Enabling Adaptive Process-aware Information Systems with ADEPT2

Manfred Reichert and **Peter Dadam**

Institute of Databases and Information Systems, University of Ulm, Germany
Web: www.uni-ulm.de/in/iui-dbis/
E-Mail: {manfred.reichert, peter.dadam}@uni-ulm.de

ABSTRACT

In dynamic environments it must be possible to quickly implement new business processes, to enable ad-hoc deviations from the defined business processes on-demand (e.g., by dynamically adding, deleting or moving process activities), and to support dynamic process evolution (i.e., to propagate process schema changes to already running process instances). These fundamental requirements must be met without affecting process consistency and robustness of the process-aware information system. In this chapter we describe how these challenges have been addressed in the ADEPT2 process management system. Our overall vision is to provide a next generation technology for the support of dynamic processes, which enables full process lifecycle management and which can be applied to a variety of application domains.

INTRODUCTION

In today's dynamic business world the economic success of an enterprise increasingly depends on its ability to quickly and flexibly react to changes in its environment. Generally, the reasons for such changes can be manifold. As examples consider the introduction of new regulations, the availability of new medical tests, or changes in customers' attitudes. Companies and organizations therefore have recognized business agility as prerequisite for being able to cope with changes and to deal with emerging trends like business-on-demand, high product and service variability, and faster time-to-market (Weber, Rinderle, & Reichert, 2007).

Process-aware information systems (PAISs) offer promising perspectives in this respect, and a growing interest in aligning information systems in a process-oriented way can be observed (Weske, 2007). As opposed to data- or function-centered information systems, PAISs separate process logic and application code. Most PAISs describe process logic explicitly in terms of a *process template* providing the schema for handling respective *business cases*. Usually, the core of the *process layer* is built by a process management system which provides generic functions for modeling, configuring, executing, and monitoring business processes. This separation of concerns increases maintainability and reduces cost of change (Mutschler, Weber, & Reichert, 2008a). Changes to one layer often can be performed without affecting other layers; e.g., changing the execution order of process activities or adding new activities to a

process template can, to a large degree, be accomplished without touching the application services linked to the different process activities (Dadam, Reichert, & Kuhn, 2000). Usually, the process logic is expressed in terms of executable *process models*, which can be checked for the absence of errors already at buildtime (e.g., to exclude deadlocks or incomplete data flow specifications). Examples for PAIS-enabling technologies include workflow management systems (van der Aalst & van Hee, 2002) and case handling tools (van der Aalst, Weske, & Grünbauer, 2005; Weske, 2007).

The ability to effectively deal with process change has been identified as one of the most fundamental success factors for PAISs (Reichert & Dadam, 1997; Müller, Greiner, & Rahm, 2004; Pesic, Schonenberg, Sidorova, & van der Aalst, 2007). In domains like healthcare (Lenz & Reichert, 2007; Dadam et al., 2000) or automotive engineering (Mutschler, Bumiller, & Reichert, 2006; Müller, Herbst, Hammori, & Reichert, 2006), for example, any PAIS would not be accepted by users if rigidity came with it. Through the described separation of concerns PAISs facilitate changes. However, enterprises running PAISs are still reluctant to adapt process implementations once they are running properly (Reijers & van der Aalst, 2005; Mutschler, Reichert, & Bumiller, 2008b). High complexity and high cost of change are mentioned as major reasons for not fully leveraging the potential of PAISs. To overcome this unsatisfactory situation more flexible PAISs are needed enabling companies to capture real-world processes adequately without leading to mismatches between computerized business processes and those running in reality (Lenz & Reichert, 2007; Reichert, Hensinger, & Dadam, 1998b). Instead, users must be able to deviate from the predefined processes if required and to evolve PAIS implementations over time. Such changes must be possible at a high level of abstraction and without affecting consistency and robustness of the PAIS.

Changes can take place at both the *process type* and the *process instance* level. Changes of single process instances, for example, become necessary to deal with exceptional situations (Reichert & Dadam, 1998a; Minor, Schmalen, Koldehoff, & Bergmann, 2007). Thus they often have to be accomplished in an ad-hoc manner. Such *ad-hoc changes* must not affect PAIS robustness or lead to errors; i.e., none of the execution guarantees ensured by formal checks at buildtime must be violated due to dynamic process changes. *Process type changes*, in turn, are continuously applied to adapt the PAIS to evolving business processes (Casati, Ceri, Pernici, & Pozzi, 1998; Rinderle, Reichert, & Dadam, 2004b; Pesic et al., 2007). Regarding long-running processes, *evolving process schemes* also require the migration of already running process instances to the new schema version. Important challenges emerging in this context are to perform *instance migrations* on-the-fly, to guarantee *compliance* of migrated instances with the new schema version, and to avoid performance penalties (Rinderle, Reichert, & Dadam, 2004a).

Off-the-shelf process management systems like *Staffware*, *WebSphere Process Server* and *FLOWer* do not support dynamic structural process changes or offer restricted change features only (Weber et al., 2007). Several vendors promise flexible process support, but are unable to cope with fundamental issues related to process change (e.g., correctness). Most systems completely lack support for ad-hoc changes or for migrating process instances to a changed process schema. Thus, application developers are forced to realize workarounds and to extend applications with respective process support functions to cope with these limitations. This, in turn, aggravates PAIS development and PAIS maintenance significantly.

In the ADEPT2 project we have designed and implemented a process management system which allows for both kinds of structural changes in a flexible and reliable manner (Reichert, Rinderle, Kreher, & Dadam, 2005). The design of such a process management technology constitutes a big challenge. First, many trade-offs exist which have to be dealt with. For example, complexity of dynamic process changes increases, the higher expressiveness of the used process modeling formalism becomes. Second, complex interdependencies between the different features of such a technology exist that must be carefully understood in order to avoid implementation gaps. Process schema evolution, for example, requires high-level change operations, schema versioning support, change logging, on-the-fly migration of running

process instances, and dynamic worklist adaptations (Weber et al., 2007). Thus the integrated treatment of these different system features becomes crucial. Third, even if the conceptual pillars of adaptive process management technology are well understood, it still will be a quantum leap to implement respective features in an efficient, robust and integrated manner.

This chapter gives insights into the ADEPT2 process management system, which is one of the few systems that provide integrated support for dynamic structural process changes at different levels. Using this next generation process management technology, new processes can be composed in a plug & play like fashion and be flexibly executed during run-time. ADEPT2 enables support for a broad spectrum of processes ranging from simple document-centred processes (Karbe & Ramsperger, 1991) to complex processes that integrate distributed application services (Khalaf, Keller, & Leymann, 2006). We illustrate how ad-hoc changes of single process instances as well as process schema changes with (optional) propagation of the changes to running process instances can be supported in an integrated and easy-to-use way.

The remainder of this chapter is structured as follows: We first give background information needed for the understanding of the chapter. Then we show how business processes can be modeled and enacted in ADEPT2. Based on this we introduce the ADEPT2 process change framework and its components. Following these conceptual considerations we sketch the architecture of the ADEPT2 system and give insights into its design principles. We conclude with a summary and outlook on future work.

BACKGROUNDS AND BASIC NOTIONS

When implementing a new process in a PAIS its logic has to be explicitly defined based on the modeling constructs provided by a *process meta model*. More precisely, for each business process to be supported, a *process type* represented by a *process schema* is defined. For one particular process type several process schemes may exist representing the different *versions* and the *evolution* of this type over time.

Figure 1 shows a simple example of a process schema (in ADEPT2 notation). It comprises seven activities which are connected through control edges. Generally, control edges specify precedence relations between activities. For example, activity *order medical examination* is followed by activity *make appointment*, whereas activities *prepare patient* and *inform patient* can be executed in parallel. Furthermore, the process schema contains a loop structure, which allows for the repetitive execution of the depicted process fragment. Finally, data flow is modeled by linking activities with data elements. Respective data links either represent a read or a write access of an activity to a data element. In our example, for instance, activity *perform examination* reads data element *patientId*, which is written by activity *order medical examination* before.

Based on a process schema new *process instances* can be created and executed. Each of these process instances logically corresponds to a different *business case*. The PAIS orchestrates the process instances according to the logic defined by their process schema. Generally, a large number of process instances, being in different states, may run on a particular process schema.

To deal with evolving processes, exceptions and uncertainty, PAISs must be flexible. This can be achieved either through *structural process changes* (Reichert & Dadam 1998a; Rinderle et al., 2004a) or by allowing for *loosely specified process models* (Sadiq, Sadiq, & Orlowska, 2001; Adams, ter Hofstede, Edmond, & van der Aalst, 2006). In the following we focus on structural schema adaptations and show how they can be accomplished in a PAIS during runtime. Loosely specified process models, in turn, enable flexibility by leaving parts of the process model unspecified at build-time and by allowing end users to add the missing information during run-time. This approach is especially useful in case of

uncertainty as it allows for deferring decisions from build- to run-time, when more information becomes available. For example, when treating a cruciate rupture for a patient we might not know in advance which treatment will be exactly performed in which execution order. Therefore, this part of the process remains unspecified during build-time and the physician decides on the exact treatment at run-time. For additional information we refer to the approaches followed by Pockets of Flexibility (Sadiq et al., 2001) and Worklets (Adams et al., 2006).

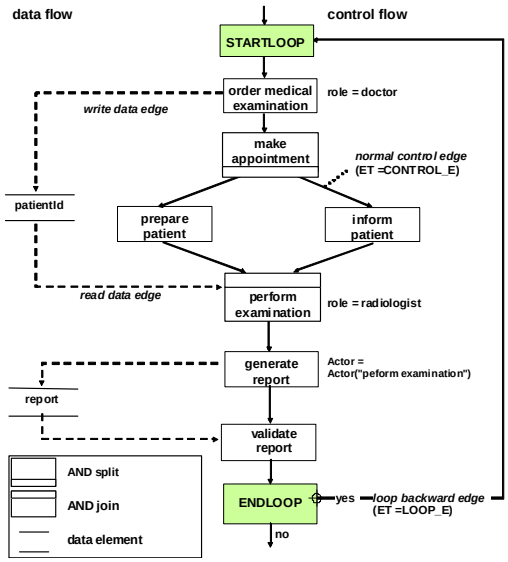


Figure 1 Example of a process schema (in ADEPT2 notation)

In general, structural adaptations of a process schema can be triggered and performed at two levels, the *process type* and the *process instance level*.

Process schema *changes at the type level* (in the following denoted as *process schema evolution*) become necessary to deal with the evolving nature of real-world processes (Rinderle et al., 2004b); e.g., to adapt the process schema to legal changes or to a redesigned business process. In PAISs process schema evolution often requires the dynamic propagation of the corresponding changes to related process instances, particularly if these instances are long-running. For example, assume that in a patient treatment process, due to a new legal requirement, patients have to be educated about potential risks of a surgery before this intervention takes place. Let us further assume that this change is also relevant for patients for which the treatment has already been started. In such a scenario, stopping all ongoing treatments, aborting them and re-starting the treatments is not a viable option. As a large number of treatment processes might be running at the same time, applying this change manually to all ongoing treatment processes is also not a feasible option. Instead system support is required to add this additional activity to all patient treatments for which this is still feasible; i.e., for which the surgery has not yet started.

Ad-hoc changes of single process instances, in turn, are usually required to deal with exceptions or unanticipated situations, resulting in an instance-specific process schema afterwards (Reichert & Dadam, 1997). In particular, such ad-hoc changes must not affect other process instances. In a medical treatment process, for example, the current medication of a particular patient might have to be discontinued due to an allergic reaction of this patient.

