Model-Driven User Interface Generation and Adaption in Process-Aware Information Systems

Jens Kolb, Paul Hübner, Manfred Reichert

Ulmer Informatik-Berichte
Nr. 2012-04
August 2012
Model-Driven User Interface Generation and Adaptation in Process-Aware Information Systems

Jens Kolb, Paul Hübner and Manfred Reichert

Institute of Databases and Information Systems
Ulm University, Germany
{jens.kolb,paul.huebner,manfred.reichert}@uni-ulm.de
http://www.uni-ulm.de/dbis

Abstract. The increasing adoption of process-aware information systems (PAISs) has resulted in a large number of implemented business processes. To react on changing needs, companies should be able to quickly adapt these process implementations if required. Current PAISs, however, only provide mechanisms to evolve the schema of a process model, but do not allow for the automated creation and adaptation of their user interfaces (UIs). The latter may have a complex logic and comprise, for example, conditional elements or database queries. Creating and evolving the UI components of a PAIS manually is a tedious and error-prone task. This technical report introduces a set of patterns for transforming fragments of a business process model, whose activities are performed by the same user role, to UI components of the PAIS. In particular, UI logic can be expressed using the same notation as for process modeling. Furthermore, a transformation method is introduced, which applies these patterns to automatically derive UI components from a process model by establishing a bidirectional mapping between process model and UI. This mapping allows propagating UI changes to the process model and vice versa. Overall, our approach enables process designers to rapidly develop and update complex UI components in PAISs.

1 Introduction

Process-aware information systems (PAISs) separate process execution from application code, i.e., the implementation of process activities. Hence, separation of concerns, which is a well-established principle in computer science is realized based on explicit process models. When initially capturing business processes in process models, focus is put on business aspects, while technical aspects concerning later process execution are excluded. Usually, the resulting business process models cover the users’ activities at a fine-grained level (cf. Figure 1a). Hence, before deploying such a business process model in the PAIS, therefore, it must be revised and customized. For example, several human tasks, forming a process fragment in the business process model, may have to be combined into one activity in the executable process model (cf. Figure 1b). This activity is then
implemented by a respective user interface (UI) component in the PAIS, e.g., a user form whose corresponds to the one of the initial process fragment (cf. Figure 1). Based on such a logic, for example, form elements may be disabled when selecting a certain check box, or database queries or web services may be run in the background to fetch or save data. Overall, both the implementation and maintenance of the UI components in a PAIS is a cumbersome and costly task. In particular, this hinders quick adaptations of process implementations [1].

Fig. 1. Deriving UI Components from a Business Process Model

The proper evolution of the processes implemented in a PAIS is a critical success factor for any company [2]. It not only requires changes of the process models, but of the associated UI components and their internal logic as well (e.g., adding new input fields). Process model evolution is a well-understood feature in modern PAIS [3,4]. There exist editors for defining and changing simple UI components of the PAISs (e.g., moving or renaming input fields in a user form). However, complex changes of the internal logic of user forms can not be accomplished by users, but require process implementers to intervene. Moreover, the automatic propagation of changes made in the UI components to business process model and vice versa is not supported.

This paper addresses these issues through the automatic generation of complex UI components out of process model fragments. In this context, we first present common patterns for transforming process model fragments to UI components. Thereby, we distinguish between elementary and complex transformations. While elementary transformation patterns (ETP) transform single activities of a process model to simple UI elements, complex transformation patterns (CTP) enable the mapping of entire process fragments and their logic to UI components, showing the same behaviour as the original process fragment does. Additionally, we provide an advanced transformation method for business process models that allows generating sophisticated UI components out of these models based on the user roles assigned to process activities. This method further allows propagating changes of UI components to the corresponding process model and vice versa. Especially, for human-centric processes, our transformation method decreases the effort for evolving PAIS to accommodate to changing needs.

The paper is structured as follows: Section 2 introduces basic notions. Section 3 describes common patterns for transforming process model fragments to UI components. Section 4 then presents a method for transforming process models
Automatic User Interface Generation

3 to a set of UI components. This method is based on transformation patterns and role-based process views. Section 5 sketches our proof-of-concept prototype. Section 6 discusses related work and Section 7 summarizes the paper.

2 Basic Notions

A process model is described in terms of a directed graph whose node set comprises activities, gateways, and data elements. An activity either corresponds to a human task and thus requires user interactions, or to a service representing an automated task. In turn, gateways can be categorized into AND, XOR and Loop and are used for modeling parallel/conditional branchings and loop structures. Edges between activities and/or gateways represent precedence relations, i.e., the control flow of the process model (cf. Figure 1a). Furthermore, data elements comprise primitive data elements and complex ones. Primitive data elements cover elementary data values of the process model and have one of the following types: integer, float, boolean, string, date, or URI. In turn, complex data elements compose primitive and/or complex data elements. Based on this, the data flow is defined by a set of directed edges connecting data elements and activities. Writing a data element is expressed through an edge pointing from an activity to the data element. In turn, reading a data element is expressed through an edge from this data element to the activity. We presume that process models are well-structured [5,6], i.e., sequences, branchings (of different semantics), and loops are specified as blocks with well-defined start and end nodes having the same gateway type. These blocks, also known as SESE (single-entry-single-exit) blocks, may be nested, but are not allowed to overlap (cf. Figure 1a).

