Proc. WETICE 2006: First IEEE Workshop on Flexibility in Process-aware Information Systems (ProFlex 2006), Manchester, June 2006 (to appear)

On Representing Instance Changes in Adaptive Process Management Systems*

Stefanie Rinderle[1], Ulrich Kreher[1], Markus Lauer[1], Peter Dadam[1]
[1]Dept. DBIS, University of Ulm, Germany
{stefanie.rinderle, ulrich.kreher, markus.lauer, peter.dadam}@uni-ulm.de

Manfred Reichert[2]
[2] IS Group, University of Twente, The Netherlands
m.u.reichert@utwente.nl

Abstract

By separating the process logic from the application code process management systems (PMS) offer promising perspectives for automation and management of business processes. However, the added value of PMS strongly depends on their ability to support business process changes which can affect the process type as well as the process instance level. This does not only impose challenging conceptual issues (e.g., correctness of process schemata after changes) but also requires sophisticated implementation concepts with respect to efficient algorithms, flexible architectures, and reasonable treatment of resources. In this paper we sketch the general implementation concepts for representing process type and process instance data as well as for realizating process schema evolution. All these concepts have been developed and are currently implemented in the ADEPT2 prototype within the AristaFlow project.

1 Introduction

More and more companies adopt process management systems (PMS) for the control, management, and monitoring of their business processes (or parts of them). An important prerequisite for the practical usage of a PMS is the adaptivity of the implemented processes [10, 11]. The PMS must be able to support changes of stored process templates (we call this *process schema evolution*) as well as modifications of running process instances. Both are needed, for example, to enable the company to react to emerging requirements or occuring exceptions. Changes of process templates represent process type changes and may become necessary due to process optimization efforts, evolving organizational structures (e.g., due to outsourcing of activities), or launching of new laws [13, 2, 8]. Changes of single process instances may have to be applied in order to deal with exceptional situations (e.g., patient breakdown or engine failure) but also, for example, to react to with late customer requirements [6].

Adaptivity imposes high challenges on process management technology. Algorithms are to be developed in order to avoid inconsistent states of *process templates* and *process instances* after process changes. If desired the PMS must also allow to *propagate* process template modifications to running process instances after having checked compliance of the instance states [10]. At this the migration of *unbiased* process instances (i.e., instances which have not been subject to individual modifications so far) and the connected state-related correctness checks build the fundament for a comprehensive schema evolution support. However, it would be out of touch with reality to only allow the migration of unbiased process instances by excluding *biased* process instances from a schema evolution. Biased process instances have been already individually modified, e.g., due to an exceptional situation. Therefore their instance-specific schema deviates from the process template the instances have been started on. For them, additional challenges arise when migrating them to the changed process template. For example, it is crucial to distinguish between *disjoint* and *overlapping* process type and process instance changes. Disjoint changes have different effects on the process (instance) schema whereas overlapping changes (partially) affect the same regions of the underlying process (instance) schema. When applying proces changes, in any case, the other functions of the PMS running in parallel (e.g., worklist management) must not be disturbed, even if several thousands process instances have to be migrated at the same time (as often the case for long-running applications). Therefore, the PMS can only make use of restricted resources (e.g., memory) for these tasks. All these challenges require a flexible and resource-saving architecture

*This work was conducted within the AristaFlow project which is funded by the State of Baden-Württemberg, Germany.

and an efficient implementation.

Commercial PMS either totally lack the possibility of changing processes during runtime or they meet the requirements presented above only in a very restricted manner. By contrast, the ADEPT2 prototype developed within the AristaFlow project offers full support of process type and process instance changes as well as of their interplay. The formal framework for this next generation PMS has been described in previous publications [10, 8]. In this paper we present an architecture for the system-internal representation of process type and process instance changes (e.g., insertion or deletion of activities or data elements) which supports instance migration in an efficient manner and needs only little memory. We also discuss how this architecture has been implemented within the ADEPT2 prototype.

In Sect. 2 we discuss the representation and migration of unbiased instances. Sect. 3 shows how to represent biased instances and how to migrate them after disjoint template changes (cf. Sect. 4). Sect. 5 presents selected features of our prototype. In Sect. 6 we discuss related work and close with a summary and outlook in Sect. 7.