-

PROCESS MODELING AND ENACTMENT IN ADEPT2

When designing an adaptive process management system several trade-offs exist which have to be carefully considered. On the one hand, as known from discussions about workflow patterns (van der Aalst, ter Hofstede, Kiepuszewski, & Barros, 2003), high expressiveness of the used process meta model allows to cover a broad spectrum of processes. On the other hand, with increasing expressiveness of the used process meta model, dynamic process changes become more difficult to handle for users (Reichert, 2000). When designing ADEPT2 we kept this trade-off in mind and we found an adequate balance between expressiveness and runtime flexibility. Though ADEPT2 uses a block-structured modeling approach, it enables a sufficient degree of expressiveness due to several modeling extensions and relaxations; for a detailed discussion we refer to (Reichert, 2000) and (Reichert, Dadam, & Bauer, 2003a).

Process Modeling in ADEPT2

The ADEPT2 process meta model allows for the integrated modeling of different process aspects including process activities, control and data flow, actor assignments, organizational, semantical, and temporal constraints, and resources. Here we focus on the basic concepts available for modeling control and data flow, and we sketch how new processes can be composed in a plug & play like fashion. We refer to reading material covering other aspects at the end of this section.

Basic concepts for control flow modeling

In ADEPT2 the control flow of a process schema is represented as attributed graph with distinguishable node and edge types (Reichert et al., 2003a). This allows for efficient correctness checks and eases the handling of loop backs. Formally, a control flow schema corresponds to a tuple (N,E, ...) with node set N and edge set E. Each control edge e $\in$ E has one of the edge types `CONTROL_E`, `SYNC_E`, or `LOOP_E`: `CONTROL_E` expresses a normal precedence relation, whereas `SYNC_E` allows to express a wait-for relation between activities of parallel branches. The latter concept is similar to *links* as used in WS-BPEL. Regarding Figure 2, for example, a necessary pre-condition for enabling activity `H` is that activity `E` either is completed or skipped before (see below). Finally, `LOOP_E` represents a loop backward edge.

Similarly, each node n $\in$ N has one of the node types `STARTFLOW`, `ENDFLOW`, `ACTIVITY`, `STARTLOOP`, `ENDLOOP`, `AND-/XOR`-Split, and `AND-/XOR`-Join. Based on these elements, we can model sequences, parallel branchings, conditional branchings, and loop backs. ADEPT2 adopts concepts from block-structured process description languages, but enriches them by additional control structures in order to increase expressiveness. More precisely, branchings as well as loops have exactly one entry and one exit node. Furthermore, control blocks may be nested, but are not allowed to overlap (cf. Figure 2). As this limits expressive power, in addition, the aforementioned synchronization edges can be used for process modeling (see Reichert & Dadam, 1998a; Reichert, 2000).

We have selected this relaxed block structure because it is quickly understood by users, allows to provide user-friendly, syntax-driven process modeling tools (see below), enables the realization of high-level change patterns guaranteeing soundness, and makes it possible to implement efficient algorithms for process analysis. Note that we provide relaxations (e.g., synchronization edges and backward failure edges) and extensions (e.g., temporal constraints, actor assignments), respectively, which allow for sufficient expressiveness to cover a broad spectrum of processes from different domains. We already applied the ADEPT1 technology in domains like healthcare, logistics, and e-commerce, and the feedback

we received was very positive (Müller et al., 2004; Bassil, Keller, & Kropf, 2004; Bassil, Benyoucef, Keller, R., & Kropf, 2002; Golani & Gal, 2006). In particular, the expressiveness of our meta model was considered as being sufficient in most cases. We are currently applying ADEPT2 in other domains like construction engineering and disaster management, and we can make similar observations here.

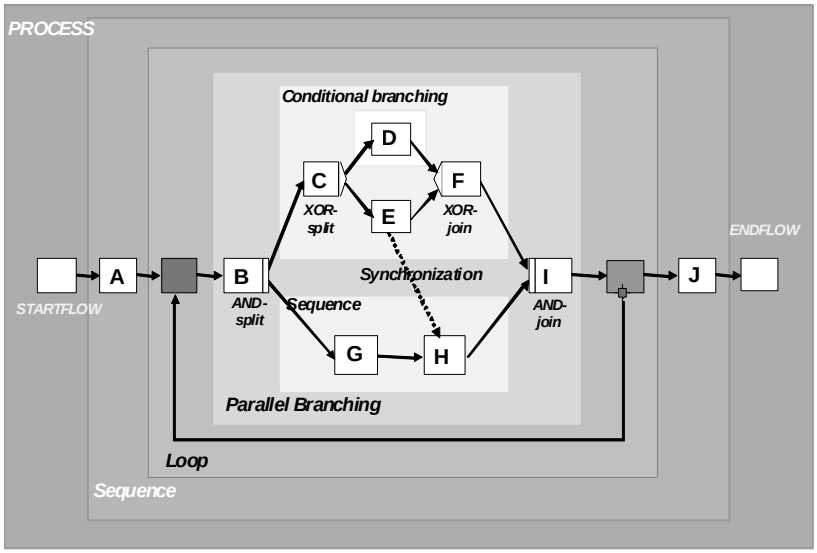


Figure 2: Block-structuring of ADEPT2 process models

Basic concepts for data flow modeling

Data exchange between activities is realized through writing and reading (global) process variables (denoted as *data elements* in the following). In this context, ADEPT2 considers both basic and complex data types. In addition, user-defined types are supported. Data elements are connected with input and output parameters of process activities. Each input parameter of a particular activity is mapped to exactly one data element by a *read data edge* and each activity output parameter is connected to a data element by a *write data edge*. An example is depicted in Figure 1. Activity *order medical examination* writes data element *patientID* which is then read by the subsequent activity *perform examination*.

The total collection of data elements and data edges constitutes the *data flow schema*. For its modeling, a number of constraints must be met. The most important one ensures that all data elements mandatorily read by an activity X must have been written before X becomes enabled; in particular, this has to be ensured independently from the execution path leading to activation of X (Reichert, 2000). Note that this property is crucial for the proper invocation of activity programs without missing input data.

Process composition by plug & play of application components

Based on the described modeling concepts a new process can be realized by creating a *process template* (i.e., *process schema*). Among other things such a template describes the control flow for the process activities as well as the data flow between them. It either has to be defined from scratch or an existing template is chosen from the process template repository and adapted as needed ("process cloning").

Afterwards application components (e.g., web services or Java components) have to be assigned to the process activities. Using the ADEPT2 process editor these components can be selected from the

component repository and be inserted into the process template by drag & drop. Following this, ADEPT2 analyzes whether the application functions can be connected in the desired order; e.g., we check whether the input parameters of application functions can be correctly supplied for all possible execution paths imposed by the process schema. Only those process templates passing all correctness checks may be released and transferred to the runtime system. We denote this feature as *correctness by construction*.

When dragging application components from the repository and assigning them to particular activities in the process template, the process designer does not need to have detailed knowledge about the implementation of these components. Instead the component repository provides an integrated, homogeneous view as well as access to the different components. Internally, this is based on a set of wrappers provided for the different types of application components. Our chosen architecture allows to add new wrappers if new component types have to be supported. Currently, ADEPT2 allows to integrate different kinds of application components like electronic forms, stand-alone executables, web services, Java library functions, and function calls to legacy systems.

Process Enactment in ADEPT2

Based on a given process schema new *process instances* can be created and started. State transitions of a single activity instance are depicted in Figure 3. Initially, activity status is set to `NOT_ACTIVATED`. It changes to `ACTIVATED` when all preconditions for executing this activity are met. In this case corresponding *work items* are inserted into the *worklists* of authorized users. If one of them selects the respective item from his worklist, activity status changes to `RUNNING` and respective work items are removed from the worklists of other users. Furthermore, the application component associated with the activity is started. At successful termination, activity status changes to `COMPLETED`.

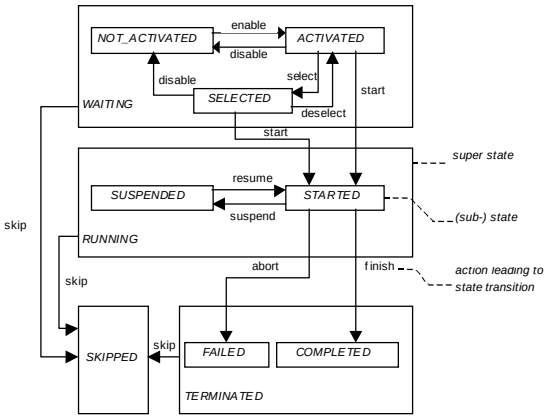


Figure 3: Internal state transitions of a process activity

To determine which activities are to be executed next, process enactment in ADEPT2 is based on a well-defined operational semantics (Reichert & Dadam, 1998a; Reichert, 2000). For each process instance we further maintain information about its current state by assigning markings to its activities and control edges respectively. Figure 4 depicts an example showing two process instances in different states.

Similar to Petri Nets, markings are determined by well defined marking and enactment rules. In particular, ADEPT2 maintains markings of already passed regions (except loop backs). Furthermore, activities belonging to non-selected paths of a conditional branching are marked as `SKIPPED`. Note that

this allows to easily check whether certain changes may be applied in the current status of a process instance or not (see later). As aforementioned, ADEPT2 ensures dynamic properties like the absence of deadlocks, proper process termination, and reachability of markings which enable the activation of particular activity. The described block structuring as well as the used node and edge types help us to accomplish this in an efficient manner. Deadlocks, for example, can be excluded if the process schema (excluding loop backs) does not contain cycles (Reichert & Dadam, 1998a).

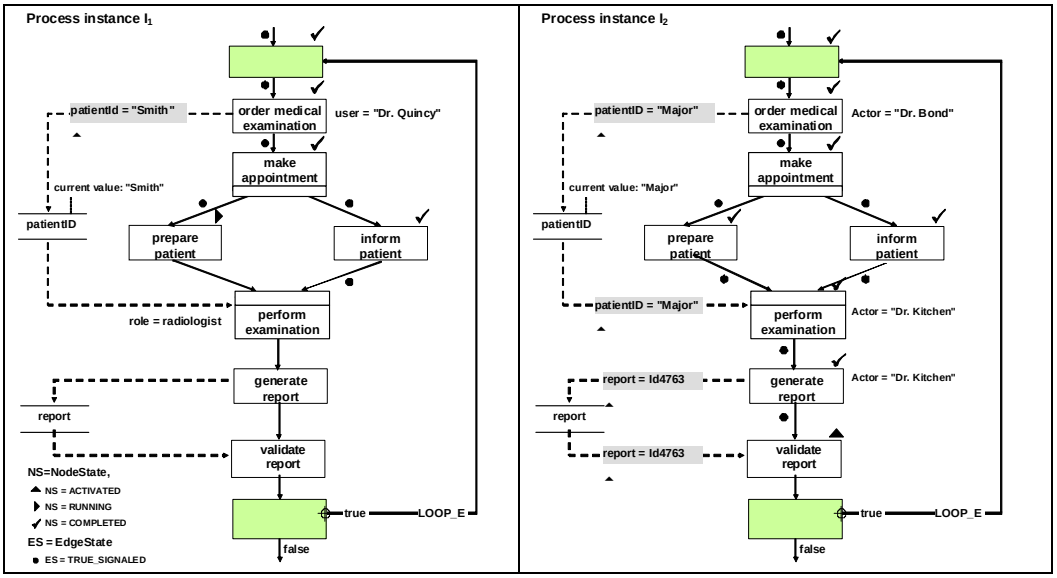


Figure 4: Examples of two process instances running on the process schema from Figure 1

For each data element ADEPT2 stores different versions of a data object during runtime if available. In more detail, for each write access to a data element, always a new version of the respective data object is created and stored in the runtime database; i.e., data objects are not physically overwritten. This allows us to use different versions of a data element within different branches of a branching with AND-Split and XOR-Join. As shown in (Reichert et al., 2003a) maintaining data object versions is also important to enable correct rollback of process instances at the occurrence of semantical errors (e.g., activity failures).