3 User Interface Transformation Patterns

This section introduces a set of well-elaborated patterns for transforming a process fragment to a corresponding UI component. Section 3.1 discusses how we identified these UI transformation patterns. Section 3.2 then introduces Elementary Transformation Patterns (ETP), which constitute the basis for the Complex Transformation Patterns (CTP) presented in Section 3.3.

3.1 Pattern Identification

The goal of our research is to identify and apply a general set of UI transformation patterns, which allow mapping process fragments to UI components. To achieve this goal, we first describe a three-step method, which we apply for identifying relevant UI transformation patterns (cf. Figure 2). In Step 1, we analyze and evaluate PAIS engineering projects in which we were involved in the past. More precisely, we analyze the business process models from these projects as well as their technical implementation and related UI components. In Step 2,
we analyze existing PAISs (e.g., IBM WebSphere Lombardi, AristaFlow BPM Suite [7]) and their support for UI generation. Finally, in Step 3 we scan existing literature on model-driven development (MDD) of UI components [8,9,10].

The empirical results of these three steps are used to specify general UI transformation patterns. These either transform single process model elements, like activities and their inputs/outputs, to simple UI elements (i.e., elementary transformation patterns (ETP)) or entire process fragments to complex UI components (i.e., complex transformation patterns (CTP)).

Based on the identified transformation patterns, Step 4 develops a transformation method allowing for the automatic generation of the UI components of the PAIS by extracting role-specific process fragments from process models and transforming them into related UI components (cf. Section 4).

3.2 Elementary Transformation Patterns

Elementary transformation patterns (ETP) enable the transformation of single process model elements to simple UI elements. For example, an activity may be transformed into a simple user form. Thereby, the respective ETP considers activity input/output data elements and maps them to form elements.

Each transformation pattern has a unique pattern name. Its description summarizes the actions required to transform the process elements to corresponding UI elements. Furthermore, a real-world application example illustrates the application of the pattern. In this context, process elements are represented using BPMN and UI elements are illustrated in terms of screen mock-ups. The problem description refers to situations in which the patterns may be applied. The implementation defines implementation-specific requirements to be met to realize the pattern in a PAIS. Related patterns refers to related transformation patterns, e.g., patterns to be co-applied with the current pattern.

ETP1 (Human Activity Transformation). This pattern describes the elementary transformation of a single activity to a Form Group Element (FGE) (cf. Table 1): i.e., for each human activity of a process model, an FGE is generated. In modern PAIS, usually, such an FGE is represented by a dialog window.

ETP2 (Service Activity Transformation). The counterpart of ETP1 is given by pattern ETP2 (cf. Table 1). ETP2 creates application stubs for automated tasks not performed by a user; i.e., no user interaction element is created, but a stub for integrating application code executed by the PAIS. In BPMN,
respective activities are denoted as service activities. For example, a service activity could fetch data from a database, which is then displayed to the human user. Pattern ETP2 is needed to automatically generate complete UI components enabling interactions with both users and backend systems (cf. Section 3.3).

<table>
<thead>
<tr>
<th>ETP1: Human Activity Transformation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong> A human activity of a business process model (i.e., an activity to be performed by a human resource) is transformed into a Form Group Element (FGE). An FGE corresponds to a UI element that contains UI elements for displaying or editing data.</td>
</tr>
<tr>
<td><strong>Example:</strong> A clerk must perform an activity, in which customer data is edited.</td>
</tr>
<tr>
<td><strong>Problem:</strong> To perform a human activity within a PAIS, a user interaction is required.</td>
</tr>
<tr>
<td><strong>Implementation:</strong> An FGE can be implemented in terms of a dialog window. In the context of CTPs, an FGE constitutes a grouping element of the UI.</td>
</tr>
<tr>
<td><strong>Related Pattern:</strong> ETP2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ETP2: Service Activity Transformation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong> A service activity is transformed into an application stub executable by the PAIS. No human interaction is required.</td>
</tr>
<tr>
<td><strong>Example:</strong> Customer data shall be retrieved from an external ERP system.</td>
</tr>
<tr>
<td><strong>Problem:</strong> Complex UI components not only require direct user interactions, but also interactions with backend information systems. Hence, service activities need to be integrated with the UI components as well.</td>
</tr>
<tr>
<td><strong>Implementation:</strong> For service activities, application stubs need to be created. Application components then may be provided by the process implementer.</td>
</tr>
<tr>
<td><strong>Related Patterns:</strong> ETP1</td>
</tr>
</tbody>
</table>

Table 1. Pattern Descriptions for Patterns ETP1 and ETP2

**ETP3 (Data Flow Transformation).** This elementary pattern allows transforming data elements and associated data flow edges to respective UI elements (cf. Table 2). Pattern ETP3 as well as related patterns ETP3.1-ETP3.4 generate Field Elements (FE) within an FGE; i.e., when generating the FGE (cf. ETP1), the data elements and edges of a process activity are transformed to input/output field elements of a UI component. In this context, patterns ETP3.1 and ETP3.2 are applied to indicate whether the respective field element is read-only or editable (with optionally pre-filled value). Further, ETP3.3 transforms the data type of a data element to the FE; e.g., a boolean data element is transformed to a radio button element with two choices. Finally, ETP3.4 allows transforming a business object to a UI component built upon ETP3.3.
**ETP3.1 (WRITE Data Flow Transformation).** *ETP3.1* transforms the read access of a process activity to a data element to an editable form element (cf. Table 2). The data type of the form element is determined by *ETP3.3*. Overall, *ETP3.1* ensures that the respective FE is editable and the user provides data, which is then mapped to the corresponding data element of the process model at run-time.