2 Migration of Unbiased Instances

We first consider unbiased process instances, i.e., instances which have not been individually modified so far. Approaches from literature [15] as well as commercial systems like Staffware [12] represent process templates and process instances as illustrated in Fig. 1. The *process logic* (e.g., control and data flow) is encapsulated within the object *process template* which represents the *process type*. The *instance objects* which represent the process instances solely contain runtime information (like execution states of activities or at least logically, the content of data elements). The associated process type is expressed by a reference to the respective process template object. Following this approach, all instances of a given process type reference the same template object. This approach constitutes the basis for the following considerations. The needed storage space is significantly reduced – especially for a large number of running instances – compared to storing a process description for each instance in a redundant way as it was the case in workflow systems like ProMInanD [3]. Furthermore it is possible to reduce the computing time since structure-changing operations occuring in the context of a schema evolution just have to be applied to the process template once instead of applying them to each running instance.

However, the template object must not be changed directly since this would result in a migration of all running instances. Doing so would not be correct for process instances for which their execution has been progressed "too far". Consider, for example, Fig. 2[1]: Instances I1 and I2

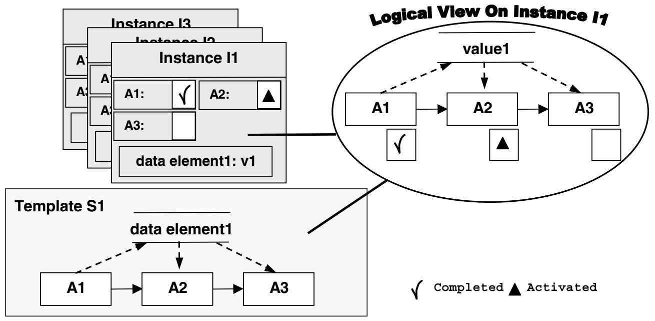


Figure 1. Representation of Process Instances

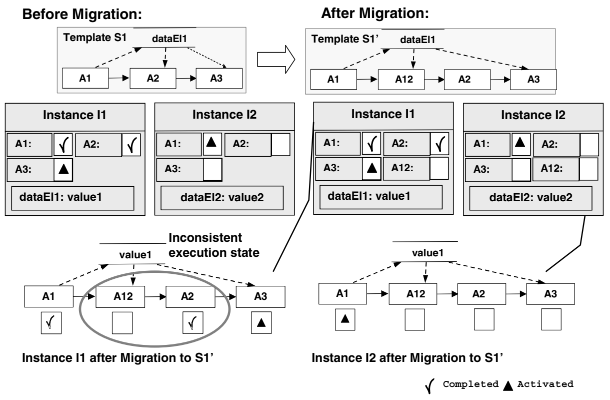


Figure 2. Incorrect Migration of Instance I1

run according to the same template S1. Since activity A2 has already been executed for instance I1 this instance is not compliant with the new version of schema S1 for which activity A12 has been inserted as a predecessor of activity A2. If this change would be directly applied to S1 instance I1 would be migrated in an incorrect manner (contrary to I2). This, in turn, would lead to an inconsistency since activity A2 of instances I1 has been completed before its new predecessor A12 is executed. Note that this is exactly the implementation we find in commercial workflow systems like, for example, Staffware [12].

The coexistence of instances which refer to the old or new schema version can be achieved by creating a copy of the template object. Then the schema changes can be applied to this copy and all compliant process instances are migrated to it. In this case the instance migration is done by re-linking the instance references to the new version. This is followed, for example, in abaXX or MQWorkflow.

[1]Note that for illustration reasons we use abstract examples in this paper. For practical scenarios on process instance and type changes refer to, e.g., [10].

3 Representation of Biased Process Instances

We now look at biased instances, i.e., instances for which their instance-specific schema deviates from the original template they were created on due to individual instance modifications. How can instance-specific modifications be implemented when using the approach discussed before? One possibility is to create a copy of the associated template object, apply the changes to it, and re-link the affected instances to the new copy version afterwards. However, doing so the original process type reference is lost. Reason is that the biased instances no longer reference the original schema object although they descend from this template. Without additional provisions such instances are no longer taken into account when further template changes and schema evolutions occur. Additionally, if the number of biased process instances is high, this approach will degenerate to the solution where the complete process description is stored for each process instance. Generally, for a particular process instance change only a small part of the original process template is adapted. Therefore, in most cases, for biased instances it is more efficient to only store the "delta" between the instance-specific schema and process template.

