

# BPMN Process Views Construction

Sira Yongchareon<sup>1</sup>, Chengfei Liu<sup>1</sup>, Xiaohui Zhao<sup>1</sup>,  
and Marek Kowalkiewicz<sup>2</sup>

<sup>1</sup> Centre for Complex Software Systems and Services  
Swinburne University of Technology  
Melbourne, Victoria, Australia  
{syongchareon, cliu, xzhao}@swin.edu.au

<sup>2</sup> SAP Research  
Brisbane, Australia  
marek.kowalkiewicz@sap.com

**Abstract.** Process view technology is catching more attentions in modern business process management, as it enables the customisation of business process representation. This capability helps improve the privacy protection, authority control, flexible display, etc., in business process modelling. One of approaches to generate process views is to allow users to construct an aggregate on their underlying processes. However, most aggregation approaches stick to a strong assumption that business processes are always well-structured, which is over strict to BPMN. Aiming to build process views for non-well-structured BPMN processes, this paper investigates the characteristics of BPMN structures, tasks, events, gateways, etc., and proposes a formal process view aggregation approach to facilitate BPMN process view creation. A set of consistency rules and construction rules are defined to regulate the aggregation and guarantee the order preservation, structural and behaviour correctness and a novel aggregation technique, called *EP-Fragment*, is developed to tackle non-well-structured BPMN processes.

## 1 Introduction

Workflow/process view technologies have been recognised as an important capability for better granularity control of process representation [5, 8-12]. A process view represents a partial view of an actual business process, and therefore separates the process representation from the executable processes. This feature highlights the benefits of process views in the areas of authority control, process visualisation, collaborative business process modelling etc.

Reluctantly, most current research on workflow/process views assumes that business processes are well structured, yet this assumption confronts a lot of conflicts when Business Process Modelling Notations (BPMN) [1] is getting popular. As a graphical modelling tool, BPMN allows users to design business processes arbitrarily, and therefore many practical BPMN processes are not strictly well structured [13]. For example, a BPMN process may have unpaired Fork and Merge or Join gateways. To apply process view technology to BPMN processes, the non-well-structured characteristics of BPMN processes have to be taken into account.

Some research efforts have been put to formalise the construction of process views, mostly by aggregating activities in the corresponding base process [3, 6]; however, their works only focus on construction from basic or compound activities without concerning related events and exceptions, which are common elements in BPMN process designing. These elements help BPMN capture more details of business processes, and they should be considered as well when creating process views for BPMN processes. Furthermore, current approaches do not provide the selective aggregation of branches in Split and Join gateways.

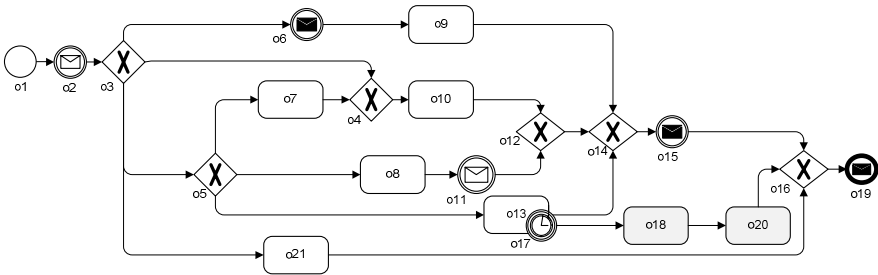


Fig. 1. Motivating example

Figure 1 shows our motivating example of BPMN process. As we can see that some parts of the structure are non-well-structured. For example, the split branches from the Fork gateway  $o_3$  flow to different Join gateways  $o_4$ ,  $o_{12}$ , and  $o_{14}$ . The timer-event  $o_{17}$  attached to the task  $o_{13}$  indicates that the subsequent execution will bypass the Join gateway  $o_{14}$  and flow through  $o_{18}$ ,  $o_{20}$ , to  $o_{16}$  if the event occurs. For the given process, users may specify the requirement for aggregating tasks  $o_7$ ,  $o_8$ ,  $o_{10}$ ,  $o_{13}$ , and event  $o_{11}$  in a process view. Two main questions are required to be answered: (1) Is the specified set of objects able to be aggregated? (2) If it is not, then what is the minimal set of objects, including the pre-specified set, for an aggregation?

Aiming at supporting process views generation for BPMN processes, we propose a BPMN process view construction approach that covers the main BPMN elements and characteristics. A set of rules is defined to regulate the view generation in compliance with structural and behavioural consistencies and correctness. Related algorithms are also developed for view checking and construction. Particularly, our approach makes the following contributions to process view research:

- Present an aggregate construction technique, called *EP-Fragment*, to tackle non-well-structured processes and selective aggregation of branches.
- Propose an algorithm for finding minimal aggregate from a set of user-specified tasks. This algorithm helps the automatic aggregation for process views.
- Consider BPMN elements, such as events, exception paths, etc., in our model.

The remainder of this paper is organised as follows. Section 2 provides a formal model of BPMN processes, syntaxes, and components for process view. Section 3 provides a process view construction methodology based on construction rules and consistency constraints; the prototype is also implemented for the proof of our approach. Section 4 reviews the related works. Finally, the concluding remarks are given in Section 5 with an indication on future work.

## 2 Formal Model of BPMN Processes

In this section, syntaxes, components, and structure of BPMN processes and process views are introduced and defined. A process view constructed on its underlying BPMN process is itself represented by the BPMN diagram. While a full range of BPMN elements are developed and proposed in BPMN 1.2 specification [1] to capture more detailed behaviour of business process, it is adequate to select only a core subset of them for the discussion on BPMN process views. This includes Tasks, Events, Gateways, Control flows, Message flows, Exception flows, and Pools.

**Definition 1 (Private process or Process).** A private BPMN process  $bp$  contains a set of tasks, events, and gateways connected together to represent the execution behaviour of the whole process. We model it as an extended directed-graph which is represented as a tuple  $(O, T, T^E, G, E, F)$ , where,

- $O$  is a finite set of BPMN element objects divided into disjoint sets of  $T, G,$  and  $E$
- $T$  is a finite set of tasks in  $bp$
- $G$  is a finite set of gateways in  $bp$
- $E$  is a finite set of events in  $bp$ ;  $event\_type: E \rightarrow \{Start, Catching-Intermediate, Throwing-Intermediate, End\}$  is a function used to specify the type of event.
- $F \subseteq O \times O$  is a finite set of control flow relations represented by a directed edge in  $bp$ . A control flow  $f = (o_i, o_j) \in F$  corresponds to the unique control flow relation between  $o_i$  and  $o_j$ , where  $o_i, o_j \in O$
- $T^E \subseteq E \times T$  is non-injective and non-surjective defining a finite set of attachment relations of intermediate events on tasks, called *Event-attached task relation*. An attachment relation of event  $e$  on task  $t$ ,  $t^e = (e, t) \in T^E$  corresponds to the intermediate trigger condition of event  $e$  for task  $t$ , where  $t \in T, e \in E$  and  $event\_type(e) = Catching-Intermediate$ .
- $F^*$  is reflexive transitive closure of  $F$ , written  $o_i F^* o_j$ , if there exists a path from  $o_i$  to  $o_j$ . In addition, we can write  $o_i (F \cup T^E)^* o_j$  if there exists a path from object  $o_i$  to  $o_j$  via control flow relations  $F$  and event-attached task relations  $T^E$ .

