Architectural Design of Flexible Process Management Technology

Manfred Reichert, Peter Dadam, Martin Jurisch,
Ulrich Kreher, Kevin Göser, Markus Lauer

Ulmer Informatik-Berichte

Nr. 2008-02
Februar 2008

Architectural Design of Flexible Process Management Technology

Manfred Reichert[1,2], Peter Dadam[1], Martin Jurisch[1],
Ulrich Kreher[1], Kevin Göser[1], Markus Lauer[1]

[1]Institute for Databases and Information Systems, Ulm University, Germany
{peter.dadam, martin.jurisch, ulrich.kreher, kevin.goeser, markus.lauer}@uni-ulm.de
[2]Information Systems Group, University of Twente, The Netherlands
m.u.reichert@utwente.de

Abstract: To provide effective support, process-aware information systems (PAIS) must not freeze existing business processes. Instead they should enable authorized users to deviate on-the-fly from the implemented processes 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 PAIS implementing this dynamics. Designing the architecture of respective technology constitutes a big challenge due to the high complexity coming with dynamic processes. Besides this, performance, robustness, security and usability of the system must not be affected by the added flexibility. In the AristaFlow project we have taken a holistic approach to master this complexity. Based on a conceptual framework for flexible process enactment and dynamic processes, 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 effective manner.

1 Introduction

In today's dynamic business world the economic success of an enterprise depends on its ability to react to changes in its environment in a quick and flexible way [LeRe07,RRD04a,WRR07]. Causes for these changes can be manifold and include the introduction of new laws or changes in customers' attitudes. For these reasons, companies have recognized business agility as a competitive advantage, which is fundamental for being able to cope with business 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 (IS) in a process-oriented way can be observed [Wesk07]. In contrast to data- or function-centered IS, 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 reducing cost of change. Changes to one layer often can be performed without affecting other layers; e.g., modifying the application which implements a particular process activity does usually not imply any change to the process layer as long as interfaces remain stable. In addition, changing the execution order of process activities or adding new activities to the process can, to a large degree, be accomplished without touching any of the application services.

The ability to efficiently deal with process change has been identified as one of the most critical success factors for PAIS [ReDa98,RRD04,WRR07]. Through the described separation of concerns PAIS facilitate changes significantly. However, enterprises are still reluctant to change process implementations once they are running properly. High complexity and cost of change are mentioned as major obstacles for not fully leveraging the potential of PAIS. To overcome 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 must be able to deviate from the predefined processes as 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 [RRD04a].

Basically, process changes can take place at the type as well as the instance level: Changes of single process instances, for example, often have to be carried out in an ad-hoc manner in order to deal with exceptional situations [ReDa98,DRK00]. Such *ad-hoc changes* must not affect system robustness or lead to errors in the sequel. Process type changes, in turn, are continuously applied in order to adapt the PAIS to evolving business processes [RRD04b]. This *process schema evolution* might require the migration of already running process instances to the new schema as well. Important challenges in this context are to perform respective migrations on-the-fly, to guarantee their compliance with the new process schema, and to avoid performance penalties.

The design of a process management technology which enables efficient process execution as well as application integration, but also provides support for the different kinds of changes, constitutes a big challenge. First, many trade-offs exist which have to be balanced. For example, complexity of dynamic changes increases with higher expressiveness of the used process modeling formalism. Second, complex interdependencies between the different features of such a system exist that must be carefully understood in order to avoid implementation gaps. Process schema evolution, for example, requires high-level change operations, versioning support, logging, on-the-fly migration of running process instances, worklist adaptations, etc. Third, even if the conceptual pillars of such a technology are well understood, it still is a quantum leap to implement respective features in an efficient, robust and integrated manner.

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

Sect. 2 presents background information needed for the understanding of this paper. Sect. 3 summarizes architectural design goals and gives an overview of ADEPT2 components. Sect. 4 shows how ad-hoc changes and schema evolution are supported within this architecture. Sect. 5 discusses related work and Sect. 6 concludes with a summary.

2 Conceptual Framework for Dynamic Changes in ADEPT2

In the ADEPT project we have developed a conceptual framework for dynamic process changes [ReDa98,RRD04b]. We use this framework as conceptual pillar for the design of the ADEPT2 architecture as well. Basically, this change framework covers process changes at both the process instance and the process type level. While the former provide *ad-hoc flexibility* to users (e.g., to deal with exceptional situations), the latter are needed to evolve process implementations over time (*process schema evolution*).

Ad-hoc flexibility. At the process instance level the ADEPT2 framework enables different kinds of ad-hoc deviations from the pre-modeled process template (e.g., to insert, delete, or move activities). Such ad-hoc changes never lead to unstable system behavior, i.e., none of the guarantees achieved by model checks at buildtime can be violated due to the dynamic change [ReDa98]. ADEPT2 offers a complete set of change operations for defining ad-hoc deviations at a high level of abstraction; e.g., authorized users may dynamically add new activities or jump forward in the flow of control. ADEPT2 ensures correctness based on formal pre-/post-conditions of these operations. All complexity associated with the adaptation of instance states, the re-mapping of input/output parameters of the affected application components, the problem of missing input data due to activity deletions, or the problem of deadlocks is hidden from users.