Other Process aspects covered in ADEPT2

Activities and their control as well as data flow are not the only viewpoints supported in our approach. ADEPT2 also considers organizational models (Rinderle & Reichert, 2007a), actor and resource assignments (Rinderle & Reichert, 2005b; Rinderle-Ma & Reichert, 2008c), and application components. In related projects, we have further looked at temporal constraints (Dadam, Reichert, & Kuhn, 2000), partitioned process schemes with distributed enactment (Reichert, Bauer, & Dadam, 1999; Bauer, Reichert, & Dadam, 2003), and configurable process visualizations (Bobrik, Bauer, & Reichert; 2006; Bobrik, Reichert, & Bauer, 2007). All these viewpoints are not only relevant for process modeling, but have to be considered in the context of (dynamic) process changes as well (Reichert & Bauer, 2007; Rinderle & Reichert, 2005b, 2007a; Dadam et al., 2000). On the one hand, each of the aspects can be primary subject to (dynamic) change. On the other hand, the different aspects might have to be adjusted due to the change of another aspect (e.g., adaptation of temporal constraints when changing the control

-

flow structure). To set a focus, however, in this chapter we restrict ourselves to control and data flow changes. The above given references provide further information on the other aspects.

Note that we consider process correctness only at the syntactical level in this chapter (e.g., absence of deadlock-causing cycles and correctness of data flow). Respective checks are fundamental for both process modeling and process change. However, errors may be still caused at the *semantical* level (e.g., due to the violation of business rules) though not affecting the robustness of the PAIS. Therefore, the integration and verification of domain knowledge flags a milestone in the development of adaptive process management technology. In the SeaFlows project, we are currently developing a framework for defining *semantic constraints* over processes in such a way that they can express real-world domain knowledge on the one hand and are still manageable concerning the effort for maintenance and semantic process verification on the other hand (Ly, Göser, Rinderle-Ma, & Dadam, 2008). This viewpoint can be used to detect semantic conflicts (e.g., drug incompatibilities) when modeling process schemes, applying ad-hoc changes at process instance level, or propagating process schema changes to already running process instances, even if they have been already individually modified themselves; i.e., SeaFlows provides techniques to ensure semantic correctness for single and concurrent changes which are, in addition, minimal regarding the set of semantic constraints to be checked. Together with further optimizations of the semantic checks based on certain process meta model properties this allows for efficiently verifying processes. Altogether, the SeaFlows framework provides the basis for process management systems which are *adaptive* and *semantic-aware* at the same time; note that this is a fundamental issue when thinking of business process compliance. For further details we refer to (Ly et al., 2008; Ly, Rinderle, & Dadam, 2008).

ADEPT2 PROCESS CHANGE FRAMEWORK

This section deals with fundamental aspects of dynamic process changes as supported by ADEPT2. Though we illustrate relevant issues along the ADEPT2 process meta model, it is worth mentioning that most of the described concepts can be applied in connection with other process modeling formalisms as well; see (Reichert, Rinderle, & Dadam, 2003b) and (Reichert & Rinderle, 2006) for examples.

Requirements

In order to adequately deal with process changes during runtime users need to be able to define them at a high level of abstraction. Several fundamental requirements, which will be discussed in the following, exist in this context:

1. *Support of structural adaptations at different levels.* Any framework enabling dynamic process changes should allow for structural schema adaptations at both the process type and the process instance level. In principle, the same set of change patterns should be applicable at both levels.
2. *Enabling a high level of abstraction when defining process changes.* It must be possible to define structural process adaptations at a high level of abstraction. In particular, all complexity associated with the adjustment of data flows or the adaptation of instance states should be hidden from users.
3. *Completeness of change operations.* To be able to define arbitrary structural schema adaptations a complete set of change operations is required; i.e., given two correct schemes it must be always possible to transform one schema into the other based on the given set of change operations.

4. *Correctness of changes*. The ultimate ambition of any change framework must be to ensure correctness of dynamic changes (Rinderle, Reichert, & Dadam, 2003). First, structural and behavioral soundness of the modified process schema should be guaranteed independent from whether the change is applied at instance level or not. Second, when performing structural schema changes at instance level, this must not lead to inconsistent process states or errors. Therefore, an adequate *correctness criterion* is needed to decide whether a given process instance is *compliant* with a modified process schema. This criterion must not be too restrictive, i.e., no process instance should not be needlessly excluded from being migrated to the new schema version.
5. *Change efficiency*. We must be able to efficiently decide whether a process instance is compliant with a modified schema or not. Furthermore, when migrating compliant instances to the modified schema, state adaptations need to be accomplished automatically and in an efficient way.

We show how ADEPT2 deals with these fundamental requirements. There exist additional challenges not treated here, but which have been considered in the design of the ADEPT2 framework as well: change authorization (Weber, Reichert, Wild, & Rinderle, 2005a), change traceability (Rinderle, Reichert, Jurisch, & Kreher, 2006b; Rinderle, Jurisch, & Reichert, 2007b), change annotation and reuse (Weber, Wild, & Breu, 2004; Rinderle, Weber, Reichert, & Wild, 2005a; Weber, Rinderle, Wild, & Reichert, 2005c; Weber, Reichert, & Wild, 2006), and change mining (Günther, Rinderle, Reichert, & van der Aalst, 2006; Günther, Rinderle-Ma, Reichert, van der Aalst, & Recker, 2008; Li, Reichert, & Wombacher, 2008b). The given references provide additional reading material on these advanced aspects.

Support of Change Patterns in ADEPT2

Two alternatives exist for realizing structural adaptations of a process schema (Weber et al., 2007). A first option is to realize the schema adaptations based on a set of change primitives like add node, remove node, add edge, and remove edge (Minor et al., 2007). Following such a low-level approach, the realization of a particular change (e.g., to move an activity to a new position) requires the combined application of multiple change primitives. To specify structural adaptations at this low level of abstraction is a complex and error-prone task. Furthermore, when applying a single change primitive, soundness of the resulting process schema cannot be guaranteed by construction; i.e., it is not possible to associate formal pre-/post-conditions with the application of single change primitives. Instead, correctness of a process schema has to be explicitly checked after applying the respective set of primitives.

Another, more favorable option is to base structural adaptations on high-level change operations (Weber et al., 2007), which abstract from the concrete schema transformations to be conducted; e.g., to insert a process fragment between two sets of nodes or to move process fragments from their current position to a new one (Reichert & Dadam, 1998a). Instead of specifying a set of change primitives the user applies one or few high-level change patterns to define a schema adaptation. Following this approach, it becomes possible to associate pre-/post-conditions with the respective change operations. This, in turn, allows the PAIS to guarantee soundness when applying the patterns (Reichert, 2000). Note that soundness will crucial if changes have to be defined by end users or – even more challenging – by intelligent software agents (Müller et al., 2004; Golani & Gal, 2006; Bassil et al., 2004). In order to meet this fundamental goal ADEPT2 only considers high-level change patterns. Of course, the same patterns can be used for process modeling as well enabling the already mentioned “correctness by construction”. A similar approach is provided in (Gschwind, Koehler, & Wong, 2008).

ADEPT2 provides a complete set of change patterns and change operations respectively based on which structural adaptations at the process type as well as the process instance level can be expressed. In

particular, this can be accomplished at a high level of abstraction. Furthermore, the change patterns are applicable to the whole process schema; i.e., the region to which the respective change operation is applied can be chosen dynamically (as opposed to late modeling of loosely specified process models where changes are usually restricted to a predefined region). This allows to flexibly deal with exceptions and to cope with the evolving nature of business processes. Furthermore, the application of a change pattern to a sound process schema results in a sound schema again, i.e., structural and behavioral soundness of the schema are preserved.

We do not present the complete set of change patterns supported by ADEPT2 (Weber et al., 2007; Weber, Reichert, & Rinderle, 2008b), but only give selected examples in the following:

- *Insert Process Fragment*: This change operation can be used to add process fragments to a given process schema. One parameter of this operation describes the position at which the new fragment is embedded in the schema; e.g., ADEPT2 allows to serially insert a fragment between two succeeding activities or to insert new fragments between two sets of activities (Reichert, 2000). Special cases of the latter variant include the insertion of a process fragment in parallel to another one (*parallel insert*) or the association of the newly added fragment with an execution condition (*conditional insert*). Figure 5a depicts an example of a parallel insertion.
- *Delete Process Fragment*. This change operation can be used to remove a process fragment. Figure 5b depicts two simple examples.
- *Move Process Fragment*. This change operation allows users to shift a process fragment from its current position in the process schema to a new one. One parameter of this operation specifies the way the fragment is re-embedded in the process schema afterwards. Though the move operation could be realized by the combined use of the insert and delete operation, ADEPT2 introduces it as separate operation since it provides a higher level of abstraction to users.

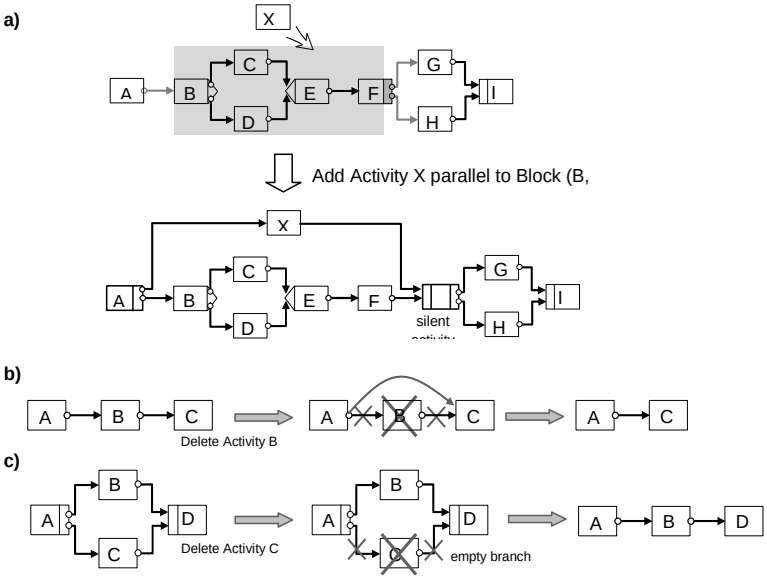


Figure 5. Insertion and deletion of process activities in ADEPT2

Other examples of ADEPT2 change operations include the embedding of a process fragment in a conditional branch or loop construct, and the addition or deletion of synchronizations between parallel

-

activities. When applying such high-level changes, ADEPT2 automatically reduces complexity through simple schema refactoring (Reichert & Dadam, 1998a); e.g., empty branches or unnecessary nodes are removed after change application (cf. Figure 5). Generally, the change patterns offered by ADEPT2 can be used for a large variety of behavior preserving process refactorings (Weber & Reichert, 2008a).

Generally, structural adaptations of a control flow schema have to be combined with adjustments of the data flow schema in order to preserve soundness. As simple example consider Figure 6 where activity B shall be deleted from the depicted process schema. To preserve schema correctness we must deal with the data dependencies activities D and E have on activity B. Figure 6 shows four basic options supported by ADEPT2 in this context: (a) cascading deletion of data-dependent activities; (b) insertion of an alternate activity which writes the respective data element; (c) insertion of an auxiliary service (e.g., an electronic form) which is invoked when deleting B, or insertion of an auxiliary service which is invoked when starting the first data-dependent activity (D in our example). Which of these four options is most favorable in a given context depends on the semantics of the activity to be primarily deleted. It therefore has to be chosen by the process designer at buildtime or by the user requesting the deletion at runtime. Regarding the example from Figure 1, for instance, deletion of activity *generate report* should be always accompanied by deletion of activity *validate report* since the second activity strongly depends on the first one; i.e., option (a) has to be applied. ADEPT2 allows to explicitly specify such strong dependencies at buildtime, which enables the runtime system to automatically apply option (a) if required. By contrast, option (c) might be favorable when deleting automated activity *make appointment* in Figure 1; e.g., in case the appointment is exceptionally made by phone and therefore can be manually entered into the system.