This delta can be represented by the change operations which have been applied to modify the instance. Alternatively, the divergence can be represented by a so called *delta layer*. This layer stores so called *delta objects* which reflect the difference between instance objects and template objects. In the following, we focus on the second variant since it offers several advantages for the management and migration of biased instances.

Figure 3 illustrates the delta layer concept. The delta layer is represented by an object which has the same interfaces as the template object and therefore offers the same operations. The difference between the delta layer object and the template object is that the delta layer object does not reflect the whole process graph but only those parts of the process schema which have been changed by instance-specific modifications. Therefore, together with the template object the delta layer object represents the instance-specific schema of the biased instance. The instance object which represents the biased instance does no longer reference the associated template object (cf. Fig 3) but the delta layer object. The delta layer object itself references the original template object and therefore preserves the assocation between instance I1 and process type S1. The unbiased instances (e.g., I2) directly reference the original process template object further on. For biased instances queries like "select all direct successors of activity X" are fired against the delta layer. For unbiased instances, by contrast, they are directly fired against the original template object.

How the delta layer can be realized depends on the representation of nodes and edges of the process graph. In our

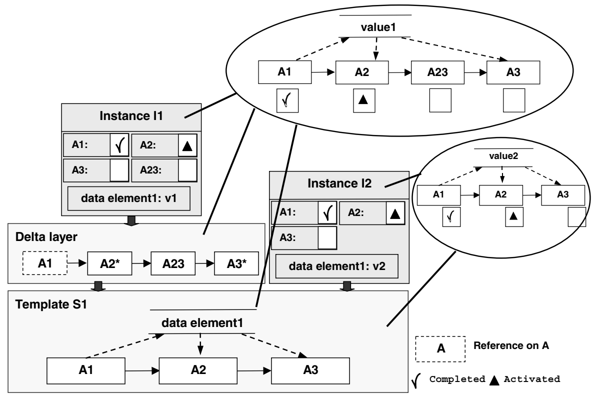


Figure 3. The Delta Layer Concept

implementation approach, for example, there are no edge objects. Instead, we explicitely store the predecessor and successor activities for each activity. Obviously, this approach is sufficient to represent the control flow.

Before a dynamic change takes place all activity objects which are affected by the modification are automatically copied into the delta layer. The change is then executed on the copies. As an example consider again Fig. 3: In order to insert activity A23 between activities A2 and A3, first of all, activity objects A2 and A3 are copied (including their ID) and inserted into the delta layer as A2* and A3*[2]. A2 and A3 are to be copied since their predecessor and successor sets are changed due to the ad-hoc modifications. Afterwards activity object A23 is created within the delta layer. Finally, A23 is inserted between A2* and A3* by adapting the predecessor and successor lists of A2*, A3*, and A23 accordingly. For an implementation using edge objects it is not necessary to copy A2 and A3 into the delta layer. Only A23 and the objects for egdes A2 $\longrightarrow$ A23 and A23 $\longrightarrow$ A3 have to be created. Additionally, A2 $\longrightarrow$ A3 has to be marked as deleted.

Assume the following scenario where query "select all direct successors of activity X" is to be fired. For an implementation not using edge objects, first of all, it is checked whether an activity object with respective ID is stored within the delta layer or not. If so, the successor list is returned. Otherwise the delta layer forwards the query to the referenced process template object. For an implementation using edge objects the delta layer, first of all, fetches all edge objects from the process template for which activity X is registered as source. Then the delta layer removes all edges from this edge set which are marked as deleted. Then it unifies this set with the set of all newly inserted edges having source X within the biased instance-specific schema afterwards. The resulting set then contains

[2]We use this notion for readability purposes

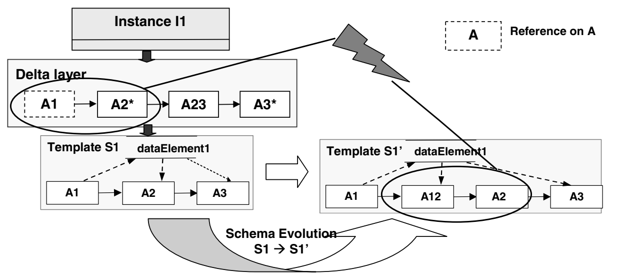


Figure 4. Migration of Biased Instances for Implementation Without Edge Objects

all edges with source X (based on which, trivially, the successors of X within the new instance-specific schema can be determined).

