Proc. Workshop Informatik '99, P. Dadam, M. Reichert (eds.): Enterprise-wide and Cross-enterprise Workflow Management: Concepts, Systems, Applications, Paderborn, Germany, October 1999
http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-24/

Enterprise-Wide and Cross-Enterprise Workflow Management: Challenges and Research Issues for Adaptive Workflows

Manfred Reichert, Thomas Bauer, Peter Dadam
Dept. Databases and Information Systems, University of Ulm
{reichert, bauer, dadam}@informatik.uni-ulm.de
http://www.informatik.uni-ulm.de/dbis

Abstract

The paper discusses important challenges and research issues for adaptive workflow management systems (WfMS), especially if they shall be applied to enterprise-wide applications. It shows that an adaptive WfMS must provide support for different kinds of (dynamic) workflow (WF) changes in order to be applicable to a broader spectrum of processes. In this context, both, requirements for WF schema evolution and issues related to ad-hoc changes of individual WF instances are discussed. A particulary interesting aspect, which is described in more detail, is how to combine such dynamic changes with a distributed execution of workflows, taking into account performance issues. For enterprise-wide and cross-enterprise workflows, the distributed execution of workflows may be attractive due to several reasons.

1 Introduction

Workflow Management Systems (WfMS) offer a promising technology that has the potential to change the implementation of enterprise-wide and cross-enterprise application systems significantly. In fact, only this technology makes it possible to realize process-oriented application systems in larger quantities and at affordable costs. To adequately support enterprise-wide and cross-enterprise applications, a WfMS must not only cope with a large number of users and concurrently active workflow (WF) instances, but it must also cover a broad spectrum of processes. In this context, high flexibility, maintainability, and scalability are essential requirements. In the ADEPT[1] project we, therefore, have spent a lot of effort especially on these subjects [BaDa97, BaDa98, BaDa99a, DaRe98, ReDa98, RHD98].

In this paper we discuss important challenges and research issues for adaptive WfMS [CCPP98, JoHe98, RHD98, Sieb98, Wesk98], especially if they shall be applied to enterprise-wide and cross-enterprise processes. We denote a WfMS as *adaptive* if it supports run-time changes of in-progress WF instances. Such adaptations become necessary, for example, when new tasks have to be added to a WF instance at run-time or when pre-defined control or data flow dependencies between WF activities have to be dynamically changed [ReDa98]. In such cases the execution of the WF instance must be (partially) suspended, the modification of its WF instance graph, its attributes and/or its state be performed, and afterwards its execution be resumed. The paper is organized as follows: In Section 2 we show that an adaptive WfMS must support different kinds of (dynamic) WF changes in order to be applicable to a broader spectrum of processes. We discuss major requirements for the evolution of WF schemes and for the concomitant run-time adaptation of corresponding WF instances. Furthermore we deal with issues related to ad-hoc changes of individual WF instances and we discuss problems that arise from the integrated support of both kinds of changes. A particularly interesting aspect is discussed in Section 3, namely how to combine dynamic WF changes with a distributed execution of workflows, taking into account performance issues. For enterprise-wide and cross-enterprise workflows, the distributed execution of workflows may be advantageous in several respects. In Section 4 we summarize further issues. The paper concludes with a summary and an outlook on future work.

[1] ADEPT stands for Application Development based on Encapsulated Pre-Modeled Process Templates.

2 Workflow Schema Evolution and Support of Ad-hoc Deviations

Enterprise-wide and cross-enterprise business processes may change rather frequently [RHD98]. Once an application system has been made to behave strictly process-oriented, it must be adjustable to process changes and to evolving organizational structures very quickly and at reasonable costs. Such adaptations may affect WF templates or other aspects of the process-oriented application system (e.g., the model capturing organizational entities). In any case, they must be performed without causing inconsistencies between the different models of the process-oriented application (e.g., faulty references).

In order to increase the robustness of the WF-based application system, it is very important to detect and to eliminate design and implementation errors as early as possible. Potential problems introduced by the concept of process-orientation are that the process may block itself during execution, may never be able to enter certain branches of the WF graph, may not meet temporal constraints (e.g., minimal or maximal time distances between the execution of two activities), or may invoke activity programs with missing or incomplete input data, for example. To avoid such problems, an adequate formal basis must be provided for WF modeling. On the one hand such a formal model must allow WF designers to capture business processes as naturally as possible and in a way understandable to the user. On the other hand, it must enable formal verifications for the absence of deadlocks, the proper termination of the flow, the correctness of the data flow, or the consistency of a time schedule, to name a few examples. With respect to consistency issues, commercial WfMS show severe limitations that disqualify them for the support of sophisticated WF applications. – In the ADEPT project we have exhaustively investigated WF modeling issues and we have developed the $ADEPT_{base}$ model as a formal basis (for details see [ReDa98]).