Process schema evolution. To cope with business process changes, the ADEPT2 change framework allows for quick and efficient adaptations of process templates (i.e., the schema of a process type) – in the following denoted as *process schema evolution* [RRD04b]. When updating a template, usually, related process instances are finished according to the old template version, while future instances are derived from the new one. However, such rigid approach is not adequate for long-running processes. The challenge is to *propagate* respective schema changes to the instances of this template as well; i.e., to migrate the instances to the new schema version of the 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 or not [RRD04a,RRD04b]. The ADEPT2 change framework uses a well-defined correctness criterion in this context, 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.

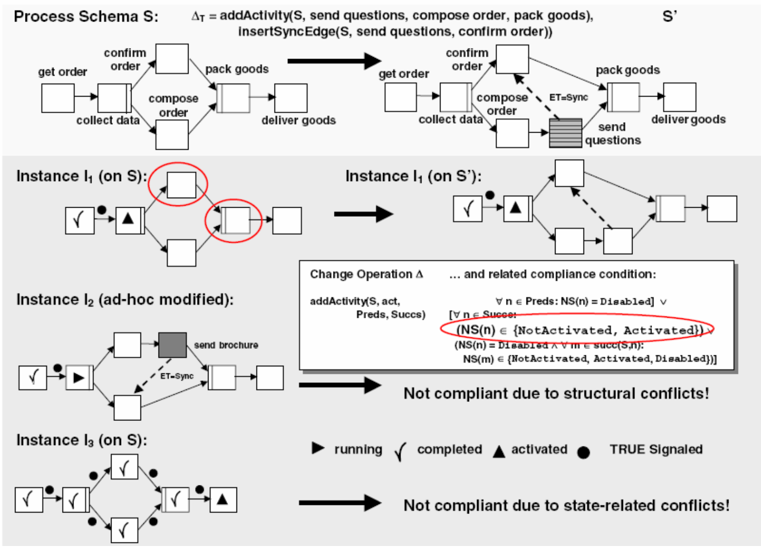

Figure 1: Process Schema Evolution (Conceptual View)

When designing ADEPT2 we have tried to look 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 interdependencies as well. For example, the correct handling of concurrent process changes is crucial in order to cover all practical cases. In this context, we have also investigated 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, the current instance schema differs from the original template. Therefore, change propagation must be accomplished under appropriate correctness constraints to avoid inconsistencies. Again, ADEPT2 applies a comprehensive correctness principle in this context, which excludes state-related, structural, and semantical conflicts between concurrent changes.

As example consider Fig. 1 where a new template version S' is created from a process template S 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. 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. The next two sections illustrate how we realize this conceptual framework within the ADEPT2 architecture.

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. The considered design principles consider 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 at the same time. This section summarizes major design principles, and gives an overview of the developed ADEPT2 architecture.

3.1 Major Design Principles

3.1.1 General Design Principles

High-end process management technology has a complexity comparable to database systems. To master this complexity a proper and modular system 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 the system components to a large extent.

The overall system architecture must 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 system components must be reusable in different context using available configuration facilities.

Process management systems must provide sophisticated buildtime and runtime components to the different user groups. This includes tools for modeling, verifying and testing processes, components for monitoring and dynamically adapting process instances, or worklist 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 should be configurable in a flexible way. On the other hand, all functions offered by the process management system should be made available via programming interfaces (APIs) as well. In particular, advanced system functions (e.g., ad-hoc changes or process schema evolution) must be accessible via API.

Implementation and maintenance of the different system components shall be as easy as possible. Therefore each component should be kept as simple as possible and only have access to the information needed for its proper functioning. Furthermore, communication details have to be hidden from component developers and independency from the used middleware components (e.g., database management systems) shall be realized.

3.1.2 Conceptual Design Goals

To provide improve maintainability, extensibility, and usability of the different system components, the conceptual design of the ADEPT2 architecture is done carefully. Due to lack of space we do not give a complete overview of all considered design issues, but illustrate our main design philosophy by means of two examples:

- *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 can be made for other components (e.g., visualization, logging, versioning, or access control). This design principle does not only reduce code redundancies, but – as a consequence – results 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 such 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). This change, however, must not affect the components realizing process schema evolution or ad-hoc flexibility. In ADEPT2, for example, we achieve this by mapping high-level change operations internally to a stable set of low-level change primitives (e.g., to add/delecte nodes).

3.2 Overview of the ADEPT2 Architecture and its Components

Figure 2 depicts the overall architecture of the ADEPT2 process management system:

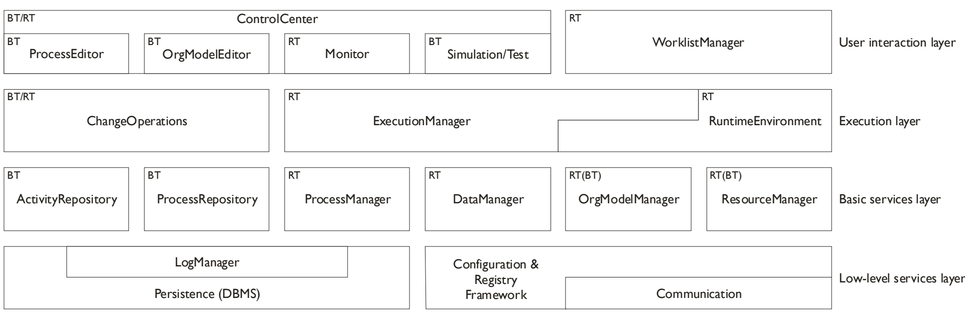

Figure 2: Basic Architecture of ADEPT2 (BT: Builtime; RT: Runtime)

Development of this architecture has been based on the design principles discussed in the previous section and on the experiences we gathered in a previous project [ReDa98]. 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 the user, including a process editor and a monitoring component.