4 Migration of Biased Process Instances

The migration of biased process instances is a challenging issue. First of all, the biased instances have to be classified along the degree of overlap between their instance-specific change and the template change [9]. Reason is that the migration strategy depends on this classification. In this paper, we focus on the migration of instances for which their bias is disjoint with the template change. However, in the ADEPT2 prototype it is possible to manage the migration of instances with arbitrary bias.

One possibility for migrating instances with disjoint bias is to re-link the reference to the original template object within the delta layer to the new template version. However this may lead to problems for the implementation without using edge objects. Consider the scenario depicted in Fig. 4. A1, A2, and A3 are three sequentially ordered actitivies. Inserting an activity A23 between A2 and A3 at instance level (instance I1) results in the creation of a copy A2* of A2 within the delta layer and the replacement of the ID of A3 by the ID of A23 within the successor list. By inserting activity A12 between A1 and A2 at the process schema level (template S1) the ID of A1 is replaced by the ID of A12 within the predecessor list of A2. The query "determine all direct predecessors of A2" still yields A1 as a predecessor of A2 since the delta layer returns the predecessor list of A2*. This list, however, has not been adapted by the schema evolution and therefore still contains the ID of A1.

In order to overcome this problem, an empty delta layer object can be attached to the template object representing the new schema version. Then we apply the instance-specific change to the empty delta layer object and re-reference it by the biased instance. The old delta layer object could be either discarded (and its used storage could be released) or be recycled to a pool of (empty) delta layers for reuse. When applying this approach the insertion of A12 (i.e., the type change) is conducted before the insertion of A23 (i.e., the instance change). Therefore A12 is already known as a predecessor of A2 before copying it into the delta layer. Therefore A12 is the predecessor of A2* after copying to the delta layer. This leads to a correct answer of query "determine all direct predecessors of A2". Swapping the order of process type and instance changes was valid since we assumed that the changes were disjoint[3].

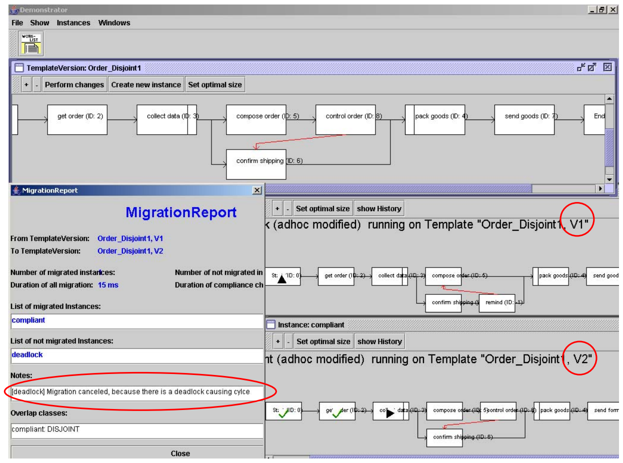
When using an implementation with edge objects we are not confronted with the problem of "forgotten predecessors" as described above. Here, the delta layer contains an activity object for A23 and the new edges A2 $\longrightarrow$ A23 and A23 $\longrightarrow$ A3 after dynamic insertion of A23. Edge A2 $\longrightarrow$ A3 is marked as deleted. At type level, edge A1 $\longrightarrow$ A2 is removed and edges A1 $\longrightarrow$ A12 and A12 $\longrightarrow$ A2 are inserted when adding A12. For processing query "determine all direct predecessors of A2" the delta layer, first of all, fetches all edges with target A2 from the process template. The template object returns only A12 $\longrightarrow$ A2 to the delta layer. Since this edge is not marked as deleted within the delta layer and no more edges with target A2 are stored, the delta layer returns A12 as predecessor of A2. However, the implementation variant using edge objects is not always advantageous. It, for example, always necessitates an access to the template object in order to determine the incoming and outgoing edges of an activity. For the implementation variant not using edges, by contrast, accessing the template object is not necessary since the associated activity is already contained within the delta layer.

5 ProofOfConceptPrototype

Our prototype internally realizes process types and instances according to the concept depicted in Fig. 3. Fig. 5 shows the ouput for the migration of instances for which their bias is disjoint with the process type change. Additionally, Fig. 5 depicts a report which summarizes information about compliant and non-compliant process instances, the reason for exclusion from migration (i.e., migration would result in structural inconsistencies like deadlock-causing cycles), and the time which was necessary for compliance checks as well as for marking adaptations.