Note that the exception flow of the task is a flow leading from an event  $e$  in  $T^E$ , and there can be one or more events attached to the task defining multiple exception flows. We also define necessary functions that will be used in the paper.

- $in(x) = |\{y \in O \mid \exists y, (y, x) \in F\}|$  returns the in-degree of node  $x$ , and  $out(x) = |\{y \in O \mid \exists y, (x, y) \in F\}|$  returns the out-degree of node  $x$
- $path(o_i \xrightarrow{f_i, f_j} o_j)$  returns a set of all objects in all possible paths leading from  $o_i$  via a control flow  $f_i$  to  $o_j$  via a control flow  $f_j$ , such that  $\exists o_i, o_j \in O, \exists f_i, f_j \in F, o_i (f_i F^* f_j) o_j$ . A set of objects in *normal path*, denoted as  $O_{NP}$ , defines a set of all objects in all possible paths from start event to the end event of a process, i.e.,  $O_{NP} = path(e_s \xrightarrow{F^*} e_e)$ , such that  $e_s \in \{E \mid event\_type(E) = Start\}$  and  $e_e \in \{E \mid event\_type(E) = End\}$ .

It is also conceived that a *well-structured* (opposite to *non-well-structured*) process must contain structures of correct pairs of Fork and Merge or Join gateways, and there must be no branch going out or coming in between the structure [15]. The

*non-well-structured* process can be detected by using graph reduction [19] or SESE decomposition technique [20].

**Definition 2 (Least Common Predecessor and Least Common Successors).** Given a set of objects  $N \subseteq O$  in a process, we define a set of least common predecessors and successors of  $N$ , denoted as  $lcp(N)$  and  $lcs(N)$ , respectively.

$$lcp(N) = \{ o^p \in OW \mid \forall o \in N (o^p F^* o \wedge (\neg \exists o^q \in OW (o^q F^* o \wedge o^q F^* o^p))) \}$$

$$lcs(N) = \{ o^s \in OW \mid \forall o \in N (o F^* o^s \wedge (\neg \exists o^q \in OW (o F^* o^q \wedge o^s F^* o^q))) \}$$

For the purpose of identifying which flow going out of the least common predecessors and which flow coming into the least common successors, we define two functions  $lcpF(N)$  and  $lcsF(N)$  as the subset of outgoing flows of  $lcp(N)$  and incoming flows of  $lcs(N)$ , respectively. These subsets only contain the flows in  $F$  that flow into or out from the set  $N$ .

$$lcpF(N) = \{ f_p \in F \mid \forall o^p \in lcp(N), \forall o^s \in lcs(N), \exists o \in N, (o^p, o) \in F \wedge \text{path}(o^p \xrightarrow{f_p, F^*} o^s) \mid > 0 \}$$

$$lcsF(N) = \{ f_s \in F \mid \forall o^s \in lcs(N), \forall o^p \in lcp(N), \exists o \in N, (o, o^s) \in F \wedge \text{path}(o^p \xrightarrow{F^*, f_s} o^s) \mid > 0 \}$$

From the  $lcs$  and  $lcp$  defined above, we can see that if any object does not exist in the *normal path* of the process, but other objects do, then  $lcs$  and  $lcp$  will not be found. For example, we can determine that  $lcp$  and  $lcs$  of a set of objects  $\{o_7, o_9\}$  in Figure 1 are  $o_3$  and  $o_{14}$ , respectively. Correspondingly, the set of flows according to  $lcp$  and  $lcs$ , i.e.,  $lcpF$  and  $lcsF$ , are  $\{(o_3, o_6), (o_3, o_5)\}$  and  $\{(o_9, o_{14}), (o_{12}, o_{14})\}$ , respectively. However, if we include  $o_{18}$  into the set, the functions  $lcp$ ,  $lcs$ ,  $lcpF$ , and  $lcsF$  will return an empty set as  $o_{18}$  does not exist in the *normal path* as same as the others.

Figure 2 illustrates an example of complex scenario showing multiple flows of multiple least common predecessors and successors in a process. Assume that  $N = \{t_2, t_3\}$ , we find  $lcp(N) = \{g_1\}$  and  $lcs(N) = \{g_4, g_5\}$ ; correspondingly,  $lcpF(N) = \{(g_1, t_2), (g_1, t_3)\}$  and  $lcsF(N) = \{(g_2, g_4), (g_2, g_5), (g_3, g_4), (g_3, g_5)\}$ . Similarly, if we assume  $N = \{t_4, t_5\}$ , then  $lcp(N) = \{g_2, g_3\}$  and  $lcs(N) = \{g_6\}$ . Therefore, we can find that  $lcpF(N) = \{(g_2, g_4), (g_2, g_5), (g_3, g_4), (g_3, g_5)\}$  and  $lcsF(N) = \{(t_4, g_6), (t_5, g_6)\}$ .

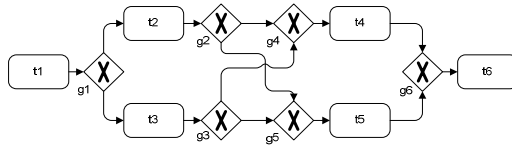


Fig. 2. An example of  $lcpF$  and  $lcsF$

**Definition 3 (Exception path).** Given a process  $bp(O, T, T^E, G, E, F)$ , an *exception path* is a set of the paths leading from a catching-intermediate event  $e$  in *Event-attached task relation*  $(e, t) \in T^E$  to any object in the *normal path* or the end event of the process.