### ETP3: Data Flow Transformation

| Description | UI components are used to display and edit data elements of the process model. Patterns ETP3 as well as patterns ETP3.1-ETP3.4 transform these data elements and associated data edges to Field Elements (FE). |
| Example | A user edits customer data, while performing a particular activity. |
| Problem | Human activities and associated data elements require UI field elements allowing the user to fill in or update information. |
| Implementation | FGEs generated by ETP1 have to be enriched with FE. This is done through the application of the elementary patterns ETP3.1-3.4. |
| Related Patterns | ETP1, ETP3.1-ETP3.4 |

### ETP3.1: WRITE Data Access Transformation

| Description | A WRITE data access of an activity to a data element requires an input field element (FE) placed on the FGE and allowing the user to enter data. The value entered in the FE is then written to the respective data element. |
| Example | A user inserts customer data when executing activity *Edit Customer.* |
| Problem | Inserting data values through UI components is crucial for users interacting with the PAIS. |
| Implementation | When generating a FE, methods are required to forward the data entered by the user to the corresponding data element in the PAIS. The respective FE must be editable. |
| Related Patterns | ETP1, ETP3 |

**ETP3.2 (READ Data Transformation).** *ETP3.2* transforms the read access of an activity to a data element to a read-only form element (e.g., a non-editable text label, cf. Table 3). At run-time, the FE is pre-filled with the value of the process data element. In certain cases, an activity has both incoming and outgoing data edges linking it with the same data element; i.e., the activity may read and update the respective data element. Hence, both patterns *ETP3.1* and *ETP3.2* are applied. As a result, a pre-filled FE is obtained whose value may be updated when executing the activity and UI component respectively.
ETP3.3 (Primitive Data Type Transformation). This pattern transforms the primitive data types integer, float, boolean, string, date, and URI to typed form elements (cf. Table 3). For example, type date is transformed to an input field with a date picker validating the input. Overall, the most relevant primitive data types used in process models are covered [11]. Additionally, the respective form element is added to the FGE of the corresponding process activity.

<table>
<thead>
<tr>
<th>ETP3.2: READ Data Access Transformation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong> A READ data access creates a read-only field element (FE) in the UI, displaying the value of the respective process data element to the user.</td>
</tr>
<tr>
<td><strong>Example:</strong> Customer data is displayed to a clerk in his UI component.</td>
</tr>
<tr>
<td>READ Data Access: Data Type: Name</td>
</tr>
<tr>
<td>READ Data Access: Data Type: Address</td>
</tr>
<tr>
<td>READ Data Access: Data Type: Date</td>
</tr>
<tr>
<td>READ Data Access: Data Type: URI</td>
</tr>
<tr>
<td>READ Data Access: Data Type: Boolean</td>
</tr>
<tr>
<td>READ Data Access: Data Type: Integer</td>
</tr>
<tr>
<td>READ Data Access: Data Type: Float</td>
</tr>
<tr>
<td>READ Data Access: Data Type: String</td>
</tr>
<tr>
<td>READ Data Access: Data Type: Date</td>
</tr>
<tr>
<td>READ Data Access: Data Type: URI</td>
</tr>
<tr>
<td>READ Data Access: Data Type: Boolean</td>
</tr>
<tr>
<td>READ Data Access: Data Type: Integer</td>
</tr>
<tr>
<td>READ Data Access: Data Type: Float</td>
</tr>
<tr>
<td>READ Data Access: Data Type: String</td>
</tr>
<tr>
<td>Problem: For displaying process data elements, read-only FEs within the UI component are required.</td>
</tr>
<tr>
<td>Implementation: Outgoing data edges of an activity are transformed to read-only FEs, which are pre-filled with the values of the respective data elements at run-time.</td>
</tr>
<tr>
<td>Related Patterns: ETP1, ETP3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ETP3.3: Primitive Data Type Transformation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong> Primitive data types require the generation of specific UI form elements in a consistent and comprehensible style.</td>
</tr>
<tr>
<td><strong>Example:</strong> A form for editing the business object customer data requires the processing of data elements like customer number or birth date. These have primitive data types and must be mapped to a respective FE.</td>
</tr>
<tr>
<td>Problem: For process data elements accessed by an activity, different form elements, depending on the respective primitive data types, have to be generated.</td>
</tr>
<tr>
<td>Implementation: Each data element, having a primitive data type, is transformed to a specific FE and placed within the FGE of the associated activity.</td>
</tr>
<tr>
<td>Related Patterns: ETP1, ETP3</td>
</tr>
</tbody>
</table>

Table 3. Pattern Descriptions for Patterns ETP3.2 and ETP3.3

ETP3.4 (Business Object Data Type Transformation). To handle complex data types (i.e., business objects), pattern ETP3.4 can be applied (cf. Table 4 [12]. It deals with data assembled of primitive or other complex data types. For example, a customer business object may consist of the primitive data elements name, street, and birth date. The implementation of pattern ETP3.4 creates form group elements for logically grouping the primitive data elements the business object consists of.
ETP3.4: Business Object Data Type Transformation

Description: A business object is transformed to an FGE. This FGE contains subordinate FEs for the data elements (with primitive data types) the business object consists of. For each primitive data type, ETP3.3 is applied. If the business object refers to other business objects, ETP3.3 is applied recursively.

Example: A customer business object contains an address business object that, in turn, consists of primitive data type attributes.