Figure 6 shows that part of the used data model which describes the representation of process templates and process instances. A `Template` object represents an orignal process template and corresponds to the template object mentioned in this paper. It is directly referenced by instance objects which represent unbiased process instances. In case of biased instances an `ModifiedTemplate` object is linked between the `Instance` objects and the

[3]Since disjoint changes are commutative [9] the order of their application can be swapped resulting in execution equivalent process schemes.



MigrationReport

From TemplateVersion: Order_Disjoint1, V1
To TemplateVersion: Order_Disjoint1, V2

Number of migrated instances: 1
Number of not migrated in
Duration of all migration: 15 ms
Duration of compliance ch

List of migrated Instances:
compliant

List of not migrated Instances:
deadlock

Notes:
[deadlock] Migration canceled, because there is a deadlock causing cylce

Overlap classes:
compliant: DISJOINT

Figure 5. Migration of Instance with Disjoint Bias (Prototype)

`Template` objects the biased instances are based upon. It fulfills the function of the delta layer and represents only process graph parts which have been modified at instance level. `Template` object and `ModifiedTemplate` object build up the runtime process schema of the associated instance. As claimed both `Template` and `ModifiedTemplate` offer the same interface - both implement interface `TemplateVersion`. Thus transparent access by an `Instance` object is ensured. As the following code fragment shows `ModifiedTemplate` delegates requests it cannot answer to the `(Modified)Template` object it adapts. The code fragement depicted in Fig. 7 returns the direct predecessors of an activity in an given process. For this purpose, first of all, it is checked whether or not the specified activity was copied to the delta layer because of being affected by ad hoc modifications (`currActivity=(Activity) activityID2ActivityMap.get(inID)`). In case `(currActivity!=null)`, method `getCtrlPred` is called on the corresponding activity object and returns the direct predecessors. Otherwise it delegates the request to the adapted `(Modified)Template` object. Note that within our prototype the data flow is handled analogously (cf. Fig. 7).

6 Related Work

From a conceptual point of view, adaptivity in PMS has been focused on by many approaches so far (e.g., see [13, 2, 4, 11, 8, 15]). From these approaches, only the INTELLIGEN project, the WASA$_2$ project [15], and ADEPT [6, 10] address the issue of process type and process instance changes within one system. All other approaches either deal with single process instance changes or process schema evolution. However, neither INTELLIGEN nor WASA$_2$ discuss how the *interplay* between process type and process instance changes can be adequately handled. For example, in WASA$_2$ ad-hoc modified process instances are excluded from further type changes Therefore ADEPT provides the only comprehensive framework in the context of process adaptivity.

```
Method getDirectPred_byCtrl(Integer inID) of ModifiedTemplate)

  public ActivityList getDirectPred_byCtrl(Integer inID){
      //Are there any information about the activity with ID inID
      //in the delta-layer?
   Activity currActivity = (Activity) activityID2ActivityMap.get(inID);
   if (currActivity!=null){
       //There are information about the activity with ID inID
       //in the delta-layer.
       //So answer the question with this information.
       return currActivity.getCtrlPred(); //answer the question
   }
   else{
       //There is no information about activity with ID inID
       //in the delta-layer, so delegate the question
       //to the original template which is modified by this delta-layer.
       return originalTemplateVersion.getDirectPred_byCtrl(inID);
   }
}
```

Figure 7. Code Fragment

There are only few approaches dealing with an efficient implementation of advanced process management functionality (e.g., [14, 4]). The functionality of existing prototypes are mostly restricted to buildtime and runtime simulations. Using such simulations it can be shown that the particular functionality is realized in principle, but not how it can be implemented in a performant way in practice. Our system is one of the few available research prototypes for adaptive, high-performance process management [7]. In order to gain usability experience we have deployed this system to different research groups [5, 1]. They have used it as platform for realizing advanced process-aware information systems in domains like e-health and e-business. The experiences have helped us to develop new system components with advanced programming interfaces.

Usually, commercial WfMS do not allow change propagation to in-progress instances when a WF schema is modified at the type level. Instead, simple versioning concepts are used to ensure that already running instances can be finished according to the old schema. One exception is offered by Staffware [12]. However, there are several critical aspects arising in this context. For example, running activities can be deleted without any user information. If the deleted activity is finished all returned results disappear.

7 Summary and Outlook

We have introduced a compact internal representation for process templates and instances based on which the migration of unbiased and biased instances can be easily realized.