Let  $teObject(e, t)$  denote the set of objects lying on the *exception path* of  $(e, t) \in T^E$ .

$$teObject(e, t) = \begin{cases} o \mid \exists o_n \in O_{NP}, e F^* o \wedge o F^* o_n, & \text{if } \text{path}(e \xrightarrow{F^*} o_n) \mid > 0 \\ o \mid \exists e_e \in \{E \mid \text{event\_type}(E) = \text{End}\}, e F^* o \wedge o F^* e_e, & \text{otherwise} \end{cases}$$

As shown in Figure 1, we want to find the objects on the exception path of timer event  $o_{17}$  which attached to the task  $o_{13}$ . As we can find that  $lcs(\{o_{17}, o_{13}\}) = \{o_{16}\}$  in which it exists in both *normal path* from start event  $o_1$  and *exception path* of  $o_{17}$ , so the set of objects in exception path  $teObject(o_{17}, o_{13}) = \{o_{18}, o_{20}\}$ .

**Definition 4 (Collaboration Process).** A collaboration process is a set of private processes that interacts each other by interchanging messages. Let  $cbp$  denote a BPMN collaboration process and it is a tuple  $(BP, M, \delta)$ , where

- $BP = \{bp_1, bp_2, \dots, bp_n\}$ ,  $bp_i \in BP (1 \leq i \leq n)$  is a process existing in  $cbp$
- $\delta: BP.O \rightarrow P$  is a bijective function describing the object-pool relations between objects in private processes and pools  $P = \{p_1, p_2, \dots, p_k\}$ , where pool  $p_i \in P (1 \leq i \leq k)$  is used as a container of private process. Correspondingly  $\delta^{-1}: P \rightarrow BP.O$  is an inverse function
- $M \subseteq \bigcup_{bp \in BP} (bp.T \cup bp.E) \times \bigcup_{bp \in BP} (bp.T \cup bp.E)$ ,  $m = \{(o_i, o_j) \in M \mid \delta(o_i) \neq \delta(o_j)\}$  is a message of the interaction between source  $o_i$  and target  $o_j$  of tasks or events such that the source and the target must be on different private processes or pools

We define *process view* as an abstract representation of its base collaboration process. The detailed construction process of a view will be introduced in the next section.

### 3 Process View Construction

Process views are constructed by a set of process view operations in which recent works on process views have summarised two primary operations: Aggregation and Hiding [2, 3]. Aggregation operation provides users to define a set of objects in the base process that has to be aggregated and replace such objects with the aggregate object, while hiding operation will simply hide the specified objects. In this paper we do not consider the hiding operation. The aggregation operation can be iterated in order to achieve the preferred process view. As such, this section will firstly define a set of consistency rules that the constructed process view and its underlying process must comply to maintain the structural and behaviour correctness between them.

#### 3.1 Preliminaries

In this section, we define some necessary terms, definitions and functions that will be used in the process view construction.

**Definition 5 (Process fragment or P-fragment).** Process fragment represents a partial structure of a private process. Let P-fragment  $Pf$  denote a nonempty connected sub-graph of a process  $bp \in cbp.BP$  and it is a tuple  $(O', T', T^{E'}, G', E', F', F_{in}, F_{out})$  where  $O' \subseteq O$ ,  $T' \subseteq T$ ,  $T^{E'} \subseteq T^E$ ,  $G' \subseteq G$ ,  $E' \subseteq E$ ,  $F' \subseteq O' \times O' \subseteq F$ , such that,

- $\forall e_s \in \{E \mid event\_type(E) = Start\}$ ,  $\forall e_e \in \{E \mid event\_type(E) = End\}$ ,  $e_s \notin E' \wedge e_e \notin E'$ , i.e.,  $Pf$  cannot contain any start or end event of  $bp$
- $\exists F_{in}, F_{out} \subseteq F$ ,  $F \cap ((O \setminus O') \times O') = F_{in} \wedge F \cap (O' \times (O \setminus O')) = F_{out}$ ;  $F_{in}$  and  $F_{out}$  are the set of entry flows and exit flows of  $Pf$ , respectively

- $\forall o_i \in O', \exists o_m, o_n \in O', \exists o_x \in O \setminus O', \exists o_y \in O \setminus O', \exists (o_x, o_m) \in F_{in}, \exists (o_n, o_y) \in F_{out}, o_x F' * o_i \wedge o_i F' * o_y$ , i.e., for every object  $o_i$  in  $Pf.O'$  there exists a path from entry flow to  $o_i$  and from  $o_i$  to exit flow
- for every object  $o \in O'$  there exists a path  $p=(e_s, \dots, f_i, \dots, o, \dots, f_o, \dots, e_e)$  starting from  $e_s$  to  $e_e$  via  $f_i \in F_{in}$ ,  $o$ , and  $f_o \in F_{out}$

Let *boundary objects* of  $Pf$  be a set of entry and exit objects of  $Pf$  which all objects  $O'$  in  $Pf$  are bounded by boundary objects, such that,

- $\exists o_x \in O \setminus O', \exists o_y \in O', (o_x, o_y) \in F_{in}$ ;  $o_x$  is the *entry object* of  $Pf$
- $\exists o_y \in O \setminus O', \exists o_x \in O', (o_x, o_y) \in F_{out}$ ;  $o_y$  is the *exit object* of  $Pf$

Figure 3 depicts an example of various P-fragments of the process in the motivating example shown in Figure 1. The biggest P-fragment  $Pf_4$  has only one entry object  $o_3$  and one exit object  $o_{16}$ . P-fragment  $Pf_3$  has two entry objects  $o_3$  and  $o_9$ , and one exit object  $o_{16}$ . Similarly,  $Pf_2$  has two entry objects  $o_3$  and  $o_5$ , and one exit object  $o_{14}$ .  $Pf_1$  has one entry object  $o_5$  but it has two exit objects  $o_{12}$  and  $o_{14}$ . From the Definition 5,  $o_{18}$  and  $o_{20}$  are not accounted for exit objects of any P-fragment because they are not in the *normal path*.

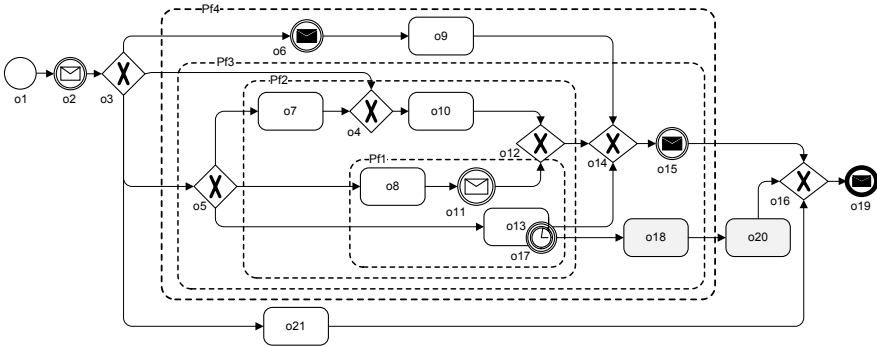


Fig. 3. P-fragments of the motivating example

### 3.2 Process View Consistency Rules

As stated before, every generated process view must preserve the structural and behaviour correctness when deriving its underlying process which can be the base business process or even inherited process views. In order to preserve such properties, a comprehensive set of *Process view consistency rules* for BPMN processes are defined. Since our previous work [4, 2] defines a set of consistency rules based on BPEL processes, we adapt and extend it as to comply with BPMN.

