Architectural Principles and Components of Adaptive Process Management Technology

Manfred Reichert[1], Peter Dadam[1], Stefanie Rinderle-Ma[1],
Martin Jurisch[2], Ulrich Kreher[2], Kevin Göser[2]

[1]Institute of Databases and Information Systems, Ulm University, Germany
{peter.dadam, manfred.reichert, stefanie.rinderle}@uni-ulm.de
[2]AristaFlow GmbH, Germany
{martin.jurisch, ulrich.kreher, kevin.goeser}@aristaflow.com

Abstract. Process-aware information systems (PAIS) must not freeze business processes, but should enable authorized users to deviate from the implemented workflows on-the-fly and to dynamically evolve them over time. While there has been a lot of work on the theoretical foundations of dynamic process changes, there is still a lack of implemented PAIS providing this dynamics. Designing the architecture of such adaptive PAIS, however, constitutes a big challenge due to the high complexity coming with dynamic changes. Besides this, performance, robustness, security and usability of the PAIS must not be affected by the added flexibility. In the AristaFlow project we follow a holistic approach to master this complexity. Based on a conceptual framework for adaptive process management, we have designed a sophisticated architecture for next generation process management technology. This paper discusses major design goals and basic architectural principles, gives insights into selected system components, and shows how change support features can be realized in an integrated and efficient manner.

1 Introduction

In today's dynamic business world enterprises must be able to quickly and flexibly react to changes in their environment [Guen06,RRD03a,RRD04a,WRR07]. Therefore, companies have recognized business agility as a competitive advantage, which is fundamental for being able to cope with trends like increasing product and service variability, faster time-to-market, and business-on-demand. *Process-aware information systems* (PAIS) offer promising perspectives in this respect and a growing interest in aligning information systems in a process-oriented way can be observed [BDR03,LeRe07, Müll06,Wesk07]. As opposed to data- or function-centered information systems, PAIS are characterized by a strict separation of process logic and application code. In particular, most PAIS describe process logic explicitly in terms of a *process template* providing the *schema* for process enactment. Usually, the core of the process layer is built by a *process management system*, which provides generic functions for modeling, executing, and monitoring processes [Wesk07]. This allows for a separation of concerns, which is a well established principle for increasing maintainability and for reducing cost of change; i.e., changes to one layer can be performed without affecting other layers.

The ability to deal with process change has been identified as one of the most critical success factors for PAIS [LRW08,RDB03,ReDa98,RRD04a,WRR07]. Through the described separation of concerns PAIS facilitate changes significantly. However, enterprises are still reluctant to adapt PAIS once they are running properly. High complexity and cost of change are mentioned as major obstacles for not fully leveraging the potential of PAIS [MRB08]. To improve this situation more flexible PAIS are needed enabling companies to capture real-world processes adequately without leading to mismatches between computerized processes and those running in reality. Instead, users need to be able to deviate from the predefined processes if required and to evolve process implementations over time [RRD03a]. Such changes must be possible at a high level of abstraction and without affecting PAIS consistency and robustness [RRD04a-d].

Basically, process changes may take place at the *instance* or the *type level*. Process instance changes often have to be carried out in an ad-hoc manner to deal with exceptional situations [ReDa98,DRK00,LeRe07]. Such *ad-hoc changes* must not affect system robustness or lead to errors in the sequel. Process type changes, in turn, correspond to continuous changes of a process schema (also denoted as *process schema evolution*) to deal with evolving needs [RRD04a-d]. Regarding long-running processes, it might also require the migration of already running process instances to the new schema version. Important challenges in this context are to perform such instance migrations on-the-fly, to guarantee compliance of the migrated process instances with the new process schema, and to avoid performance penalties [Rind06,RRD04a-b].

The design of adaptive process management technology constitutes an enormous challenge. On the one hand such technology should allow for efficient process enactment as well as for enterprise application integration. On the one hand it has to provide support for dynamic process changes. When designing such a technology we have to cope with many trade-offs. For example, complexity of dynamic changes increases the higher expressiveness of the used process modeling language becomes. Further, complex interdependencies between the different features of adaptive PAIS exist that must be carefully understood to avoid implementation gaps. Process schema evolution, for example, requires integrated support for high-level change patterns, schema versioning, change logging, on-the-fly instance migrations, and dynamic worklist adaptations. Finally, even if the conceptual pillars of such a next generation process management technology are well understood, it will still be a quantum leap to implement advanced features in an integrated, efficient and robust manner.

In the AristaFlow project we have followed a holistic approach to tackle these challenges. Based on a sophisticated conceptual framework for dynamic process changes, which we developed in the ADEPT project [ReDa98,RRD04b], we have designed the architecture of the ADEPT2 process management system and prototypically implemented it. From the very beginning, one of the primary design goals has been process flexibility. This paper summarizes major architectural principles, gives insights into ADEPT2 system components and their interactions, and shows how change support features can be realized in an integrated and efficient manner within such architecture.

The remainder of this paper is organized as follows: Section 2 presents background information needed for understanding this paper. Section 3 summarizes architectural principles applied during the design of the ADEPT2 system and its components. Section 4 shows how ad-hoc changes and schema evolution are enabled within this architecture. Section 5 discusses related work and Section 6 concludes with a summary.

2 Conceptual Framework for Dynamic Changes in ADEPT2

We developed a framework for dynamic process changes in previous projects [ReDa98, RRD04b]. In this paper we use it as conceptual pillar for designing the ADEPT2 system architecture. ADEPT2 covers changes at both the process instance and the process type level. This, in turn, allows for *ad-hoc flexibility* as well as for *process schema evolution*.

