# User Interface Derivation Based on Role-Enriched Business Process Model

Lei Han, Weiliang Zhao(✉), and Jian Yang

Macquarie University, North Ryde, NSW 2109, Australia
lei.han1@students.mq.edu.au, {weiliang.zhao,jian.yang}@mq.edu.au

**Abstract.** This work proposes an approach for User Interface (UI) derivation based on a role-enriched Business Process (BP) model with the capability to describe the details of the control flow and data operations in a BP. For each user role, data relationships are extracted according to the identified control flow patterns and data operation patterns. A set of mandatory and recommended UI derivation rules are specified as the cornerstones to derive the UI logic from a BP. The algorithm for UI derivation is provided. This UI derivation approach provides the basis for UI development and maintenance.

**Keywords:** User interface · Business process · Control flow pattern

## 1 Introduction

A business process is a collection of linked tasks that provides services. Each task is a unit of work performed by human users or applications. Users participate a BP through User Interfaces. Generally speaking the UI development for the BP needs a lot of hard coding efforts which normally constitute 70 % to 80 % of the manually written codes of a BP implementation (e.g. the interaction between BP and database/application systems/services). Because BPs and their UIs are tightly coupled, the existing UIs can not easily adapt to the BP changes without recoding. The realization and maintenance of UIs are often costly and effort-consuming, which impedes the quick adaptations of BP realization [3,6].

As a scenario example, Fig. 1(a) shows a recruitment process at the human resource department of a company. This process is represented with Business Process Model and Notation (BPMN) [9]. Three user roles as `personnel officer`, `applicant`, and `referee` are involved. In the recruitment process, there are tasks as: (1) the `personnel officer` announces a job vacancy; (2) an `applicant` lodges his application; (3) a `referee` writes a reference letter to support the application; (4) the `personnel officer` arranges an interview for the `applicant`; (5) the `applicant` confirms the interview; (6) the `personnel officer` reviews the reference letter; (7) the `personnel officer` conducts the interview; (8) the `personnel officer` makes the decision according to the evaluation of the reference letter and the interview report. Task 4, 5 are in parallel with task 6.

Q.Z. Sheng et al. (Eds.): ICSOC 2016, LNCS 9936, pp. 450–464, 2016.
DOI: 10.1007/978-3-319-46295-0_28

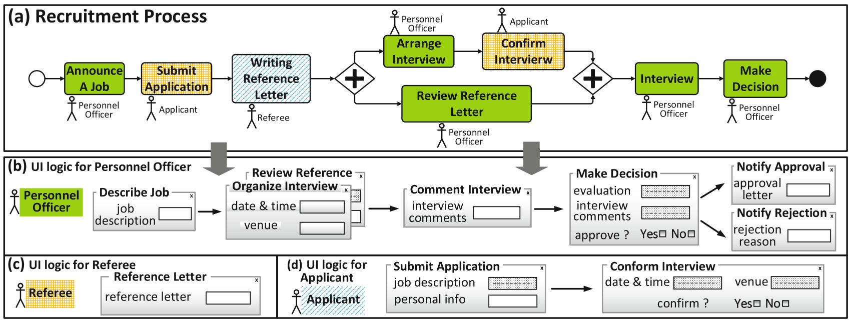

**Fig. 1.** Deriving UI logics from a business process

Users provide inputs to and retrieve information from a BP through UIs. The operation flow of input/output data for a specific user role in a BP is referred to as the UI logic of this user role. Input/output data will be grouped and put in different UI containers. For instance, Fig. 1(d) shows the UI logic for the user role `applicant`. There are two UI containers (`Submit Application` and `Confirm Interview`) and they will turn up in a sequential order. With `Submit Application`, an `applicant` can read the job information and provide the details of his application. With `Confirm Interview`), the `applicant` can check the interview date, time, location and confirm his attendance.

There are several works of UI derivation from BP models. The UI derivation approach provided in [5,6] is based on a BPMN process model, in which the detailed data operations inside tasks have not been covered. The derived UI logic cannot reflect UI requirements originated from detailed data operations in individual tasks of a BP. [14] proposes an artifact-centric BP model to derive the UI logic; PHILharmonicFlows [7,8] derives the UI logic based on an object-aware BP model. These BP models are lacking of the capability to describe the relations between a user role and the control flow of tasks in a BP. The derived UI logic by these approaches can not differentiate the UI logics of different user roles.

We believe the UI logic of a BP should have the following features: (1) each participating user role should have a UI logic; (2) each UI logic consists of a set of containers and the execution constraints of these containers; (3) each container includes a set of data items specified with access types (*read*, *write*). To support UI derivation, we propose a role-enriched business process model with the capability to specify complex control flow patterns and capture the details of data operations inside individual tasks. Based on this role-enrich BP model, the UI derivation steps are as follows:

1. The BP is abstracted and aggregated for each user role based on the role-enrich BP model. For each user role, the tasks related to this user role are reserved; the tasks not related to this user role are abstracted.

2. Data relationships are extracted from the abstracted and aggregated business process (AABP) for each user role. A set of elementary operations are developed according to the data operations inside individual tasks and the identified control flow patterns in the AABP. The extracted data relationships are the foundation to analyze and derive the UI logic.
3. The UI logic are derived from the extracted data relationships. A set of mandatory and recommended rules are specified. The UI logic is derived by using the UI derivation algorithm with these rules as cornerstones.

