history entry of a workflow instance typically comprises entry type (Short, 2 Bytes), activity identifier (Long, 8 Bytes), originator (Long, 8 Bytes), time stamp (Long, 8 Bytes), iteration counter (Short, 2 Bytes), and decision statement (Short, 2 Bytes) (e.g., the MXML format [vdAea07]). Altogether, this results in 30 Bytes per trace entry. Contrary, the size of an activity marking is 1 Byte. Then we obtain an average trace size of 35 MB for 20.000 running workflow instances. Contrary, the necessary marking information is bounded by 0,2 MB.

For data-consistent traces (cf. Def. 4), trace size becomes even bigger. Even if we only assume data values of type Short per entry (additional 4 Bytes), trace size increases to 40 MB for the above example. For values of type Long, trace size becomes 55 MB and if values of type String are involved, trace size can be arbitrarily big. Thus, checking compliance by replaying data-consistent traces might become way to expensive for realistic scenarios. Contrary, the costs for checking the compliance conditions presented in Theorem 1 are again bounded by 0,2 MB regardless which kind of data values are written.

6 Related Work

A detailed discussion on correctness criteria for dynamic workflow change can be found in [RRD04a]. All approaches have mainly focused on control flow changes and their correctness so far. Some of the proposed correctness criteria come with automatic guarantees for data flow correctness after control flow changes. One example is the compliance criterion [CCPP98] based on which not correctly supplied input data of activities is prohibited. However, approaches do not sufficiently deal with correctness after data flow changes. Case-handling [vWG04] focuses on the data-driven adaptation of workflows. However, explicit control and data flow changes as proposed by the ADEPT2 approach, for example, are needed if a WfMS also aims at supporting concurrent changes (i.e., changing workflow schema and running instances at the same time).

7 Summary and Outlook

In this paper we showed how data flow correctness can be preserved in the context of dynamic workflow change. First, we discussed structural and behavorial conditions for control flow changes which automatically ensure data flow correctness of the associated workflow schema. For data flow changes, the data consistency problem (i.e., how to preserve correctness for data flow changes) was introduced. It can be tackled when workflow execution traces are extended by information about data read and write accesses. Based on data-consistent traces, compliance for data flow changes as well as precise compliance conditions for data flow changes can be defined. Using compliance conditions for data flow changes, correctness considerations for control and data flow in the context of dynamic change become complete and thus applicable in practice. All concepts on dynamic change are implemented within powerful adaptive WfMS ADEPT2. An important concern of this project is to support the evolution of WfMSs in a correct, efficient, usable, and holistic way. The latter refers to support of evolution of all different aspects related to a WfMS (e.g., the controlled evolution of access control mechanisms [RR07]).

References

[BRKK05] S. Bassil, S. Rinderle, R. Keller, and M. Kropf, P. Reichert. Preserving The Context of Interrupted Business Process Activities. In *Proc. Int'l Conf. on Enterprise Information Systems*, pages 38–45, 2005.

[CCPP98] F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Workflow Evolution. *Data and Knowledge Engineering*, 24(3):211–238, 1998.

[DRR$^+$08] P. Dadam, M. Reichert, S. Rinderle, M. Jurisch, H. Acker, K. Göser, U. Kreher, and M. Lauer. Towards Truly Flexible and Adaptive Process-Aware Information Systems. In *UNISCON 2008*, pages 72–83, 2008.

[RD98] M. Reichert and P. Dadam. ADEPT$_{flex}$ - Supporting Dynamic Changes of Workflows Without Losing Control. *JIIS*, 10(2):93–129, 1998.

[Rei00] M. Reichert. *Dynamic Changes in Workflow-Management-Systems.* PhD thesis, University of Ulm, Computer Science Faculty, 2000. (in German).

[RJR07] S. Rinderle, M. Jurisch, and M. Reichert. On Deriving Net Change Information From Change Logs – The DELTALAYER-Algorithm. In *Conf. Datenbanksysteme in Business, Technologie und Web (BTW'07)*, pages 364–381, 2007.

[RMRW08] S. Rinderle-Ma, M. Reichert, and B. Weber. Relaxed Compliance Notions in Adaptive Process Management Systems. In *Proc. Int'l Conference on Conceptual Modeling (ER'08)*, 2008. (accepted for Publication).

[RR07] S. Rinderle and M. Reichert. A Formal Framework for Adaptive Access Control Models. *Journal on Data Semantics*, (IX):82–112, 2007.

[RRD04a] S. Rinderle, M. Reichert, and P. Dadam. Correctness Criteria for Dynamic Changes in Workflow Systems – A Survey. *Data and Knowledge Eng.*, 50(1):9–34, 2004.

[RRD04b] S. Rinderle, M. Reichert, and P. Dadam. Flexible Support Of Team Processes By Adaptive Workflow Systems. *Distributed and Parallel Databases*, 16(1):91–116, 2004.

[vB02] W.M.P van der Aalst and T. Basten. Inheritance of Workflows: An Approach to Tackling Problems Related to Change. *Theoret. Comp. Science*, 270(1-2):125–203, 2002.

[vdAea07] W.M.P. van der Aalst and et al. ProM 4.0: Comprehensive Support for Real Process Analysis. In *Proc. of Application and Theory of Petri Nets and Other Models of Concurrency*, pages 484–494, 2007.

[vWG04] W.M.P van der Aalst, M. Weske, and D. Grünbauer. Case Handling: A New Paradigm for Business Process Support. *Data & Knowledge Engineering*, 2004. (to appear).

[Wes00] M. Weske. Workflow Management Systems: Formal Foundation, Conceptual Design, Implementation Aspects. University of Münster, Germany, 2000. Habilitation Thesis.

[WRSRM08] B. Weber, M. Reichert, and Stefanie S. Rinderle-Ma. Change Patterns and Change Support Features - Enhancing Flexibility in Process-Aware Information Systems. *Data and Knowl. Engineering*, 66(3):438–466, 2008.