Ad-hoc flexibility. At the instance level the ADEPT2 framework allows for ad-hoc deviations from the pre-modeled process schema (e.g., to insert, delete, or move activities). Such ad-hoc changes do not lead to unstable system behavior, i.e., none of the guarantees (e.g., absence of deadlocks) achieved by formal model checks at buildtime are violated due to dynamic changes [ReDa98]. ADEPT2 provides a complete set of high-level change patterns for defining ad-hoc deviations; e.g., authorized users may dynamically add new activities or jump forward in the flow of control [RDB03, WRW05]. ADEPT2 defines pre- and post-conditions for all operations to ensure correctness. Further, all complexity associated with the adaptation of instance states, the re-mapping of activity inputs/ outputs, the handling of missing input data, or the problem of deadlocks is hidden to a large degree from users.

Process schema evolution. To cope with business process changes (e.g., due to reengineering efforts), ADEPT2 allows for quick and efficient adaptations of process templates (i.e., schema changes at type level) – in the following denoted as *process schema evolution* [RRD04b]. When updating a process template, usually, related process instances are finished according to the old schema version, while future instances are derived from the new one. However, such rigid approach is not adequate for long-running processes [LeRe07]. Here, the challenge is to *propagate* respective schema changes to already running instances of this process template as well; i.e., to migrate these process instances to the new schema version of the respective process template .

The on-the-fly migration of a collection of process instances to a modified process template must not violate correctness and consistency properties of these instances. Therefore, we need a general principle for arguing whether a process instance is compliant with an updated schema [RRD04a,RRD04b]. The ADEPT2 change framework uses a well-defined correctness criterion, which is independent of the underlying process meta model and which is based on a relaxed notion of trace equivalence. 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 are defined for each change operation (see Fig. 1 for an example). Finally, ADEPT2 automatically adapts the states of compliant instances when migrating them to an updated schema.

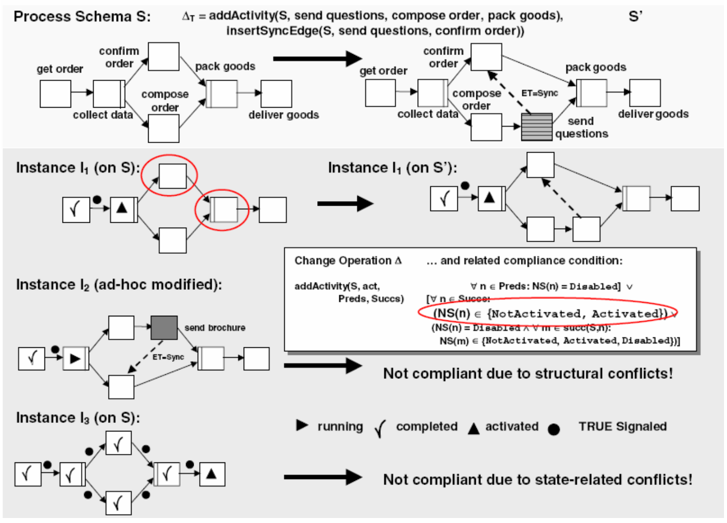
Process Schema S:
ΔT = addActivity(S, send questions, compose order, pack goods), insertSyncEdge(S, send questions, confirm order))
S'
get order
confirm order
collect data
compose order
pack goods
deliver goods
ET=Sync
send questions
Instance I1 (on S):
Instance I1 (on S'):
Instance I2 (ad-hoc modified):
send brochure
ET-Sync
Change Operation Δ
... and related compliance condition:
addActivity(S, act, Preds, Succs)
∀ n ∈ Preds: NS(n) = Disabled] ∨
[∀ n ∈ Succs:
(NS(n) ∈ {NotActivated, Activated}) ∨
(NS(n) = Disabled ∧ ∀ m ∈ succ(S,n):
NS(m) ∈ {NotActivated, Activated, Disabled})]
Not compliant due to structural conflicts!
Instance I3 (on S):
running
completed
activated
TRUE Signaled
Not compliant due to state-related conflicts!

Figure 1: Process Schema Evolution (Conceptual View)

When designing ADEPT2 we have looked at the picture as a whole. In particular, we have not considered the different kinds of changes in an isolated manner, but have investigated their complex interdependencies as well. For example, the correct handling of concurrent process changes is crucial in order to cover all practical cases [RRD04c-d]. In this context, we have dealt with the question how to propagate process template changes to related process instances which are in different states and to which various ad-hoc modifications have been previously applied. For such *biased* instances, current instance schema differs from the schema of the original template. Therefore, change propagation must be accomplished while considering certain correctness constraints to avoid inconsistencies. In this context, ADEPT2 excludes state-related, structural, and semantical conflicts between concurrent changes [RRD03a,RRD04c,RRD04d].

As example consider Fig. 1 where a new template version S' is created from a process template S based on which three instances are running. Instance I_1 can be migrated to the new process template 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 template schema. Though there is no state conflict regarding I_2 this instance can also not migrate to S'. I_2 has been individually modified by an ad-hoc change which is conflicting with the template change. More precisely, when propagating the process template change to I_2 a deadlock-causing cycle would occur. The ADEPT2 change framework provides efficient means to detect such structural conflicts [RRD03]. Basic to this are sophisticated conflict tests. In summary, we restrict propagation of a *process template change* to those instances for which the change does not conflict with instance state or previous ad-hoc changes. – So far, we have focused on our conceptual change framework, which constitutes the basis for the proper design of the ADEPT2 system architecture. Following sections illustrate how we have implemented this conceptual framework.