The remainder of the paper is organized as follows. Section 2 proposes an role-enriched BP model and identifies a set of control flow patterns and data operation patterns. Section 3 introduces BP abstraction and aggregation for each involved user role. Section 4 discusses how to extract data relationships. Section 5 presents a set of UI derivation rules and the algorithm for the UI derivation. Section 6 reviews existing works. Section 7 gives a conclusion.

## 2 Role-Enriched BP Model

Beyond existing works, the proposed BP model has the capability to specify: (1) how user roles are involved in tasks; (2) how data are operated in individual tasks; (3) how complex control flow patterns affect data relationships. This section provides the syntax of the proposed Role-enriched BP model. A set of control flow patterns and data operation patterns are specified with an extended Business Process Model and Notation.

### 2.1 Formal Syntax

We propose a role-enriched BP model to specify a BP with a set of tasks and control flow relations between these tasks. The participating user roles are labeled in individual tasks. The data operation flow in each task specifies a set of data items and the operation flow relations between these data items. Each data item has an access type as ((*read* or *write*)).

**Definition 1: Data Operation Flow.** A data operation flow is denoted as a tuple $df = (N_d, type_d, type_A, SF_d^{fix}, SF_d^{free})$, where:

- $N_d = \{e_d^s, e_d^e\} \cup G_d \cup A$ where $e_d^s$ and $e_d^e$ are the start event and end event respectively; $G_d = G_d^{in} \cup G_d^{out}$, $G_d^{in}$ is a finite set of entry gateways and $G_d^{out}$ is a finite set of exit gateways; $A$ is a finite set of data items.
- $type_d$: $G_d \rightarrow \{Sequential, Parallel\text{-}A, Parallel\text{-}B, Conditional, Loop\}$ is a mapping function to give each gateway a type. These gateways are elements that control how the data flow diverges or converges.
- $type_A : A \rightarrow \{read, write\}$ is a mapping function to specify the access type of each data item $a \in A$. There are two kinds of access types as *read* and *write*.
- $SF_d^{fix}$ and $SF_d^{free}$ represent fixed-order sequence flow and free-order sequence flow respectively.

Note that fixed-order sequence flow means that the sequence of involved entities must be in a fixed order; free-order sequence flow means that the order in the sequence of involved entities is free.

**Definition 2: Role-enriched Business Process Model.** A role-enriched BP model is denoted as $rm = (N_t, type_t, SF_t^{fix}, SF_t^{free}, refine, R, \rho)$, where:

- $N_t = \{e_t^s, e_t^e\} \cup G_t \cup T$ where $e_t^s$ and $e_t^s$ are start event and end event respectively; $G_t = G_t^{in} \cup G_t^{out}$, $G_t^{in}$ is a finite set of entry gateways and $G_t^{out}$ is a finite set of exit gateways; $T$ is a finite set of tasks.
- $type_t$: $G_t \rightarrow$\{*Sequential*, *Parallel-A*, *Parallel-B*, *Conditional*, *Loop*\} is a mapping function to give each gateway a type. These gateways are elements that control how the task flow diverges or converges.
- $SF_t^{fix}$ and $SF_t^{free}$ represent fixed-order sequence flow and free-order sequence flow respectively.
- $refine : T \rightarrow DF$ is a refinement function on tasks. $DF = \{df_1, df_2, ..., df_n\}$ stands for a set of data operation flows. The refinement function links tasks and their corresponding data operation flows.
- $R$ is a finite set of user roles.
- $\rho = T \times R$ specifies relationships between user roles and tasks.

## 2.2 Control Flow Patterns of Role-Enriched BP Model

This sub section describes the identified control flow patterns. These patterns are the basis of building up elementary operations for data relationship extraction later. The BPMN [9] is extended to specify: (1) relationships between tasks and user roles; and (2) complex control flow patterns and data operation patterns. Figure 2 shows the graphical notations with details as:

- A solid line with an arrow denotes the fixed-order sequence flow.
- A dashed line denotes the free-order sequence flow.
- The upper label of a task denotes its user roles.
- The upper label of a data item denotes its access type.
- A bar on the left/right side of a circle denotes an entrance/exit gateway.
- *Sequential*, *Parallel-A*, *Parallel*$-$*B*, *Conditional*, and *Loop* denote the gateway types.