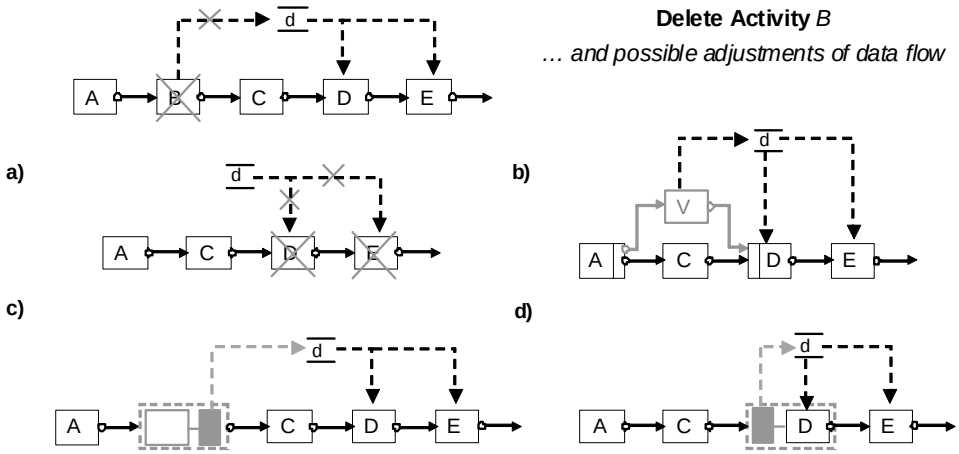


Figure 6: Adjusting data flow in the context of an activity deletion

In summary, ADEPT2 provides a complete set of high-level change operations which can be used for specifying structural adaptations as well as for accomplishing structural comparisons of process schemes (Li, Reichert, & Wombacher, 2008a). In particular, these high-level operations cover most of the change patterns described in (Weber et al. 2007; Weber et al., 2008b). Finally, the application of ADEPT2 operations to a correct process schema results in a correct schema again. Basic to the latter is the formal semantics defined for the supported change patterns (Rinderle-Ma, Reichert, & Weber, B.; 2008b).

-

Ensuring Correctness of Dynamic Changes

So far, we have only looked at structural schema adaptations without considering the state of the process instances running on the respective schema. In this subsection we discuss under which conditions a structural schema change can be applied at the process instance level as well. Obviously, structural adaptations have to be restricted with respect to the current state of an instance. As example consider Figure 7a. Activity X is serially added between activities A and B resulting in a correct process schema afterwards. Consider now process instance I from Figure 7b. When applying the schema change to this instance, an inconsistent state would result; i.e., activity B would have state COMPLETED though its preceding activity X would still be in state ACTIVATED.

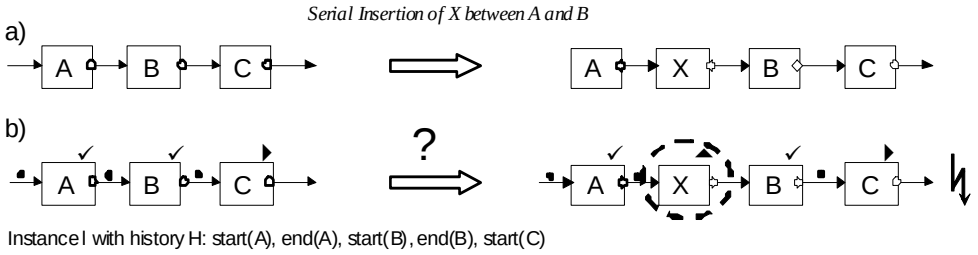


Figure 7. Schema change and inconsistency due to uncontrolled change propagation at instance level

To avoid such inconsistencies we need a formal foundation for dynamic changes. In the following, let I be an instance running on process schema S and having marking M_S. Assume further that S is transformed into another correct process schema S′ by applying change Δ. Then the following two issues arise:

1. Can Δ be correctly propagated to process instance I, i.e., can Δ be applied to I without causing inconsistencies? For this case, I is denoted as being *compliant* with the modified schema S′.
2. How can we migrate a compliant instance I to S′ such that further execution of I can be based on S′? Which state adaptations become necessary and how can they be automatically accomplished?

Both issues are fundamental for any adaptive process management system. While the first one concerns *pre-conditions* on the state of the respective instance, the second one is related to *post-conditions* to be satisfied after the dynamic change. We need an efficient method allowing for automated compliance checks and instance migrations. Intuitively, instance I would be compliant with the modified schema S′ if it could have been executed according to S′ as well and had produced the same effects on data elements (Rinderle et al., 2004b; Casati et al., 1998). Trivially, this will be always the case if instance I has not yet entered the region affected by the change. Generally, we need information about the previous execution of instance I to decide on this and to determine a correct follow-up marking when structurally adapting it. At the logical level we make use of the *execution history* (i.e., trace) kept for each process instance. We assume that this execution history logs events related to the start and completion of activity executions. Obviously, an instance I with history H will be compliant with modified schema S′ and therefore can migrate to S′ if H can be produced on S′ as well. We then obtain a correct new state (i.e., marking) for instance I by "replaying" all events from H on S′ in the order they occurred.

Taking our example from Figure 7b this property does not hold for instance I. Therefore the depicted schema change must not be applied to this instance. As another example consider the process instance from Figure 8a and assume that activity C shall be moved to the position between activities A and B

resulting in schema S′. Since the execution history of I can be produced on S′ as well the instance change will be allowed (cf. Figure 8b). Note that we have to deactivate activity B and activate activity C in this context before proceeding with the flow of control. Similar considerations hold for the instance from Figure 8a when moving activity C to a position parallel to activity B resulting in process schema S″. Again this change is valid since the execution history of I can be produced on S″ as well (cf. Figure 8c).

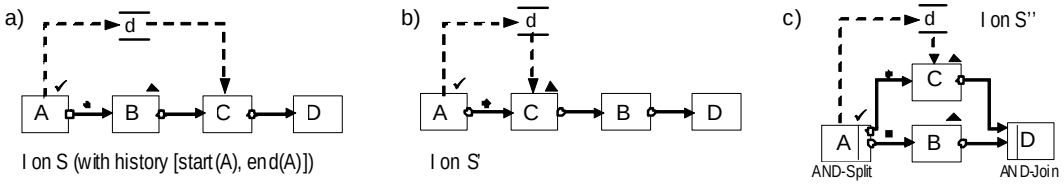


Figure 8: Process instance I and two possible changes (movement of activity C)

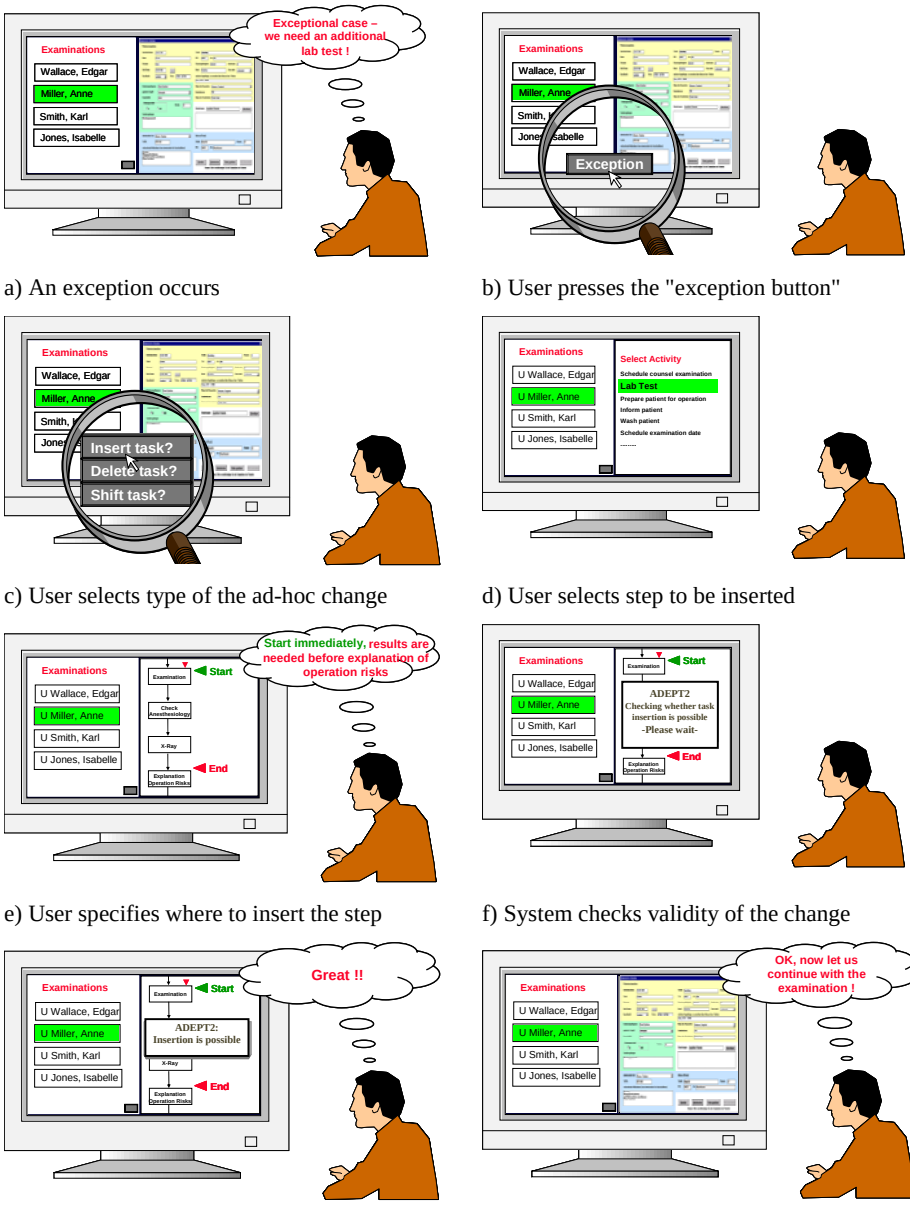
Note that the described compliance criterion is still too restrictive to serve as general correctness principle. Concerning changes of a loop structure, for example, it might needlessly exclude instances from migration, particularly if the loop is its n^{th} run (n>1) and previous iterations do not comply with the new schema version. We refer to (Rinderle et al., 2004b) for relaxations provided in this context.

Generally, it would be no good idea to guarantee compliance and to determine follow-up markings of compliant instances by accessing the whole execution history and by trying to replay it on the modified schema. This would cause a performance penalty, particularly if a large number of instances were running on the schema to be modified (see below). ADEPT2 therefore utilizes the semantics of the applied change operations as well as information on the change context to efficiently check for compliance and to adapt state markings of compliant instances when migrating them to the new schema version (Rinderle et al., 2004b). For example, an activity in state `COMPLETED` or `RUNNING` must be not deleted from a process instance. Or when adding a new activity to a process instance or moving an existing one, the corresponding execution history must not contain any entry related to successor activities of the added or shifted activity. This would be the case, for example, if the successor nodes had marking `NOT_ACTIVATED` or `ACTIVATED`. Obviously, this does not hold for the scenario depicted in Figure 7.

In summary, the ADEPT2 change framework is based on a well-defined correctness criterion, which is independent of the ADEPT2 process meta model and which is based on an adapted notion of trace equivalence (Rinderle et al., 2004a). This compliance criterion considers control as well as data flow changes, ensures correctness of instances after migration, works correctly in connection with loop backs, and does not needlessly exclude instances from migrations. To enable efficient compliance checks, precise and easy to implement compliance conditions have been defined for each change operation. ADEPT2 automatically adapts the states of compliant instances when migrating them to an updated schema. Finally, we are currently working on the relaxation of the described compliance criterion in order to increase the number of process instances that can be dynamically and automatically migrated to a new process schema version (Rinderle-Ma, Reichert, & Weber, 2008a).