The use of a formal WF model and the provision of corresponding analysis algorithms are also prerequisites for keeping maintenance costs low. Structural modifications of a WF schema (*WF type changes*), like the insertion, deletion, or shift of process steps, must not lead to undesired side-effects and program failures. Instead, the system must assist the WF designer in detecting all concomitant schema modifications, which become necessary in order to guarantee a robust and correct execution for (new) WF instances of the changed WF type. To maintain the correctness of a WF schema, however, is only one of several requirements for *WF schema evolution*. Changing a WF type does also mean, in general, that we still have WF instances active in the system that follow the "old" schema. Especially for long-running processes, it is desirable to automatically adapt them to the new process structure as well (as far as this is possible). As the WF instances may be in different states, however, the respective schema modifications may be propagated only to a subset of them [CCPP98, JoHe98]. For example, an already completed activity must not be deleted from a WF instance graph. For the same reason, a new activity cannot be inserted into a region of a WF instance graph, which has already been processed. For modifications of a loop body, another important aspect has to be considered: A change, which is not valid in the current state of a loop iteration, may become applicable when the loop enters its next iteration. In such a situation, it is favorable to record the change and to apply it when this loop back will be performed. In any case, the WfMS must ensure a correct and stable execution behavior afterwards. Again, the provision of a formal model can help a lot, since it will allow the system to check whether a particular change is valid in the current state of a WF instance or whether it is not. In the former case, the WfMS must also adapt the state of the WF instance after its modification [ReDa98]. Since there may be a large number of active instances of the same WF type in the system, the necessary checks and adaptations must be performed efficiently and automatically by the WfMS. Finally, the WfMS must cope with the co-existence of WF instances following either the old or the new schema. This presumes appropriate concepts and mechanisms for the versioning of WF schemes and for the adequate representation of WF instances [JoHe98].

Things become even more complicated if *ad-hoc deviations* from the pre-defined WF schema must be supported at the instance level. Examples for such run-time deviations are the deletion of one or more WF steps (e.g., to skip their execution) from a WF instance graph or the dynamic insertion of a new activity. Such interventions into the control of a WF instance may become necessary to handle *exceptional situations* [MüRa99, RHD98] or to model parts of the WF that cannot be completely pre-defined at build-time (*late modeling*). In our experience, due to combinatorial reasons, for more complex processes it is neither possible nor cost-effective to capture all possible task sequences and all exceptions in advance. But even for simple workflows, unpredictable situations may occur that require ad-hoc deviations from the pre-planned process at run-time [RHD98]. Such ad-hoc deviations must not lead to consistency problems or to an unstable system behavior (e.g., program failures due to the invocation of an activity with missing input data). This means, in fact, that one has to show that none of the correctness guarantees or assertions,

which have been achieved by formal checks at build-time, are being violated by the introduction of ad-hoc changes at run-time. Instead the system must assist the user in detecting all concomitant adaptations that are necessary to ensure a robust and correct execution behavior afterwards. To achieve this, all aspects of the WF model (control flow, data flow, temporal constraints, etc.) and their possible interactions have to be taken into consideration. For example, when deleting a step from a WF instance graph, the data flow may have to be adapted as well in order to preserve its correctness. Due to the numerous interdependencies that exist between the different aspects of a WF model, with increasing expressiveness of the used WF meta model, it becomes more and more difficult to handle ad-hoc changes in a correct and consistent manner. Therefore, it is extremely important to hide details of a change (like e.g., the complexity arising from the re-mapping of input and output parameters of WF activities) from the user. Again, an adequate formal model can help a lot. Such a model must offer a clear semantics, which enables the system to argue on correctness issues and which covers all possible cases, either by supporting the desired action or by rejecting it (no implementation holes). Ideally, such a model enables the system to restrict the necessary analysis to a portion of the WF instance graph in most cases and, by doing so, to perform the necessary checks very efficiently. Further details and a discussion of other issues related to dynamic WF changes can be found in [ReDa98, RHD98].