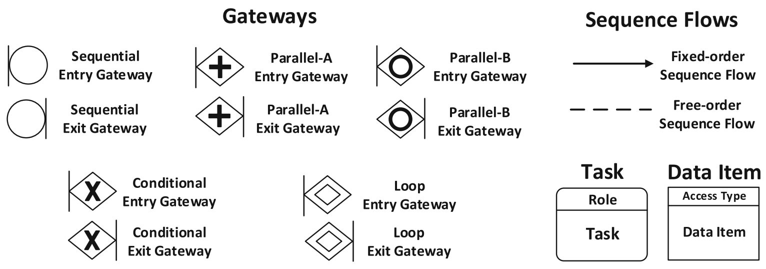

**Fig. 2.** Extended BPMN

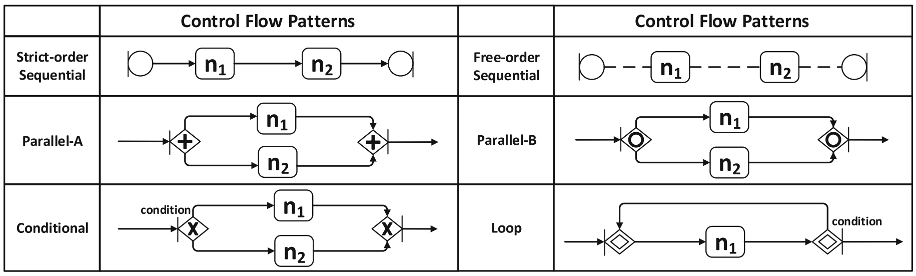

**Fig. 3.** Control flow patterns

Figure 3 shows the identified set of control flow patterns as:

– **Strict-order Sequential** specifies that nodes must be executed in a strict sequential order.
– **Free-order Sequential** specifies that nodes must be executed one after another but the order is free.
– **Parallel-A** specifies that all branches must be executed in parallel and the pattern completes when all branches have completed.
– **Parallel-B** specifies that all branches must be executed in parallel and the pattern completes when any branch has completed.
– **Conditional** specifies that the executed branch is decided according to the runtime condition.
– **Loop** specifies that nodes in the loop must be executed iteratively until the "jumping-out condition" is met.

In **Strict-order Sequential**, **Free-order Sequential**, **Conditional**, and **Loop**, each node $n_i$ $(i = 1, 2)$ can be either a task or a data item inside a task. In **Parallel-A** and **Parallel-B**, each node $n_i$ $(i = 1, 2)$ must be a task. When considering relationships between/among data items, each branch of the **Parallel-A** block must be completed by the same user role, and this user role

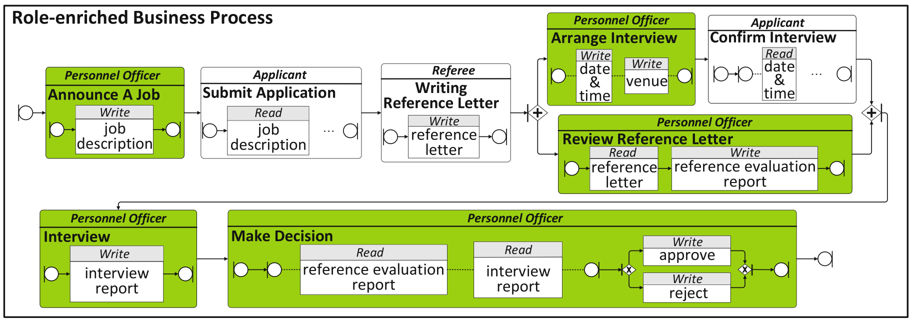

**Fig. 4.** Recruitment process specified with role-enriched BP model

must execute these branches one by one in a free order. One and only one branch of the **Parallel-B** block must be completed. When considering data operation patterns, **Parallel-A** equals to **Free-order Sequential**; **Parallel-B** equals to **Conditional**. **Parallel-A** or **Parallel-B** will never turn up in the data operation patterns.

Figure 4 illustrates how the role-enriched BP model specifies business process for the recruitment scenario example. Here, we only highlight details of data operation flows of the tasks participated by the user role `personnel officer`.

## 3 BP Abstraction and Aggregation

Different user roles have different UI logics. The role-enriched BP is abstracted and aggregated for each user role. The BP abstraction and aggregation are processed based on a set of identified control flow patterns. An abstracted and aggregated business process (AABP) for a particular user role contains tasks all related to this user role, abstracted nodes, as well as the control flow relations between the tasks and the abstracted nodes. Inside each task, the operated data items and their execution orders are specified. This part of work is relatively independent and its details will be provided in a separate paper.

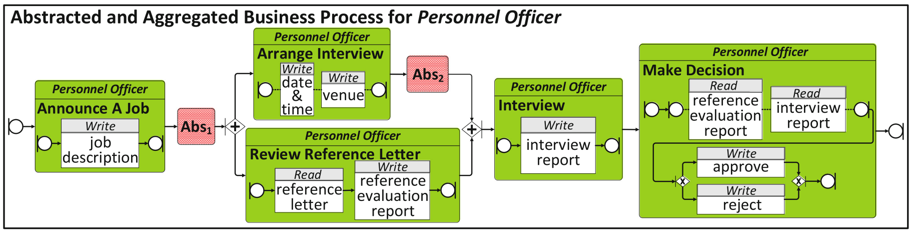

**Fig. 5.** AABP for personnel officer

Here we use the recruitment scenario example to show the result of BP abstraction and aggregation. Figure 5 shows the AABP for the user role `personnel officer`. The tasks for the `personnel officer` are all kept, and the tasks for the `applicant/referee` are abstracted (the tasks `Submit Application` and `Writing Reference Letter` are abstracted as $Abs_1$; the task `Confirm Interview` is abstracted as $Abs_2$).

## 4 Data Relationship Extraction

Data relationships are extracted and specified as a tree graph from an AABP. A series of elementary operations are specified on identified control flow patterns and data operation patterns of the AABP. The algorithm for data relationship extraction is built up on these elementary operations.