Assume that  $v_1$  is a process view based on underlying collaboration process  $cbp$ , and  $v_2$  is a process view constructed by applying an aggregation operation on process view  $v_1$ , then  $v_1$  and  $v_2$  must satisfy all consistency rules defined below.

**Rule 1 (Order preservation).** For any two objects belonging to process views  $v_1$  and  $v_2$ , their execution order must be consistent if such objects exists in  $v_1$  and  $v_2$ , i.e.,

If  $o_1, o_2 \in \bigcup_{bp \in v_1.BP} bp.O \cap \bigcup_{bp \in v_2.BP} bp.O$ , such that  $o_1 F o_2$  in  $v_1$ , then  $o_1 F o_2$  in  $v_2$

**Rule 2 (Branch preservation).** For any two objects belonging to process views  $v_1$  and  $v_2$ , the branch subjection relationship of them must be consistent, i.e.,

If  $o_1, o_2 \in \bigcup_{bp \in v_1.BP} bp.O \cap \bigcup_{bp \in v_2.BP} bp.O$ , such that  $\neg(o_1 F^* o_2 \vee o_2 F^* o_1)$  then  $lcp(\{o_1, o_2\})$  in  $v_1 = lcp(\{o_1, o_2\})$  in  $v_2$  and  $lcs(\{o_1, o_2\})$  in  $v_1 = lcs(\{o_1, o_2\})$  in  $v_2$

**Rule 3 (Event-attached task preservation).** For any event-attached task relation belonging to  $v_1$  and  $v_2$ , an existence of all coherence objects on the exception path led from such attached event must be consistent, i.e.,

If  $(e, t) \in \bigcup_{bp \in v_1.BP} bp.T^E \cap \bigcup_{bp \in v_2.BP} bp.T^E$ , such that  $teObject(e, t)$  exists in  $v_1$ , then  $teObject(e, t)$  exists in  $v_2$ .

**Rule 4 (Message flow preservation).** For any message flow exists in  $v_1$  and  $v_2$ , the message flow relation of its source and target objects must be consistent, i.e.,

If  $o_1, o_2 \in \bigcup_{bp \in v_1.BP} (bp.T \cup bp.E) \cap \bigcup_{bp \in v_2.BP} (bp.T \cup bp.E)$ , such that  $(o_1, o_2) \in v_1.M$  then  $(o_1, o_2) \in v_2.M$ .

### 3.3 Constructing an Aggregate

In this section, we define a set of aggregation rules and introduce a formal approach by extending the concept of *P-fragment* to validate the specified set of objects in the process whether it is able to be aggregated. If it is valid, then the result of aggregation is constructed and represented by single atomic task.

#### 3.3.1 Aggregation Rules

*Aggregation rules* specify the requirements when constructing an aggregate. Let  $O^A \subseteq O$  denote a set of objects in process view  $v_1$  that have to be aggregated and let  $agg(O^A)$  return an aggregate task in process view  $v_2$  constructed from  $O^A$  such that every object in  $O^A$  exists in the *normal path* in  $v_1$  and the aggregate satisfies every aggregation rule. We also demonstrate that this proposed set of aggregation rules conforms to *Process view consistency rules*, thus the aggregation operation maintain structural and behaviour correctness between  $v_1$  and  $v_2$ .

**Aggregation Rule 1 (Atomicity of aggregate).** An aggregate behaves as an atomic unit of processing (task); therefore, it must preserve the execution order for every task and event within it, as well as between itself and the process.

It is conceived that the structure and behaviour of every object to be aggregated the aggregate remain internally unchanged. However, the relation and behaviour among those objects in  $O^A$  and the other objects  $O \setminus O^A$  that are not in the aggregate need to be considered such that there must exist only one in-degree and out-degree of the aggregate which are the least common predecessor of  $O^A$  and the least common successor of  $O^A$ , respectively, i.e.,

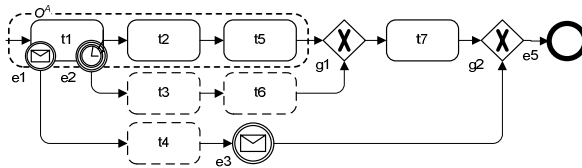
$$\forall o \in O^A \text{ in } v_1, lcp(agg(O^A)) = lcp(O^A) \wedge lcs(agg(O^A)) = lcs(O^A) \wedge |in(agg(O^A))| = |out(agg(O^A))| = 1$$

This rule demonstrates the conformance to *Process view consistency rules*: (1) Order preservation and (2) Branch preservation.

**Aggregation Rule 2 (Objects in exception path).** If the task in event-attached task relation is in the aggregate then every object in its exception path must be hidden in the process view; thus, it is not considered to be in the aggregate, i.e.,

If there exists task  $t \in O^A \cap T$  and event  $e \in E$  such that  $(e, t) \in T^E$ , then every object  $o \in teObject(e, t)$  must be hidden.

The concept behind this rule is that every object in the exception path is treated as an internal behaviour of a task having an event attached to, if the task is to be aggregated then, consequently, such event is to be hidden. Figure 4 shows an example of an application of this rule. If a set of objects  $\{t_1, t_2, t_5\}$  is to be aggregated, then the set  $\{e_1, e_2, e_3, t_3, t_4, t_6\}$  resulted from  $teObject(e_1, t_1) \cup teObject(e_2, t_1)$  must be hidden.



**Fig. 4.** Aggregating tasks with event-attached task

This rule demonstrates the conformance to *Process view consistency rules*: (3) Event-attached task preservation.

### 3.3.2 Structure Validation

In this section we propose an approach for structure validation of a given set of objects to be aggregated, called *Enclosed P-fragment*. This approach mainly checks the atomicity of the structure according to Aggregation rule 1. If it is valid, then the aggregate is able to be constructed. However, if it is not valid, we also propose the technique to find the minimum set of objects based on a given set in the next section.