Generally, an adaptive WfMS has to consider both kinds of changes, i.e., it must support adaptations of a potentially large number of WF instances to modifications of their WF type as well as ad-hoc changes of single WF instances. To adjust in-progress WF instances to a type change, however, is a non-trivial problem if ad-hoc changes have to be supported as well. In this context, the main problem arises from the fact that the instances of a WF type may not only be in different states when a type change shall be propagated, but may also have a process structure (represented by the WF instance graph) that deviates from that of their original schema. While in some cases this may not affect the applicability of the type change to a particular instance, in other cases there may exist unresolvable conflicts between previously applied ad-hoc changes of a WF instance graph and the type change. Well, how can WF type changes be efficiently propagated to a potentially large number of WF instances under these conditions? In principle, for each instance of a modified WF type, the system must check whether the type change is currently applicable to the corresponding WF instance graph or not. As a large number of WF instances may have to be adapted to the type change, the necessary checks should be automatically performed by the WfMS without causing a performance penalty. Costly user interactions, which are also not tolerable due to robustness and correctness reasons, must be avoided.

The simplest solution would be to disallow the propagation of a WF type change to all WF instances whose execution graph was "locally" modified due to some exceptional situation. This approach could be simply handled, but it is rather unsuited for the support of long-running processes [RHD98]. Another solution would be to completely reapply all formal checks for each of these WF instances. For performance reasons, however, this approach would be disadvantageous, especially if a large number of instances have to be adapted. The challenge, therefore, is to provide efficient mechanisms, which enable the system to propagate WF type changes to a large number of WF instances, independently from whether they possess a process structure that differs from that of their original schema or not. To achieve this goal, for all WF aspects that may be subject of a change, proper conflict relations have to be defined. On top of this, it must be possible to detect potential conflicts between concurrent changes by means of simple conflict tests. For example, the WfMS must be able to efficiently check, under which conditions conflicting changes of the data flow of a WF instance may lead to inconsistencies or errors in the sequel. Having a closer look at the nature of ad-hoc changes, however, this approach is not as simple as it looks like at first glance. The reason for this is that ad-hoc changes of a WF instance graph may have a different durability. While some of them may be permanently valid until the termination of the instance, others may be only of temporary nature and must therefore be removed from the WF instance graph at the occurrence of corresponding events (e.g., when a loop back is performed).

3 Scalability Issues in Adaptive Workflow Systems

Enterprise-wide and cross-enterprise WF-based applications are characterized by a large number of users and concurrently active WF instances. As a consequence the WF servers have to cope with a very high load. Already the processing of a single WF activity may require the transfer of multiple messages between the WF server and its clients, e.g., to transmit input and output data of the activity, to update worklists, to invoke activity programs, or to exchange application data between activity programs and external

data sources. Obviously, this amount of communication may overload both, WF servers and subnets, if the number of concurrently active WF instances increases. In addition, the organizational units, which participate in a cross-organizational WF, are often connected by slow wide area networks. The load of the communication system, therefore, is extremely critical for the performance of the system. In order to keep communication local within one network segment as far as possible, in many cases, it is advantageous to dynamically migrate the control of in-progress WF instances to a new WF server in another network segment. Apart from this, due to autonomy reasons, a business company will not always tolerate that its activities are controlled by the WfMS of a foreign company. For cross-organizational workflows this means, that they cannot be completely executed by the WfMS of a single company, but may have to be controlled by distributed, (potentially) heterogeneous WfMS.

At least for some WF classes, it would be very attractive to distribute WF control onto several servers. In the WF literature, a number of distribution models have been proposed [BaDa99a, Muth98, ShKo97]. In the ADEPT project, we have also developed such a model, which is called $ADEPT_{distribution}$ [BaDa97, BaDa98]. It supports the WF designer in partitioning a WF schema and in distributing the different partitions across several WF servers. This distribution is done in a way that prevents single system components (WF servers, network segments, and gateways) from becoming overloaded at run-time. Like other distribution models, so far, $ADEPT_{distribution}$ has assumed that the structure of a WF instance graph is not changed during run-time. As shown in Section 2, this assumption does not hold for adaptive WfMS. Which additional issues arise if dynamic WF changes have to be supported in such environments? And how can we ensure that the advantages of a distributed WF control do not get lost if dynamic changes have to be supported as well? In order to be able to discuss relevant issues, first of all, we sketch the basic ideas of the $ADEPT_{distribution}$ approach in Section 3.1. Based on this model, in Section 3.2 we discuss important issues arising from the integrated support of dynamic WF changes and distributed WF control.