3.2.1 Layer with Low-level Services

This first layer comprises basic services which accomplish tasks like logging, persistency, configuration support, and communication. In particular, idiosyncrasies of the used communication services or storage management component are hidden from upper-layer components. This allows us to use different database 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 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 system components. Thereby, communication idiosyncrasies (e.g., concerning the used transport protocols, interaction styles or message formats) are hidden from the components using this interface. For example, it remains transparent for them whether the services they request are running locally or remotely.

LogManager: ADEPT2 allows to log all relevant system events occurring at build- and runtime. This includes events like changes in the state of a process instance, changes of a process template or process instance, or access to process data elements. The *LogManager* provides a generic interface based on which upper-layer components can log the 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 components (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 executing the respective activity. In particular, it connects the activity to an application component. In this context, details of the used component model (e.g., Web services, Enterprise Java Beans, or (D)COM) 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. Note that this allows for the realization of sub processes in an easy and intuitive manner. Further, *ProcessRepository* manages all versions of a process template and the information needed to derive them from each other (e.g. change logs).

***ProcessManager*:** While the above components manage buildtime data, the *Process-Manager* provides exactly those information needed for process enactment during runtime. This includes, for example, schemes of active process templates and in-progress process instances as well as current instance states. In particular, *ProcessManager* restores the specific schemes of instances to which ad-hoc changes were applied.

As opposed to *ProcessRepository*, the *ProcessManager* has no knowledge about the evolution of process or activity templates; i.e., it does not know about the different template versions and their relations. This minimalism allows for efficient process enactment. As we discuss in Sect. 4, *ProcessManager* also deals with the migration of (compliant) 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 and 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 also accessible by external components, 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*:** To define potential actors for an activity, this activity can be associated with an actor assignment. Such an assignment refers to organizational entities (e.g., units, project teams, roles, actors) or organizational relations (e.g., “is-manager-of”) as captured in an organizational model. The *OrgModelManager* maintains this organizational model and corresponding actors. It further accepts an actor assignment as input and delivers all actors qualifying for the respective expression as result.

***ResourceManager*:** Besides actors, additional resources are usually required during process execution. Examples include rooms, machines, and software licenses. ADEPT2 allows to model respective resources and considers this information during runtime as well (e.g., for determining bottlenecks in advance).

3.2.3 Execution Layer

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

ChangeOperations: This component comprises the change operations that can be applied to processes in different context (e.g., to add, delete or move activities). First, change operations are required when modeling new process templates or adapting existing ones. In the latter case the respective schema changes can be propagated to already running process instances as well (cf. Sect. 2); i.e., we (logically) apply the operations at the instance level. The same applies with respect to ad-hoc instance changes (cf. Sect. 2). Note that in all these cases same or similar change operations are needed. Our basic design principles (cf. Sect. 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 after its change. Note that these pre-/post-conditions are varying depending on whether the operation is applied at the type or instance level.

ExecutionManager: This component coordinates the execution of process instances in an efficient and correct way. For example, it evaluates predicates on instance data to choose between alternative branches or to loop back during runtime. As a prerequisite the *ExecutionManager* needs information about the current schema as well as the state of respective instances. This information is provided by the *ProcessManager*, i.e., a lower-layer component. For the *ExecutionManager* it remains transparent whether a process instance is still running on its original schema or on a modified schema (due to 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 as well as to control their execution (e.g., to abort or suspend component execution). Finally, the *Runtime-Environment* 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 in this context are made available via APIs as well.

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

3.3 Summary

All described 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.

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 deeper insights into the realization of our dynamic 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 General procedure of a process schema evolution

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

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 operator set can be applied as when modeling 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*.

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 the *ProcessManager* checks whether it is compliant with target schema `S'` or not. For this purpose the *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 changes and the ones captured by `Delta`.
7. The *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 the *ProcessManager*.

This (simplified) procedure already demonstrates that multiple system components are needed to enable a feature like process schema evolution.

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 change. 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 the LogManager can be reused in different context, which improves maintainability of the ADEPT2 architecture.

***ProcessRepository*:** If process schema evolution and related instance migrations have to be supported we must 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 process schema evolution, and is therefore discussed in more detail. First, the *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 the *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.

For handling instances with ad-hoc changes a more sophisticated approach is needed. In ADEPT2 we have developed the *delta layer concept* (see also [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 the *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., the *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.)