Scenarios for Dynamic Process Changes in ADEPT2

After having introduced the basic pillars of the ADEPT2 change framework we now sketch how ADEPT2 supports *dynamic* process changes at different levels.



a) An exception occurs

b) User presses the "exception button"

c) User selects type of the ad-hoc change

d) User selects step to be inserted

e) User specifies where to insert the step

f) System checks validity of the change

g) Change can be applied

h) User continues work

Figure 9: Ad-hoc change in ADEPT2 (user view)

Ad-hoc Changes of Single Process Instances

Figure 9 a – h illustrate how the interaction between the ADEPT2 system and the end user looks like when performing an ad-hoc change. In this example, we assume that during the execution of a particular process instance (e.g., the treatment of a certain patient under risk) an additional lab test becomes necessary. Assume that this medical test has not been foreseen at buildtime (cf. Figure 9a). As a consequence, this particular process instance will have to be individually adapted if the change request is approved by the system. After the user has pressed the "exception button" (cf. Figure 9b), he can specify the type of the intended ad-hoc change (cf. Figure 9c). If an insert operation shall be applied, for example,

-

the system will display the tasks that can be added in the given context and for which the user has respective authorization (cf. Figure 9d). As aforementioned, these tasks can be based on simple or complex application components (e.g., *write letter* or *send email*), or even be complete processes.

Generally, authorized users can retrieve the task to be dynamically added to a particular process instance from the ADEPT2 activity repository. This repository organizes the tasks in different categories, provides query facilities to retrieve them, and maintains the information necessary to plug the tasks into an instance schema (e.g., interface specification and task attributes). We restrict access to exactly those tasks that can be added in the given context; i.e., selectable tasks depend on the profile of the current user, the process type, the process instance, etc. For details we refer to (Weber et al.; 2005a). Finally, ADEPT2 also allows for the reuse of ad-hoc changes previously applied in a similar problem context. Basic to this reusability are case-based reasoning techniques (Weber, Reichert, Wild, & Rinderle-Ma, 2008c).

Following this task selection procedure, the user simply has to state after which activities in the process the execution of the newly added activity shall be started and before which activities it shall be finished (cf. Figure 9e). Finally, the system checks whether the desired structural adaptation is valid in the given state of the instance (cf. Figure 9f and Figure 9g). In this context, the same checks are performed as during buildtime (e.g., to ensure for the absence of deadlocks). In addition, the current process instance state is taken into account when modifying the instance.

As already discussed, such adaptations can be specified at a high level of abstraction (e.g. "Insert Step X between activity set A1 and activity set A2"), which eases change definition significantly. All change operations are guarded by pre-conditions which are either automatically checked by the system when the operation is invoked or which are used to hide non-allowed changes from users. Post-conditions guarantee that the resulting process instance is correct again. Furthermore, all change operations and change patterns respectively are made available via the ADEPT2 API (*application programming interface*) as well. The same applies for the querying interface of the ADEPT2 repository. This allows for the implementation of sophisticated end user clients or even automated agents (Müller et al., 2004).

To enable change traceability ADEPT2 stores process instance changes in change logs (Rinderle et al., 2006b, 2007b). Together with execution logs, which capture enactment information of process instances, the structure and state of a particular process instance can be reconstructed at any time. Both change and execution log are also valuable sources for process learning and process optimization (Günther et al., 2008; Li et al., 2008b).

By performing the described ad-hoc deviation inside the PAIS the added task becomes an integral part of the respective process instance. This way full system support becomes possible relieving the user from handling the exception; i.e., task execution can be fully coordinated by the PAIS, the task can be automatically assigned to user worklists, its status can be monitored by the PAIS, and its results can be analyzed and evaluated in the context of the respective process instance. By contrast, if the exception had been handled manually, i.e. outside the PAIS, it would be the intellectual responsibility of the end user to accomplish task execution, monitoring and analysis, and to relate the task to the respective process instance (e.g., by attaching a “post-it” to his screen). As we know from healthcare the latter approach unnecessarily burdens users resulting in organizational overload and omissive errors (Lenz & Reichert, 2007).

Process Schema Evolution

Though the support of ad-hoc modifications is very important, it is not yet sufficient. As motivated, for long-running processes it is often required to adapt the process schema (from which new instances can be created afterwards) due to organizational changes. Then process instances currently running on this

process schema can be affected by the change as well. If processes are of short duration only, already running instances can be usually finished according to the old schema version. However, this strategy will not be applicable for long running processes. Then the old process schema version may no longer be applicable, e.g., when legal regulations have changed or when the old process reveals severe problems.

One solution would be to individually modify each of the running process instances by applying corresponding ad-hoc changes (as described above). However, this would be too inefficient and error-prone if a multitude of running process instances had been involved. Note that the number of active process instances can range from dozens up to thousands (Bauer, Reichert, & Dadam, 2003); i.e., compliance checking and change propagation might become necessary for a large number of instances.

An adaptive process management system must be able to support correct changes of a process schema and their propagation to already running process instances if desired. In other words, if a process schema is changed and thus a new version of this schema is created, process instances should be allowed to migrate to the new schema version (i.e., to be transferred and re-linked to the new process schema version). In this context, it is of particular importance that ad-hoc changes of single process instances and instance migrations do not exclude each other since both kinds of changes are needed for the support of long-running processes (Rinderle, Reichert, & Dadam, 2004c + 2004d).

The ADEPT2 technology implements the combined handling of both kinds of changes. Process instances which have been individually modified can be also migrated to a changed process schema if this does not cause inconsistencies or errors in the following. All correctness checks (on the schema and the state of the instances) needed and all adaptations to be accomplished when migrating the instances to the new process schema version are performed by ADEPT2. The implementation is based on the change framework and the formal foundations described before. ADEPT2 can precisely state under which conditions a process instance can be migrated to the new process schema version. This allows for checking the compliance of a collection of process instances with the changed schema version in an efficient and effective manner. Finally, concurrent and conflicting changes at the process type and the process instance level are managed in a reliable and consistent manner as well.

Figure 10 a – c illustrate how such a process schema evolution is conducted from the user's point of view in ADEPT2. The process designer loads the process schema from the process template repository, adapts it (using the ADEPT2 process editor and the change patterns supported by it), and creates a new schema version (cf. Figure 10a). Then the system checks whether the running process instances can be correctly migrated to the new process schema version (cf. Figure 10 b+c). These checks are based on state conditions and structural comparisons (in order to ensure compliance and soundness respectively). Furthermore, the system calculates which adaptations become necessary to perform the migration at the process instance level. The ADEPT2 system analyzes all running instances of the old schema and creates a list of instances which can be migrated as well as a list of instances for which this is not possible (together with a report which explains the different judgments). When pressing the "migration button" ADEPT2 automatically conducts the migration for all selected process instances (see Figure 10d).

In ADEPT2, the on-the-fly migration of a collection of process instances to a modified process schema does not violate correctness and consistency properties of these instances. At the system level this is ensured based on the correctness principle introduced in the previous section. As example consider Figure 11 where a new schema version S' is created from schema S on which three instances are running. Instance I_1 can be migrated to the new process schema version. By contrast, instances I_2 and I_3 cannot migrate. I_3 has progressed too far and is therefore not compliant with the updated schema. Though there is no state conflict for I_2 this instance can also not migrate to S'. I_2 was individually modified by a previous ad-hoc change conflicting with the depicted schema change at the type level. More precisely, when propagating the type change to I_2 a deadlock-causing cycle would occur. The ADEPT2 change

framework provides efficient means to detect such conflicts. Basic to this are sophisticated conflict tests (see Rinderle, Reichert, & Dadam, 2004d). In summary, we restrict propagation of a type change to those instances for which the change does not conflict with instance state or previous ad-hoc changes.

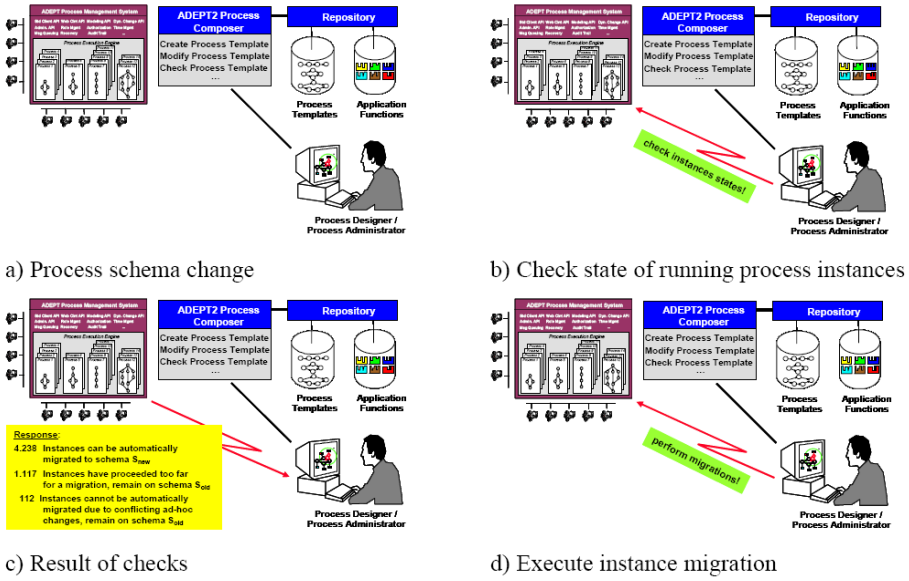


Figure 10: Process schema evolution in ADEPT2 (user perspective)

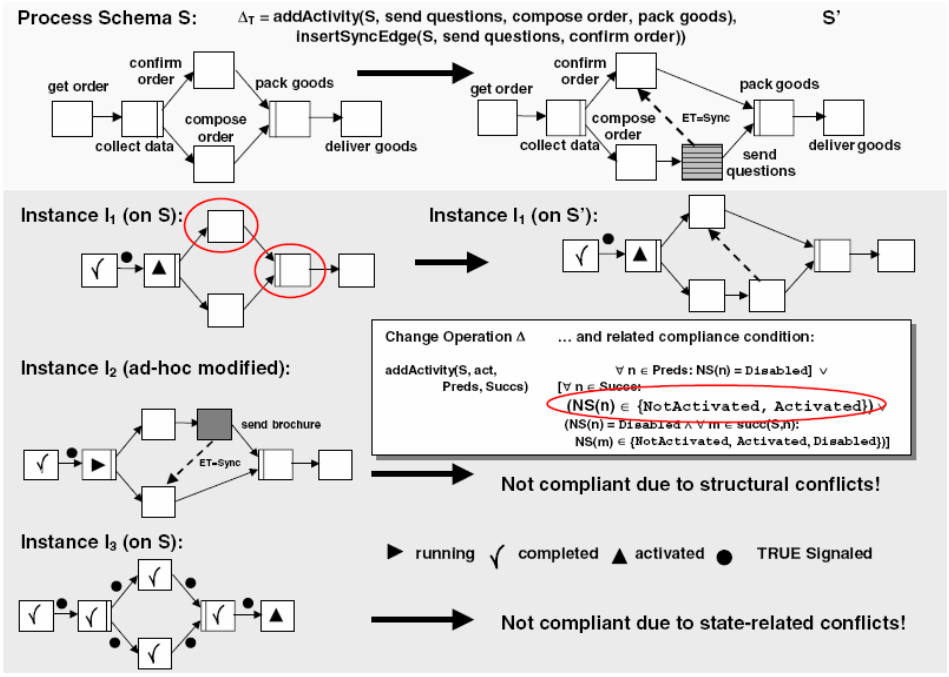


Figure 11: Process schema evolution in ADEPT2 (system perspective)

-

Full Process Lifecycle Support Through Adaptive Processes

As shown, adaptive process management technology like ADEPT2 extends traditional PAISs with the ability to deal with dynamic structural changes at different process levels. This enables full life cycle support as depicted in Figure 12 (Weber, Reichert, Rinderle, & Wild, 2005b).