3 Design Principles and Components of the ADEPT2 Architecture

The design of the ADEPT2 system has been governed by a number of well-defined principles in order to realize a sustainable and modular system architecture. Considered design principles include general architectural aspects as well as conceptual issues concerning the different system features. Our overall goal is to enable ad-hoc flexibility and process schema evolution (cf. Section 2), together with other process support features, in an integrated way, while ensuring robustness, correctness, extensibility, performance and usability of the system at the same time. This section summarizes major design principles of the ADEPT2 architecture and gives an overview of its components.

3.1 General Design Principles and Goals

High-end process management technology has a complexity comparable to database systems. To master this complexity a proper and modular architecture is needed 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 system components to a large extent. To meet these goals the overall architecture should be layered. Thereby, components of lower layers must hide as much complexity as possible from upper layers. Basic components must be combinable in a flexible way to realize higher-level services like *ad-hoc flexibility* or *process schema evolution*. To achieve this, ADEPT2 components are reused in different context making use of sophisticated configuration facilities.

Process management systems should provide sophisticated buildtime and runtime components to the different user groups (e.g., process participants, process administrators, process designers). This includes buildtime tools for modeling, verifying and testing processes, runtime components for monitoring and dynamically adapting process instances, and worklist clients for accessing upcoming tasks. Many applications, however, require adapted user interfaces and functions to integrate process support features in the best possible way. On the one hand, provided user components should be configurable in a flexible manner. On the other hand, all functions offered by the process management system should be made available via application programming interfaces (APIs) as well. In particular, advanced system functions (e.g., ad-hoc changes or process schema evolution) should be accessible via such programming interfaces as well.

Implementation and maintenance of the different system components should be as easy as possible. Therefore 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) is realized.

To enable maintainability, extensibility and usability of the different system components we need a proper conceptual design for the ADEPT2 system architecture. We do not give an in-depth discussion of all considered design goals, but illustrate our main philosophy by means of two examples. Though these basic design principles apply to modern software architectures in general, we explicitly list them here to make clear how they affect the design of adaptive process management technology:

- *Reuse of code fragments*: A major design goal for any complex system architecture is 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 components (e.g., visualization, logging, versioning, and access control). This design principle does not only reduce code redundancies, but – as a consequence – 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 updates 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 provide higher level change patterns to users [WRR07]). This change, however, should neither affect the component realizing process schema evolution nor the one enabling ad-hoc flexibility. In ADEPT2 we achieve this by internally mapping high-level change operations to a stable set of low-level primitives (e.g., to add/delete nodes) [Rind06].

3.2 Overview of the ADEPT2 Architecture and its Components

Figure 2 depicts the overall architecture of the ADEPT2 process management system. Its development has been based on the design principles discussed before and on the experiences we gathered during implementation of ADEPT1 [RRD03b]. ADEPT2 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 templates 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 users, including a process editor and a process monitoring component.

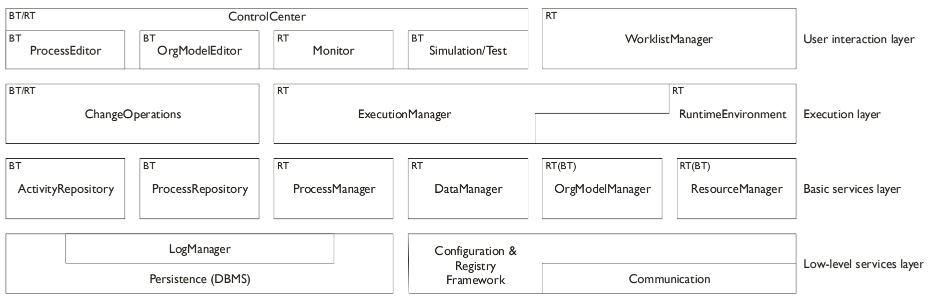

Figure 2: Basic Architecture of ADEPT2 (BT: Buildtime; RT: Runtime)

3.2.1 Layer with Low-level Services

This first layer of the ADEPT2 architecture comprises basic services which accomplish tasks like logging, persistency management, configuration support and communication. Idiosyncrasies of the used middleware services are hidden from upper-layer components. This allows us to use different database management systems or to exchange communication middleware without need for adapting implementations of upper layers.

Configuration & Registry Framework: This component provides the basic infrastructure for configuring and managing the different system components of the ADEPT2 architecture, and for enabling inter-component communication. The developed framework allows to start, manage and terminate ADEPT2 components (e.g., *ProcessManager*) as well as their services (e.g., managing instance data), and to flexibly configure them for use in different context. In addition, a generic interface is provided to realize communication between ADEPT2 components. Thereby, communication details (e.g., concerning transport protocols, interaction styles or message formats) are hidden from the components using this interface. For example, it remains transparent for these components whether the services they request are running locally or remotely.

LogManager: ADEPT2 logs all system events occurring at build- and runtime [Rind06, RJR07]. This includes events like changes in the state of a process instance, structural changes at type or instance level, or access to process data elements. The *LogManager* provides a generic interface based on which upper-layer components can log whatever events they want. Persistency is handled by a separate sub-component of the *LogManager*, which hides details of the underlying storage management component. This allows us to use different persistency managers (e.g., relational DBMS, XML files, flat files) without affecting implementation of upper layers.

3.2.2 Layer with Basic Services

Components of this layer provide basic services for managing build- and runtime data of the process management system and for making it available to upper-layer components.

ActivityRepository**:** This system component manages the activity templates based on which processes can be composed and executed. An activity template encapsulates all information needed for working on a particular task. In particular, it connects the activity to an application component. Thereby, details of the used component model (e.g., Enterprise Java Beans, (D)COM or Web services) are hidden from other ADEPT2 system components. Activity templates comprise additional information needed for proper activity execution. Based on it, for example, one can figure out whether the associated application component can be interrupted or aborted during runtime.