3.1 Distributed Workflow Control in $ADEPT_{distribution}$

At build-time, the schema of a WF is divided into several *partitions*. Each partition is assigned to a WF server, which controls the activities of this partition during run-time. For this purpose, at each server a copy of the (complete) WF schema is stored. If a WF instance reaches a transition between two partitions, its control *migrates* to the WF server of the target partition. Before this WF server may proceed with the execution of the WF instance, WF control data and WF relevant data have to be transmitted. Activities from parallel execution branches can be controlled by different WF servers in this approach, so that more than one WF server may be involved in the current execution of a particular WF instance. In order to keep communication costs low, generally, $ADEPT_{distribution}$ does not require that these WF servers synchronize with each other. An example of a WF instance graph, which is controlled by multiple WF servers, is depicted in Figure 1. In the current state, two servers – S_2 and S_3 – participate in the processing of this instance. Note that S_2 does not know the execution state of the lower branch, i.e., it does not know whether this branch is still controlled by S_1, the control has already been migrated to S_3 (as in the example shown), or it has been given back to S_1 (in order to control the partition P_4). Conversely, S_3 has no knowledge about the processing state of the partition P_2, which is controlled by S_2.

$ADEPT_{distribution}$ partitions a WF schema in a way that minimizes the communication load of the system at run-time. For this purpose the WF designer is supported by a set of algorithms, which allow him or her to

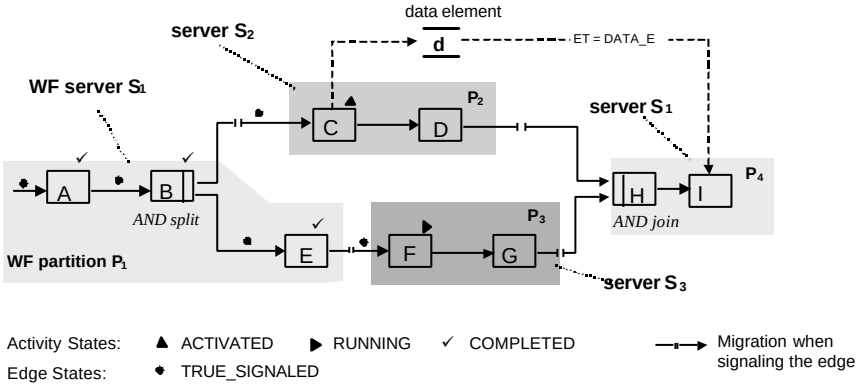

Figure 1: *Distributed execution of workflows in ADEPT*

automatically calculate optimal server assignments for the activities of a given WF schema – a partition then consists of a subgraph of which the activities are assigned to the same WF server. In doing so, it is assumed that a WF activity may be controlled by an arbitrary WF server of the WfMS, unless the WF designer explicitly excludes it from the control of this activity. To determine optimal server assignments, we use a formal cost model that allows us to evaluate the quality of a selected distribution. Among other things, this model considers costs for the transfer of parameter data, for the update of worklists, and for the migration of WF instances. In most cases, a WF activity will be controlled by a WF server, which is located nearby its potential actors. Since migrations do also generate communication costs, however, they are only used if they improve the communication behavior of the system. Details on this work and descriptions of the developed algorithms for partitioning WF schemes can be found in [BaDa98]. These algorithms do also consider so-called *variable server assignments*. Unlike static server assignments, where the WF servers that control the activities of a WF, are completely pre-defined at build-time, variable server assignments enable the WfMS to select the WF server of a particular WF activity dynamically, depending on the control data of preceding activity executions[2] (see [BaDa99b] for details). This approach contributes to improve the communication behavior of the system, especially if *dependent actor assignments*[3] are used [BaDa98, BaDa99b].