**Definition 6 (Enclosed P-fragment or EP-Fragment).** Let  $Pf(O', T', T^E, G', E', F', F_{in}, F_{out})$  define a P-fragment of a process by the Definition 5. If  $Pf$  has only one entry object and one exit object as its boundary, then it is *enclosed*, called *Enclosed P-fragment* or *EP-Fragment*. We can claim that any *EP-Fragment* itself guarantees the atomicity of its whole structure.

Revisiting our motivating example in Figure 3, we can see that  $Pf_1$  is not enclosed since there are two exit objects  $o_{12}$  and  $o_{14}$ ; while  $o_{18}$  is not accounted for exit object as it is on the exception path from event  $o_{17}$  attached to task  $o_{13}$ . Similarly,  $Pf_2$  and  $Pf_3$  are unenclosed. The former has two entry objects  $o_3$  and  $o_5$ , and one exit object  $o_{16}$ ; likewise, the latter has two entry objects  $o_3$  and  $o_9$ .

From the Definition 6, multiple entries and multiple exits are allowed for defining *EP-Fragment*. This also enables the selective aggregation of branches feature as illustrated by  $Pf_4$  in Figure 3. The fragment  $Pf_4$  is enclosed because it has only one entry object  $o_3$  and one exit object  $o_{16}$ , although there are multiple branches coming in and going out from its fragment.

In order to validate the structure of the given set of objects to be aggregated, we have to find whether the given set of objects is able to form an *EP-Fragment*. To do so, two auxiliary functions are required: *forward walk* and *backward walk*.

Given any two flows in a process:  $f_s=(o_x, o_s) \in F$  as an entry flow and  $f_e=(o_e, o_y) \in F$  as an exit flow, we want to find two sets of objects, denoted as  $\rho Fwd(f_s, o_y)$  and  $\rho Bwd(f_e, o_x)$ , by walking forward along all possible paths starting from  $f_s$  to  $o_y$  and by walking backward along all possible paths from  $f_e$  to  $o_x$ , respectively.

- A *forward walk* function  $\rho Fwd(f_s, o_y)$  returns a set of objects by walking forward from  $f_s$  to  $o_y$  as well as from  $f_s$  to the end event of the process
- A *backward walk* function  $\rho Bwd(f_e, o_x)$  returns a set of objects by walking backward from  $f_e$  to  $o_x$  as well as from  $f_e$  to the start event of the process.

These two functions can be implemented by extending the depth-first search algorithm so we do not detail them in this paper. Apart from them, we also require two functions to identify a set of objects that does not exist in forward walk but it is found in backward walk, and vice versa. Such functions will help us to validate the *EP-Fragment* as the technique will be described later.

Let function  $objOutBwd(f_e, o_x)$  return a set of objects  $O^{OB} \subseteq O$  such that it does not exist in forward walk but exists in backward walk and each of such object's flow directly links to the object which exists in both forward and backward walks, i.e.,  $\forall o_b \in O^{OB}, \exists o \in \rho Fwd(f_s, o_y) \cap \rho Bwd(f_e, o_x), (o_b, o) \in F$ . Inversely, function  $objOutFwd(f_s, o_y)$  returns a set of objects  $O^{OF} \subseteq O$  such that  $\forall o_f \in O^{OF}, \exists o \in \rho Fwd(f_s, o_y) \cap \rho Bwd(f_e, o_x), (o, o_f) \in F$ .

We can see that if  $\rho Fwd(f_s, o_y) = \rho Bwd(f_e, o_x)$ , then  $objOutFwd(f_s, o_y)$  and  $objOutBwd(f_e, o_x)$  return  $\emptyset$ . This also implies that there exists only one entry flow to the forward walk from  $f_s$  to  $f_e$  and only one exit flow from the backward walk  $f_e$  to  $f_s$ .

From Figure 5, we want to find objects in forward and backward walks between an entry flow  $f_s = (t_2, g_2)$  and an exit flow  $f_e = (g_5, t_7)$ . The result of  $\rho Fwd(f_s, t_7)$  is  $\{g_2, t_3, t_4, t_5, t_6, g_5, g_4, g_6, t_8\}$  and  $\rho Bwd(f_e, t_2)$  is  $\{g_2, t_3, t_4, t_5, t_6, g_5, g_4, g_1, t_1\}$ . Consequently,  $objOutFwd(f_s, t_7)$  returns  $\{g_6\}$  since it exists in forward walk but does not exist in backward walk. Correspondingly,  $objOutBwd(f_e, t_2)$  returns  $\{g_1\}$ .

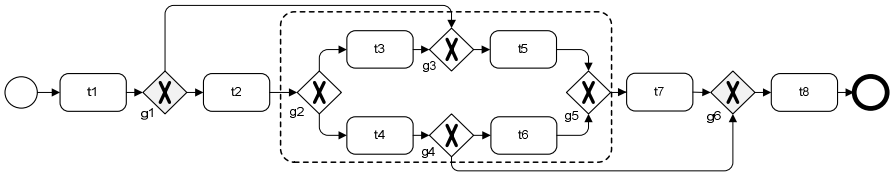


Fig. 5. An example of forward and backward walk in a process

**Lemma 1:** Given a set of objects  $N \subseteq O$  in a process  $bp(O, T, T^E, G, E, F)$ , an *EP-Fragment*  $Pf(O', T', T^{E'}, G', E', F', F_{in}, F_{out})$  can be formed by  $N$ , if and only if,

$$\begin{aligned}
 - \bigcup_{(f_s, o_y) \in lcpF(N) \times lcs(N)} \rho Fwd(f_s, o_y) &= \bigcup_{(f_e, o_x) \in lcpF(N) \times lcs(N)} \rho Bwd(f_e, o_x) \\
 , \text{ i.e., the forward walks and backward walks of all combinations of } lcpF &\text{ and } lcsF \\
 \text{flows return the same result set identical to } N \text{ in } bp & \tag{1}
 \end{aligned}$$

$$- \forall f_p \in lcpF(N), \exists o \in N, f_p = (o_x, o), \text{ i.e., there exists only one entry object } o_x \tag{2}$$

$$- \forall f_s \in lcsF(N), \exists o \in N, f_s = (o, o_y), \text{ i.e., there exists only one exit object } o_y \tag{3}$$

From Figure 3, assuming that  $N = \{o_8, o_{11}, o_{13}\}$ , then we can find that  $lcp(N) = \{o_5\}$  and  $lcs(N) = \{o_{14}\}$ . Correspondingly, we will find  $lcpF(N) = \{(o_5, o_8), (o_5, o_{13})\}$  and  $lcsF(N) = \{(o_{12}, o_{14}), (o_{13}, o_{14})\}$ . Because  $lcpF$  returns entry flows with only one entry object  $o_5$  and  $lcsF$  returns exit flows with only one exit object  $o_{14}$ , therefore  $N$  satisfies condition (2) and (3). After having applied both functions for every combination of  $lcpF(N)$  and  $lcsF(N)$ , the result sets of  $\rho Fwd$  and  $\rho Bwd$  are  $\{o_8, o_{11}, o_{12}, o_{13}\}$  and  $\{o_8, o_{11}, o_{12}, o_{13}, o_{10}, o_7, o_4, o_3, o_2, o_1\}$ , respectively. As we can see that only the result from  $\rho Fwd$  is identical to  $N$ , but  $\rho Bwd$  is not (condition (1) is not satisfied), thus  $N$  cannot be formed as an *EP-Fragment*.