Problem: Business objects may comprise primitive data elements as well as business objects (complex data elements). All these data elements must be extracted and respective UI elements be generated for them.

Implementation: Before generating FEs, all primitive data elements of a business object have to be extracted recursively and grouped according to the schema of the business object.

Related Patterns: ETP3, ETP3.3

Table 4. ETP3.4: Business Object Data Type Transformation

3.3 Complex Transformation Patterns

Complex Transformation Patterns (CTPs) allow transforming entire process fragments of a process model, whose activities shall be performed by the same user, to corresponding UI components. When creating such a UI component both the control and data flow of the respective process fragment are considered. Hence, each of the generated UI components covers parts of the overall process logic. Furthermore, by combining role-specific activities in the same UI component, unnecessary UI context switches can be avoided. To structure such a UI component, tab elements—called Tab Container Elements (TCE)—are introduced. Moreover, a CTP interconnects TCEs according to the control flow of the process fragment to which it is applied. In this context, single activities and related data elements of the process fragment are transformed using ETPs.

CTP1 (Process Model Transformation). For a process fragment whose activities shall be processed by the same user or user role, pattern CTP1 generates a User Interface Dialog (UID) (cf. Table 5). Such a UID is a toplevel container and is represented by a dialog window in the PAIS, which contains a number of UI elements representing the activities and data elements.

CTP2 (Sequence Block Transformation). This pattern deals with the transformation of a sequence of activities (and SESE blocks respectively) to TCEs (cf. Table 5). For each activity (or SESE block) of the sequence, a TCE element is generated and linked to other TCEs according to the given activity sequence.
CTP1: Process Model Transformation

**Description:** For a particular process fragment, a surrounding User Interface Dialog (UID), i.e., a toplevel container window, is generated. Following this, all other UI elements related to activities of this fragment are generated based on ETPs and CTPs, and are then embedded in the UID.

**Example:** All interactions with a clerk shall be done using the same UI component.

**Problem:** The UI elements related to the activities and data elements of a particular process fragment need to be mapped to a toplevel container window. The UI flow logic (e.g., the ordering in which field elements may be displayed or written) corresponds to the control flow of the given process fragment.

**Implementation:** For each process fragment, a UID element (dialog window) is generated. 

**Related Patterns:** None

CTP2: Sequence Block Transformation

**Description:** A sequence of activities (and SESE blocks respectively) is transformed into a sequence of Tab Container Elements (TCE) to be processed in the same sequential order.

**Example:** A clerk first edits the customer data. Afterwards, he edits the corresponding contact data.

**Problem:** Human activities, performed in sequence by the same user (role), shall be accomplished using the same UI component, instead of using separate UI components (e.g., dialog windows) for each activity.

**Implementation:** For each activity (or SESE block), a TCE element is created. The order in which these TCEs may be processed, corresponds to the one of the respective activities.

**Related Patterns:** ETP1, ETP2, ETP3, CTP3

Table 5. Pattern Descriptions for Patterns CTP1 and CTP2

CTP3 (Parallel Block Transformation). This pattern transforms parallel activities (or SESE blocks) of a process fragment to respective UI elements within the same UID. The UI elements may then be accessed concurrently (cf. Table 6). The resulting UI component is similar to the one of a single activity (cf. ETP1, Table 1). However, pattern CTP3 not only covers the transformation of activities or SESE blocks arranged in parallel. It also enables the concurrent processing of arbitrary SESE blocks arranged in parallel to the respective UI elements.

CTP4 (XOR Block Transformation). This pattern is applied for transforming an XOR branching of a given process fragment to the UI component (cf. Table 6). CTP4 generates independent TCEs for each branch of the XOR branching. The decision, which branch and hence which TCE shall be selected, is made during run-time based on process data; e.g., whether the TCE element for creating a new customer or the one for editing an existing customer shall be displayed, depends on decision data that only becomes available during process
execution. In particular, run-time data for deciding which of the branches of an XOR branching shall be executed, is required.

### CTP3: Parallel Block Transformation

**Description:** A parallel block and its activities are mapped to a single TCE for their processing. This TCE allows for their concurrent.

**Example:** While editing the address of a customer, the contact type the customer wants to use for communication can be entered in parallel.

![Diagram of parallel block transformation](image)

**Problem:** Activities (or SESE blocks) of a process fragment, which are performed in parallel by the same user (role), shall be mapped to the same UI component. UI elements then must be displayable/editable concurrently.

**Implementation:** When applying CTP3, for each parallel branch, FGEs are added to the TCE.

**Related Patterns:** ETP1, ETP2, ETP3, CTP2

### CTP4: XOR Block Transformation

**Description:** An XOR block and its branches are usually mapped to multiple UI components in the PAIS, one for each branch. By contrast, CTP4 transforms them to one TCE, if the corresponding activities are performed by the same user role. At run-time, this TCE then only displays the UI elements corresponding to the activities (or SESE blocks) of the selected branch.

**Example:** If a customer entry already exists in the customer database, the form for editing customer data is displayed. Otherwise, the form for creating a new customer is displayed.

![Diagram of XOR block transformation](image)

**Problem:** Branches of an XOR branching require the generation of several UI elements for each branch. At run-time, only the UI elements of the selected branch shall be displayed to the user.

**Implementation:** A placeholder TCE is inserted at build-time. At run-time, the decision is made which concrete UI elements are dynamically added to that TCE.