At build-time an initial representation of a process is created by explicitly modeling its template from scratch (based on analysis results), by cloning an existing process template and adapting it, or by discovering a process model through the mining of execution logs (1). The first two options have been described earlier in this chapter; the latter one requires support by a sophisticated process mining tool like ProM (van Dongen, de Medeiros, Verbeek, Weijters, & van der Aalst, 2005).

At run-time new process instances can be derived from the predefined process template (2). In general, an instance is enacted according to the process template it was derived from. While automated activities are executed without user interaction, non-automated activities are assigned to the worklists of users to be worked on (3). The latter is based on actor assignment rules associated with the non-automated activity.

If exceptional situations occur during run-time, process participants may deviate from the predefined schema (4). ADEPT2 balances well between flexibility and security in this context; i.e., process changes are restricted to authorized users, but without nullifying the advantages of a flexible system by handling authorizations in a too rigid way. In (Weber, Reichert, Wild, & Rinderle, 2005a) we discuss the requirements relevant in this context and propose a comprehensive access control (AC) model with special focus on adaptive PAISs. We support both the definition of user dependent and process type dependent access rights, and allow for the specification of access rights for individual change patterns. If desired, access rights can be specified at an abstract (i.e., coarse-grained) level (e.g., for a whole process category or process template). Fine-grained specification of access rights (e.g., concerning the deletion of a particular process activity) is supported as well, allowing context-based assistance of users when performing a change. Generally, the more detailed the respective specifications, the more costly their definition and maintenance becomes. Altogether our AC approach allows for the compact definition of user dependent access rights restricting process changes to authorized users only. Finally, the definition of process type dependent access rights is supported to only allow for those change commands which are applicable within a particular process context. For further details we refer to (Weber et al., 2005a).

While execution logs record information about the start and completion of activities as well as their ordering, process changes are recorded in change logs (5). The analysis of respective logs by a process engineer and by business process intelligence tools, respectively, allows to discover malfunctions or bottlenecks (Li, Reichert, & Wombacher, 2008c). In (Li, Reichert, & Wombacher, 2008b) we additionally provide an approach which fosters learning from past ad-hoc changes; i.e., an approach which allows for mining instance variants. As result we obtain a generic process model for which the average distance between this model and the respective instance variants becomes minimal. By adopting this generic model as new template in the PAIS, need for future ad-hoc adaptation decreases; i.e., mining execution and change logs can result in an evolution of the process schema; i.e., an updated process schema version (6). In addition, it becomes possible to provide recommendations to user about future process enactment based on execution logs (e.g., Schonenberg, Weber, van Dongen, & van der Aalst, 2008).

If desired and possible, running process instance migrate to the new schema version and continue their execution based on the new schema (7).

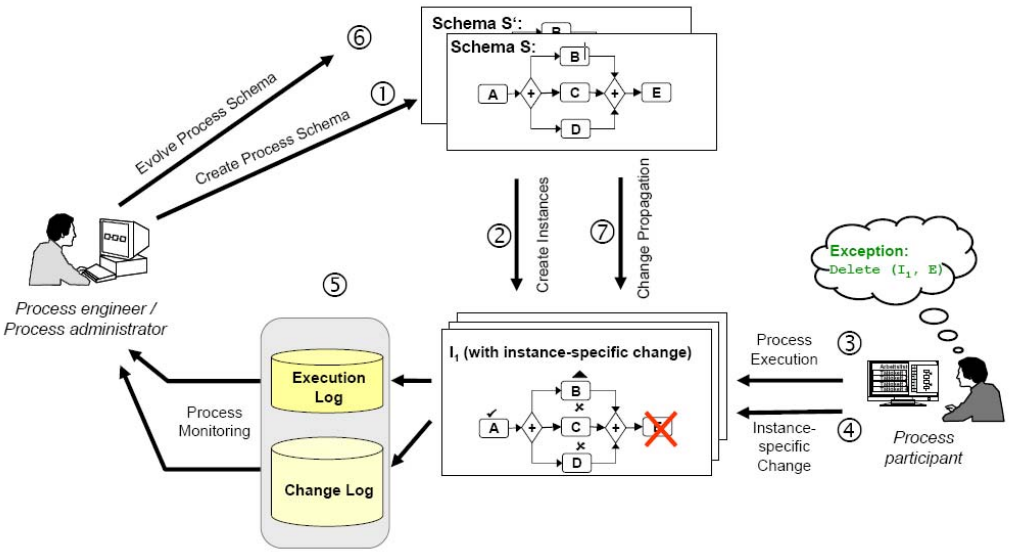


Figure 12: Process lifecycle management in ADEPT2 (see Weber et al., 2005b)

ARCHITECTURE OF THE ADEPT2 PROCESS MANAGEMENT SYSTEM

The design of the ADEPT2 system has been governed by a number of principles in order to realize a sustainable and modular system architecture. The considered design principles refer to general architectural aspects as well as to conceptual issues concerning the different system features. Our overall goal was to enable ad-hoc flexibility and process schema evolution, together with other process support features, in an integrated way, while ensuring robustness, correctness, extensibility, performance and usability at the same time. This section summarizes major design principles and gives an overview of the developed system architecture.

High-end process management technology like ADEPT2 has a complexity comparable to database management systems. To master this complexity a proper and modular system architecture has been chosen for ADEPT2 with clear separation of concerns and well-defined interfaces. This is fundamental to enable exchangeability of implementations, to foster extensibility of the architecture, and to realize autonomy and independency of the system components to a large extent. The overall architecture of ADEPT2 is layered (cf. Figure 13). Thereby, components of lower layers hide as much complexity as possible from upper layers. Basic components are combinable in a flexible way to realize higher-level services like ad-hoc flexibility or process schema evolution. To foster this, ADEPT2 system components are reusable in different context using powerful configuration facilities.

To make implementation and maintenance of the different system components as easy as possible, each component is kept as simple as possible and only has access to the information needed for its proper functioning. Furthermore, communication details are hidden from component developers and independency from the used middleware components (e.g., database management systems) has been realized. Two important design goals concern avoidance of code redundancies and system extensibility:

- *Avoidance of code redundancies*. One major design goal for the ADEPT2 system architecture was to avoid code redundancies. For example, components for process modeling, process schema

-

evolution, and ad-hoc process changes are more or less based on the same set of change operations. This suggests to implement these operations by one separate system component, and to make this component configurable such that it can be reused in different context. Similar considerations have been made for other ADEPT2 components (e.g., visualization, logging, versioning, and access control). This design principle does not only reduce code redundancies, but also results in better maintainability, decreased cost of change, and reduced error rates.

- *Extensibility of system functions*. Generally, it must be possible to add new components to the overall architecture or to adapt existing ones. Ideally, such extensions or changes do not affect other components; i.e., their implementations must be robust with respect to changes of other components. As example assume that the set of supported change operations shall be extended (e.g., to offer additional change patterns to users). This extension, however, must not affect the components realizing process schema evolution or ad-hoc flexibility. In ADEPT2 we achieve this by mapping high-level change operations internally to a stable set of low-level change primitives (e.g., to add/delete nodes).

Figure 13 depicts the overall architecture of the ADEPT2 process management system, which features a layered and service-oriented architecture. Each layer comprises different components offering services to upper-layer components. The first layer is a thin abstraction on SQL, enabling a DBMS independent implementation of persistency. The second layer is responsible for storing and locking different entities of the process management system (e.g., process schemes and process instances). The third layer encapsulates essential process support functions including process enactment and change management. The topmost layer provides different buildtime and runtime tools to the user, including a process editor and a monitoring component.

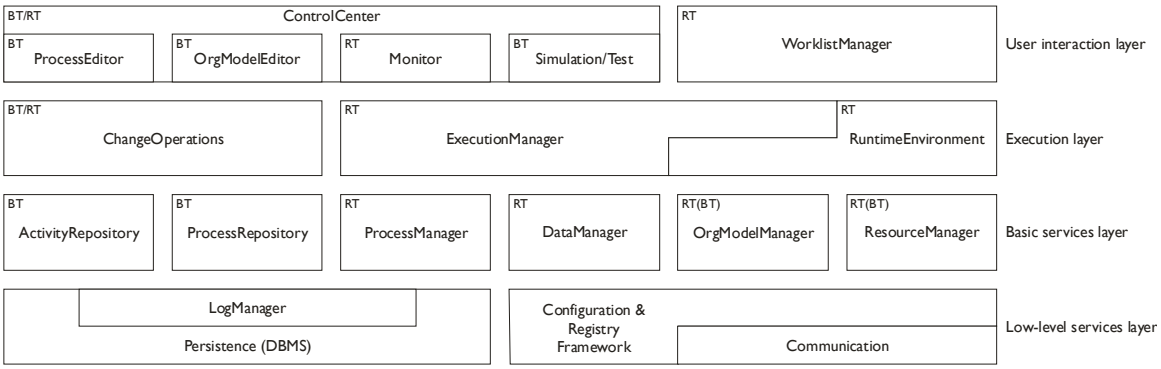


Figure 13: Basic Architecture of ADEPT2 (BT: Buildtime; RT: Runtime)

Components of the ADEPT2 architecture are loosely coupled enabling the easy exchange of component implementations. Furthermore, basic infrastructure services like storage management or the techniques used for inter-component communication can be easily exchanged. Additional plug-in interfaces are provided which allow for the extension of the core architecture, the data models, and the user interface.

Implementation of the different components of the ADEPT2 architecture raised many challenges, e.g., with respect to storage representation of schema and instance data: Unchanged instances are stored in a redundant-free manner by referencing their original schema and by capturing instance-specific data (e.g., activity states). As example consider instances I1, I3, I4, and I6 from Figure 14. For changed (”biased”)

-

instances, this approach is not applicable. One alternative would be to maintain a complete schema for each biased instance, another to materialize instance-specific schemes on-the-fly. ADEPT2 follows a hybrid approach: For each biased instance we maintain a minimal substitution block that captures all changes applied to it so far. This block is then used to overlay parts of the original schema when accessing the instance (I2 and I5 in our example from Figure 14).

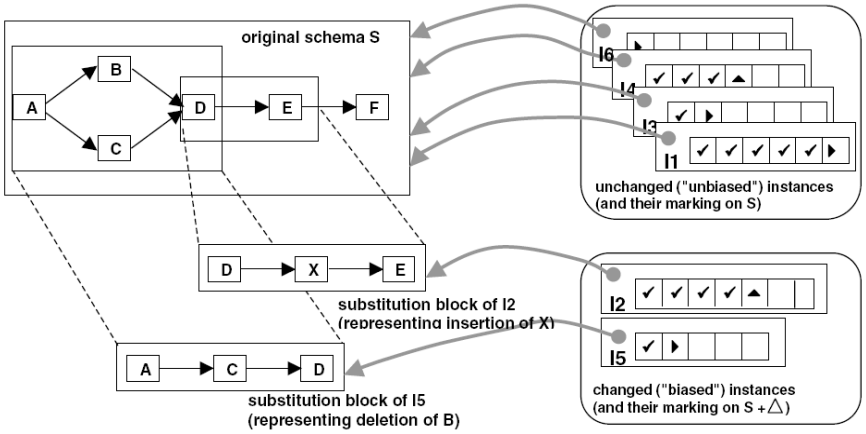


Figure 14: Managing Template and Instance Objects in the ProcessManager (Logical View)

ADEPT2 provides sophisticated buildtime and runtime components to the different user groups. This includes tools for modeling, verifying and testing process schemes, components for monitoring and dynamically adapting process instances, and different worklist clients (incl. Web clients). Many applications, however, require adapted user interfaces and functions to integrate process support features the best possible way. On the one hand, the provided user components are configurable in a flexible way. On the other hand, all functions (e.g., ad-hoc changes) offered by the process management system are made available via programming interfaces (APIs) as well.