ProcessRepository**:** This component manages process templates and their meta data. Similar to activity templates, process templates can be used as building blocks when composing a new process model. Note that this allows for the realization of sub processes in an easy and intuitive manner. Furthermore, the *ProcessRepository* component manages all versions of a process template and the information needed to derive them from each other; i.e. change logs [RJR07] regarding the process templates.

ProcessManager**:** While the above components manage buildtime data, the *Process-Manager* component provides exactly those information needed for process enactment. This includes, for example, schemes of active process templates and running process instances as well as current instance states. In particular, *ProcessManager* restores instance-specific schemes for those process instances that were subject of previous ad-hoc changes. As opposed to *ProcessRepository*, *ProcessManager* has no knowledge about the evolution of process or activity templates; i.e., it is not aware of different template versions and their relations. This minimalism allows for efficient process enactment. As we discuss in Section 4, *ProcessManager* also deals with the migration of running process instances to a new process template version. It then has to interact with the *ProcessRepository* in order to retrieve the information required in this context; i.e., the schemes of the old and the updated process template as well as their difference.

DataManager**:** For each process instance the *DataManager* maintains all process (relevant) data created during process enactment; i.e., all data elements and their values written by certain activities and read by other ones. Since process relevant data can become quite extensive and must be accessible by external components as well, they are not maintained within the *ProcessManager,* but through a separate component. The *DataManager* keeps all versions of a data element and creates a log entry each time the data element is accessed (in cooperation with the *LogManager*). Finally, the *DataManager* allows for implementing access functions for user-defined data types.

OrgModelManager and ***ResourceManager*** To define potential actors for a particular activity, it can be associated with an actor assignment. Such assignment refers to organizational entities (e.g., organizational units, project teams, roles, actors) or organizational relations (e.g., "is-manager-of") as captured in an organizational model. The *Org-ModelManager* maintains this organizational model. It further accepts an actor assignment as input and delivers all actors qualifying for the respective expression as result. Besides actors, additional resources (e.g. rooms, machines and software licenses) can be maintained using the *ResourceManager* component.

3.2.3 Execution Layer

This layer comprises functional components of the ADEPT2 architecture which allow for the correct enactment and adaptation of process instances and related activities.

ChangeOperations: This component comprises high-level change operations that can be applied to processes in different context (e.g., to add or delete activities). First, change operations are required when creating new process templates or when adapting existing ones. In the latter case respective schema changes can be propagated to already running process instances as well; i.e., we (logically) apply the operations at instance level. The same applies with respect to ad-hoc instance changes. Note that in all these cases same or similar change operations are needed. Our basic design principles (cf. Section 3.1) therefore suggest to implement these change operations in a separate component to avoid code redundancies and to improve code maintainability. Each change operation realizes certain process graph transformations and is based on well-defined pre-/post-conditions in order to guarantee soundness of a process schema after its change. Note that conditions are varying depending on whether the change is applied at type or instance level.

ExecutionManager: This component coordinates the execution of process instances in an efficient and correct way; e.g., it evaluates predicates on instance data to choose between alternative branches or to loop back during runtime. As a prerequisite *ExecutionManager* needs information about the current schema as well as the state of respective instances. This information is provided by *ProcessManager*, i.e., a lower-layer component. For *ExecutionManager* it remains transparent whether a process instance is still running on its original schema or on a modified schema (as result of applied ad-hoc changes). When an activity is started the *ExecutionManager* provides the invoked application component with needed input data; when the activity completes, in turn, the *ExecutionManager* takes over its output data and forwards it to the *DataManager*.

RuntimeEnvironment: This component provides the container for executing arbitrary applications. It retrieves the input data of the respective application from the DataManager and prepares it for application invocation; i.e., the invoked application component does not need any knowledge about the specific process context in which it is executed. After completing an application execution successfully, in turn, the container receives the application output data and forwards it to the *DataManager*. Besides this, the *RuntimeEnvironment* allows to start application components and to control their execution (e.g., to abort or suspend component execution). Finally, the *RuntimeEnvironment* informs the ExecutionManager when the execution of an application fails.

3.2.4 User Interaction Layer

This layer comprises those components of the ADEPT2 architecture with which the different user groups interact. According to our basic philosophy all functions provided by these interactive components are made available via *application programming interfaces* (APIs) as well. This allows users to replace standard tools (e.g. for editing process templates or for managing user worklists) by own tool implementations.

ControlCenter: The ADEPT2 *ControlCenter* provides advanced buildtime and runtime components for user interactions. This includes the *ProcessEditor*, the *OrgModelEditor*, *Test Clients*, and the *Runtime Monitor*. The *ProcessEditor*, for example, constitutes the major component for modeling process templates and for guaranteeing model correctness (see Section 5). The *TestClient*, in turn, is a fully-fledged test environment for process execution. Unlike commonly known simulation tools, it runs on a lightweight instance of the process management system itself. As such, various execution modes between pure simulation and production mode are possible.

WorklistManager: This component manages worklists. When an activity becomes activated *WorklistManager* dissolves the corresponding actor assignment (in cooperation with *OrgModelManager*) and updates the respective worklists. The *WorklistManager* also considers deputy arrangements and allows to delegate work items to other users (even if the respective activity has been already started). Finally, escalation will be provided if a selected work item is not processed within a specified duration of time.

In summary, all described ADEPT2 system components 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 model and the user interface.