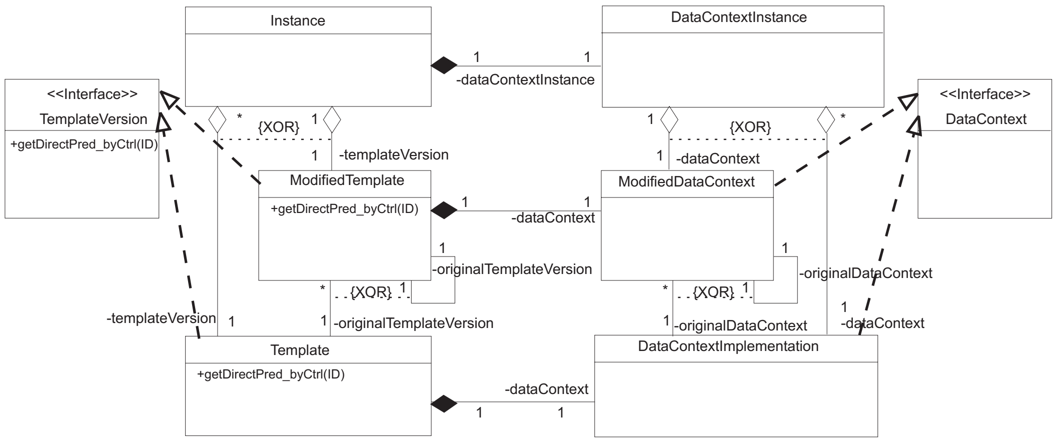


Figure 6. Class Diagram Prototype

The migration itself is quick and efficient since it is based on a simple re-linking of references in most cases. Furthermore the necessary storage space could be reduced compared to other approaches since, for example, process template objects can be reused and the delta layer only stores those parts of the process graph which have been modified. With these achievements the applicability of adaptive process management becomes highly realistic in practice. Further on, the delta layer and migration approach can be transferred to other process meta models which store information about already executed activities [8]. Due to lack of space we have omitted the results of our simulations which will be subject of further publications.

The practical applicability of the presented appproach can be still increased. For example, a coordination of concurrent accesses on process templates and instances by adequate locking mechanism is important. Additionally, we want to address the question how to conduct an instance migration if some of the instances are partitioned and controlled by different process engines. Finally, in this paper, the approach is restriced to disjoint process type and instance changes whereas the handling of overlapping process changes is to be analyzed as well. All these considerations are taken into account within the implementation of the new process management system ADEPT2 within the AristaFlow project (www.aristaflow.de).

References

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

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

[3] B. Karbe. Flexible workflow management with proMInandD. In *CSCW – Computer Supported Cooperative Work: Information Systems for decentralized enterprise structures*. Addison-Wesley, 1994. (in German).

[4] K. Kochut, J. Arnold, A. Sheth, J. Miller, E. Kraemer, B. Arpinar, and J. Cardoso. IntelliGEN: A distributed workflow system for discovering protein-protein interactions. *DPD*, 13(1):43–72, 2003.

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

[6] M. Reichert and P. Dadam. ADEPT$_{flex}$ - supporting dynamic changes of workflows without losing control. *JIIS*, 10(2):93–129, 1998.

[7] M. Reichert, S. Rinderle, U. Kreher, and P. Dadam. Adaptive process management with ADEPT2. In *ICDE'05*, pages 1113–1114, 2005.

[8] S. Rinderle, M. Reichert, and P. Dadam. Correctness criteria for dynamic changes in workflow systems – a survey. *DKE*, 50(1):9–34, 2004.

[9] S. Rinderle, M. Reichert, and P. Dadam. Disjoint and overlapping process changes: Challenges, solutions, applications. In *CoopIS'04*, pages 101–120, 2004.

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

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

[12] Staffware. *Staffware Process Suite Version 2 – White Paper*. Staffware PLC, Maidenhead, UK, 2003.

[13] W. v.d. Aalst and T. Basten. Inheritance of workflows: An approach to tackling problems related to change. *Theoret. Comp. Science*, 270(1-2):125–203, 2002.

[14] M. Weske. Object-oriented design of a flexible workflow management system. In *ADBIS98*, pages 119–131, 1998.

[15] M. Weske. Formal foundation and conceptual design of dynamic adaptations in a workflow management system. In *HICSS-34*, 2001.