**Related Patterns:** ETP1, ETP2, ETP3, CTP2

### CTP5 (Loop Block Transformation)

This pattern transforms a loop block to elements of a UI component (cf. Table 7). For each loop, CTP5 generates a TCE as well as corresponding UI elements for activities (or SESE blocks) nested in this loop (cf. CTP1, Table 5). Additionally, a decision element is required to decide whether to leave the loop after completion of a particular iteration or trigger the next loop iteration. The iteration can be triggered by data elements processed during loop execution (e.g., evaluating input data for validity) or by external criteria (e.g., repeat calling someone until getting an answer).
CTP6 (Background Activity Transformation). Table 7 shows an advanced scenario for transforming process fragments to UI components. CTP6 is a pattern reflecting the need for dynamically loading data elements by a service activity. More precisely, data has to be fetched from or stored to a backend system, while the user concurrently works on human activities by a user.

The next section introduces our overall UI transformation method. It applies the introduced transformation patterns in connection with process views [13].

CTP5: Loop Block Transformation

<table>
<thead>
<tr>
<th>Description</th>
<th>A loop block is transformed to a TCE which then may contains the UI elements corresponding to the activities located inside this loop block.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example</td>
<td>Customer data shall be first edited and then validated. Depending on the validation result, editing may have to be redone (e.g., if invalid data was entered).</td>
</tr>
<tr>
<td>Problem</td>
<td>Loop blocks may consist of nested activities and SESE blocks. Each loop iteration allow the user (role) to edit previously entered data.</td>
</tr>
<tr>
<td>Implementation</td>
<td>For each loop block, a TCE is inserted. The latter is executed as long as the loop exit condition evaluates to false. Furthermore, all nested UI elements of the loop block are associated with the TCE.</td>
</tr>
<tr>
<td>Related Patterns</td>
<td>ETP1, ETP2, ETP3, CTP2, CTP3, CTP4, CTP6</td>
</tr>
</tbody>
</table>

CTP6: Background Activity Transformation

<table>
<thead>
<tr>
<th>Description</th>
<th>While human activities are performed by human resources, service activities may automatically fetch or save data concurrently in the background.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example</td>
<td>A user selects a customer name in order to edit respective customer data. After selecting the name, in the background, all available customer information is retrieved from the database and displayed to the user.</td>
</tr>
<tr>
<td>Problem</td>
<td>A service activity needs to be executed concurrently to human activities performed by the same user (role). This requires the concurrent fetching/storing of data from a database as well as the automated and dynamic displaying of new form elements.</td>
</tr>
<tr>
<td>Implementation</td>
<td>Dynamic forms, which contain background activities, need a change listener mechanism to detect user inputs and to react on them.</td>
</tr>
<tr>
<td>Related Patterns</td>
<td>ETP1, ETP2, CTP2</td>
</tr>
</tbody>
</table>

Table 7. Pattern Descriptions for Patterns CTP5 and CTP6
4 Transforming Process Models to User Interfaces

Section 4 shows how the presented patterns can be used to transform process fragments into sophisticated UI components of the PAIS. In particular, we introduce a transformation method (cf. Section 4.1) and discuss how process fragments can be adapted through changes of the UI components (cf. Section 4.2).

4.1 User Interface Transformation Method

To transform a complete business process model, consisting of several process fragments, to multiple UI components, we introduced five steps (cf. Figure 3). Thereby, the number of generated UI components depends on the number of different users and user roles involved in the process.

**Step 1.** Role-specific process views are created for the given process model. To be more precise, a process view abstracts from certain aspects of the process model. For example, it may only represent the activities of a particular user role or organisational unit. In our context, a role-specific process view constitutes the basis for creating a role-specific UI component.

**Step 2.** For each process view, a User Interface Dialog (UID) is created. As aforementioned, a UID acts as a toplevel container that includes all UI elements required for processing the activities of a specific process view. For this purpose, pattern CTP1 is applied (cf. Table 5).

**Step 3.** In order to transform complete process fragments to UI elements, complex transformation patterns CTP2-6 are applied. For each CTP applied, a Tab Container Element (TCE) is generated. Each TCE is represented in the tab bar area (cf. Figure 3, Step 3). When clicking on an item in this area, the corresponding UI elements are displayed. If there are nested SESE blocks, they are displayed in a hierarchical tree in the tab bar area.

**Step 4.** Single activities (ETP1+2) are transformed into Form Group Elements (FGE). Basically, each FGE represents one activity in the process model. In case of a parallel branching, multiple activities are displayed on a TCE element in the UI (cf. Figure 3, Step 4).

**Step 5.** All data elements of the process view are transformed to Field Elements (FE). These are layouted and positioned within an FGE. In this context, there exist different kinds of FEs depending on the data type of the respective data element (cf. ETP3.3, Table 3).

The structure of the resulting UI component is represented through a User Interface Model (UIM). Figure 4b shows an example of such a UIM. This tree-based
schema describes the hierarchical structure of the UI as generated by our transformation method. Section 4.2 shows that this UIM can be also used to propagate changes of the UI component to the process model and vice versa.

4.2 Synchronizing Process Model and UI Changes

After generating complex UI components for a process model through process views and deploying them in the PAIS, users may want to modify the UI. For example, they might add a new FE or re-position elements within the UI.

