UniFlexView: A Unified Framework for Consistent Construction of BPMN and BPEL Process Views

Journal:	*Concurrency and Computation: Practice and Experience*
Manuscript ID	CPE-18-1294.R2
Editor Selection:	Prof. David Walker
Wiley - Manuscript type:	Research Article
Date Submitted by the Author:	26-Nov-2019
Complete List of Authors:	Yongchareon, Sira; Auckland University of Technology, IT and Software Engineering Liu, Chengfei; Swinburne University of Technology Zhao, Xiaohui; Federation University Australia
Keywords:	

UniFlexView: A Unified Framework for Consistent Construction of BPMN and BPEL Process Views

Sira Yongchareon[1,2], Chengfei Liu[2], and Xiaohui Zhao[2,3]

[1]Department of IT and Software Engineering
Auckland University of Technology, New Zealand
sira.yongchareon@aut.ac.nz
[2]Department of Computer Science and Software Engineering
Swinburne University of Technology, Australia
cliu@swin.edu.au
[3] School of Science, Engineering and Information Technology
Federation University Australia
x.zhao@federation.edu.au

Abstract. Process view technologies allow organizations to create different granularity levels of abstraction of their business processes, therefore enabling a more effective business process management, analysis, inter-operation, and privacy controls. Existing research proposed view construction and abstraction techniques for block-based (i.e., BPEL) and graph-based (i.e., BPMN) process models. However, the existing techniques treat each type of the two types of models separately. Especially, this brings in challenges for achieving a consistent process view for a BPEL model that derives from a BPMN model. In this paper, we propose a unified framework, namely UniFlexView, for supporting automatic and consistent process view construction. With our framework, process modelers can use our proposed view definition language to specify their view construction requirements disregarding the types of process models. Our UniFlexView's system prototype has been developed as a proof of concept and demonstration of the usability and feasibility of our framework.

Keywords: Business process models; process views; process model abstraction; workflow management

1. INTRODUCTION

A concept of workflow or process views adapted from database view has been considered as a necessary capability that should be applied for inter-organizational business processes since it helps driving business process management technology to thrust more supports in cooperation, autonomy, and openness [22]. Workflows or process views are considered a promising conceptual approach to selectively hide details of private workflows, whilst providing a process-oriented interface to facilitate the state-oriented communication between trading partners [33, 41].

The implementation of process-view technologies, as part of Process-Aware Information Systems (PAISs), allows each business partner in its collaboration network to restrict how much of its own private (local) business processes can be revealed to other partners in the collaboration [9, 13, 37]. A process view inherited from a complete business process provides less detailed or more abstracted than the base process. Apart from workflow and process view perspective, business process modeling techniques are also in the interest of how those process view concepts can be realized and implemented in the real world based on current industry's standards, such as Business Process Execution Language (BPEL) and its Web-Service version (WS-BPEL) [2] and Business Process Modeling Notation (BPMN) [23]. These process modeling standards are used for model business processes in a Web Service

environment. While there are continuous changes in business conditions among partners to stay competitive in the market, they play a very vital and important role in facilitating and cooperating business processes among business partners in a loosely coupled relationship.

In practice, BPMN is commonly used to define business processes at a conceptual level while BPEL is used to define service execution. Different stakeholders may use different meta-models and/or different modeling tools to create and maintain their version of the process model [49]. As a result, there could be a potential inconsistency between a process view of BPEL and another view of BPMN where both derive from the same process model. Especially the issue becomes more complicated as BPEL process models tend to be more structured but process models defined using BPMN can be unstructured. Yet we can perform manual (or semi-automatic) consistency checking, however, it can be time-consuming and could result in errors. To address these challenges, we propose a unified framework for supporting process view construction for both the languages. Our framework allows process modelers to define a single process view definition that can be applied to both BPMN and BPEL to create two consistent versions of a process view.

The main contributions of this work can be summarized as follows.

—We propose a unified framework named *UniFlexView* that consists of a business process model, a process view model and a comprehensive set of rules that can be used to guarantee the consistency between a derived process view and its original process model defined by BPEL and BPMN considering unstructuredness.

— We propose a unified language, namely *View Definition Language* (*VDL*), to define business process views regardless of process modeling languages used. The language has been developed based on an XPath-liked syntax.

— We have developed a *UniFlexView*'s system prototype as a proof of concept to evaluate the feasibility and usability of our framework.

The remainder of this paper is organized as follows. Section 2 discusses the background and related work. Section 3 introduces our proposed *UniFlexView* framework. Section 4 discusses our view definition language and view operations used within the framework. Section 5 presents the implementation of our *UniFlexView* system and case studies. Finally, the conclusion and future work are provided in Section 6

2. BACKGROUND AND RELATED WORKS

In this section, we reviewed related process views and business process modeling research.