## 4.1 Data Relationships

After the AABP is generated for a user role, the data relationships, including operated data and data operation flow, are extracted from the AABP for this specific user role. A tree graph represents these extracted data relationships and it can be recorded with a JSON String. Figure 6 shows a tree graph representing the data relationships extracted from the AABP for `personnel officer`.

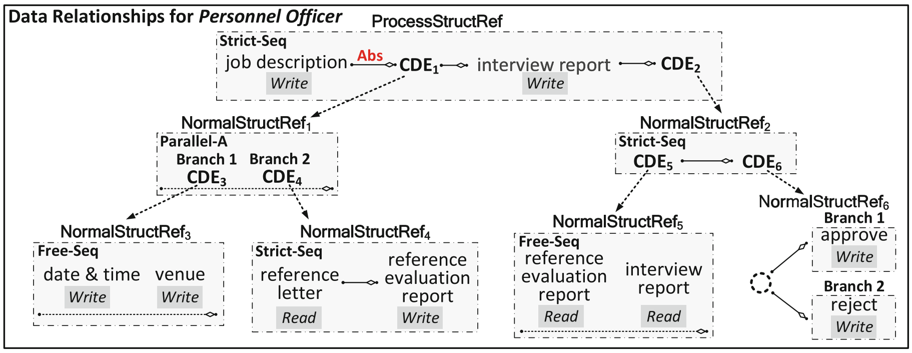

**Fig. 6.** Data relationships extracted from the AABP for personnel officer

In a tree graph, there are four types of nodes: (1) `Attribute Node` is represented by "Attr". It refers to a data item operated inside a single task of an AABP. (2) `Information Node` is represented by "INFO". It contains a piece of information for a specific user role. (3) `CDE Node` is represented by "CDE". It refers to a fragment of the tree graph. (4) `Virtual Node` is represented by a dashed circle. It starts a conditional data relationship pattern. For example, *approve* and *reject* in Fig. 6 are `Attribute Nodes` that follow a `Virtual Node`.

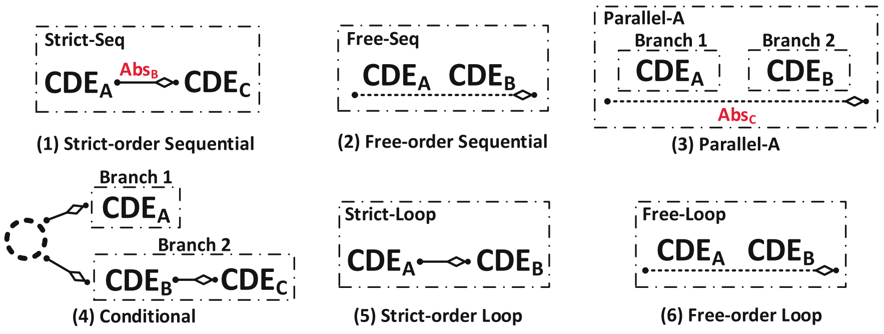

**Fig. 7.** Data relationship patterns

There are six data relationship patterns: **Strict-order Sequential**, **Free-order Sequential**, **Parallel-A**, **Conditional**, **Strict-Order Loop**, and **Free-Order Loop**. Figure 7 shows examples of these patterns with `CDE Nodes`. Except **Conditional**, a pattern is represented with a dot-dash rectangle, a pattern name, and node relationships. **Conditional** is represented with a `Virtual Node` and a set of branches with node relationships.

An `Abs Label` is used to represent control flow relations between tasks and abstracted nodes in the AABP. In a tree graph, an `Abs Label` only turns up above a solid edge (see $Abs_B$ in (1) of Fig. 7) or under a dashed edge (see $Abs_C$ in (3) of Fig. 7). The `ProcessStructRef` is the root of a tree graph. The `NormalStructRef` is the reference of a tree graph fragment.

## 4.2 From AABP to Data Relationhips

### 4.2.1 Elementary Operations

Figure 8 illustrates a series of elementary operations for data relationship extraction. Each elementary operation extracts data relationships from a particular control flow pattern of an AABP or a particular data operation pattern inside an individual task of an AABP. A control flow relation between/among tasks is represented by a set of `CDE Nodes` and edges in a tree graph. An abstracted node in an AABP is removed away totally or recorded by an `Abs Label` in a tree graph. A data operation relation between/among data items is represented by a set of `Attribute Nodes` and edges in a tree graph.

The **Conditional-Data-Deriv-1** in Fig. 8 shows how an elementary operation generates data relationships from a **Conditional** control flow pattern in an AABP. Each branch of the **Conditional** control flow pattern is inherited by the corresponding data relationship pattern. For a branch, there are rules: (1) If it is composed of only one abstracted node ($AbsNode_A$), an `Information Node` ($INFO_A$) is used to represent the execution status of this branch. (2) If a branch is composed of only tasks, `CDE Nodes` ($CDE_B$ and $CDE_C$) are used to represent these tasks ($Task_B$ and $Task_C$). (3) If a branch is composed of both tasks and abstracted nodes, a `CDE Node` is used to represent a task; an `Abs Label`($Abs_E$) is used to represent an abstracted node ($AbsNode_E$) between tasks; and an abstracted node ($AbsNode_G$) at one end of a branch is removed away.