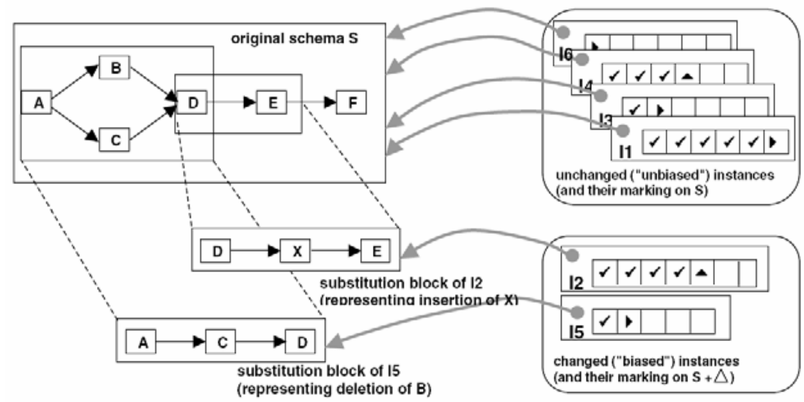

Figure 3: Managing Template and Instance Objects in the ProcessManager (Logical View)

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 from the process in order to ensure traceability. 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 for also evolving organizational models and related actor assignments in a controlled way [RiRe07]. This new feature, in turn, will require extensions of the *OrgModelManager* (e.g., the ability to maintain different versions of an organizational model or to adapt actor assignments semi-automatically when the underlying organizational model is changed).

ChangeOperations: As mentioned this component allows to use the same change operations for modeling and adapting process templates as well as for defining instance-specific changes. As not all change operations might be needed in a given context, the set of accessible operations can be restricted. Further, 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 fundemantal property is achieved by internally transforming high-level change operations into a set of stable, basic change primitives (e.g., add/delecte node or edge).

When modeling process templates structural schema changes are enable 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. Complexity of these comparisons has been be significantly reduced using the delta layer concept (see above).

Schema evolution and instance-specific 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 a change the ExecutionManager re-evaluates the execution state of the modified instances in order to correctly proceed in the flow of control.

WorklistManager: The *ExecutionManager* notifies the *WorklistManager* when new activities become activated or running activities are completed. The *WorklistManager* then updates the user worklists accordingly; i.e., it adds new work items to worklists when enabling activities and removes items from worklists when completing activities. Basically, the same functions can be used to adapt user worklists when applying ad-hoc changes 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 change.

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

We have implemented selected components of the described architecture in a proof-of-concept prototype in order to demonstrate major flexibility concepts and their interplay. A detailed description can be found in [Göse07]. For example, Fig. 4 shows a screen of the ADEPT2 process editor, which constitutes the main system component for modeling and adapating 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.

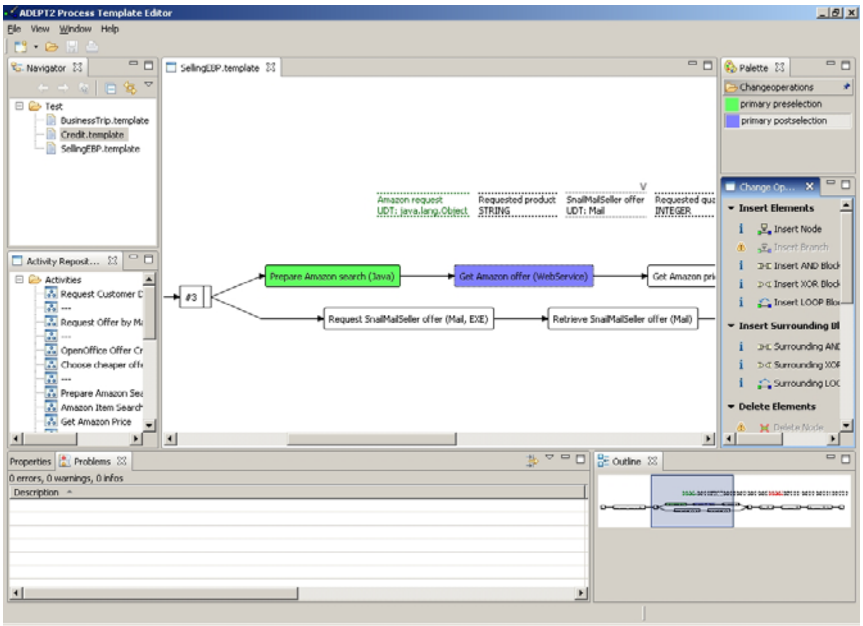

Figure 4: Screenshot of ADEPT2 Process Editor

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 the 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 is 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.

5 Related Work

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 (see [WRR07] for an overview). Examples include adaptive process management [MGR04,MSK07,ReDa98,RRD04a+b,Wesk00], case handling [AWG05,MWR08], declarative workflows [Pesi07,SSO05], and late binding/modeling [Adam06]. However, there is still a lack of comprehensive 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, adaptivity and maintainability.

Like ADEPT2, CAKE2 [MSK07] and WASA2 [Wesk00] allow for structural run-time 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 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].

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., v.d.Aalst,W.M.: 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.

[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 Programm, Brisbane, Australia, September 2007, pp. 3-6.

[LeRe07] Lenz, R.; Reichert, M.: IT Support for Healthcare Processes - Premises, Challenges, Perspectives, Data and Knowledge Engineering (1) (2007) 39{58.

[MGR04] Müller; Greiner, U.; E. Rahm, AgentWork: A workow system supporting rule-based workow adaptation., DKE 51 (2) (2004) 223{256.

[MSK07] Minor, M.; Schmalen, D.; Koldeho, A. workflow supported by a suspension. Proc. WETICE'07, 2007.

[MWR08] Mutschler, B.; Weber, B.; Reichert, M..: Workflow Management versus Case Handling: Results from a Controlled Software Experiment. Proc. SAC'08 (to appear)

[Pesi07] Pesic, M.; Schonenberg, M.;Sidorova, N.; van der Aalst, W.M.P.: Constraint-Based Workow Models: Change Made Easy., Proc. CoopIS'07, 2007.