4 Architectural Support for Dynamic Process Changes in ADEPT2

So far we have introduced the ADEPT2 conceptual framework for dynamic process changes and we have sketched the different layers of the ADEPT2 system architecture. In this section we give insights into the realization of the ADEPT2 change framework within this architecture. Taking *process schema evolution* as example, we show in which way the different architectural components contribute to realize this feature and how they interact with each other to do this in a proper and efficient way.

4.1 A 2-phase procedure for realizing process schema evolution

When considering the ADEPT2 system architecture from Fig. 2 the general procedure for performing a *process schema evolution* comprises two phases (note that this procedure is simplified and does not consider interactions with lower-level services):

Phase I: Preparation Phase

1. Load an existing process template into the *ProcessEditor* and adapt its schema `S` using the change operations provided by *ChangeOperations*. Exactly the same set of change operations can be applied as when creating new process templates.
2. Record the modified process template (i.e., its target schema `S'`), together with the applied changes (i.e., the difference between `S'` and `S`), in the *ProcessRepository*.

Phase II: Schema Evolution Phase

3. Suspend (i.e. freeze) all process instances which are running on original process schema `S` and which shall be migrated to target schema `S'` (if possible).
4. Load target schema `S'`into *ProcessManager*. New instances are created based on `S'`.
5. Select original schema `S` and target schema `S'`in the *ProcessRepository* and transmit information about the schema difference `Delta` to the *ProcessManager*.
6. Based on `Delta`, for each frozen instance *ProcessManager* checks whether it is compliant with target schema `S'`. For this purpose *ProcessManager* considers the current instance state as well as instance-specific deviations from original schema `S`. The latter is required to detect conflicts between ad-hoc instance changes and the schema changes as captured by `Delta`.
7. *ProcessManager* migrates all compliant instances to target schema `S'`. Among other things this is accompanied by state adaptations of the instances to be migrated.
8. Where appropriate, adapted instances whose deviations conflict with process schema changes are adapted manually. This can be done using the components *ProcessEditor* and *ChangeOperations*. Again the migration is performed by *ProcessManager*.

This procedure already demonstrates that multiple system components are needed to enable process schema evolution in conjunction with other process support features.

4.2 How do architectural components of ADEPT2 support process changes?

For selected components of the ADEPT2 architecture we exemplarily show how they contribute to process flexibility in terms of schema evolution and ad-hoc changes. We revisit the described design principles and discuss their benefits in the given context.

***LogManager*:** Ad-hoc changes of single process instances as well as template changes have to be logged. The interfaces provided by the *LogManager* are generic; i.e., both kinds of changes can be logged with this component. Thus, LogManager can be reused in different context, which improves maintainability of the ADEPT2 architecture.

***ProcessRepository*:** If process schema evolution and instance migrations are supported we will have to maintain information about the different schema versions and their differences. This task is accomplished by the *ProcessRepository.*

***ProcessManager*:** This component is fundamental for the support of ad-hoc changes as well as of process schema evolution. It is therefore discussed in more detail. First, *ProcessManager* maintains the control data needed for proper and efficient execution of unchanged as well as changed process instances. Second, in the context of schema evolution, this component migrates compliant process instances to the new schema.

One major challenge is to efficiently represent template and instance objects within *ProcessManager*. Unchanged instances, for example, should be represented in a non-redundant way. The *ProcessManager* keeps one instance object for each of these unchanged instances, which captures instance-specific data (i.e., instance states) and refers to the original template schema (denoted as *template object* in the following). As example, consider instances I_1, I_3, I_4, and I_6 as depicted in Fig. 3.

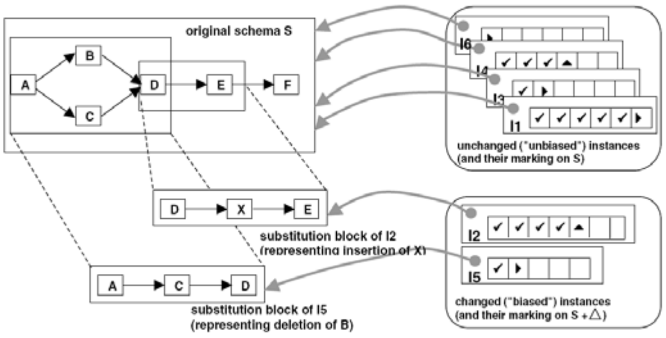
original schema S
A
B
C
D
E
F
I6
I4
I3
I1
unchanged ("unbiased") instances (and their marking on S)
D
X
E
substitution block of I2 (representing insertion of X)
I2
I5
A
C
D
substitution block of I5 (representing deletion of B)
changed ("biased") instances (and their marking on S + △)

Figure 3: Managing Template and Instance Objects in the ProcessManager (Logical View)

For representing process instances with ad-hoc changes a more sophisticated approach is needed. In ADEPT2 we have developed the *delta layer concept* [Rind06, RJR07] for this purpose. It allows to efficiently represent the difference between template and instance objects. Simply speaking, the delta layer is represented by an object with same interfaces as the process template object and therefore the same methods can be applied. However, a delta layer object does not reflect the whole process schema, but only those parts which have been adapted due to instance-specific changes. As examples consider instances I_2 and I_5 as shown in Fig. 3. Together with the template object the delta layer object allows to restore the instance-specific process schema. The instance objects which belong to changed process instances do no longer reference the associated template object but the delta layer object. The delta layer object itself references the original template object and therefore keeps the link between instance object and original template [Rind06].