2.1 Workflow and Process Views

Chiu et al. [32] adapted the concepts of views from databases to workflows which help balance trust and security by meaning that only information necessary for process enactment, enforcement and monitoring of the service can be made available to participating parties in a fully controlled and comprehensive manner and employed a virtual workflow view for the inter-organizational collaboration instead of the real instance, to hide internal information. In addition, each party requires only minor, or none, modification to its own workflow to successfully meet at a commonly agreed upon and interoperable interface. Since an organization is required to interoperate with many other different organizations, different views of a workflow can be used in order to provide different organizations with individual views according to their requirements. Based on a conceptual model for workflow views and their theoretical

bases as the fundamental support for workflow interoperability and visibility by external parties, a workflow view is an externally accessible subset of a private workflow.

Later, Schulz and Orlowska [33] proposed a model for tiering business processes into the private business processes of the organization and those shared business processes that interconnect them. Private business processes can expose interaction points, and shared processes can link to these points so that an overall business process may span two or more organizations. The interaction points can selectively expose information about the processes and process tasks of an organization. This workflow view model is provided for multi-granular privacy for workflows, given they believe that a workflow view needs to be protected from the unauthorized interaction. Finally, they used a Petri-net based representation for the basis for consideration of state dependencies between tasks in a workflow and the adjacent task in a workflow view.

Similar to [33], Jiang et al. [41] proposed a process-view and timed-colored Petri net (TCPN) combined approach to manage cross-organizational workflows. Their approach includes the mapping from TCPN workflow models to process-view workflow models in which the aspects of control flow and data flow are considered together and the collaborative execution mechanisms of cross-organizational workflow instances. They have developed a hybrid P2P based decentralized workflow management system combined with the process-view approach to providing a flexible and scalable architecture for cross-organizational workflows management.

Van der Aalst [13], Chebbi et al. [37], and Lin [9] proposed similar designs and architectures for inter-organizational workflows based on the notion of local process views. Each collaborating partner defines its own local process view. Similarly, Eshuis et al. [42] proposed a collaboration framework that supports the outsourcing of business processes between parties based on process views. The framework considers both the construction of public process views by projection (via principles of hiding, omitting and aggregation) of internal processes and the matching of consumer and provider views to measure their similarity. Liu and Shen [36] presented an algorithm to construct a process view with an ordering-preserved approach from a given workflow, but they did not discuss its correctness with respect to inter-organizational workflows. Chie et al. [32] also proposed that the interoperation model is consistent if it satisfies the criteria of integrity and correctness. The integrity criteria concern the consistency between workflow views and their parent workflows, while the correctness criteria concern the consistency between workflow views and their target communication scenarios. Grefen and Eshuis [12] 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 the second phase lets a consumer constructs a customized process view tailored to its needs to filter out unwanted process information. The customized process view reveals only those activities requested by the consumer; the remaining activities of the underlying process view are hidden or omitted. However, this approach focuses on a block-structured process model only and it does not consider a graph structure since they claim that block-structured process models have the advantage of not containing structural errors such as deadlocks. Therefore, it allows for a simple and efficient (tractable) procedure for constructing customized views. They also defined a set of construction rules for aggregation of nodes (activities) as consistency criteria between a generated process view and its base business process.

Zhao et al. [43] presented a framework to support view abstraction and concretization of WS-BPEL processes with a rigorous view model proposed to specify

the dependency and correlation between structural components of process views with emphasis on the characteristics of WS-BPEL, and a set of rules are defined to guarantee the structural consistency between process views during transformation. Then, [44] extended [43] to incorporate role dependencies into process view derivation in order to support organizational privacy protection, authorization control, and runtime updates to the process view perceivable to a user with specific view-merging operations. However, both approaches are designed to support only a block-structured language, WS-BPEL in this case, constructing views for un-structured workflows is not possible with these approaches. Our *UniFlexView* framework presented in this paper extends the framework presented in [43] to fully support constructing process views from structured and non-structured typed workflow modeling languages.

Eshuis et al. [48] developed a formal approach to propagate consistency-preserving changes from an internal business process to its derived process view. It allows a process view to evolve while maintaining its consistency with its underlying internal processes. Their definitions of process models and process views are based on BPEL, while our framework is not confined to only BPEL. On the order hand, Küster et al. [49] proposed a Shared Process Model that provides different stakeholder views at different abstraction levels and that synchronizes changes made to any view. However, their approach only supports BPMN models.