[ReDa98] Reichert, M.; Dadam, P.: ADEPTflex – Supporting Dynamic Changes of Workflows Without Losing Control. J 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.

[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

[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)

[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

Liste der bisher erschienenen Ulmer Informatik-Berichte
Einige davon sind per FTP von `ftp.informatik.uni-ulm.de` erhältlich
Die mit * markierten Berichte sind vergriffen

List of technical reports published by the University of Ulm
Some of them are available by FTP from `ftp.informatik.uni-ulm.de`
Reports marked with * are out of print

91-01 *Ker-I Ko, P. Orponen, U. Schöning, O. Watanabe*
Instance Complexity

91-02* *K. Gladitz, H. Fassbender, H. Vogler*
Compiler-Based Implementation of Syntax-Directed Functional Programming

91-03* *Alfons Geser*
Relative Termination

91-04* *J. Köbler, U. Schöning, J. Toran*
Graph Isomorphism is low for PP

91-05 *Johannes Köbler, Thomas Thierauf*
Complexity Restricted Advice Functions

91-06* *Uwe Schöning*
Recent Highlights in Structural Complexity Theory

91-07* *F. Green, J. Köbler, J. Toran*
The Power of Middle Bit

91-08* *V.Arvind, Y. Han, L. Hamachandra, J. Köbler, A. Lozano, M. Mundhenk, A. Ogiwara, U. Schöning, R. Silvestri, T. Thierauf*
Reductions for Sets of Low Information Content

92-01* *Vikraman Arvind, Johannes Köbler, Martin Mundhenk*
On Bounded Truth-Table and Conjunctive Reductions to Sparse and Tally Sets

92-02* *Thomas Noll, Heiko Vogler*
Top-down Parsing with Simulataneous Evaluation of Noncircular Attribute Grammars

92-03 *Fakultät für Informatik*
17. Workshop über Komplexitätstheorie, effiziente Algorithmen und Datenstrukturen

92-04* *V. Arvind, J. Köbler, M. Mundhenk*
Lowness and the Complexity of Sparse and Tally Descriptions

92-05* *Johannes Köbler*
Locating P/poly Optimally in the Extended Low Hierarchy

92-06* *Armin Kühnemann, Heiko Vogler*
Synthesized and inherited functions -a new computational model for syntax-directed semantics

92-07* *Heinz Fassbender, Heiko Vogler*
A Universal Unification Algorithm Based on Unification-Driven Leftmost Outermost Narrowing

92-08* *Uwe Schöning*
On Random Reductions from Sparse Sets to Tally Sets

92-09* *Hermann von Hasseln, Laura Martignon*
Consistency in Stochastic Network

92-10 *Michael Schmitt*
A Slightly Improved Upper Bound on the Size of Weights Sufficient to Represent Any Linearly Separable Boolean Function

92-11 *Johannes Köbler, Seinosuke Toda*
On the Power of Generalized MOD-Classes

92-12 *V. Arvind, J. Köbler, M. Mundhenk*
Reliable Reductions, High Sets and Low Sets

92-13 *Alfons Geser*
On a monotonic semantic path ordering

92-14* *Joost Engelfriet, Heiko Vogler*
The Translation Power of Top-Down Tree-To-Graph Transducers

93-01 *Alfred Lupper, Konrad Froitzheim*
AppleTalk Link Access Protocol basierend auf dem Abstract Personal Communications Manager

93-02 *M.H. Scholl, C. Laasch, C. Rich, H.-J. Schek, M. Tresch*
The COCOON Object Model

93-03 *Thomas Thierauf, Seinosuke Toda, Osamu Watanabe*
On Sets Bounded Truth-Table Reducible to P-selective Sets

93-04 *Jin-Yi Cai, Frederic Green, Thomas Thierauf*
On the Correlation of Symmetric Functions

93-05 *K.Kuhn, M.Reichert, M. Nathe, T. Beuter, C. Heinlein, P. Dadam*
A Conceptual Approach to an Open Hospital Information System

93-06 *Klaus Gaßner*
Rechnerunterstützung für die konzeptuelle Modellierung

93-07 *Ullrich Keßler, Peter Dadam*
Towards Customizable, Flexible Storage Structures for Complex Objects

94-01 *Michael Schmitt*
On the Complexity of Consistency Problems for Neurons with Binary Weights

94-02 *Armin Kühnemann, Heiko Vogler*
A Pumping Lemma for Output Languages of Attributed Tree Transducers

94-03 *Harry Buhrman, Jim Kadin, Thomas Thierauf*
On Functions Computable with Nonadaptive Queries to NP

94-04 *Heinz Faßbender, Heiko Vogler, Andrea Wedel*
Implementation of a Deterministic Partial E-Unification Algorithm for Macro Tree Transducers

94-05 *V. Arvind, J. Köbler, R. Schuler*
On Helping and Interactive Proof Systems

94-06 *Christian Kalus, Peter Dadam*
Incorporating record subtyping into a relational data model

94-07 *Markus Tresch, Marc H. Scholl*
A Classification of Multi-Database Languages

94-08 *Friedrich von Henke, Harald Rueß*
Arbeitstreffen Typtheorie: Zusammenfassung der Beiträge

94-09 *F.W. von Henke, A. Dold, H. Rueß, D. Schwier, M. Strecker*
Construction and Deduction Methods for the Formal Development of Software