Figure 6 shows a process with P-fragment  $Pf_1$  in the loop structure. Assume that  $N = \{t_1, t_2, t_3\}$ . Since we find that  $lcpF(N) = \{(g_1, t_1)\}$  and  $lcsF(N) = \{(t_3, g_1)\}$ , then conditions (2) and (3) are satisfied. However, when applying  $\rho Fwd$  and  $\rho Bwd$  functions,  $\{t_1, t_2, t_3, g_2, t_4\}$  and  $\{t_1, t_2, t_3, g_2\}$  are returned, respectively. The non-identical results from both functions prove that objects  $N$  in  $Pf_1$  can not form an *EP-Fragment* by not satisfying condition (1). In contrast and clearly, objects in  $Pf_2$  can form an *EP-Fragment*.

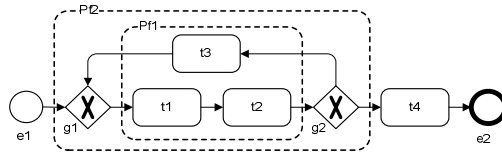


Fig. 6. P-fragments in a loop structure

**Theorem 1:** A P-fragment  $Pf(O', T', T^E, G', E', F', F_{in}, F_{out})$  in a process  $bp(O, T, T^E, G, E, F)$  can be aggregated if and only if it is enclosed.

**Proof:** We prove the claim in two steps: (1) we present that the *EP-Fragment* can be aggregated and it complies with the Aggregation rules 1 and 2; (2) we show that the aggregate can form an *EP-Fragment*.

- (1) Let  $Pf(O', T', T^E, G', E', F', F_{in}, F_{out})$  be an *EP-Fragment* and every object  $O'$  in  $Pf$  is to be aggregated. Aggregation Rule 1 is naturally satisfied by *EP-Fragment* (by the Definition 6). Similarly, Aggregation Rule 2 is satisfied by P-fragment (by the Definition 5).
- (2) Let object  $agg(O^A)$  be an aggregate represented as a single atomic task with one incoming flow  $(o_x, agg(O^A))$  and one outgoing flow  $(agg(O^A), o_y)$ . Let  $N = \{agg(O^A)\}$  and then we find  $lcp(N) = \{o_x\}$  and  $lcs(N) = \{o_y\}$ ; therefore

condition (2) and (3) of Lemma 1 are satisfied. Apply forward and backward walk functions will return identical result  $\{agg(O^A)\}$ , therefore condition (1) of Lemma 1 is satisfied. So we can conclude that  $agg(O^A)$  is an *EP-Fragment*.

### 3.3.3 Minimal Aggregate

As aforementioned, if a given set of objects cannot be aggregated, i.e., not able to form an *EP-Fragment* by Lemma 1, we facilitate users to be able to do so, by using our proposed minimal aggregate function. For a given set  $O^A$ , we can find the minimal aggregate of  $O^A$  which satisfies every Aggregation rule. We define  $minAgg(O^A)$  as the function that returns a minimal set of objects that can be aggregated, and hides every object on exception paths. The implementation of  $minAgg(O^A)$  is illustrated in Algorithm 1.

---

**Algorithm 1.** Finding minimal set of objects for the aggregation

---

```

minAgg:  $O^A \rightarrow O$ 
1 let  $O_{min} = \{\}, O^F = \{\}, O^B = \{\}, O^{OF} = \{\}, O^{OB} = \{\}$ 
2 let  $O^{FT} = \{\}, O^{BT} = \{\}$ 
3 let  $O_{temp} = O^A$ 
4 do
5   for each  $(f_s: (o_x, o_s), f_e: (o_e, o_y)) \in lcpF(O_{temp}) \times lcsF(O_{temp})$ 
6      $O^F = \rho Fwd(f_s, o_y)$ 
7      $O^B = \rho Bwd(f_e, o_x)$ 
8      $O^{FT} = O^{FT} \cup (O^F \setminus O^{FT})$ 
9      $O^{BT} = O^{BT} \cup (O^B \setminus O^{BT})$ 
10     $O^{OF} = O^{OF} \cup (objOutBwd(f_e, o_x) \setminus O^{OF})$  // find the adjacent exit object
11     $O^{OB} = O^{OB} \cup (objOutFwd(f_s, o_y) \setminus O^{OB})$  // find the adjacent entry object
12  end for
13  if  $(O^{FT} = O^{BT}) \wedge (\forall f_p \in lcpF(O^{FT}), \exists o \in O^{FT}, f_p = (o_x, o) \wedge (\forall f_s \in lcsF(O^{FT}), \exists o \in O^{FT}, f_s = (o, o_y))$  then break //break the loop if Agg Rule1 is satisfied
14   $O_{temp} = O_{temp} \cup O^{OF} \cup O^{OB}$ 
15 while  $O^{FT} \neq O^{BT}$ 
16  $O_{min} = O^{FT}$ 
17 for each  $t \in O_{min} \cap T$ 
18   if  $\exists e \in E, (e, t) \in T^E$  then
19     //hide all objects belonging to path of event-attached task(Agg Rule 2)
20     for each  $o \in teObject(e, t)$ 
21       hide object  $o$  and its corresponding flows
22     end for
23   end if
24 end for
25 return  $O_{min}$ 

```

---

We explain why  $minAgg(O^A)$  returns a minimal set of objects of  $O^A$ . Firstly, the given set of objects are validated whether it can form an *EP-Fragment* or not by applying Lemma 1. If it is able to form an *EP-Fragment*, then it is returned which initially satisfies Lemma 1 without extending its boundary. However, if it cannot form an *EP-Fragment*, then a set of adjacent objects resulted from *objOutFwd* and *objOutBwd* functions are added to the object set for each loop (lines 10-11). Since these two

functions return only a set of direct adjacent objects which is necessary required intuitively; thus the additional set is minimal then we conclude that this algorithm guarantees the minimum expansion of the object set to form an *EP-Fragment*.

