# Automatically Generating and Updating User Interface Components 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 need to be able to quickly adapt these process implementations. Current PAISs only provide mechanisms to evolve the schema of a process, but do not allow for support the automated creation and adaptation of user interfaces (UIs). The latter may have a complex logic and comprise conditional elements or database queries. Creating and evolving UIs manually is a tedious and error-prone task. This paper introduces a set of patterns for transforming fragments of a business process, whose activities are performed by the same user role, to UIs 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 UIs 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 UIs in PAISs.

## 1 Introduction

*Process-aware information systems (PAISs)* separate process execution from application code. Hence, a separation of concerns 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 process execution are excluded. Usually, respective process models cover the users' activities at a fine-grained level (cf. Fig. 1a). Hence, before deploying such a process model in a PAIS, it must be revised and customized. For example, several human tasks, forming a process fragment in the process model, may be combined into one activity in the executable process model (cf. Fig. 1b). This activity is then implemented by a *user interface (UI) component* in the PAIS, e.g., a user form whose logic corresponds to the one of the initial process fragment (cf. Fig. 1c). Based on this logic, for example, form elements may be disabled when selecting a certain check box, or web services may be called in the background. Overall,

both the implementation and maintenance of the UI components in a PAIS is a cumbersome and costly task. This hinders quick adaptations of process implementations [1].
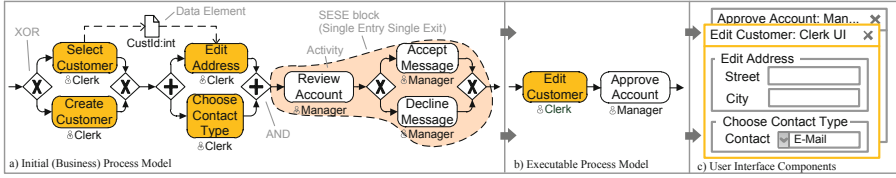


**Fig. 1.** Deriving UI Components from a Business Process Model

The process evolution of the processes implemented in a PAIS is a critical success factor for any company. Such an evolution requires changes of the process models and their associated UI components. Process model evolution is a well-understood feature in modern PAISs [2,3]. There exist editors for defining simple UI components of the PAIS (e.g., moving or renaming input fields in a UI). However, complex changes of the logic of UIs can not be done by users, but require process implementers. Moreover, the automatic propagation of changes made in the UI components to the process model and vice versa is not supported. We address these issues through the automatic generation of UI components out of process fragments, and present patterns for transforming process fragments to UI components. While *elementary transformation patterns (ETP)* transform single activities 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 process fragment. Next, we provide an advanced transformation method that allows generating UI components out of a process model based on the user roles assigned to activities. This method allows propagating changes of UI components to the process model and vice versa. Our transformation method decreases the effort for evolving PAIS 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 presents a method for transforming process models to UI components, which is based on transformation patterns and role-based process views. Section 5 discusses related work and Section 6 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 loops. Edges between activities and/or gateways represent precedence relations, i.e., the *control*

*flow* of the process model (cf. Fig. 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*. 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 by an edge from this data element to the activity. We presume that process models are *well-structured* [4], 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 (cf. Fig. 1a), may be nested, but are not allowed to overlap.

## 3  User Interface Transformation Patterns

The goal of our research is to *identify and apply a general set of UI transformation patterns to map process fragments to UI components*. To achieve this goal, we first describe a three-step method, which we apply for identifying UI transformation patterns (cf. Fig. 2). *Step 1* analyzes and evaluates PAIS projects in which we were involved. More precisely, we analyze the process models from these projects as well as their technical implementation. *Step 2* analyzes existing PAISs and their support for UI generation. *Step 3* scans related literature. The empirical results are used to specify general UI transformation patterns. Based on these patterns, *Step 4* develops a transformation method for the automatic generation of role-specific UI components (cf. Section 4).