![Fig. 3. Transformation Method](image-url)

Basically, two categories of UI changes can be distinguished. Local changes are changes not affecting the associated process view. For example, assume that a user re-positions the FGE *Edit Address* within the TCE *Edit Contact (AND)* in Figure 4a. Such change would not affect the execution order of the activities.
in the corresponding process view, i.e., it only affects the visual representation of the UI component. Hence, the change needs not be propagated to the view. By contrast, global changes modify the logic of the UI as well as the associated process view. For example, the user may want to add the FGE Edit Phone Number together with the respective FE Phone (cf. Figure 4a). This change then affects the control flow of the underlying process view as well. The correct position of the change within the UIM can be determined by the hierarchical structure of the GUI (cf. Figure 4b). The changes of the UIM are then propagated to the respective process view; note that the latter is represented by the UIM. Finally, the change of the process view has to be propagated to the basis process model on which the view is created. For this propagation the concepts developed in the proView project can be applied \[16,17\].

![Fig. 4. Adapting Process Models through Changes in the UI Component](image)

5 Proof-of-Concept Prototype

We developed a proof-of-concept prototype for the described UI generation approach. In particular, we implemented the component enabling process view generation as well as the described transformation patterns and method. This UI generation prototype has been realized in the context of the proView research project, which tries to reduce process model complexity for end-users by generating personalized and tailored views on process models for them \[13,14\]. The prototypical implementation is based on a layered REST architecture using the vaadin MVC web framework (cf. Figure 5). To evaluate the general validity of the transformation patterns and method, we used the prototype to generate

\[1\] http://www.dbis.info/proView
UI components for process models we gathered in several PAIS projects. This evaluation has shown remarkable results in the sense that the UI components generated by our approach are more or less the same as those manually implemented in respective projects. Hence, both implementation and maintenance effort in PAIS development can be significantly reduced.

Our evaluation has further shown that for sequences comprising numerous activities, the generated UI component may become too complex. In this case, the tab bar located on the left side of the UI contains too many entries. Here, additional transformation patterns need to be identified. Finally, run-time issues have not been fully considered yet. For example, it must be specified what shall happen if a user wants to step back in the UI, i.e., derived from the defined form logic.

![User Form Generated by the Proof-of-Concept Prototype](image)

**Fig. 5.** User Form Generated by the Proof-of-Concept Prototype

6 Related Work

Generally, UI development is a time consuming and expensive task in the context of PAIS development [18].

**Task Models:** Task models describe the actions to be performed by a user when interacting with an information system and the respective UI to reach a specific goal. Different variants of task models exist: Goals, Operators, Methods and Selection rules (GOMS) [19], or Hierarchical Task Analysis (HTA) [20]. These approaches describe the goals, steps and operations of a user interface. Concurrent Task Trees (CTT), in turn, provide a hierarchical task model supporting various types of tasks (e.g., automatic vs. manual task) and relationships between tasks (e.g., sequential vs. parallel execution) [21,22]. An overview of task
modeling approaches is provided in [23].

UI Description Language: As opposed to task models, UI description languages describe the concrete elements a user interface consists of, instead of their behaviour. Nevertheless, some of these languages include basic task modeling concepts as well [8]. Furthermore, respective languages are independent from a specific programming language and enable software developers to describe a user interface once, but deploy it on various devices (e.g., PC and tablets). Examples of UI languages include: User Interface Markup Language (UIML) [24], Abstract User Interface Markup Language (AUIML) [25], and User Interface Extensible Markup Language (UsiXML) [26]. Based on these languages, UI descriptions can be created at both built- and run-time.

Model-Driven UI Development: Model-driven UI development applies the principles of Model-Driven Development (MDD) to UI development. Although a lot of competing approaches exist, an accepted standard for model-driven UI development is missing [8,27,28,29,30]. FlowiXML [8], for example, provides a methodology to develop UIs for business processes, taking the organizational structure as well as the process model into account. FlowiXML suggests an eight-step-approach based on various model types (e.g.: CTT, Petri Nets, and UsiXML). However, it does not allow for the automated generation of UIs. Based on FlowiXML, [9] introduces the Business Alignment Framework. This four-step-approach provides task models (i.e., CTT) for user tasks within the process model. Furthermore, a domain model supplements the task model. Based on these models, an abstract UI description is generated and transformed into a concrete UI component during process execution. This approach allows for changes based on UIs and discuss how to manually align them with process models. However, automatic propagation is not supported.

The approach provided in [31] transforms a process model into a human interaction perspective, which allows UI designers to specify data elements, user roles, tasks, and UI layout. This perspective enables detailed views on the process model. After manually refining them, corresponding UIs are generated during run-time. Furthermore, data-centered process management approaches offer a different (data-centered) view on business processes. Therefore, state transitions of process-related data elements are described. Based on this information, UIs can be generated as well [10,32].

UI Generation in existing PAIS: Contemporary PAIS are able to create certain UIs automatically (e.g., IBM WebSphere Lombardi [33]). To be more precise, single activities of a process model can be transformed into simple user forms, taking associated data elements into account similar to the ETPs. However, more complex scenarios are not covered. None of the presented approaches allows for the automatic generation of complex user interfaces based on process models. Finally, the adaption of process models based on changes of the corresponding UI is only considered rudimentarily.
7 Conclusion

In this paper, we showed how UI components can be automatically created from entire process fragments and process models respectively. For this purpose, elementary and complex transformation patterns were identified and described. Furthermore, a transformation method, which applies these patterns to create complex UIs based on process views, were introduced. Our approach further enables the propagation of UI changes (e.g., adding new input fields) to the associated process model and vice versa. Finally, we implemented our UI generation approach in a powerful proof-of-concept prototype. In summary, our approach will contribute to reduce costs for PAIS development and maintenance.