**Theorem 2:** *A set of objects  $O^A \subseteq O$  in a process  $bp(O, T, T^E, G, E, F)$  satisfies all aggregation rules if and only if  $O^A = \text{minAgg}(O^A)$ .*

**Proof:** To prove this theorem, we need to construct an aggregate by  $\text{minAgg}(O^A)$  that satisfies both Aggregation Rule 1 and Rule 2.

**Aggregation Rule 1:** From the Algorithm 1 for  $\text{minAgg}$ , initially we find the  $\text{lcpF}$  and  $\text{lcsF}$  of  $O^A$ . Then, all objects within the paths between  $\text{lcpF}$  and  $\text{lcsF}$  are found by  $\rho\text{Fwd}$  and  $\rho\text{Bwd}$  (lines 5-7). The while loop check if  $\rho\text{Fwd}$  does not return the result as identical to the result of  $\rho\text{Bwd}$  (line 4), then  $Pf$  is not enclosed (by Lemma 1, condition 1), then the adjacent objects resulted from  $\text{objOutFwd}$  and  $\text{objOutBwd}$  functions are added to  $O^A$  (line 14). Then  $O^A$  is repetitively computed for finding the  $\text{lcpF}$  and  $\text{lcsF}$  again finding the entry and exit flows that will be the boundary of the enlarged  $O^A$ .  $\rho\text{Fwd}$  and  $\rho\text{Bwd}$  are used to compare the result and then  $O^A$  is validated by  $\text{lcpF}$  and  $\text{lcsF}$  again to check whether it can form an *EP-Fragment* (lines 4-15). If it concludes that such new result set of both  $\rho\text{Fwd}$  and  $\rho\text{Bwd}$  are identical, and  $\text{lcpF}$  returns flows with one entry object and  $\text{lcsF}$  returns flows with one exit object (line 13), then  $O^A$  can form an *EP-Fragment* (by Lemma 1). This concludes that the result aggregate satisfies Aggregation Rule 1.

**Aggregation Rule 2:** For each object in the result aggregate set that satisfies Aggregation Rule 1, if there exists event  $e$  attached to task  $t$  in  $O^A$ , then such event and every object  $o \in \text{teObject}(e, t)$  is not included into the aggregate and it is also hidden (lines 17-22). Thus, it satisfies Aggregation Rule 2.

If a given set  $O^A$  initially satisfies every condition in Lemma 1 (line 13) and every object in the exception path of every event that attached to task in  $O^A$ , then the result set will return the same as an original given set. Therefore, this concludes that  $O^A = \text{minAgg}(O^A)$ . In contrast, if the  $O^A$  is not able to satisfy Lemma 1, then the result set will be expanded; hence  $|\text{minAgg}(O^A)| > |O^A|$ . Thus, this concludes that  $O^A$  is not able to be aggregated.  $\square$

### 3.3.4 Effect to Message Flows of an Aggregate

The aggregation has to preserve the consistency of message flow interactions among the process and its participants in the collaboration process. For every incoming and outgoing message flow of the object that have to be aggregated, it also remains for the aggregate, such that,

- $\forall m_x \in v_1.M, \exists o_j \in O^A, \exists o_k \in \bigcup_{bp \in v_1.BP} bp.O \setminus O^A, m_x = (o_j, o_k) \rightarrow (t_{agg}, o_k) \in v_2.M$
- $\forall m_y \in v_1.M, \exists o_m \in \bigcup_{bp \in v_1.BP} bp.O \setminus O^A, \exists o_n \in O^A, m_y = (o_m, o_n) \rightarrow (o_m, t_{agg}) \in v_2.M$

Figure 7 illustrates an example of object aggregation with message flows. All incoming and outgoing messages (a) are rearranged to the aggregate task (b).

This conditional effect of the aggregation satisfies *Process view consistency rules*: (4) Message flow preservation.

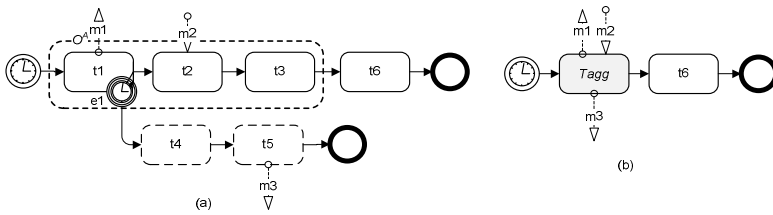


Fig. 7. Aggregation with messages

By considering the Aggregation Rules (1 and 2) and the conditional effect of message flows, we therefore can see that the aggregate task resulted from our aggregation approach satisfies to all Process view consistency rules.

### 3.4 Prototype

The prototype implementation, named *FlexView*, is currently being developed to support process view construction for BPMN process based on the approach proposed in this paper. The system initially loads the base BPMN file, and then allows users to specify which elements in the process will be aggregated. When the operation is completed, the process view is generated as an output of the system. Figure 8 shows the main screen of the system and an example of process view constructed from the base BPMN process displayed on the BizAgi Process Modeller [17]. Due to limited space, we do not show much detail in the prototype here.

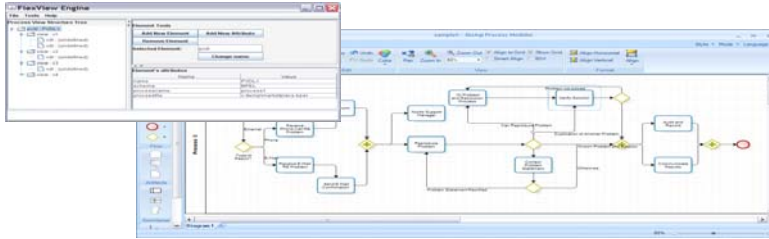


Fig. 8. *FlexView* engine (left) and result process view

## 4 Related Work and Discussion

Zhao, Liu, Sadiq, and Kowalkiewicz [4] proposed the process view approach based on the perspective of role-based perception control. A set of rules on consistency and validity between the constructed process views and their underlying process view is defined. Compared with our work, they neither provide how each process view is constructed nor consider non-well-structured processes.