The delta layer concept is also useful in the context of process schema evolution. In particular, it allows to quickly check whether instance-specific adaptations and template changes are conflicting with each other. Since *ProcessManager* supports ad-hoc changes anyway, schema evolution does not cause additional efforts when realizing this component. Note that we have decided to manage the different template versions and their *deltas* through a separate component (i.e., *ProcessRepository*). This historical information is only needed in the context of process schema evolution and should therefore not affect normal process enactment. (Here we assume that template changes constitute "exceptional cases" in comparison to normal process enactment.)

DataManager: To support instance-specific changes the *DataManager* must be able to dynamically add or delete process data elements. In this context, ADEPT2 deletes data elements and their values only logically in order to ensure traceability in all cases. Regarding schema evolution no additional functionality is required.

OrgModelManager: The support of template as well as instance changes imposes security issues as the process management system becomes more vulnerable to misuse. Therefore, the application of respective changes must be restricted to authorized users. We have developed an access control framework for (dynamic) process changes [WRW05] which can be based on the information managed by the *OrgModelManager* (see Section 3); i.e., similar to actor assignments specified in the context of process activities, we can define access control constraints for process changes (see [WRW05] for details). However, this requires no extensions of the *OrgModelManager* component. Currently, we are working on an advanced framework in order to enable evolving organizational models and the controlled adaptation of related actor assignments [RiRe07,RiRe08]. This new feature, in turn, will require extensions of *OrgModelManager;* e.g., the ability to maintain different versions of an organizational model or to adapt actor assignments on-the-fly when the underlying organizational model is changed.

ChangeOperations: As aforementioned this component allows to use the same change operations for modeling and adapting process templates as well as for defining instance-specific ad-hoc changes. As not all change operations might be needed in a given context, the set of accessible operations can be restricted. Furthermore, this component allows to add new change operations through well-defined interfaces. Finally, respective extensions do not influence the implementation of any other ADEPT2 component. This fundamental property is achieved by internally transforming high-level change operations into a basic set of stable change primitives (e.g., to add or delete nodes).

When modeling process templates structural schema changes are enabled by *ChangeOperations*. Regarding instance-specific changes, in addition, state adaptations become possible. Finally, process schema evolution requires the comparison of instance-specific changes with respective template changes conducted at the process type level. In our experience the complexity of these comparisons can be significantly reduced when using the delta layer concept as described before.

Schema evolution and instance-specific ad-hoc changes can be based on similar mechanisms. While for instance-specific adaptations the change operations and the respective state adaptations are applied in sequence, for schema evolution the structural changes and the subsequent state adaptations are applied all at once. In case of unchanged instances this only requires the re-linking of the instance objects to the new template object.

ExecutionManager: This component partially locks execution of running process instances when applying dynamic changes to them. After such change Execution-Manager re-evaluates the execution state of the modified instances in order to correctly proceed in the flow of control.

WorklistManager: *ExecutionManager* notifies *WorklistManager* when new activities become activated or running activities are completed. *WorklistManager* then updates user worklists accordingly; i.e., it adds new work items to worklists when enabling activities and removes items from them when completing activities. Basically, same functions can be used to adapt worklists when applying ad-hoc changes (e.g., for activating newly inserted activities) or when migrating process instances to an updated template.

RuntimeEnvironment: The *RuntimeEnvironment* only deals with the execution of single activities and related application components respectively. Therefore no specific functions with respect to schema evolution or instance-specific changes are needed.

ProcessEditor: To define template as well as instance-specific changes the *ProcessEditor* can be used (see Fig. 4). Among other features this component triggers the logging of the applied process changes. As aforementioned all functionality provided by *ProcessEditor* is made available via programming interfaces (APIs) as well.

Monitor: When changing an instance, which is currently displayed by the ADEPT2 monitor, the respective visualization is adapted automatically.

4.3 Proof-of-concept prototype

Except for the *ResourceManager* component, which will be added at a later stage of the project, we have implemented all components of the described architecture (cf. Fig. 2). In particular, our proof-of-concept prototype allows to demonstrate major flexibility concepts and their interplay with other process support functions. While core features of the components from the basic service layer and the execution layer have been implemented for most parts, we have not yet fully realized the intended tool components from the user interaction layer. For example, in the current release of the ADEPT2 system, ad-hoc changes of single process instances have to be accomplished using the ADEPT2 process editor. While the user interface provided by this editor is sufficient for expert users, we will have to replace it if ad-hoc changes are to be defined by end users. Therefore, more sophisticated clients are under implementation. The same applies to clients enabling user interactions that become necessary to deal with non-decidable cases in the context of instance migrations. So far, ADEPT2 is able to automatically migrate all process instances that are compliant with the new schema version according to some correctness notion (see [RRD04b] for details). Furthermore, we have implemented a web client version of the WorklistManager. Regarding the low-level services layer, we have realized all basic functionality as needed by upper-layer components. However, there are still a lot of things to be done, e.g. concerning communication between system components and persistency management [Göse07]. Fig. 4 shows a screen of the ADEPT2 process editor, which constitutes the main system component for modeling and adapting process templates. 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.