94-10 *Axel Dold*
Formalisierung schematischer Algorithmen

94-11 *Johannes Köbler, Osamu Watanabe*
New Collapse Consequences of NP Having Small Circuits

94-12 *Rainer Schuler*
On Average Polynomial Time

94-13 *Rainer Schuler, Osamu Watanabe*
Towards Average-Case Complexity Analysis of NP Optimization Problems

94-14 *Wolfram Schulte, Ton Vullinghs*
Linking Reactive Software to the X-Window System

94-15 *Alfred Lupper*
Namensverwaltung und Adressierung in Distributed Shared Memory-Systemen

94-16 *Robert Regn*
Verteilte Unix-Betriebssysteme

94-17 *Helmuth Partsch*
Again on Recognition and Parsing of Context-Free Grammars:
Two Exercises in Transformational Programming

94-18 *Helmuth Partsch*
Transformational Development of Data-Parallel Algorithms: an Example

95-01 *Oleg Verbitsky*
On the Largest Common Subgraph Problem

95-02 *Uwe Schöning*
Complexity of Presburger Arithmetic with Fixed Quantifier Dimension

95-03 *Harry Buhrman,Thomas Thierauf*
The Complexity of Generating and Checking Proofs of Membership

95-04 *Rainer Schuler, Tomoyuki Yamakami*
Structural Average Case Complexity

95-05 *Klaus Achatz, Wolfram Schulte*
Architecture Indepentent Massive Parallelization of Divide-And-Conquer Algorithms

95-06 *Christoph Karg, Rainer Schuler*
Structure in Average Case Complexity

95-07 *P. Dadam, K. Kuhn, M. Reichert, T. Beuter, M. Nathe*
ADEPT: Ein integrierender Ansatz zur Entwicklung flexibler, zuverlässiger kooperierender Assistenzsysteme in klinischen Anwendungsumgebungen

95-08 *Jürgen Kehrer, Peter Schulthess*
Aufbereitung von gescannten Röntgenbildern zur filmlosen Diagnostik

95-09 *Hans-Jörg Burtschick, Wolfgang Lindner*
On Sets Turing Reducible to P-Selective Sets

95-10 *Boris Hartmann*
Berücksichtigung lokaler Randbedingung bei globaler Zieloptimierung mit neuronalen Netzen am Beispiel Truck Backer-Upper

95-12 *Klaus Achatz, Wolfram Schulte*
Massive Parallelization of Divide-and-Conquer Algorithms over Powerlists

95-13 *Andrea Mößle, Heiko Vogler*
Efficient Call-by-value Evaluation Strategy of Primitive Recursive Program Schemes

95-14 *Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
A Generic Specification for Verifying Peephole Optimizations

96-01 *Ercüment Canver, Jan-Tecker Gayen, Adam Moik*
Formale Entwicklung der Steuerungssoftware für eine elektrisch ortsbediente Weiche mit VSE

96-02 *Bernhard Nebel*
Solving Hard Qualitative Temporal Reasoning Problems: Evaluating the Efficiency of Using the ORD-Horn Class

96-03 *Ton Vullinghs, Wolfram Schulte, Thilo Schwinn*
An Introduction to TkGofer

96-04 *Thomas Beuter, Peter Dadam*
Anwendungsspezifische Anforderungen an Workflow-Mangement-Systeme am Beispiel der Domäne Concurrent-Engineering

96-05 *Gerhard Schellhorn, Wolfgang Ahrendt*
Verification of a Prolog Compiler - First Steps with KIV

96-06 *Manindra Agrawal, Thomas Thierauf*
Satisfiability Problems

96-07 *Vikraman Arvind, Jacobo Torán*
A nonadaptive NC Checker for Permutation Group Intersection

96-08 *David Cyrluk, Oliver Möller, Harald Rueß*
An Efficient Decision Procedure for a Theory of Fix-Sized Bitvectors with Composition and Extraction

96-09 *Bernd Biechele, Dietmar Ernst, Frank Houdek, Joachim Schmid, Wolfram Schulte*
Erfahrungen bei der Modellierung eingebetteter Systeme mit verschiedenen SA/RT–Ansätzen

96-10 *Falk Bartels, Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
Formalizing Fixed-Point Theory in PVS

96-11 *Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
Mechanized Semantics of Simple Imperative Programming Constructs

96-12 *Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
Generic Compilation Schemes for Simple Programming Constructs

96-13 *Klaus Achatz, Helmuth Partsch*
From Descriptive Specifications to Operational ones: A Powerful Transformation Rule, its Applications and Variants

97-01 *Jochen Messner*
Pattern Matching in Trace Monoids

97-02 *Wolfgang Lindner, Rainer Schuler*
A Small Span Theorem within P

97-03 *Thomas Bauer, Peter Dadam*
A Distributed Execution Environment for Large-Scale Workflow Management Systems with Subnets and Server Migration

97-04 *Christian Heinlein, Peter Dadam*
Interaction Expressions - A Powerful Formalism for Describing Inter-Workflow Dependencies

97-05 *Vikraman Arvind, Johannes Köbler*
On Pseudorandomness and Resource-Bounded Measure

97-06 *Gerhard Partsch*
Punkt-zu-Punkt- und Mehrpunkt-basierende LAN-Integrationsstrategien für den digitalen Mobilfunkstandard DECT