### 4.2.2 Algorithm for Data Relationship Extraction

In order to extract the data relationships from an AABP, the control flow patterns of the AABP and the data operation patterns inside each individual task are identified and elementary operations are applied accordingly. A tree graph representing the extracted data relationships is built up by using **Algorithm** 1.

As an example, according to elementary operations **Sequential-Data-Deriv-1** and **Parallel-A-Data-Deriv-3** in Fig. 8, $Abs_1$ in Fig. 5 is inherited by label $Abs$ in Fig. 6, and $Abs_2$ in Fig. 5 does not appear in Fig. 6.

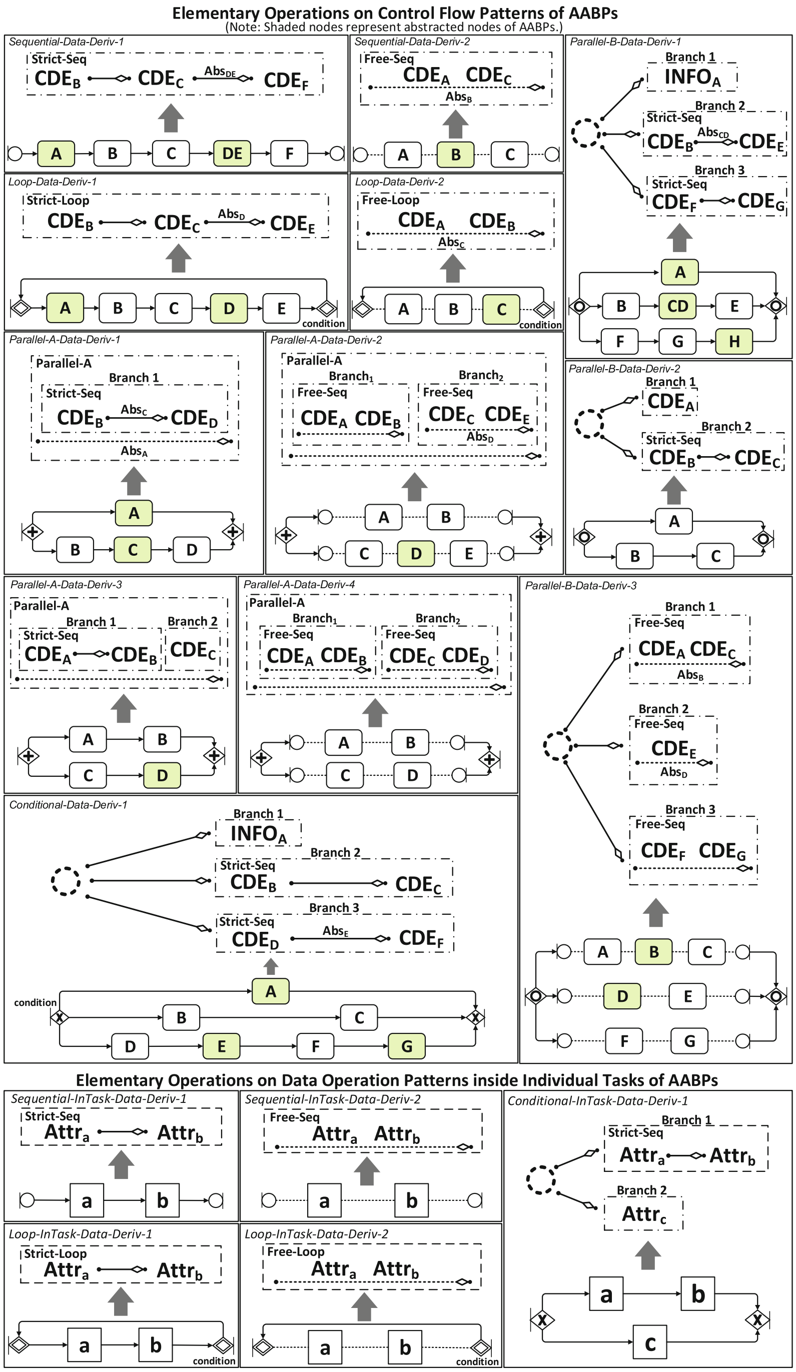

**Fig. 8.** Elementary operations for data relationship extraction from AABPs

**Algorithm 1.** Data Relationship Extraction

```
Function DeriveDataRelationships(AbsAggBusinessProcess $aabp$)
    identify $cfPattern$ and $cfElementSet$ at coarsest granularity level of $aabp$;
    $TreeGraph = \emptyset$;
    foreach $cfElement$ in $cfElementSet$ do
        if $cfElement$ is Task then
            get data operation flow $df\_cfElement$ from $cfElement$;
            $result$ = HandleDataOperationFlowInTask($df\_cfElement$);
            add $result$ to $TreeGraph$;
        else if $cfElement$ is ComplexStructure then
            $cfElement$ = DeriveDataRelationships($cfElement$);
    transform $cfElementSet$ to $treeFragment$ using elementary operation;
    if $cfPattern$ is on the coarsest granularity level then
        assign ProcessStructRef to $treeFragment$;
    else
        assign NormalStructRef to $treeFragment$;
    add $treeFragment$ to $TreeGraph$;
    return $TreeGraph$;
Function HandleDataOperationFlowInTask(DataOperationFlow $df$)
    $TreeFragmentSet = \emptyset$;
    identify $dfElementSet$ at coarsest granularity level of $df$;
    foreach $dfElement$ in $dfElementSet$ do
        if $dfElement$ is not DataItem then
            $dfElement$ = HandleDataOperationFlowInTask($dfElement$);
    transform $dfElementSet$ to $treeFragment'$ using elementary operation;
    assign NormalStructRef to $treeFragment'$;
    add $treeFragment'$ to $TreeFragmentSet$;
    return $TreeFragmentSet$;
```

## 5 User Interface Derivation

This section introduces the UI derivation from the extracted data relationships. The UI flow is derived for each user role involved in the BP. To derive the UI flow, a series of UI derivation rules are specified including constrains and recommendations. The algorithm for UI derivation is developed by utilizing these rules.