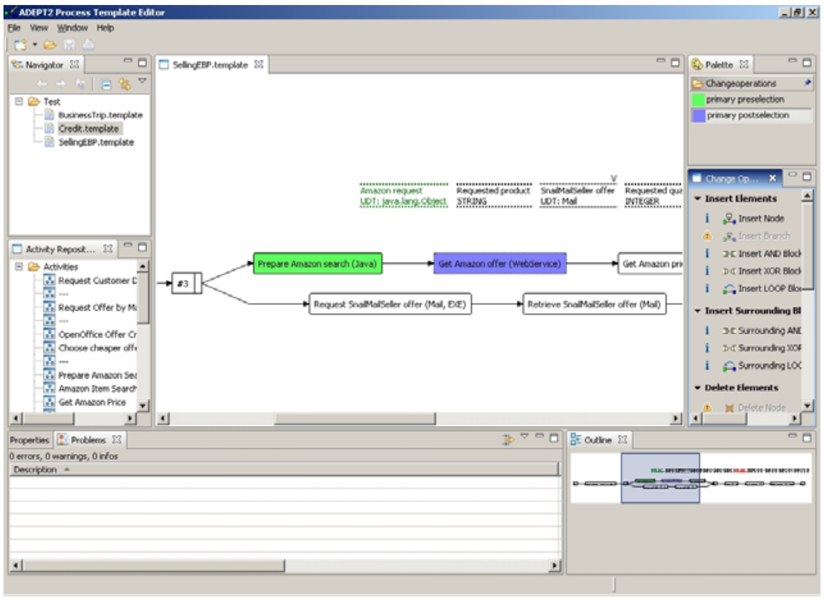

Figure 4: Screenshot of ADEPT2 Process Editor

Due to lack of space we cannot give a more detailed description of the provided functionality. We refer to [Göse07,ReDa98, RRD04b,WRR07] for a detailed overview of the concepts on which the implemented change features are grounded.

4.4 Summary

We have discussed how schema evolution and instance-specific changes have been considered in the ADEPT2 architecture. On the one hand we have shown that this architecture is able to cope with the different kinds of (dynamic) process changes. On the other hand, the given illustrations make clear that realization of schema evolution and ad-hoc changes within one system is far from being trivial. A proper system architecture with clear separation of concerns constitutes one necessary prerequisite in this context. Another one is a solid conceptual framework. When designing the ADEPT2 proof-of-concept prototype we have considered both perspectives. In future work we will evaluate the implemented concepts based on a series of lab experiments.

5 Related Work

Off-the-shelf process management systems like Staffware, FLOWer and IBM Process Server do not adequately support dynamic process changes or offer very restricted change features only [WRR07,WRR08]. Several vendors promise flexible process support, but are unable to cope with fundamental needs 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 "enrich" applications with respective process support functions to cope with these limitations. This aggravates PAIS development and PAIS maintenance significantly.

The need for flexible and easily adaptable PAIS has been recognized and several competing paradigms for addressing process changes and process flexibility have been developed. Examples include adaptive processes [MGR04,MSK07,ReDa98,RRD04a+b, Wesk00], case handling [AWG05,MWR08], data-driven processes [MRH08], declarative workflows [Pesi07,SSO05], process refactoring [WeRe08] and late modeling [Adam06]. However, there is still a lack of implementations of respective technologies offering sufficient support to be applied for experimental use. Furthermore, only little work has been done with respect to the architectural design of respective systems considering requirements like extensibility, scalability, flexibility and maintainability.

Like ADEPT2, CAKE2 [MSK07] and WASA2 [Wesk00] allow for structural runtime adaptations at the process instance level. Both approaches only support change primitives (i.e., adding / removing nodes and edges respectively), while ADEPT2 provides support for a wide range of high-level change operations [WRR07]. ADEPT2 is the only system which provides common support for both process schema evolution and ad-hoc changes [WRR07,RRD04a]. Worklets [Adam06] allow for the late binding of sub-processes following a rule-based approach. Except for the dynamic replacement of activities no support for ad-hoc changes is provided. Similar considerations can be made for the case handling tool FLOWer [AWG05.MWR08], which allows to delete activities, but does not support other kinds of ad-hoc changes. Neither Worklets nor FLOWer have considered issues related to process schema evolution. Finally, among all these approaches ADEPT2 scores best in respect to high-level change operations [WRR07,WRR08].

6 Summary

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 other approaches 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 is one of the very few systems which has tried to consider both conceptual and architectural considerations in the design of a next generation process management system.

References

[Adam06] Adams, M.; ter Hofstede, A.H.M.; Edmond, D.; van der Aalst,W.: A service-oriented implementation of dynamic flexibility in workflows. Proc. Coopis'06 (2006)

[AWG05] van der Aalst, W.; Weske, M.; Grünbauer, D.: Case handling: a new paradigm for business process support, Data and Knowledge Engineering. 53 (2) (2005) 129-162.

[BDR03] Bauer, T.; Reichert, M.; Dadam, P.: Intra-subnet load balancing in distributed workflow management systems. Int'l Journal of Cooperative Information Systems , 12(3):205-323, 2003

[DRK00] Dadam, P.; Reichert, M.; Kuhn, K.: Clinical workflows – the killer application for process-oriented information systems? Proc. 4th Int'l Conf. on Business Information Systems (BIS'2000), Poznan, Poland, April 2000, pp. 36-59.

[Göse07] Göser, K. et al.: Next-generation process management with ADEPT2. Proc. of the BPM'07 Demonstration Program, Brisbane, Australia, September 2007, pp. 3-6.

[Guen06] Guenther, C.W.; Rinderle, S.; Reichert, M.; van der Aalst, W.M.P. (2006) Change mining in adaptive process management systems. In: Proc. 14th Int'l Conf. on Coop. Information Systems (CoopIs'06), 2006, Montpellier. LNCS 4275, pp. 309-326

[LeRe07] Lenz, R.; Reichert, M.: IT support for healthcare processes - premises, challenges, perspectives, Data and Knowledge Engineering, 61(1):39-58, 2007.

[LRW08] Li, C.; Reichert, M.; Wombacher, A.: Discovering reference process models by mining process variants. In: Proc. 6th Int'l Conf. on Web Services (ICWS'08), September 2008, Beijing, China. IEEE Computer Society Press

[MGR04] Müller, R.; Greiner, U.; Rahm, E.: AgentWork: A workflow system supporting rule-based workflow adaptation, Data and Knowledge Engineering 51 (2) (2004) 223-256.