**Fig. 2.** Transformation Pattern Identification Method

### 3.1  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.

**ETP1 (Human Activity Transformation)** transforms 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.

**Table 1.** ETP1: Human Activity Transformation

| ETP1: Human Activity Transformation | |
|---|---|
| **Description:** | 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. |
| **Example:** | A clerk must perform an activity, in which customer data is edited. |
| |  |
| **Problem:** | To perform a human activity within a PAIS, a user interaction is required. |
| **Implementation:** | 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. |

**ETP2 (Service Activity Transformation)**[1] creates application stubs for automated tasks not performed by a user (e.g., fetching data from a database). ETP2 is needed to generate complete UI components enabling interactions with both users and backend systems (cf. Section 3.2).

**ETP3 (Data Flow Transformation)**[1] transforms data elements and data flow edges to UI elements. *ETP3* as well as related patterns *ETP3.1-ETP3.3* 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 FEs of a UI component. In this context, sub-patterns *ETP3.1* and *ETP3.2* are applied to indicate whether the FE is read-only or editable. Finally, *ETP3.3* transforms the type of a data element to a specific FE; e.g., a boolean data element is transformed to a radio button element with two choices.
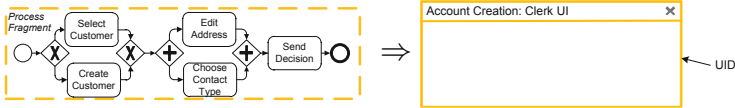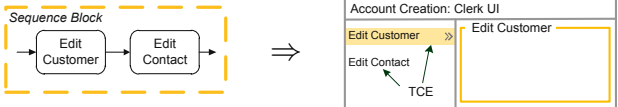
## 3.2   Complex Transformation Patterns

Complex Transformation Patterns (CTPs) allow transforming entire fragments of a process model, whose activities shall be performed by the same user, to UI components. When creating such a UI component both the control and data flow of the process fragment are considered. Hence, each generated UI component covers parts of the overall process logic. 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 used. A CTP interconnects TCEs according to the control flow of the process fragment to which it is applied. Single activities and data elements related to the process fragment are transformed using ETPs.

**CTP1 (Process Model Transformation).** generates a *User Interface Dialog (UID)* for process fragments whose activities are processed by the same user role (cf. Table 2). A UID is a toplevel container, which is represented by a dialog window in the PAIS containing UI elements representing activities and data elements.

---

[1] A more detailed description of all ETPs can be found in [5].

**Table 2.** Pattern Descriptions for Patterns CTP1 and CTP2

| **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. |

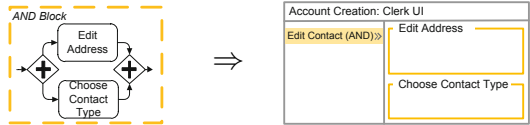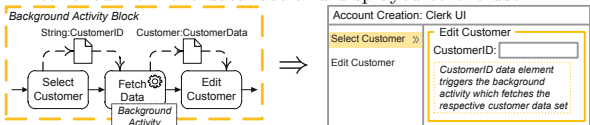| **CTP2: Sequence Block Transformation** | |
|---|---|
| **Description:** | A sequence of activities (and SESE blocks) is transformed into a sequence of *Tab Container Elements (TCE)* to be processed in the same sequential order. |
| **Example:** | A clerk first edits the customer data and then the 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 are processed relates to the one of the respective activities. |

**CTP2 (Sequence Block Transformation)** deals with the transformation of a sequence of activities (and SESE blocks respectively) to TCEs (cf. Table 2). 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.

**CTP3 (Parallel Block Transformation)** transforms parallel activities (or SESE blocks) of a process fragment to UI elements within the same UID. These elements may then be accessed concurrently (cf. Table 3). The UI component is similar to the one of a single activity (cf. ETP1). However, CTP3 not only covers the transformation of activities arranged in parallel, but enables the concurrent processing of SESE blocks arranged in parallel to the respective UI elements.

**CTP4 (XOR Block Transformation)**[2] transforms an XOR branching of a process fragment to a UI component. 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; e.g., whether the TCE element for creating a new customer or the one for editing an existing customer shall

---

[2] A more detailed description of this pattern can be found in [5].