97-07 *Manfred Reichert, Peter Dadam*
$ADEPT_{flex}$ - Supporting Dynamic Changes of Workflows Without Loosing Control

97-08 *Hans Braxmeier, Dietmar Ernst, Andrea Mößle, Heiko Vogler*
The Project NoName - A functional programming language with its development environment

97-09 *Christian Heinlein*
Grundlagen von Interaktionsausdrücken

97-10 *Christian Heinlein*
Graphische Repräsentation von Interaktionsausdrücken

97-11 *Christian Heinlein*
Sprachtheoretische Semantik von Interaktionsausdrücken

97-12 *Gerhard Schellhorn, Wolfgang Reif*
Proving Properties of Finite Enumerations: A Problem Set for Automated Theorem Provers

97-13 *Dietmar Ernst, Frank Houdek, Wolfram Schulte, Thilo Schwinn*
Experimenteller Vergleich statischer und dynamischer Softwareprüfung für eingebettete Systeme

97-14 *Wolfgang Reif, Gerhard Schellhorn*
Theorem Proving in Large Theories

97-15 *Thomas Wennekers*
Asymptotik rekurrenter neuronaler Netze mit zufälligen Kopplungen

97-16 *Peter Dadam, Klaus Kuhn, Manfred Reichert*
Clinical Workflows - The Killer Application for Process-oriented Information Systems?

97-17 *Mohammad Ali Livani, Jörg Kaiser*
EDF Consensus on CAN Bus Access in Dynamic Real-Time Applications

97-18 *Johannes Köbler,Rainer Schuler*
Using Efficient Average-Case Algorithms to Collapse Worst-Case Complexity Classes

98-01 *Daniela Damm, Lutz Claes, Friedrich W. von Henke, Alexander Seitz, Adelinde Uhrmacher, Steffen Wolf*
Ein fallbasiertes System für die Interpretation von Literatur zur Knochenheilung

98-02 *Thomas Bauer, Peter Dadam*
Architekturen für skalierbare Workflow-Management-Systeme - Klassifikation und Analyse

98-03 *Marko Luther, Martin Strecker*
A guided tour through *Typelab*

98-04 *Heiko Neumann, Luiz Pessoa*
Visual Filling-in and Surface Property Reconstruction

98-05 *Ercüment Canver*
Formal Verification of a Coordinated Atomic Action Based Design

98-06 *Andreas Küchler*
On the Correspondence between Neural Folding Architectures and Tree Automata

98-07 *Heiko Neumann, Thorsten Hansen, Luiz Pessoa*
Interaction of ON and OFF Pathways for Visual Contrast Measurement

98-08 *Thomas Wennekers*
Synfire Graphs: From Spike Patterns to Automata of Spiking Neurons

98-09 *Thomas Bauer, Peter Dadam*
Variable Migration von Workflows in *ADEPT*

98-10 *Heiko Neumann, Wolfgang Sepp*
Recurrent V1 – V2 Interaction in Early Visual Boundary Processing

98-11 *Frank Houdek, Dietmar Ernst, Thilo Schwinn*
Prüfen von C–Code und Statmate/Matlab–Spezifikationen: Ein Experiment

98-12 *Gerhard Schellhorn*
Proving Properties of Directed Graphs: A Problem Set for Automated Theorem Provers

98-13 *Gerhard Schellhorn, Wolfgang Reif*
Theorems from Compiler Verification: A Problem Set for Automated Theorem Provers

98-14 *Mohammad Ali Livani*
SHARE: A Transparent Mechanism for Reliable Broadcast Delivery in CAN

98-15 *Mohammad Ali Livani, Jörg Kaiser*
Predictable Atomic Multicast in the Controller Area Network (CAN)

99-01 *Susanne Boll, Wolfgang Klas, Utz Westermann*
A Comparison of Multimedia Document Models Concerning Advanced Requirements

99-02 *Thomas Bauer, Peter Dadam*
Verteilungsmodelle für Workflow-Management-Systeme - Klassifikation und Simulation

99-03 *Uwe Schöning*
On the Complexity of Constraint Satisfaction

99-04 *Ercument Canver*
Model-Checking zur Analyse von Message Sequence Charts über Statecharts

99-05 *Johannes Köbler, Wolfgang Lindner, Rainer Schuler*
Derandomizing RP if Boolean Circuits are not Learnable

99-06 *Utz Westermann, Wolfgang Klas*
Architecture of a DataBlade Module for the Integrated Management of Multimedia Assets

99-07 *Peter Dadam, Manfred Reichert*
Enterprise-wide and Cross-enterprise Workflow Management: Concepts, Systems, Applications. Paderborn, Germany, October 6, 1999, GI–Workshop Proceedings, Informatik ’99

99-08 *Vikraman Arvind, Johannes Köbler*
Graph Isomorphism is Low for ZPP^{NP} and other Lowness results

99-09 *Thomas Bauer, Peter Dadam*
Efficient Distributed Workflow Management Based on Variable Server Assignments

2000-02 *Thomas Bauer, Peter Dadam*
Variable Serverzuordnungen und komplexe Bearbeiterzuordnungen im Workflow-Management-System ADEPT

2000-03 *Gregory Baratoff, Christian Toepfer, Heiko Neumann*
Combined space-variant maps for optical flow based navigation