[MSK07] Minor, M.; Schmalen, D.; Koldeho, A. Structural adaptation of workflows supported by a suspension mechanism and by case-based reasoning. Proc. WETICE'07, 2007.

[MRB08] Mutschler, B.; Reichert, M.; Bumiller, J.: 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, 2008.

[MRH08] Müller, D.; Reichert, M.; Herbst, J.: A new paradigm for the enactment and dynamic adaptation of data-driven process structures. Proc. 20th Int'l Conf. on Advanced Information Systems Engineering (CAiSE'08), Montpellier, LNCS 5074, pp. 48-63

[Müll06] Müller, D.; Herbst, J.; Hammori, M.; Reichert, M.: IT Support for release management processes in the automotive industry. In: Proc. 4th Int'l Conf. on Business Process Management (BPM'06), Vienna, Austria. LNCS 4102, pp. 368-377

[MWR08] Mutschler, B.; Weber, B.; Reichert, M..: Workflow management versus case handling: results from a controlled software experiment. Proc. 23rd Annual ACM Symposium on Applied Computing (SAC'08), Fortaleza, Ceará, Brazil, March 2008, pp. 82-89

[Pesi07] Pesic, M.; Schonenberg, M.; Sidorova, N.; van der Aalst, W.M.P.: Constraint-based workflow models: change made easy, Proc. CoopIS'07 Conf., 2007.

[RDB03] Reichert, M.; Dadam, P.; Bauer, T.: Dealing with forward and backward jumps in workflow management systems. Int'l Journal Software and Systems Modeling (SOSYM), 2(1):37-58, 2003

[ReDa98] Reichert, M.; Dadam, P.: ADEPTflex – Supporting dynamic changes of workflows without losing control. Journal of Intelligent Information Systems, 10(2):93-129, 1998

[Rind06] Rinderle, S.; Reichert, M; Jurisch, M.; Kreher, U.: On representing, purging and utilizing change logs in process management systems. Proc. 4th Int'l Conf. Business Process Management (BPM'06), Vienna, LNCS 4102, September 2006, pp. 241-256

[RiRe07] Rinderle, S.; Reichert, M.: A formal framework for adaptive access control models. Journal on Data Semantics IX, LNCS 4601, Springer 2007, pp. 82-112.

[RiRe08] Rinderle-Ma, S.; Reichert, M.: Managing the Life Cycle of Access Rules in CEOSIS. Proc. 12th IEEE Int. Enterprise Computing Conference (EDOC'08), September 2008, Munich, Germany.

[RJR07] Rinderle, S.; Jurisch, M.; Reichert, M.: On Deriving Net Change Information From Change Logs – The DELTALAYER-Algorithm. Proc. BTW'07, 2007, pp. 364-381

[RRD03a] Reichert, M.; Rinderle, S.; Dadam, P.: 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

[RRD03b] Reichert, M.; Rinderle, S.; Dadam, P.: ADEPT Workflow Management System Flexible Support for Enterprise-Wide Business Processes. Proc. 1st Int'l Conf. on Business Process Management (BPM '03), Eindhoven, LNCS 2678, pp. 371-379

[RRD04a] Rinderle, S.; Reichert, M.; Dadam, P.: Correctness criteria for dynamic changes in workflow systems - a survey. Data and Knowledge Engineering, 50(1):9-34 (2004)

[RRD04b] Rinderle, S.; Reichert, M.; Dadam, P.: Flexible support of team processes by adaptive workflow systems. Distributed and Parallel Databases, 16(1):91-116, 2004

[RRD04c] Rinderle, S.; Reichert, M.; Dadam, P.: On Dealing with structural conflicts between process type and instance changes. In: Proc. 2nd Int'l Conf. Business Process Management (BPM'04), Potsdam, Germany. LNCS 3080, 2004, pp. 274-289

[RRD04d] Rinderle, S., Reichert, M.; Dadam, P.: Disjoint and overlapping process changes: challenges, solutions, applications. In: Proc. 11th Int'l Conf. on Cooperative Information Systems (CooplS'04), Agia Napa, Cyprus. LNCS 3290, 2004, pp. 101-121

[SSO05] Sadiq, S.; Sadiq, W.; Orlowska, M.: A framework for constraint specification and validation in flexible workflows. Information Systems 30, 349-378, 2005

[Wesk07] Weske, M.: Business Process Management, Springer, 2007.

[Wesk00] Weske, M.: Workflow Management Systems: Formal foundation, conceptual design, implementation aspects. Habilitationsschrift, University of Münster, 2000

[WRW05] Weber, B. ; Reichert, M. ; Wild, W. ; Rinderle, S.: Balancing flexibility and security in adaptive process management systems. Proc. CoopIS'05, LNCS 3760, pp. 59-76

[WRR07] Weber, B.; Rinderle, S.; Reichert, M.: Change patterns and change support features in process-aware information systems. Proc. 19th Int'l Conf. on Advanced Inform. Sys. Engineering (CAiSE'07), LNCS 4495, Trondheim, June 2007, pp. 574-588

[WRR08] Weber, B.; Reichert, M.; Rinderle-Ma, S. (2008) Change patterns and change support features - enhancing flexibility in pocess-aware information systems. Data and Knowledge Engineering, 66(3):438-466

[WeRe08] Weber, B.; Reichert, M.: Refactoring process models in large process repositories. Proc. 20th Int'l Conf. on Advanced Information Systems Engineering (CAiSE'08), June 2008, Montpellier, France, LNCS 5074, pp. 124-139.