**Table 3.** Pattern Descriptions for Patterns CTP3 and CTP6

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

| CTP6: Background Activity Transformation | |
|---|---|
| **Description:** | While human activities are performed by human resources, service activities may automatically fetch or save data concurrently in the background. |
| **Example:** | 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. |
| |  |
| **Problem:** | 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. |
| **Implementation:** | Dynamic forms, which contain background activities, need a change listener mechanism to detect user inputs and to react on them. |

be displayed. In particular, run-time data for deciding which branch of an XOR branching shall be executed is required.

**CTP5 (Loop Block Transformation)**[2] transforms a loop block to elements of a UI component. For each loop, CTP5 generates a TCE and corresponding UI elements for nested activities or SESE blocks (cf. CTP1). Additionally, a decision element is required to decide whether to exit the loop after completion of a particular iteration or trigger the next loop iteration either based on data elements processed during loop execution (e.g., evaluating data for validity) or external criteria (e.g., calling someone until getting an answer).

**CTP6 (Background Activity Transformation)** reflects the need for dynamically loading data elements by a service activity (cf. Table 3). More precisely, data has to be fetched from or stored to a backend system, while the user concurrently works on human activities.

# 4  Transforming Process Models to User Interfaces

Section 4.1 shows how the presented patterns are used to transform process fragments to UI components of the PAIS. Further, we 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 process model, consisting of several process fragments, to multiple UI components, we introduce a five step method (cf. Fig. 3). Thereby, the number of generated UI components depends on the number of different user roles involved in the process.
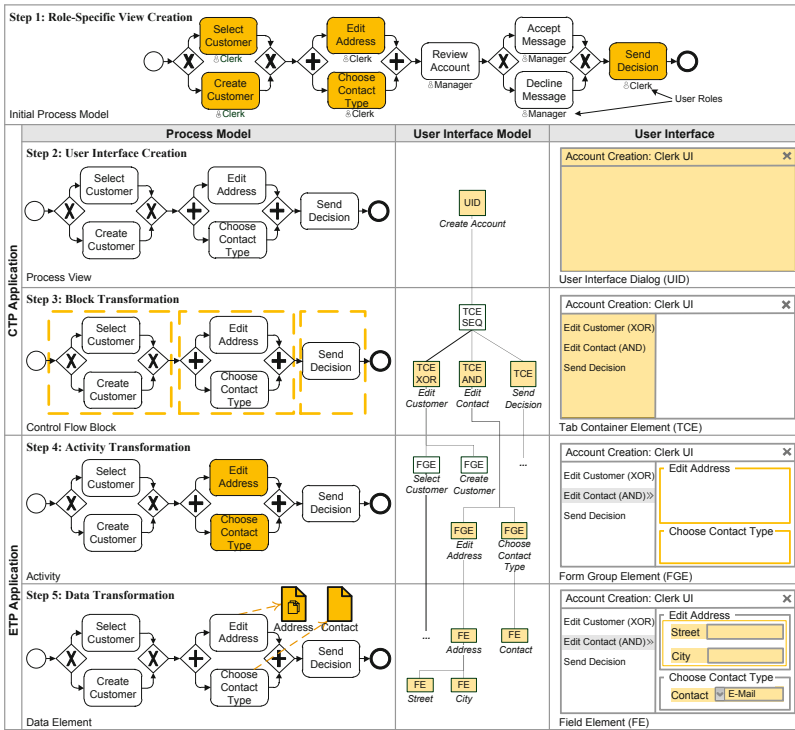


**Fig. 3.** Transformation Method

**Step 1.** *Role-specific process views* [6,7,8] are created for the given process model. A process view abstracts from certain aspects of the process model (e.g., it only contains activities of a particular user role). 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. A UID acts as a toplevel container including all UI elements required for processing the activities of a process view. For this purpose, CTP1 is applied (cf. Table 2).

**Step 3.** CTP2-6 are applied to transform complete process fragments to UI elements. For each CTP applied, a *Tab Container Element (TCE)* is generated. Each TCE is represented in the tab bar area (cf. Fig. 3, Step 3). When clicking such an item, 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. Fig. 3, Step 4).