We have implemented the described architecture in a proof-of-concept prototype in order to demonstrate major flexibility concepts and their interplay. Figure 15 shows a screen of the ADEPT2 process editor, which constitutes the main system component for modeling and adapting process schemes.

This editor allows to quickly compose new process templates out of pre-defined activity templates, to guarantee schema correctness by construction and on-the-fly checks, and to integrate application components (e.g., web services) in a plug-and-play like fashion. Another user component is the ADEPT2 Test Client. It provides a fully-fledged test environment for process execution and change. Unlike common test tools, this client runs on a light-weight variant of the ADEPT2 process management system. As such, various execution modes between pure simulation to production mode become possible.

SUMMARY AND OUTLOOK

The ADEPT2 technology meets major requirements claimed for next generation process management technology. It provides advanced functionality to support process composition by plug & play of arbitrary application components, it enables ad-hoc flexibility for process instances without losing control, and it supports process schema evolution in a controlled and efficient manner. As opposed to many other PAISs all these aspects work in interplay as well. For example, it is possible to propagate process schema

changes to individually modified process instances or to dynamically compose processes out of existing application components. All in all such a complex system requires an adequate conceptual framework and a proper system architecture. ADEPT2 considers both conceptual and architectural issues in the design of a next generation process management system.

Challenges on which we are currently working include the following ones: dynamic changes of distributed processes and process choreographies (Reichert & Bauer, 2007; Rinderle, Wombacher, & Reichert, 2006c), data-driven modeling, coordination and adaptation of large process structures (Rinderle & Reichert, 2006a; Müller, Reichert, & Herbst, 2007 + 2008), process configuration (Hallerbach, Bauer, & Reichert, 2008; Thom, Reichert, Chiao, Iochpe, & Hess, 2008), process variants mining (Li et al., 2008b), process visualization and monitoring (Bobrik et al., 2006, 2007), dynamic evolution of other PAIS aspects (Rinderle & Reichert, 2005b and 2007; Ly, Rinderle, Dadam, & Reichert, 2005), and evaluation models for (adaptive) PAISs (Mutschler, Reichert, & Rinderle, 2007; Mutschler & Reichert, 2008c). All these activities target at full process lifecycle support in process-aware information systems (Weber, Reichert, Wild, & Rinderle-Ma, 2008c).

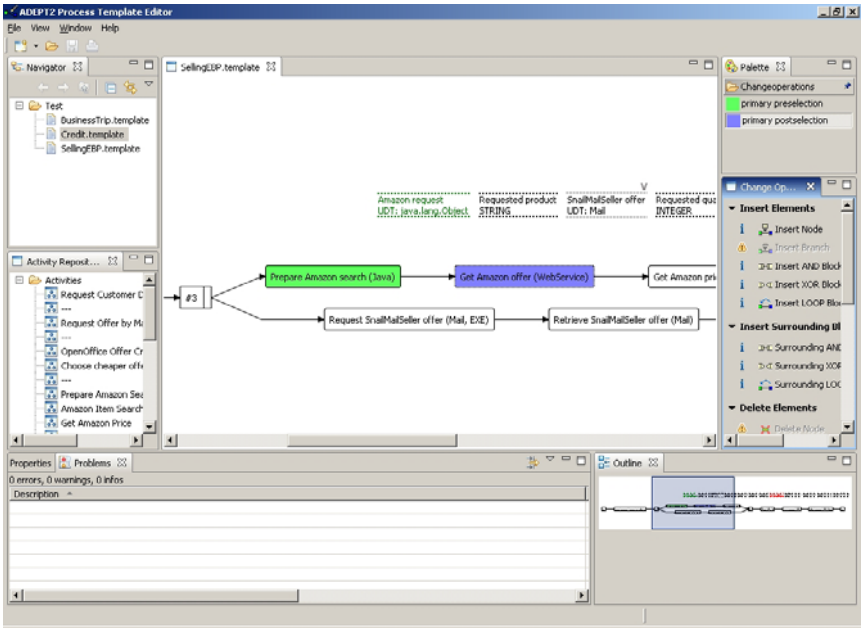


Figure 15: Screenshot of ADEPT2 Process Editor

REFERENCES

Adams, M., ter Hofstede, A., Edmond, D., & van der Aalst, W.M.P. (2006). A service-oriented implementation of dynamic flexibility in workflows. In *Proceedings of the 14th Int'l Conf. on Cooperative Information Systems (CoopIS'06)*, Montpellier, France, LNCS 4275, pp. 291-308.

Bassil, S., Benyoucef, M., Keller, R., Kropf, P. (2002): Addressing dynamism in e-negotiations by workflow management systems. In *Proceedings DEXA'02 Workshops*, pp. 655-659.

Bassil, S., Keller, R., & Kropf, P. (2004). A workflow-oriented system architecture for the management of container transportation. In *Proceedings of the 2nd Int'l Conf. on Business Process Management (BPM'04)*, Potsdam, Germany, LNCS 3080, pp. 116-131.

-

Bauer, T., Reichert, M., & Dadam, P. (2003). Intra-subnet load balancing in distributed workflow management systems. *Int'l Journal Cooperative Information Systems (IJCIS)*, 12(3), 295-323.

Bobrik, R., Bauer, T., & Reichert, M. (2006) Proviado – personalized and configurable visualizations of business processes. In *Proceedings 7th Int'l Conf. on Electronic Commerce and Web Technologies (EC-WEB'06)*, Krakow, Poland, LNCS 4082, pp. 61-71.

Bobrik, R., Reichert, M, & Bauer, T. (2007). View-based process visualization. In *Proceedings of the 5th Int'l Conf. on Business Process Management (BPM'07)*, Brisbane, Austalia, LNCS 4714, pp. 88-95.

Casati, F., Ceri, S., Pernici, B., & Pozzi, G. (1998). Workflow evolution. *Data and Knowledge Engineering*, 24(3), 211-238.

Dadam, P., Reichert, M., & Kuhn, K. (2000). Clinical workflows - the killer application for process-oriented information systems? In *Proceedings of the 4th Int'l Conf. on Business Information Systems (BIS'2000)*, Poznan, Poland, Springer, pp. 36-59.

Golani, M. & Gal, A. (2006). Optimizing exception handling in workflows using process restructuring. In *Proceedings of the 4th Int'l Conf. Business Process Management (BPM'06)*, Vienna, Austria, LNCS 4102, pp. 407-413.

Gschwind, T., Koehler, J., & Wong, J. (2008). Applying patterns during business process modeling. In *Proceedings of the 6th Int'l Conf. Business Process Management (BPM'08)*, Milan, Italy, LNCS 5240, pp. 4-19.

Günther, C.W., Rinderle, S., Reichert, M., & van der Aalst, W.M.P. (2006). Change mining in adaptive process management systems. In *Proceedings of the 14th Int'l Conf. on Cooperative Information Systems (CoopIS'06)*, Montpellier, France, LNCS 4275, pp. 309-326.

Günther, C.W., Rinderle-Ma, S., Reichert, M., van der Aalst, W.M.P., & Recker, J. (2008). Using process mining to learn from process changes in evolutionary systems. *Int'l Journal of Business Process Integration and Management*, 3(1), 61-78.

Hallerbach, A., Bauer, T., & Reichert, M. (2008). Managing process variants in the process lifecycle. In: *Proceedings of the 10th Int'l Conf. on Enterprise Information Systems (ICEIS'08)*, Barcelona, Spain, pp. 154-161.

Karbe, B. & Ramsperger, N. (1991): Concepts and implementation of migrating office processes. *Wissensbasierte Systeme*, pp. 136-147.

Khalaf, R., Keller, A., & Leymann, F. (2006). Business processes for web services: principles and applications. *IBM Systems Journal*, 45(2), 425-446.

Lenz, R. & Reichert, M. (2007). IT support for healthcare processes – premises, challenges, perspectives. *Data and Knowledge Engineering*, 61(1), 39-58.

Li, C., Reichert, M., & Wombacher, A. (2008a). On measuring process model similarity based on high-level change operations. In *Proceedings of the 27th Int'l Conf. on Conceptual Modeling (ER'08)*, Barcelona, Spain. Springer, LNCS, 2008.

Li, C., Reichert, M., & Wombacher, A. (2008b). Discovering reference process models by mining process variants. In *Proceedings of the 6th Int'l Conference on Web Services (ICWS'08)*, Beijing, China. IEEE Computer Society Press, 2008.

Li, C., Reichert, M., & Wombacher, A. (2008c). Mining based on learning from process change logs. In *Proceedings BPM'08 workshops – 4th Int'l Workshop on Business Process Intelligence (BPI'08)*, Milan, Italy. LNBIP (to appear).

Ly, L.T., Rinderle, S., Dadam, P., & Reichert, M. (2005) Mining staff assignment rules from event-based data. In *Proceedings of the BPM'05 workshops*, Nancy, France. Springer, LNCS 3812, pp. 177-190.

Ly, L.T., Göser, K., Rinderle-Ma, S., & Dadam, P. (2008) Compliance of semantic constraints – a requirements analysis for process management systems. In *Proceedings 1st Int'l Workshop on Governance, Risk and Compliance - Applications in Information Systems (GRCIS'08)*, Montpellier, France.

-

Ly, L.T., Rinderle, S., & Dadam, P. (2008) Integration and verification of semantic constraints in adaptive process management systems. *Data and Knowledge Engineering* , 64(1), pp. 3-23.

Minor, M., Schmalen, D., Koldehoff, A., & Bergmann, R. (2007). Structural adaptation of workflows supported by a suspension mechanism and by case-based reasoning. In *Proceedings of the WETICE'07 workshops*, IEEE Computer Press, pp. 370-375.

Müller, R., Greiner, U., & Rahm, E. (2004). AgentWork: A workflow system supporting rule-based workflow adaptation. *Data and Knowledge Engineering*, 51 (2), 223-256.

Müller, D., Herbst, J., Hammori, M., & Reichert, M. (2006). IT support for release management processes in the automotive industry. In *Proceedings of the 4th Int'l Conf. on Business Process Management (BPM'06)*, Vienna, Austria. LNCS 4102, pp. 368-377.

Müller, D., Reichert, M., & Herbst, J. (2007). Data-driven modeling and coordination of large process structures. In *Proceedings of the 15th Int'l Conf. on Cooperative Information Systems (CoopIS'07)*, Vilamoura, Algarve, Portugal, LNCS 4803, pp. 131-149.

Müller, D., Reichert, M., & Herbst, J. (2008). A new paradigm for the enactment and dynamic adaptation of data-driven process structures. In *Proceedings of the 20th Int'l Conf. on Advanced Information Systems Engineering (CAiSE'08)*, Montpellier, France, LNCS 5074, pp. 48-63.

Mutschler, B., Bumiller, J., & Reichert, M. (2006). Why process-orientation is scarce: an empirical study of process-oriented information systems in the automotive industry. In *Proceedings of the 10th Int'l Conf. on Enterprise Computing (EDOC '06)*, Hong Kong, IEEE Computer Press, pp. 433-440.

Mutschler, B., Reichert, M., & Rinderle, S. (2007). Analyzing the dynamic cost factors of process-aware information systems: a model-based approach. In *Proceedings of the 19th Int'l Conf. on Advanced Information Systems Engineering (CAiSE'07)*, Trondheim, Norway. LNCS 4495, pp. 589-603.