In future research, we will address the execution aspects of process models and associated UIs as well. In this context, features such as jumping back to an already edited UI element will be supported by adapting the corresponding process instance to assure a valid history of its execution. Further, we started to implement a UI editor which uses a set of predefined UI widgets and templates for creating sophisticated UI components for process models. In this editor, we use our transformation method and change propagation features for deriving complete process models from the created complex UIs.

References

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 Simultaneous 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
<table>
<thead>
<tr>
<th>ID</th>
<th>Authors</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>92-08*</td>
<td>Uwe Schöning</td>
<td>On Random Reductions from Sparse Sets to Tally Sets</td>
</tr>
<tr>
<td>92-09*</td>
<td>Hermann von Hasseln, Laura Martignon</td>
<td>Consistency in Stochastic Network</td>
</tr>
<tr>
<td>92-10</td>
<td>Michael Schmitt</td>
<td>A Slightly Improved Upper Bound on the Size of Weights Sufficient to Represent Any Linearly Separable Boolean Function</td>
</tr>
<tr>
<td>92-11</td>
<td>Johannes Köbler, Seinosuke Toda</td>
<td>On the Power of Generalized MOD-Classes</td>
</tr>
<tr>
<td>92-12</td>
<td>V. Arvind, J. Köbler, M. Mundhenk</td>
<td>Reliable Reductions, High Sets and Low Sets</td>
</tr>
<tr>
<td>92-13</td>
<td>Alfons Geser</td>
<td>On a monotonic semantic path ordering</td>
</tr>
<tr>
<td>92-14*</td>
<td>Joost Engelfriet, Heiko Vogler</td>
<td>The Translation Power of Top-Down Tree-To-Graph Transducers</td>
</tr>
<tr>
<td>93-01</td>
<td>Alfred Lupper, Konrad Froitzheim</td>
<td>AppleTalk Link Access Protocol basierend auf dem Abstract Personal Communications Manager</td>
</tr>
<tr>
<td>93-03</td>
<td>Thomas Thierauf, Seinosuke Toda, Osamu Watanabe</td>
<td>On Sets Bounded Truth-Table Reducible to P-selective Sets</td>
</tr>
<tr>
<td>93-04</td>
<td>Jin-Yi Cai, Frederic Green, Thomas Thierauf</td>
<td>On the Correlation of Symmetric Functions</td>
</tr>
<tr>
<td>93-05</td>
<td>K.Kuhn, M.Reichert, M. Nathe, T. Beuter, C. Heinlein, P. Dadam</td>
<td>A Conceptual Approach to an Open Hospital Information System</td>
</tr>
<tr>
<td>93-06</td>
<td>Klaus Gaßner</td>
<td>Rechnerunterstützung für die konzeptuelle Modellierung</td>
</tr>
<tr>
<td>93-07</td>
<td>Ullrich Keßler, Peter Dadam</td>
<td>Towards Customizable, Flexible Storage Structures for Complex Objects</td>
</tr>
<tr>
<td>94-01</td>
<td>Michael Schmitt</td>
<td>On the Complexity of Consistency Problems for Neurons with Binary Weights</td>
</tr>
<tr>
<td>94-02</td>
<td>Armin Kühnemann, Heiko Vogler</td>
<td>A Pumping Lemma for Output Languages of Attributed Tree Transducers</td>
</tr>
<tr>
<td>94-03</td>
<td>Harry Buhrman, Jim Kadin, Thomas Thierauf</td>
<td>On Functions Computable with Nonadaptive Queries to NP</td>
</tr>
<tr>
<td>94-04</td>
<td>Heinz Faßbender, Heiko Vogler, Andrea Wedel</td>
<td>Implementation of a Deterministic Partial E-Unification Algorithm for Macro Tree Transducers</td>
</tr>
</tbody>
</table>
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

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 Independent 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

95-11  Thomas Beuter, Peter Dadam:  
Prinzipien der Replikationskontrolle in verteilten Systemen

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

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

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

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

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

Jochen Messner
Pattern Matching in Trace Monoids

Wolfgang Lindner, Rainer Schuler
A Small Span Theorem within P

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

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

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

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

Manfred Reichert, Peter Dadam
ADEPTflex - Supporting Dynamic Changes of Workflows Without Loosing Control

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

Christian Heinlein
Grundlagen von Interaktionsausdrücken

Christian Heinlein
Graphische Repräsentation von Interaktionsausdrücken

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 Wennemers
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

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 Wennemers
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  

99-08  Vikraman Arvind, Johannes Köbler  
Graph Isomorphism is Low for ZPPNP 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.)