3.2 Dynamic Workflow Changes and Distributed Workflow Execution

In the ADEPT project [DaRe98] we have developed the $ADEPT_{flex}$ calculus, which provides a complete and minimal set of change operations for dynamic WF modifications in process-oriented WfMS [ReDa98, RHD98]. Most of the concepts offered by $ADEPT_{flex}$ have been prototypically implemented in the ADEPT-WfMS. $ADEPT_{flex}$ uses the same formal WF meta model as the $ADEPT_{distribution}$ approach described in the previous section. So far, $ADEPT_{distribution}$ does not consider aspects related to dynamic WF changes. Conversely, up to now $ADEPT_{flex}$ has assumed that a WF instance is controlled by one central WF server. From a logical point of view, this assumption is helpful for verifying the correctness of a change [ReDa98]. The deletion of a WF activity from a WF instance graph, for example, may lead to missing input data of subsequent activities. In order to preserve a proper data flow, these data dependent steps either have to be deleted as well (cascading deletion) or the correctness of the data flow specification has to be restored by additional adaptations (e.g., by adding so-called data provision steps to the WF instance graph) [RHD98]. If a WF is controlled by multiple servers, such a change may affect more than one server. In the WF instance graph from Figure 1, for example, the deletion of activity C (change within the partition P_2) would entail concomitant changes of the partition P_4, like the deletion of the data-dependent activity I or the addition of a preceding data provision step to I. Generally, a structural change may affect several partitions of a WF instance graph that may be controlled by different servers.

How can dynamic WF changes be supported in a distribution model like $ADEPT_{distribution}$, without loosing the advantages offered by the distributed control of workflows? First of all, it is important to minimize the synchronization effort, which becomes necessary when a dynamic change is performed. A naive solution would be to synchronize all servers that have already been involved in the processing of the WF instance or that may become active in the future. Generally, such a strict synchronization is not required and – in the case of variable server assignments (cf. Section 3.1) – it is also not possible. Instead, it is more favorable to synchronize only those WF servers, which are currently involved in the processing of the instance. If a WF instance graph does not contain AND-splits (i.e., there are no parallel execution branches), at each point in time only one WF server is responsible for the control of this instance. Consequently, no additional synchronization effort results. For parallel execution branches, however, several servers may be concurrently involved, so that a dynamic WF change may require a synchronization between them.

We will illustrate this by a simple example. Taking the change operations provided by $ADEPT_{flex}$ [ReDa98] and the WF instance graph from Figure 1, it is possible to dynamically insert a new activity X into this graph, of which the execution may not start before step F is completed and must terminate before activity D can be activated. Internally, this change is realized by the application of a well-defined set of graph transformation and graph reduction rules [ReDa98]. After its insertion, the step X constitutes a new branch of the parallel branching defined by the AND-split B and the corresponding AND-join H. The

---

[2] An example of a variable server assignment may be "*Server*(A_2)*:= Domain(Actor*(A_1)*)*". This means that activity A_2 is dynamically assigned to the server that is located in the domain of the actor who worked on activity A_1.

[3] With this, we mean logical assignments like "Activity A_2 must be executed by the same actor who has worked on the preceding activity A_1".

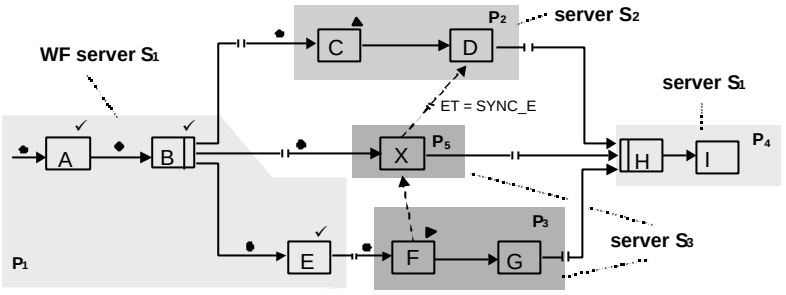

Figure 2: *The same instance graph after performing a dynamic change.*

desired execution order (X after F, X before D) is enforced by the additional insertion of the two synchronization edges[4] $F \rightarrow X$ and $X \rightarrow D$ (cf. Figure 2). Assume that this change is initiated by a client connected to the WF server S_3. Obviously, the desired modifications are only allowed, as long as D has not been started. In order to check this, first of all, the WF server S_3 must retrieve the current state of activity D from the WF server S_2 (Note that S_3 itself does not know the state of the upper branch). Conversely, the change must not only be applied to the WF instance graph stored at S_3, but it must also be considered for the copy of this graph stored at S_2. The latter becomes necessary in order to ensure that S_2 delays the execution of D until the newly inserted step X will be completed. WF servers that may become active in the future (like S_1 in our example) must not be immediately notified about the change. Instead, it is sufficient to transmit the corresponding information, when the control of the WF instance migrates to this server. This approach, however, causes additional communication costs for "normal" migrations, which should be kept as minimal as possible.