Mutschler, B., Weber, B., & Reichert, M. (2008a). Workflow management versus case handling: results from a controlled software experiment. In *Proceedings of the 23rd Annual ACM Symposium on Applied Computing (SAC'08)*, Fortaleza, Brazil, pp. 82-89.

Mutschler, B., Reichert, M., & Bumiller, J. (2008b): Unleashing the effectiveness of process-oriented information systems: problem analysis, critical success factors and implications, *IEEE Transactions on Systems, Man, and Cybernetics*, 38(3), 280-291.

Mutschler, B. & Reichert, M. (2008c). On modeling and analyzing cost factors in information systems engineering. In *Proceedings of the 20th Int'l Conf. on Advanced Information Systems Engineering (CAiSE'08)*, Montpellier, France. LNCS 5074, pp. 510-524.

Pesic, M., Schonenberg, M., Sidorova, N., & van der Aalst, W.M.P. (2007). Constraint-based workflow models: change made easy. In *Proceedings of the 15th Int'l Conf. on Cooperative Information Systems (CoopIS'07)*, Vilamoura, Algarve, Portugal, LNCS 4803, pp. 77-94.

Reichert, M. & Dadam, P. (1997). A framework for dynamic changes in workflow management systems. In *Proc. 8th Int'l Workshop on Database and Expert Systems Applications*, Toulouse, pp. 42–48.

Reichert, M. & Dadam, P. (1998a). ADEPTflex – supporting dynamic changes of workflows without losing control. *Journal of Intelligent Information Systems*, 10(2), 93-129.

Reichert, M., Hensinger, C., & Dadam, P. (1998b). Supporting adaptive workflows in advanced application environments. In *Proceedings of the EDBT Workshop on Workflow Management Systems (in conjunction with EDBT'98 conference)*, Valencia, Spain, pp. 100-109.

Reichert, M., Bauer, T., & Dadam, P. (1999): Enterprise-wide and cross-enterprise workflow-management: challenges and research issues for adaptive workflows. In *Proceedings of the Informatik'99 Workshop on Enterprise-wide and Cross-enterprise Workflow Management*, CEUR Workshop Proceedings, Vol. 24, pp. 56-64.

Reichert, M. (2000): Dynamische Ablaufänderungen in Workflow Management Systemen. *Dissertation*, Universität Ulm, Fakultät für Informatik, Juli 2000.

Reichert, M., Dadam, P., & Bauer, T. (2003a). Dealing with forward and backward jumps in workflow management systems. *Int'l Journal Software and Systems Modeling*, 2(1), 37-58.

Reichert, M., Rinderle, S., & Dadam, P. (2003b). On the common support of workflow type and instance changes under correctness constraints. In *Proc. 11th Int'l Conf. Cooperative Information Systems (CooplS '03)*, Catania, Italy, LNCS 2888, pp. 407-425.

Reichert, M., Rinderle, S., Kreher, U., & Dadam, P. (2005). Adaptive process management with ADEPT2. In *Proceedings of the 21st Int'l Conf. on Data Engineering (ICDE'05)*, Tokyo.

Reichert, M. & Rinderle, S. (2006). On design principles for realizing adaptive service flows with BPEL. In *Proceedings EMISA'06*, Hamburg, Lecture Notes in Informatics (LNI) P-95, pp. 133-146.

Reichert, M. & Bauer, T. (2007): Supporting ad-hoc changes in distributed workflow management systems. In *Proceedings of the 15th Int'l Conf. on Cooperative Information Systems (CoopIS'07)*, Vilamoura, Algarve, Portugal, LNCS 4803, pp. 150-168.

Reijers, H. & van der Aalst, W.M.P. (2005). The effectiveness of workflow management systems: predictions and lessons learned. *Int'l Journal of Information Management*, 5, 457–471.

Rinderle, S., Reichert, M., & Dadam, P. (2003). Evaluation of correctness criteria for dynamic workflow changes. In *Proceedings of the 1st Int'l Conf. on Business Process Management (BPM '03)*, Eindhoven, Netherlands. Springer, LNCS 2678, pp. 41-57.

Rinderle, S., Reichert, M., & Dadam, P. (2004a). Correctness criteria for dynamic changes in workflow systems - a survey. *Data and Knowledge Engineering*, 50(1), 9-34.

Rinderle, S., Reichert, M., & Dadam, P. (2004b): Flexible support of team processes by adaptive workflow systems. *Distributed and Parallel Databases*, 16(1), 91-116.

Rinderle, S., Reichert, M., Dadam, P. (2004c): Disjoint and overlapping process changes - challenges, solutions, applications. In *Proceedings of the 12th Int'l Conf. Cooperative Information Systems (CoopIS'04)*, Agia Napa, Cyprus, LNCS 3290, pp. 101-120.

Rinderle, S., Reichert, M., & Dadam, P. (2004d). On dealing with structural conflicts between process type and instance changes. In *Proceedings of the 2nd Int'l Conf. Business Process Management (BPM'04)*, Potsdam, Germany, LNCS 3080, pp.274-289.

Rinderle, S., Weber, B., Reichert, M., & Wild, W. (2005a). Integrating process learning and process evolution - a semantics based approach. In *Proceedings of the 3rd Int'l Conf. Business Process Management (BPM'05)*, Nancy, France, LNCS 3649, pp. 252-267.

Rinderle, S. & Reichert, M. (2005b). On the controlled evolution of access rules in cooperative information systems. In *Proceedings of the 13th Int'l Conf. on Cooperative Information Systems (CoopIS'05)*, Agia Napa, Cyprus. Springer, LNCS 3760, pp. 238-255.

Rinderle, S. & Reichert, M. (2006a). Data-driven process control and exception handling in process management systems. In *Proceedings of the 18th Int'l Conf. on Advanced Information Systems Engineering (CAiSE'06)*, Luxembourg, LNCS 4001, pp. 273–287.

Rinderle, S., Reichert, M., Jurisch, M., & Kreher, U. (2006b). On representing, purging and utilizing change logs in process management systems. In *Proceedings of the 4th Int'l Conf. Business Process Management (BPM'06)*, Vienna, Austria, LNCS 4102, pp. 241-256.

Rinderle, S., Wombacher, A., & Reichert, M. (2006c). Evolution of process choreographies in DYCHOR. In *Proceedings of the 14th Int'l Conf. on Cooperative Information Systems (CoopIS'06)*, Montpellier, France, LNCS 4275, pp. 273-290.

Rinderle, S. & Reichert, M. (2007a). A formal framework for adaptive access control models. *Journal on Data Semantics*, IX, LNCS 4601, pp. 82-112.

Rinderle, S., Jurisch, M., Reichert, M. (2007b). On deriving net change information from change logs – the DELTALAYER algorithm. In *Proceedings of the 12th Conf. on Database Systems in Business, Technology and Web (BTW'07)*, Aachen, Lecture Notes in Informatics, LNI-103, pp. 364-381.

Rinderle-Ma, S., Reichert, M., & Weber, B. (2008a). Relaxed compliance notions in adaptive process management systems. In *Proceedings of the 27th Int'l Conference on Conceptual Modeling (ER'08)*, Barcelona, Spain. Springer, LNCS.

Rinderle-Ma, S., Reichert, M., & Weber, B. (2008b). On the formal semantics of change patterns in process-aware information systems. In *Proceedings of the 27th Int'l Conference on Conceptual Modeling (ER'08)*, Barcelona, Spain. Springer, LNCS.

Rinderle-Ma, S. & Reichert, M. (2008c) Managing the lfe cycle of access rules in CEOSIS. In *Proceedings of the 12th IEEE Int'l Enterprise Computing Conference (EDOC'08)*, Munich, Germany.

Sadiq, S., Sadiq, W., Orlowska, M. (2001). Pockets of flexibility in workflow specifications. In *Proceedings of the 20th Int'l Conference on Conceptual Modeling (ER'01)*, Yokohama, Japan, LNCS 2224, pp. 513-526.

Schonenberg, H., Weber, B., van Dongen, B., & van der Aalst, W.M.P. (2008). Supporting flexible processes by recommendations based on history. In *Proceedings of the 6th Int'l Conf. on Business Process Management (BPM'08)*. Milan, Italy, LNCS 5240, pp. 51-66.

Thom, L., Reichert, M., Chiao, C., Iochpe, C., & Hess, G. (2008). Inventing less, reusing more and adding intelligence to business process modeling. In *Proceedings of the 19th Int'l Conference on Database and Expert Systems Applications (DEXA '08)*, Turin, Italy. LNCS 5181, pp. 837-850.

Van der Aalst, W.M.P., van Hee, K.M. (2002): Workflow management: models, methods, and systems MIT Press.

Van der Aalst, W.M.P., ter Hofstede, A., Kiepuszewski, B., & Barros, A. (2003). Workflow patterns, *Distributed and Parallel Databases*, 14 (1), 5–51.

Van der Aalst, W.M.P., Weske, M., & Grünbauer, D. (2005). Case handling: a new paradigm for business process support. *Data and Knowledge Engineering*, 53 (2), 129-162.

Van Dongen, B., de Medeiros A., Verbeek, H., Weijters, A., & van der Aalst, W.M.P. (2005). The ProM framework: a new era in process mining tool support. In *Proceedings 26th Int'l Conf. on the Applications and Theory of Petri Nets (ICATPN'05)*, Miami, USA, LNCS 3536, pp. 444-454.

Weber, B., Wild, W., & Breu, B. (2004). CBRFlow. enabling adaptive workflow management through conversational case-based reasoning. In *Proceedings of the ECCBR'04 conference*. Madrid, Spain, LNCS 3155, pp. 434-448.

Weber, B., Reichert, M. Wild, W., & Rinderle, S. (2005a). Balancing flexibility and security in adaptive process management systems. In *Proceedings of the 13th Int'l Conf. on Cooperative Information Systems (CoopIS'05)*, Agia Napa, Cyprus, LNCS 3760, pp. 59-76.

Weber, B., Reichert, M., Rinderle, S., & Wild, W. (2005b). Towards a framework for the agile mining of business processes. In *Proceedings of the BPM'05 Workshops*, Nancy, France, LNCS 3812, pp. 191-202.

Weber, B., Rinderle, S., Wild, W., & Reichert, M. (2005c) CCBR–driven business process evolution. In Proceedings of the 6th Int'l Conf. on Case-Based Reasoning (ICCBR'05), Chicago. LNCS 3620, pp. 610-624.

Weber, B., Reichert, M., & Wild, W. (2006) Case-base maintenance for CCBR-based process evolution. In Proceedings of the 8th European Conf. on Case-Based Reasoning (ECCBR'06), Ölüdeniz/Fethiye, Turkey. LNCS 4106, pp. 106-120.

Weber, B., Rinderle, S., & Reichert, M. (2007). Change patterns and change support features in process-aware information systems. In *Proceedings of the 19th Int'l Conf. on Advanced Information Systems Engineering (CAiSE'07)*, Trondheim, Norway, LNCS 4495, pp. 574-588.

Weber, B. & Reichert, M. (2008a). Refactoring process models in large process repositories. In *Proceedings of the 20th Int'l Conf. on Advanced Information Systems Engineering (CAiSE'08)*, Montpellier, France, LNCS 5074, pp. 124-139.

-

Weber, B., Reichert, M., & Rinderle-Ma, S. (2008b). Change patterns and change support features – enhancing flexibility in process-aware information systems. *Data and Knowledge Engineering* , 66(3), 438-466.

Weber, B., Reichert, M., Wild, W., & Rinderle-Ma, S. (2008c). Providing integrated life cycle support in process-aware information systems. *Int'l Journal of Cooperative Information Systems (IJCIS)*, World Scientific Publ. (to appear)

Weske, M. (2000). Workflow management systems: formal foundation, conceptual design, implementation aspects., University of Münster, Germany, Habilitation Thesis, 2000.

Weske, M. (2007). Business Process Management. Berlin:Springer.

-