### 5.1 User Interface Flow

The UI flow has two granularity levels: the operation flow between UI containers, and data items included inside each UI container. Each data item needs to be specified with the **Access type** including *read* and *write*. A UI container holds the maximum amount of data items to be operated by a user role. The entire set of data items of the UI container will be shown to end users (see Fig. 10). The UI designers can divide this container into sub-containers. The UI containers can have operation flow relations as: **Strict-order Sequential**, **Free-order Sequential**, **Conditional**, **Strict-order Loop**, or **Free-order Loop**.

## 5.2 From Data Relationships to UIs

### 5.2.1 UI Derivation Rules

This sub section coins a set of rules for deriving the UI flow from the tree graph described previously. These UI derivation rules can be classified into two categories as **Constraints** and **Recommendations**. The **Constraints** include rules that must be followed by the UI designers. The **Recommendations** include rules that are recommended to be followed by the UI designers. Figure 9 describes these rules in a graphical view. The container flows are represented in the same way as data relationships. Note that all the "Node" labels in the following figures can represent either an `Attribute Node`, an `Information Node`, or a `CDE Node` in the tree graph. We have formalized each rule in Fig. 9. Due to space limitations, we only provide details of the rule **Sequential-Constraint-1** in Fig. 9 as an example.

**Sequential-Constraint-1**:
$\forall m$: $\text{“}TreeGraph[m]''.\text{“}dataRelationshipPattern'' == \text{“}Strict - Seq''$,
**iff** $\forall i, j$: $\text{“}TreeGraph[m]''.\text{“}graphNodes[i]''.\text{“}postAbsOfNode'' ==$
$\text{“}TreeGraph[m]''.\text{“}graphNodes[j]''.\text{“}preAbsOfNode''$,
**then** $(\text{“}TreeGraph[m]''.\text{“}graphNodes[i]'' \in con_i) \wedge$
$(\text{“}TreeGraph[m]''.\text{“}graphNodes[j]'' \in con_j) \wedge$
$((con_i, con_j) == \text{“}Strict - Seq'')$.

**Sequential-Constraint-1** specifies that in the **Strict-order Sequential** data relationship pattern, if there exists an `Abs Label` between two adjacent nodes, these two nodes must be separated into different containers.

All the UI generation rules are divided into two groups. *Group 1* includes rules about data relationships between inside data of a specific pattern, while *Group 2* includes rules about data relationships between inside data and outside data of a specific pattern. In Fig. 9, the rules in *Group 1* are not shaded; and the rules in *Group 2* are shaded.

### 5.2.2 Algorithm for UI Derivation

Here we provide **Algorithm** 2 to derive the UI flow from an AABP. The tree graph is obtained by using **Algorithm** 1 with the AABP as input. Then the UI flow is derived from the tree graph by applying UI derivation rules. The UI flow is represented by a set of containers and the operation flow of these containers.

Figure 10 shows the derived UI Flow for `Personnel Officer`. This UI flow contains seven containers with involved operation flow relations as **Strict-order Sequential**, **Free-order Sequential**, and **Conditional**. As an example of the details of the derivation process, data pointed by $CDE_3$ and $CDE_4$ in Fig. 6 are put into two containers `Organize Interview` and `Review Reference Letter`.