For distributed workflows the correct handling of dynamic WF changes is a non-trivial problem. In the following, we discuss important issues that arise in connection with the approach described above:

- How must a migration be performed, if the corresponding WF instance graph has been changed? – As already mentioned, the target server of a migration may only possess an old version of the schema of the WF instance. On the one hand, it should be avoided that a complete description of the modified WF instance graph is transferred to the target server. On the other hand, the new process structure must be made available at this server, in order to correctly proceed with the flow and to provide a proper basis for subsequent changes. The use of execution and change histories, which already exist in $ADEPT_{flex}$ [ReDa98], offers a promising approach for reducing the communication amount. In the example from Figure 2, when migrating the control from the S_2 (or S_3) to S_1 (S_1 controls the partition P_4), in addition to WF control data and WF relevant data, the corresponding entries from the change history have to be transmitted as well. The WF server S_1 must then apply the modifications to its local copy of the WF instance graph, before the execution may proceed. As far as possible, the redundant transfer of information about a change should be avoided.
- Is it possible to perform a dynamic WF change without synchronizing all WF servers currently involved in the control of the WF instance? – Similarly to the execution of parallel branches, it is desirable to perform dynamic changes of single branches without costly communication with other WF servers. As shown in the example, for changes that affect multiple partitions of a WF instance graph this will not always be possible. Instead, it must be ascertained for how long and in which mode the WF instance has to be locked at the respective WF servers. As far as possible, long-term locks should be avoided, so that the WF execution is not blocked unnecessarily long. There are numerous examples for dynamic WF changes, for which such a strict synchronization does not become necessary. Taking the WF instance graph from Figure 1, for example, the WF server S_2 might insert a new activity between C and D or delete the activity C (incl. the deletion of the outgoing data edge connected to d and the deletion of the data-dependent activity I) without synchronizing this change with S_3. This is possible, since S_2 does not require information about the state of the lower branch in order to apply the change. Conversely, S_3 must not be informed about the modification of the upper branch in order to proceed with the control. Such local modifications do occur often in practice. Therefore it does make

[4] Synchronization edges are a special edge type of our graph-based WF meta model $ADEPT_{base}$ [ReDa98]. They can be used for modeling different kinds of "wait-for" situations between activities from parallel execution branches.

sense to differentiate between different classes of changes with respect to a WF instance graph (e.g., local change of a partition, changes of several partitions controlled by the same WF server, changes of partitions controlled by different WF servers, etc.) and to offer optimized procedures for their application. Generally, if a WF server wants to perform a dynamic change, it must determine which information have to be retrieved from which other servers to perform the desired modifications and which servers must be informed about the change afterwards.

- Which adaptations will become necessary regarding server assignments, if a WF instance graph is structurally changed? – For newly inserted activities, for example, appropriate WF servers must also be assigned to. On the one hand, it seems to be attractive to use the same distribution algorithms that are applied at build-time [BaDa98] (Note that this may lead to changed server assignments of other WF activities as well). On the other hand, if these run-time calculations are too costly, simpler approaches for the (dynamic) assignment of servers may be favorable. In the example from Figure 2, the newly inserted activity X has been assigned to the WF server S_3, which initiated the change.
- Assume that a WF server *S*, which wants to apply a dynamic change to a WF instance, controls a particular partition from a parallel branch of the WF instance graph. How can this WF server find out, which other WF servers are currently involved in the processing of this WF instance (see above)? – The main problem in this context arises from the fact that, generally, *S* has no information about the state of activities from parallel branches (cf. Section 3.1), unless *S* controls these activities itself or has obtained additional information when an incoming synchronization edge was signaled [ReDa98]. The server *S* does also not know, whether the processing of a parallel branch is in a state before or after a migration. Using the information locally available, *S* does not know, which other WF servers are currently active. With respect to the WF instance graph from Figure 1, this applies to the WF server S_2, for example. This server does not know, whether the lower branch is currently controlled by S_1 or S_3. This problem even gets worse if variable server assignments (cf. Section 3.1) are used.
- Which additional problems arise in connection with variable server assignments? – If the server selection of an activity A_2 depends on the execution of a preceding activity A_1 (cf. Section 3.1), it must be ensured that a WF server can be assigned to A_2, even if A_1 will be dynamically removed from the WF instance graph. In this context, it seems to be attractive to apply similar exception handling mechanisms, as they have been used in $ADEPT_{flex}$ to guarantee the provision of activity input parameters in the case of structural WF changes [ReDa98].

Although we have used the ADEPT workflow model for illustration purposes, most of the issues discussed, do also apply to other flexible WF models.

4 Further Issues

We shortly discuss two other basic requirements, which are essential for the flexible support of large-scale WF-based application systems.