Liu and Shen [6] presented an algorithm to construct a process view with an ordering-preserved approach from a given set of conceptual-level activities in a base process. In their approach, the aggregate activities called virtual activities requires to conform membership rule, atomicity rule, and order preservation rule. Compare with our work, they only focus on basic activity aggregation while they do not consider non-well-structured processes and the relation setting of activity in a collaborative

process such as messages and event attachments (exception). We extend their work to allow such relations.

Van der Aalst, Dumas et al [16] proposed the framework for staged correctness-preserving configuring reference process models regarding the correctness of syntax and behavioural semantic captured by propositional logic formula. The proposed framework is based on *WF-net* and a set of transition variants used for the configuration: allowed, hidden, and blocked. Compare with our work, they do not provide an aggregation approach to construct the abstracted process model.

Bobrik, Reichert et al [18] presented a visualization approach to tackle inflexibility of building and visualizing personalized views of managed processes. They introduced two basic view operations: reducing and aggregating, and properties of process views. Graph reduction and graph aggregation techniques (by defining SESE region) are used for such operations. This work has some similarities compared with our *P-fragment*; however, the *EP-Fragment* allows multiple entries and exits to be applicable for selective aggregation of branches. In addition, their work focuses on process visualizing thus relaxing the preservation of structural and behaviour consistencies between base process and its resulted view, while our work is based on the comprehensive set of consistency rules. Their work also does not consider other aspects of BPMN properties, such as exception, but ours does.

Grefen and Eshuis [3] proposed a formal approach to construct a customized process view on a business process. The approach consists of two main phases: a process provider constructs a process view that hides private internal details of the underlying business process, and second phase let a consumer constructs a customized process view tailored to its needs to filter out unwanted process information. However, their approach focuses on block-structured process model represented by hierarchy tree model only and it does not take a graph structure into account. While it is too restrictive and unlikely to see those well-structured process in BPMN process, the approach presented in this paper adapted and extended from their work and our previous work [2] by considering non-well-structured process and event attachments features of BPMN.

Vanhatalo, Volzer, and Koehler [14] proposed a technique for decomposing workflow graphs into a modular and fine fragment by finding Canonical Fragments, and generate the Refine Process Structure Tree. In short, we aim at proposing an aggregate approach that satisfies aggregation rules specifically for BPMN process, while they only focus on finding the finest fragment of graphs.

## 5 Conclusion and Future Work

This paper presented a novel approach for constructing an aggregate for BPMN process views. The main contribution of this approach is that the core subset of current BPMN standard is taken into account in order to define a comprehensive set of construction rules and consistency rules. Since BPMN is likely to allow processes to be non-well-structured unlike some other standards such as BPEL which are strictly well-structured (block-structure), it is necessary to validate its structure using the *EP-Fragment* validation technique proposed in this paper. Our future work is to support process views for choreography processes in the BPMN 2.0.

**Acknowledgement.** The research work reported in this paper is supported by Australian Research Council and SAP Research under Linkage Grant LP0669660.

## References

1. Object Management Group: Business Process Modeling Notation, V1.2 (January 2009), <http://www.omg.org/spec/BPMN/1.2>
2. Zhao, X., Liu, C., Sadiq, W., Kowalkiewicz, M., Yongchareon, S.: On Supporting Abstraction and Concretisation for WS-BPEL Business Processes. In: Zhou, X., et al. (eds.) DAS-FAA 2009. LNCS, vol. 5463, pp. 405–420. Springer, Heidelberg (2009)
3. Eshuis, R., Grefen, P.: Constructing customized process views. *Data & Knowledge Engineering* 64, 419–438 (2008)
4. Zhao, X., Liu, C., Sadiq, W., Kowalkiewicz, M.: Process View Derivation and Composition in a Dynamic Collaboration Environment. In: CoopIS 2008, pp. 82–99 (2008)
5. Liu, C., Li, Q., Zhao, X.: Challenges and opportunities in collaborative business process management. *Information System Frontiers* (May 21, 2008)
6. Liu, D., Shen, M.: Workflow modeling for virtual processes: an order-preserving process-view approach. *Information Systems* (28), 505–532 (2003)
7. Liu, D., Shen, M.: Business-to-Business workflow interoperation based on process-views. *Decision Support Systems* (38), 399–419 (2004)
8. Chiu, D.K.W., Cheung, S.C., Till, S., Karlapalem, K., Li, Q., Kafeza, E.: Workflow View Driven Cross-Organizational Interoperability in a Web Service Environment. *Information Technology and Management* 5, 221–250 (2004)
9. Chebbi, I., Dustdar, S., Tata, S.: The view-based approach to dynamic inter-organizational workflow cooperation. *Data & Knowledge Engineering* 56, 139–173 (2006)
10. Zhao, X., Liu, C., Yang, Y.: An Organisational Perspective on Collaborative Business Processes. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) BPM 2005. LNCS, vol. 3649, pp. 17–31. Springer, Heidelberg (2005)
11. Schulz, L.A., Orłowska, M.E.: Facilitating cross-organisational workflows with a workflow view approach. *Data & Knowledge Engineering* 51, 109–147 (2004)
12. Van der Aalst, W.M.P., Weske, M.: The P2p Approach to Interorganizational Workflows. In: Dittrich, K.R., Geppert, A., Norrie, M.C. (eds.) CAiSE 2001. LNCS, vol. 2068, pp. 140–156. Springer, Heidelberg (2001)
13. Van der Aalst, W.M.P., Ouyang, C., Dumas, M., ter Hofstede, A.H.M.: Pattern-Based Translation of BPMN Process Models to BPEL Web Services. *International Journal of Web Services Research* (2007)
14. Vanhatalo, J., Volzer, H., Koehler, J.: The Refined Process Structure Tree. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 100–115. Springer, Heidelberg (2008)
15. Liu, R., Kumar, A.: An Analysis and Taxonomy of Unstructured Workflows. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) BPM 2005. LNCS, vol. 3649, pp. 268–284. Springer, Heidelberg (2005)
16. Van der Aalst, W.M.P., Dumas, M., Gottschalk, F.H.M., ter Hofstede, A., La Rosa, M., Mendling, J.: Correctness-Preserving Configuration of Business Process Models. In: FASE 2009, pp. 46–61 (2009)
17. BizAgi Process Modeller, BizAgi, <http://www.bizagi.com>
18. 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)
19. Sadiq, W., Orłowska, M.E.: Analyzing process models using graph reduction techniques. In: CAiSE, pp. 117–134 (2000)
20. Vanhatalo, J., Volzer, H., Leymann, F.: Faster and More Focused Control-Flow Analysis for Business Process Models Through SESE Decomposition. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 43–55. Springer, Heidelberg (2007)