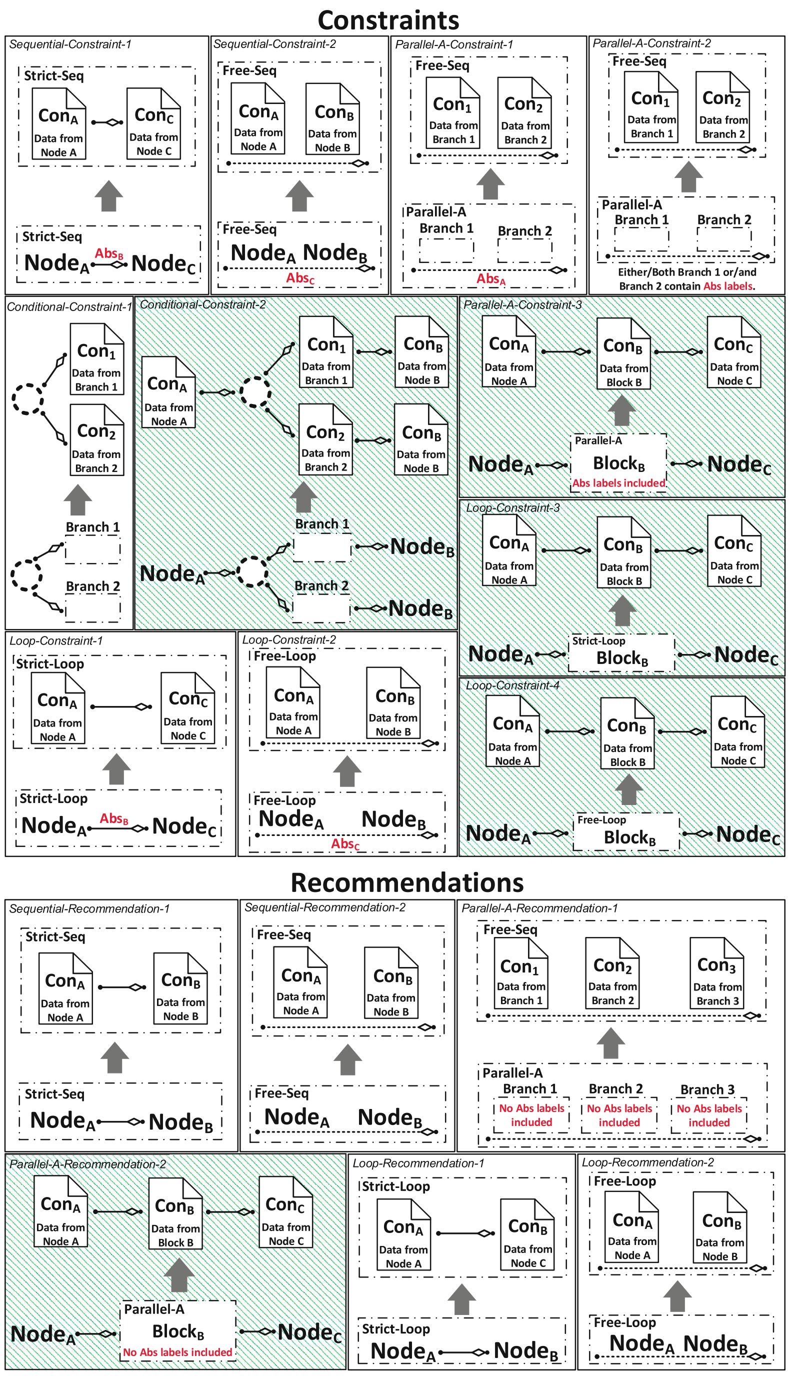

**Fig. 9.** UI derivation rules

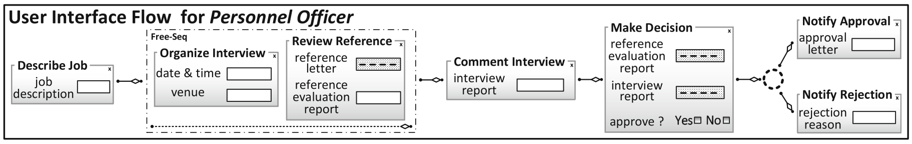

**Fig. 10.** UI flow for personnel officer

## 6 Related Work

J. Kolb et al. [5,6] propose a two-step method to generate the UI logic of a BPMN process, in which the role-specific views are derived at first, then a series of elementary and complex patterns are identified to support the derivation of the UI logic from the role-specific views. In their BP model, only four basic control flow patterns (sequential, parallel, exclusive, and loop) have been specified and the execution flows between data items inside a task of BP have not been covered. Their UI derivation rules cover less situations comparing to our rules described in Sect. 5.2.1. K. Sousa et al. [11] develop an approach to derive UIs from a business process with four steps as process modelling, task derivation, task refinement, and UI model derivation. These UI derivation methods have no capability to differentiate between constraints and recommendations.

V. Kunzle et al. in [7,8] propose an object-aware approach for BP modelling, in which the evolutions of data objects and constraints between data objects are

**Algorithm 2.** UI Derivation

```
Input  : $AABP$
Output: $UIFlow$
1  use Algorithm 1 to extract $TreeGraph$ from AABP;
2  retrieve $headTreeFragment$ with ProcessStructRef from $TreeGraph$;
3  generate $UIFlowFrag$ from $headTreeFragment$ by applying rules in Group 1;
4  set $UIFlow = UIFlowFrag$;
5  UIDerivation($headTreeFragment$);
6  process $UIFlow$ by applying rules in Group 2;
7  return $UIFlow$;
8  Function UIDerivation(TreeFragment $treeFrag$)
9      get $structRef$ from $treeFrag$;
10     if $structRef$ is not ProcessStructRef then
11         generate $UIFlowFrag$ from $treeFrag$ by applying rules in Group 1;
12         update the container holding $structRef$ with $UIFlowFrag$;
13     foreach $node$ of $treeFrag$ do
14         if $node$ is CDENode then
15             get the value $structRef'$ from $node$;
16             retrieve $treeFrag'$ with $structRef'$ from $TreeGraph$;
17             UIDerivation($treeFrag'$);
```

specified. Their derived UI logic can only cover limited UI flow types due to that their BP model only includes the sequential and conditional data execution flow types.