4.1 Semantic Rollback of Workflows in Adaptive WfMS

An adaptive WfMS must offer adequate concepts to cope with semantic failures of single WF activities at run-time. During the last years, a number of advanced transaction concepts have been developed [Alon96, JaKe97, Leym95]. With few exceptions [LiPu98, MüRa99], however, most of them are based on the assumption that the flow structure of the transaction or the workflow, respectively, is completely known at build-time. As shown in Section 2, this assumption is only valid for completely static workflows and does not apply to adaptive WfMS. The proposed concepts are therefore not suited for the flexible support of a broader spectrum of workflows.

Normally, for the transactional support of long-running processes, we cannot strictly enforce the atomicity of the whole composite transaction (workflow). Instead, intermediate results may become visible and may be modified by other transactions. As a consequence ordinary transaction rollback is no longer applicable in this context. An extended transaction mechanism must allow the WF designer to define compensation steps for WF activities in order to enable the WfMS to support some kind of "logical rollback" for already completed and committed activities, if semantic failures occur at run-time [JaKe97]. One important problem in this context, which has not been satisfactorily solved in the literature so far, arises from the fact that the kind of compensation of a WF step may depend on the current state of the WF instance, on the point in time a failure occurs, or on other factors. Obviously, these dependencies compli-

cate the definition of appropriate “recovery spheres” [Leym95] and they also complicate application development. It will be one of the key factors for the success of adaptive WF technology, whether it will be possible to develop sophisticated concepts for describing compensation spheres, for implementing WF activities and their (perhaps different) compensation steps, and for dealing with dynamic cases and dynamic WF changes (see above). In connection with dynamic changes, among other things, it must be considered that the definition of compensation spheres may have to be adapted when the structure of the WF instance graph is dynamically modified.

In addition to these requirements, for enterprise-wide and cross-enterprise workflows numerous other problems have to be solved. It has to be specified, for example, under which conditions an actor is authorized to reset WF activities that are controlled by the WfMS of a foreign company. In this context, we must not only deal with technical issues (e.g., support of different commit protocols), but we must also consider numerous other aspects (privacy, documentation, differences in law for companies from different countries, etc.). All these issues must be carefully analyzed and understood, taking into consideration all the non-trivial interdependencies with other features of the system, like e.g., the support of different kinds of changes (cf. Section 2) or the distributed execution of workflows (cf. Section 3).

4.2 Security Issues in Adaptive WfMS

Very often, a WfMS processes data for which high standards must be set with respect to privacy and data security. Generally, in a WfMS the access rights of a person are determined by the roles or functions he or she may take. Already for the static (“unflexible”) case, however, the definition of appropriate roles and organizational rules for substitutions may become very complex and poses high requirements for the WfMS, especially when looking at the maintenance of organizational models. For the “flexible” case, in addition, we must ensure that dynamic changes of a WF instance do not lead to “security gaps”. For cross-organizational workflows we have to deal with the difficulty that WF changes, which are reasonable from the point of view of a single organizational unit, may be in conflict with superior process goals (e.g., temporal constraints, quality requirements, etc.). For these reasons, it must be possible to control, at a very fine level of granularity, which persons or roles may perform which changes with respect to a particular WF instance or with respect to instances of a particular WF type.

Most of the challenges discussed in the previous sections apply to "normal" WF applications as well. Nevertheless, in the context of enterprise-wide and cross-enterprise workflows they have a special relevance and an extreme importance.

5 Summary and Outlook

To be applicable to enterprise-wide and cross-enterprise workflows, a process-oriented WfMS must cover a broad spectrum of processes and it must support different kinds of dynamic changes at run-time. These changes must be accomplished in an efficient and secure manner and without affecting the robustness of the system. This must also apply if the number of users and concurrently active WF instances is high. In this context, it is important that the "normal" WF execution is not affected too much by the additional features of an adaptive WfMS. In the ADEPT project the described challenges have been exhaustively investigated. Formal issues and the basic principles of dynamic WF changes are well understood in the meantime. Furthermore they have been prototypically implemented in the ADEPT-WfMS [ReDa98, RHD98]. We also work on issues related to WF schema evolution and their interrelation with ad-hoc deviations. The same applies to scalability issues [BaDa97, BaDa98, BaDa99a] as well as to some other aspects (for an overview see [DaRe98]). An important next step will be, to interweave the different concepts and to realize common implementations.