**Step 5.** Data elements of the process view are transformed to *Field Elements (FE)* and positioned within an FGE. There exist different kinds of FEs depending on the data type of the respective data element (cf. pattern ETP3.3 in [5]).

The internal structure of the resulting UI component is represented through a *User Interface Model (UIM)* (cf. Fig. 4b). This tree-based schema describes the UI structure generated by our transformation method.

### 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. 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 Fig. 4a. Such a change would not affect the execution order of the activities in the process view, i.e., it only affects the visual representation of the UI component. Hence, the change needs not be propagated to the view. *Global changes* modify the logic of the UI and the control flow of associated process views as well (e.g., adding FGE *Edit Phone Number* and respective FE *Phone*, cf. Fig. 4a). The correct position of the change within the UIM can be determined by the hierarchical structure of the UI (cf. Fig. 4b). The changes of the UIM are then propagated to the 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*[3] project can be applied [9,10].

## 5  Related Work

**Task Models** describe the actions to be performed by a user when interacting with an information system. Different variants of task models exist [11]. These approaches describe the goals, steps and operations of a UI. *Concurrent Task Trees (CTT)*, in turn, provide a hierarchical model supporting various types of tasks (e.g., automatic vs. manual task) and relationships between tasks (e.g., sequential vs. parallel execution) [12].

---
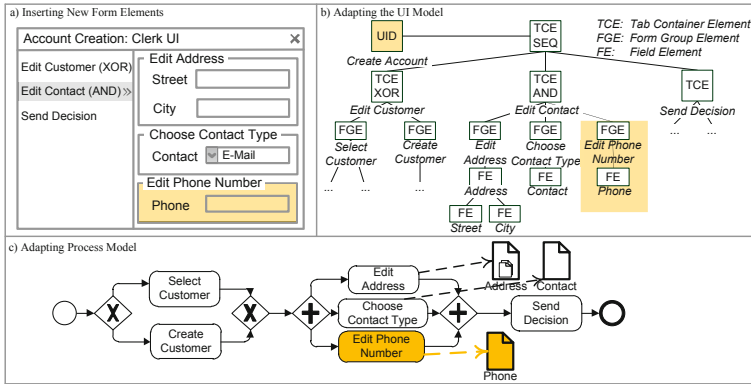
[3] `http://www.dbis.info/proView`

**Fig. 4.** Adapting Process Models through Changes in the UI Component

**Model-Driven UI Development** applies the principles of *Model-Driven Development* to UI development. Although a lot of competing approaches exist, an accepted standard is missing [13,14]. FlowiXML [15], for example, provides a methodology to develop UIs for business processes, taking the organizational structure as well as the process model into account. However, it does not allow for the automated generation of UIs. Based on FlowiXML, [16] describes user tasks through task models (i.e., CTT) within a process model. Based on these models, an abstract UI description is generated and transformed into a UI component at run-time. This approach allows for changes based on UIs and discusses how to manually align them with process models. However, automatic propagation is not supported. [17] transforms a process model into a human interaction perspective, which allows specifying data elements, user roles, tasks, and UI layout. After manually refining them, corresponding UIs are generated during run-time. Furthermore, data-centered process management offers a different (i.e., data-centered) view on processes. In particular, state transitions of process-related data elements are described. Based on this, UIs can be generated as well [18].

**UI Generation in Existing PAIS** is able to create UIs automatically (e.g., IBM Lombardi [19]). Single activities of a process model can be transformed into simple UIs, taking associated data elements into account (cf. ETPs). More complex scenarios are not covered. None of the approaches allows for the automatic generation of complex UIs based on process models. The adaptation of process models based on changes of the UI is only considered rudimentarily.

## 6    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, was 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 [5]. In summary, our approach will contribute to reduce costs for PAIS development and maintenance.

Future research 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 process instance.