Artifact-centric approach is another paradigm to model BPs. It focuses on the evolutions of artifacts and associated constraints [2,10]. A BP is specified with artifacts and data dependencies are explicitly described accordingly. S. Yongchareon et al. [14] develop a framework for UI derivation based on artifact techniques. An IBM team [1,4] develops the Siena and its successor Barcelona for supporting UI derivation from artifact-centric process models. These works cannot generate the UI flow types originated from BP control flow patterns such as **Free-order Sequential**, and **Parallel-B**.

N. Sukaviriya et al. [12,13] propose an approach to transform a process model into a human interaction perspective. This approach is very limited in providing details of UI layouts and UI flows based on the specified data elements, user roles, tasks.

## 7 Conclusion and Future Work

This paper proposes an approach of the UI derivation based on a BP model. It aims to support the analysing, developing, and updating of real UIs. The role-enriched BP model is proposed and how to abstract/aggregate a business process for each user role has been summarized. A tree graph representing the data relationships is extracted from an abstracted and aggregated BP for a specific user role. UI derivation rules are coined as a set of constraints and recommendations. By applying these rules, UI flows are derived from the tree graph. Our proposed approach has the capability to maintain the consistency between BPs and UIs. It provides an enable tool to derive UI logic from a BP. In the future work, the change management of BPs and UIs will be studied in a unified framework.

**Acknowledgments.** This work is supported by the Australian Research Council Linkage Project (LP120200231) and the China Scholarship Council.

## References

1. Cohn, D., Dhoolia, P., Heath, F., Pinel, F., Vergo, J.: Siena: from powerpoint to web app in 5 minutes. In: Bouguettaya, A., Krueger, I., Margaria, T. (eds.) ICSOC 2008. LNCS, vol. 5364, pp. 722–723. Springer, Heidelberg (2008). doi:10.1007/978-3-540-89652-4_63
2. Cohn, D., Hull, R.: Business artifacts: a data-centric approach to modeling business operations and processes. Bull. IEEE Comput. Soc. Tech. Committee Data Eng. **32**(3), 3–9 (2009)
3. Han, L., Zhao, W., Yang, J.: An approach towards user interface derivation from business process model. In: Cao, J., Liu, X., Ren, K. (eds.) PAS 2015. CCIS, vol. 602, pp. 19–28. Springer, Singapore (2015)

4. Heath, F.T., Boaz, D., Gupta, M., Vaculín, R., Sun, Y., Hull, R., Limonad, L.: Barcelona: a design and runtime environment for declarative artifact-centric BPM. In: Basu, S., Pautasso, C., Zhang, L., Fu, X. (eds.) ICSOC 2013. LNCS, vol. 8274, pp. 705–709. Springer, Heidelberg (2013). doi:10.1007/978-3-642-45005-1_65
5. Kolb, J., Hübner, P., Reichert, M.: Automatically generating and updating user interface components in process-aware information systems. In: Meersman, R., et al. (eds.) OTM 2012. LNCS, vol. 7565, pp. 444–454. Springer, Heidelberg (2012). doi:10.1007/978-3-642-33606-5_28
6. Kolb, J., Hübner, P., Reichert, M.: Model-driven user interface generation and adaptation in process-aware information systems. Open Access Repositorium der Universität Ulm (2012). http://dx.doi.org/10.18725/OPARU-2439
7. Künzle, V., Reichert, M.: A modeling paradigm for integrating processes and data at the micro level. In: Halpin, T., Nurcan, S., Krogstie, J., Soffer, P., Proper, E., Schmidt, R., Bider, I. (eds.) BPMDS/EMMSAD-2011. LNBIP, pp. 201–215. Springer, Heidelberg (2011). doi:10.1007/978-3-642-21759-3_15
8. Künzle, V., Reichert, M.: Philharmonicflows: towards a framework for object-aware process management. J. Softw. Maintenance Evol. Res. Pract. **23**(4), 205–244 (2011)
9. Model, B.P.: Notation (bpmn) version 2.0. OMG Specification, Object Management Group (2011)
10. Nigam, A., Caswell, N.S.: Business artifacts: an approach to operational specification. IBM Syst. J. **42**(3), 428–445 (2003)
11. Sousa, K., Mendonça, H., Vanderdonckt, J., Rogier, E., Vandermeulen, J.: User interface derivation from business processes: a model-driven approach for organizational engineering. In: Proceedings of the 2008 ACM Symposium on Applied Computing, pp. 553–560. ACM (2008)
12. Sukaviriya, N., Mani, S., Sinha, V.: Reflection of a year long model-driven business and UI modeling development project. In: Gross, T., Gulliksen, J., Kotzé, P., Oestreicher, L., Palanque, P., Prates, R.O., Winckler, M. (eds.) INTERACT 2009. LNCS, vol. 5727, pp. 749–762. Springer, Heidelberg (2009). doi:10.1007/978-3-642-03658-3_80
13. 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., Palanque, P., Abascal, J., Barbosa, S.D.J. (eds.) INTERACT 2007. LNCS, vol. 4662, pp. 165–178. Springer, Heidelberg (2007). doi:10.1007/978-3-540-74796-3_17
14. Yongchareon, S., Liu, C., Zhao, X., Xu, J.: An artifact-centric approach to generating web-based business process driven user interfaces. In: Chen, L., Triantafillou, P., Suel, T. (eds.) WISE 2010. LNCS, vol. 6488, pp. 419–427. Springer, Heidelberg (2010). doi:10.1007/978-3-642-17616-6_38