2000-04 *Wolfgang Gehring*
Ein Rahmenwerk zur Einführung von Leistungspunktsystemen

2000-05 *Susanne Boll, Christian Heinlein, Wolfgang Klas, Jochen Wandel*

Intelligent Prefetching and Buffering for Interactive Streaming of MPEG Videos

2000-06 *Wolfgang Reif, Gerhard Schellhorn, Andreas Thums*
Fehlersuche in Formalen Spezifikationen

2000-07 *Gerhard Schellhorn, Wolfgang Reif (eds.)*
FM-Tools 2000: The 4th Workshop on Tools for System Design and Verification

2000-08 *Thomas Bauer, Manfred Reichert, Peter Dadam*
Effiziente Durchführung von Prozessmigrationen in verteilten Workflow-Management-Systemen

2000-09 *Thomas Bauer, Peter Dadam*
Vermeidung von Überlastsituationen durch Replikation von Workflow-Servern in ADEPT

2000-10 *Thomas Bauer, Manfred Reichert, Peter Dadam*
Adaptives und verteiltes Workflow-Management

2000-11 *Christian Heinlein*
Workflow and Process Synchronization with Interaction Expressions and Graphs

2001-01 *Hubert Hug, Rainer Schuler*
DNA-based parallel computation of simple arithmetic

2001-02 *Friedhelm Schwenker, Hans A. Kestler, Günther Palm*
3-D Visual Object Classification with Hierarchical Radial Basis Function Networks

2001-03 *Hans A. Kestler, Friedhelm Schwenker, Günther Palm*
RBF network classification of ECGs as a potential marker for sudden cardiac death

2001-04 *Christian Dietrich, Friedhelm Schwenker, Klaus Riede, Günther Palm*
Classification of Bioacoustic Time Series Utilizing Pulse Detection, Time and Frequency Features and Data Fusion

2002-01 *Stefanie Rinderle, Manfred Reichert, Peter Dadam*
Effiziente Verträglichkeitsprüfung und automatische Migration von Workflow-Instanzen bei der Evolution von Workflow-Schemata

2002-02 *Walter Guttmann*
Deriving an Applicative Heapsort Algorithm

2002-03 *Axel Dold, Friedrich W. von Henke, Vincent Vialard, Wolfgang Goerigk*
A Mechanically Verified Compiling Specification for a Realistic Compiler

2003-01 *Manfred Reichert, Stefanie Rinderle, Peter Dadam*
A Formal Framework for Workflow Type and Instance Changes Under Correctness Checks

2003-02 *Stefanie Rinderle, Manfred Reichert, Peter Dadam*
Supporting Workflow Schema Evolution By Efficient Compliance Checks

2003-03 *Christian Heinlein*
Safely Extending Procedure Types to Allow Nested Procedures as Values

2003-04 *Stefanie Rinderle, Manfred Reichert, Peter Dadam*
On Dealing With Semantically Conflicting Business Process Changes.

2003-05 *Christian Heinlein*
Dynamic Class Methods in Java

2003-06 *Christian Heinlein*
Vertical, Horizontal, and Behavioural Extensibility of Software Systems

2003-07 *Christian Heinlein*
Safely Extending Procedure Types to Allow Nested Procedures as Values (Corrected Version)

2003-08 *Changling Liu, Jörg Kaiser*
Survey of Mobile Ad Hoc Network Routing Protocols)

2004-01 *Thom Frühwirth, Marc Meister (eds.)*
First Workshop on Constraint Handling Rules

2004-02 *Christian Heinlein*
Concept and Implementation of C+++, an Extension of C++ to Support User-Defined Operator Symbols and Control Structures

2004-03 *Susanne Biundo, Thom Frühwirth, Günther Palm(eds.)*
Poster Proceedings of the 27th Annual German Conference on Artificial Intelligence

2005-01 *Armin Wolf, Thom Frühwirth, Marc Meister (eds.)*
19th Workshop on (Constraint) Logic Programming

2005-02 *Wolfgang Lindner (Hg.), Universität Ulm , Christopher Wolf (Hg.) KU Leuven*
2. Krypto-Tag – Workshop über Kryptographie, Universität Ulm

2005-03 *Walter Guttmann, Markus Maucher*
Constrained Ordering

2006-01 *Stefan Sarstedt*
Model-Driven Development with ACTIVECHARTS, Tutorial

2006-02 *Alexander Raschke, Ramin Tavakoli Kolagari*
Ein experimenteller Vergleich zwischen einer plan-getriebenen und einer leichtgewichtigen Entwicklungsmethode zur Spezifikation von eingebetteten Systemen

2006-03 *Jens Kohlmeyer, Alexander Raschke, Ramin Tavakoli Kolagari*
Eine qualitative Untersuchung zur Produktlinien-Integration über Organisationsgrenzen hinweg

2006-04 *Thorsten Liebig*
Reasoning with OWL - System Support and Insights –

2008-01 *H.A. Kestler, J. Messner, A. Müller, R. Schuler*
On the complexity of intersecting multiple circles for graphical display

2008-02 *Manfred Reichert, Peter Dadam, Martin Jurisch,l Ulrich Kreher, Kevin Göser, Markus Lauer*
Architectural Design of Flexible Process Management Technology

Ulmer Informatik-Berichte
ISSN 0939-5091

Herausgeber:
Universität Ulm
Fakultät für Ingenieurwissenschaften und Informatik
89069 Ulm