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, Ulrich Kreher, Kevin Göser, Markus Lauer
Architectural Design of Flexible Process Management Technology

2008-03  Frank Raiser
Semi-Automatic Generation of CHR Solvers from Global Constraint Automata

2008-04  Ramin Tavakoli Kolagari, Alexander Raschke, Matthias Schneiderhan, Ian Alexander
Entscheidungsdocumentation bei der Entwicklung innovativer Systeme für produktlinien-basierte Entwicklungsprozesse

2008-05  Markus Kalb, Claudia Dittrich, Peter Dadam
Support of Relationships Among Moving Objects on Networks

2008-06  Matthias Frank, Frank Kargl, Burkhard Stiller (Hg.)
WMAN 2008 – KuVS Fachgespräch über Mobile Ad-hoc Netzwerke

2008-07  M. Maucher, U. Schöning, H.A. Kestler
An empirical assessment of local and population based search methods with different degrees of pseudorandomness

2008-08  Henning Wunderlich
Covers have structure

2008-09  Karl-Heinz Niggl, Henning Wunderlich
Implicit characterization of FPTIME and NC revisited

2008-10  Henning Wunderlich
On span-P^cc and related classes in structural communication complexity

2008-11  M. Maucher, U. Schöning, H.A. Kestler
On the different notions of pseudorandomness

2008-12  Henning Wunderlich
On Toda’s Theorem in structural communication complexity

2008-13  Manfred Reichert, Peter Dadam
Realizing Adaptive Process-aware Information Systems with ADEPT2

2009-01  Peter Dadam, Manfred Reichert
The ADEPT Project: A Decade of Research and Development for Robust and Flexible Process Support
Challenges and Achievements

2009-02  Peter Dadam, Manfred Reichert, Stefanie Rinderle-Ma, Kevin Göser, Ulrich Kreher, Martin Jurisch
2009-03  Alena Hallerbach, Thomas Bauer, Manfred Reichert  
Correct Configuration of Process Variants in Provop

2009-04  Martin Bader  
On Reversal and Transposition Medians

2009-05  Barbara Weber, Andreas Lanz, Manfred Reichert  
Time Patterns for Process-aware Information Systems: A Pattern-based Analysis

2009-06  Stefanie Rinderle-Ma, Manfred Reichert  
Adjustment Strategies for Non-Compliant Process Instances

Statistical Computing 2009 – Abstracts der 41. Arbeitstagung

2009-08  Ulrich Kreher, Manfred Reichert, Stefanie Rinderle-Ma, Peter Dadam  
Effiziente Repräsentation von Vorlagen- und Instanzdaten in Prozess-Management-Systemen

2009-09  Dammertz, Holger, Alexander Keller, Hendrik P.A. Lensch  
Progressive Point-Light-Based Global Illumination

2009-10  Dao Zhou, Christoph Müssel, Ludwig Lausser, Martin Hopfensitz, Michael Kühl, Hans A. Kestler  
Boolean networks for modeling and analysis of gene regulation

2009-11  J. Hanika, H.P.A. Lensch, A. Keller  
Two-Level Ray Tracing with Recordering for Highly Complex Scenes

2009-12  Stephan Buchwald, Thomas Bauer, Manfred Reichert  
Durchgängige Modellierung von Geschäftsprozessen durch Einführung eines Abbildungsmodells: Ansätze, Konzepte, Notationen

2010-01  Hariolf Betz, Frank Raiser, Thom Frühwirth  
A Complete and Terminating Execution Model for Constraint Handling Rules

2010-02  Ulrich Kreher, Manfred Reichert  
Speichereffiziente Repräsentation instanzspezifischer Änderungen in Prozess-Management-Systemen

2010-03  Patrick Frey  
Case Study: Engine Control Application

2010-04  Matthias Lohrmann und Manfred Reichert  
Basic Considerations on Business Process Quality

2010-05  HA Kestler, H Binder, B Lausen, H-P Klenk, M Schmid, F Leisch (eds):  
Statistical Computing 2010 - Abstracts der 42. Arbeitstagung

2010-06  Vera Künzle, Barbara Weber, Manfred Reichert  
Object-aware Business Processes: Properties, Requirements, Existing Approaches
2011-01 Stephan Buchwald, Thomas Bauer, Manfred Reichert
Flexibilisierung Service-orientierter Architekturen

2011-02 Johannes Hanika, Holger Dammertz, Hendrik Lensch
Edge-Optimized À-Trous Wavelets for Local Contrast Enhancement with Robust Denoising

2011-03 Stefanie Kaiser, Manfred Reichert
Datenflussvarianten in Prozessmodellen: Szenarien, Herausforderungen, Ansätze

2011-04 Hans A. Kestler, Harald Binder, Matthias Schmid, Friedrich Leisch, Johann M. Kraus (eds):

2011-05 Vera Künzle, Manfred Reichert
PHILharmonicFlows: Research and Design Methodology

2011-06 David Knuplesch, Manfred Reichert
Ensuring Business Process Compliance Along the Process Life Cycle

2011-07 Marcel Dausend
Towards a UML Profile on Formal Semantics for Modeling Multimodal Interactive Systems

2011-08 Dominik Gessenharter
Model-Driven Software Development with ACTIVECHARTS - A Case Study

2012-01 Andreas Steigmiller, Thorsten Liebig, Birte Glimm
Extended Caching, Backjumping and Merging for Expressive Description Logics

2012-02 Hans A. Kestler, Harald Binder, Matthias Schmid, Johann M. Kraus (eds):
Statistical Computing 2012 - Abstracts der 44. Arbeitstagung

2012-03 Felix Schüssel, Frank Honol, Michael Weber
Influencing Factors on Multimodal Interaction at Selection Tasks

2012-04 Jens Kolb, Paul Hübner, Manfred Reichert
Model-Driven User Interface Generation and Adaption in Process-Aware Information Systems

2012-05 Matthias Lohrmann, Manfred Reichert
Formalizing Concepts for Efficacy-aware Business Process Modeling*

2012-06 David Knuplesch, Rüdiger Pryss, Manfred Reichert
A Formal Framework for Data-Aware Process Interaction Models