References

[Alon96] Alonso, G.; Agrawal, D.; El Abbadi, A.; Kamath, M.; Günthör, R.; Mohan, C.: *Advanced Transaction Models in Workflow Contexts*. Proc. 12th Int'l Conf. on Data Engineering. New Orleans, February 1996

[BaDa97] Bauer, T.; Dadam, P.: *A Distributed Execution Environment for Large-Scale Workflow Management Systems with Subnets and Server Migration*. Proc. CoopIS '97, Kiawah Island, June 1997, pp. 99-108

[BaDa98] Bauer, T.; Dadam, P.: *Variable Migration of Workflows in ADEPT* (in German). Ulmer Informatik-Berichte, Nr. 98-09, September 1998

[BaDa99a] Bauer, T.; Dadam, P.: *Distribution Models for Workflow Management Systems – Classification and Simulation* (in German). Ulmer Informatik-Berichte, Nr. 99-02, University of Ulm, March 1999

[BaDa99b] Bauer, T.; Dadam, P.: *Efficient Distributed Control of Enterprise-Wide and Cross-Enterprise Workflows.* Proc. Workshop Enterprise-wide and Cross-enterprise Workflow Management: Concepts, Systems, Applications, 29. Jahrestagung der GI (Informatik'99). Paderborn, October 1999

[CCPP98] Casati, F.; Ceri, S.; Pernici, B.; Pozzi, G.: *Workflow Evolution*. Data & Knowledge Engineering, 24(3), Jan. 1998, pp. 211-238

[DaRe98] Dadam, P.; Reichert, M.: *The ADEPT WfMS Project at the University of Ulm.* Presented at the 1st European Workshop on Workflow and Process Management (WPM'98) – "Workflow Management Research Projects", Zürich, October 1998

[JaKe97] Jajodia, S.; Kerschberg, L. (Eds.): *Advanced Transaction Models and Architectures.* Kluwer Academic Publishers, 1997

[JoHe98] Joeris, G.; Herzog, O.: *Managing Evolving Workflow Specifications.* Proc. CoopIS'98, New York, August 1998.

[Leym95] Leymann, F.: *Supporting Business Transactions via Partial Recovery in Workflow Management Systems*, Proc. Datenbanksysteme in Büro, Technik und Wissenschaft (BTW '95), Dresden, March 1995, pp. 51-70

[LiPu98] Liu, L.; Pu, C.: *Methodical Restructuring of Complex Workflow Activities.* Proc. 14th Int'l Conf. On Data Engineering (ICDE'98), Orlando, Florida, February 1998, pp. 342-350.

[MüRa99] Müller, R.; Rahm, E.: *Rule-Based Dynamic Modification of Workflows in a Medical Domain.* Proc. Datenbanksysteme in Büro, Technik und Wissenschaft (BTW '99), Freiburg, March 1999, pp. 429-448.

[Muth98] Muth, P.; Wodtke, D.; Weißenfels, J.; Kotz-Dittrich, A.; Weikum, G.: *From Centralized Workflow Specification to Distributed Workflow Execution.* Journal of Intelligent Information Systens, Special Issue on Workflow Management Systems, Vol. 10, March/April 1998, pp. 159-184

[ReDa98] Reichert, M.; Dadam, P.: *ADEPT$_{flex}$ – Supporting Dynamic Changes of Workflows Without Losing Control*. Journal of Intelligent Information Systems, Special Issue on Workflow Management Systems, Vol. 10, March/April 1998, pp. 93-129

[RHD98] Reichert, M.; Hensinger, C.; Dadam, P.: *Supporting Adaptive Workflows in Advanced Application Environments.* Proc. EDBT-Workshop on Workflow Management Systems, Valencia, March 1998, pp. 100-109

[ShKo97] Sheth, A.; Kochut, K.: *Workflow Applications to Research Agenda: Scalable and Dynamic Work Coordination and Collaboration Systems*. Proc. NATO Advanced Study Institute on Workflow Man Sys and Interop. Istanbul, August 1997, pp. 12-21

[Sieb98] Siebert, R.: *An Open Architecture for Adaptive Workflow Management Systems*. Proc. 3rd Biennial World Conf on Integrated Design and Process Techn, Vol 2, Berlin, July 1998, pp. 79-85.

[Wesk98] Weske, M.: *Flexible Modeling and Execution of Workflow Activities*. Proc. 31st Hawaii Int'l Conf. on System Sciences (HICSS–31), Software Technology Track (Vol VII), 1998, pp. 713–722