## References

1. Pradeep, H.: Process-User Interface Alignment: New Value From a New Level of Alignment. Align Journal (October 3, 2007)
2. Weber, B., Reichert, M., Mendling, J., Reijers, H.A.: Refactoring Large Process Model Repositories. Computers in Industry 62(5), 467–486 (2011)
3. Reichert, M., Dadam, P.: ADEPTflex - Supporting Dynamic Changes of Workflows Without Losing Control. Journal of Intelligent Inf. Sys. 10(2), 93–129 (1998)
4. La Rosa, M., Wohed, P., Mendling, J., ter Hofstede, A.H.M., Reijers, H.A., van der Aalst, W.M.P.: Managing Process Model Complexity Via Abstract Syntax Modifications. IEEE Transactions on Industrial Informatics 7(4), 614–629 (2011)
5. Kolb, J., Hübner, P., Reichert, M.: Model-Driven User Interface Generation and Adaptation in Process-Aware Information Systems. Technical report, UIB 2012-04, Ulm University (2012)
6. Reichert, M., Kolb, J., Bobrik, R., Bauer, T.: Enabling Personalized Visualization of Large Business Processes through Parameterizable Views. In: Proc. ACM SAC 2012, Riva del Garda (Trento), Italy (2012)
7. Kolb, J., Reichert, M., Weber, B.: Using Concurrent Task Trees for Stakeholder-centered Modeling and Visualization of Business Processes. In: Oppl, S., Fleischmann, A. (eds.) S-BPM ONE 2012. CCIS, vol. 284, pp. 237–251. Springer, Heidelberg (2012)
8. Bobrik, R., Reichert, M., Bauer, T.: View-Based Process Visualization. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 88–95. Springer, Heidelberg (2007)
9. Kolb, J., Kammerer, K., Reichert, M.: Updatable Process Views for User-centered Adaption of Large Process Models. In: Proc. Intl. Conf. on Service Oriented Computing (ICSOC 2012), Shanghai, China (to appear, 2012)
10. Kolb, J., Kammerer, K., Reichert, M.: Updatable Process Views for Adapting Large Process Models: The proView Demonstrator. In: Proc. of the Business Process Management 2012 Demonstration Track, Tallinn, Estonia (to appear, 2012)
11. Limbourg, Q., Vanderdonckt, J.: Comparing Task Models for User Interface Design. The Handbook of Task Analysis for Human-Computer Interaction 6 (2004)
12. Paternò, F., Mancini, C., Meniconi, S., Maria, V.S.: ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. In: Proc. IFIP TC13 Int'l Conf. on Human-Computer Interaction, pp. 362–369 (1997)
13. Traetteberg, H., Molina, P.J.: Making Model-Based UI Design Practical: Usable and Open Methods and Tools. In: Proc. IUI 2004, pp. 376–377 (2004)
14. Lu, X.: Model Driven Development of Complex User Interface. In: Proc. MoDELS 2007, Workshop on Model Driven Development of Advanced User Interfaces (2007)

15. Garcia, J.G., Vanderdonckt, J., Calleros, J.M.G.: FlowiXML: A Step Towards Designing Workflow Management Systems. Int'l Journal of Web Engineering and Technology 4(2), 163–182 (2008)
16. Sousa, K., Mendonça, H., Vanderdonckt, J., Rogier, E., Vandermeulen, J.: User Interface Derivation from Business Processes: A Model-Driven Approach for Organizational Engineering. In: Proc. ACM SAC 2008, pp. 553–560 (2008)
17. Sukaviriya, N., Sinha, V., Ramachandra, T., Mani, S., Stolze, M.: User-Centered Design and Business Process Modeling: Cross Road in Rapid Prototyping Tools. In: Baranauskas, C., Abascal, J., Barbosa, S.D.J. (eds.) INTERACT 2007. LNCS, vol. 4662, pp. 165–178. Springer, Heidelberg (2007)
18. Künzle, V., Reichert, M.: PHILharmonicFlows: Towards A Framework for Object-Aware Process Management. Journal Software Maintenance and Evolution: Research & Practice 23(4), 205–244 (2011)
19. Yang, S., Sun, Y., Waterhouse, J., Lau, D., Al-Hamwy, T.: Modeling and Implementing a Business Process Using WebSphere Lombardi Edition 7.1. In: Proc. CASCON 2010, pp. 374–375 (2010)