2.2 BPEL vs. BPMN

BPEL and BPMN are entirely different yet complementary standards. BPEL is an execution language designed to provide a definition of web services orchestration. BPEL defines only the executable aspects of a process when that process is dealing exclusively with web services and XML data. BPEL is a block-structured programming language [1, 20, 38] and the structure can be represented by a tree model. The node of a tree represents each activity and a control flow dependency between activities hold by its parent. For example, a sequence node containing a set of ordered activities (which represents that all children of this sequence node) are executed sequentially. Concurrent processing is modeled using a flow element with a set of concurrent activities defined as children of the element.

On the other hand, BPMN is conceived of as a graph-structured language with additional concepts to handle blocks [5, 14, 19, 38]. Routing between activities is handled by transitions between them. Unlike BPEL activities, the activities in a BPMN process are delineated as the nodes of a directed graph, with the transitions being the edges connecting between each set of two nodes. Conditions associated with the transitions determine at execution time when activity or activities should be executed next. BPEL does not define the graphical diagram, human-oriented processes, sub-process, and many other aspects of a modern business process. It simply was never defined to carry the business process diagram. This nature distinguishes BPMN from BPEL which focuses exclusively on the executable aspects of the process.

According to Shaprio [38], there is no significant difficulty when translating blocked-structured flow control in BPEL into a graph structure like in BPMN, but the reverse is very problematic. There are some works proposing a transformation of BPEL process to Petri Nets [15], and XPDL to Petri Nets [14], as well as the reverse transformation from Petri Nets and graph structure to BPEL [16, 17, 18] in order to do a formal analysis of the process structure. Note XML Process Definition Language (XPDL) is a format standardized by the Workflow Management Coalition (WfMC) to interchange business process definitions between different workflow products [5]. One of the earlier work [46] has proposed a view mechanism and a set of process consistency rules for constructing BPMN process views and our research presented in this paper

is extended based on that work. In [24], the transformation strategies between graph-oriented and block-oriented process modeling languages are proposed but they are restricted to a structured process graph. Although there is one recent research work proposing a tool named WFTXB [19] for translating between XPDL and BPEL proposed for providing mapping algorithms between them, however, it does not consider the poorly structured XPDL process model unlike the work for BPEL in [16]. More recently, [45] studied the problem of transforming a process model with an arbitrary topology into an equivalent well-structured process model using refined process structure trees [47] and a mathematical approach to reason about equivalences of restructured process models. The proposed method has been implemented as a tool that supports BPMN and EPC process notations. However, their work does not consider BPEL specifications, especially the synchronization link characteristic which makes view constructions for BPEL not straightforward.

3. UNIFLEXVIEW FRAMEWORK

3.1 Business Process Model

Workflow and business process models are typically modeled by Petri nets because of their formal semantics, graphical representation benefits, expressiveness, analysis techniques, and tools for modeling and analyzing workflow processes [10, 13]. In this research the process view model might frequently change by consistency checking procedures, aiming at process structure adjustment and rectification for correctness and consistency according to its underlying base process model as a similar reason to [9], therefore, flexibility is the most crucial factor of choosing the workflow or process model. Hence, our process graph model is developed to represent all aspects of a process view model and its behavior rather than Petri nets.

Here, we define a *process model* to capture a structural component of a business process representing how activities are structured. A process model is represented by an enclosed block structure that has only one start activity at the beginning of its structure and only one end activity at the ending of its structure. Such activities can be an atomic activity or a gateway. A gateway is defined as an atomic activity representing a structure of control flow. There are eight types of gateways, namely *OR-Split*, *OR-Join, XOR-Split, XOR-Join, AND-Split, AND-Join, LOOP-Begin, LOOP-End*. The *OR-Split/Join* and *LOOP-Begin/End* gateways may contain the conditions for restriction of the control flow.

Definition 1: (Process model). Let p denote a process model and it is a tuple (A, G, E, L, m_s, m_e), where

—A is a set of atomic activities,

—G is a set of gateways and function $gateway\_type$: $G \rightarrow Type$ ($Type \in$ {*AND-Split, OR-Split, XOR-Split, AND-Join, OR-Join, XOR-Join, LOOP-Begin, LOOP-End*})

—E is a set of directed edges and $e = (n_x, n_y) \in E$ corresponds to the control dependency between nodes n_x and n_y, where $n_x, n_y \in A \cup G$,

—L is a set of *synchronization links* that represent the synchronization dependency between two activities or gateways on different branches in *AND-Split* and *AND-Join* gateways,

—m_s and m_e are the start activity and the end activity of the process, respectively.

We can construct a new process model to mimic the model that uses *synchronization links* by replacing the links with *AND-Split/Join* gateways structure. The semantics of

synchronization links are based on the existing BPEL standard [1]. An example of this substitution is shown in Figure 1.

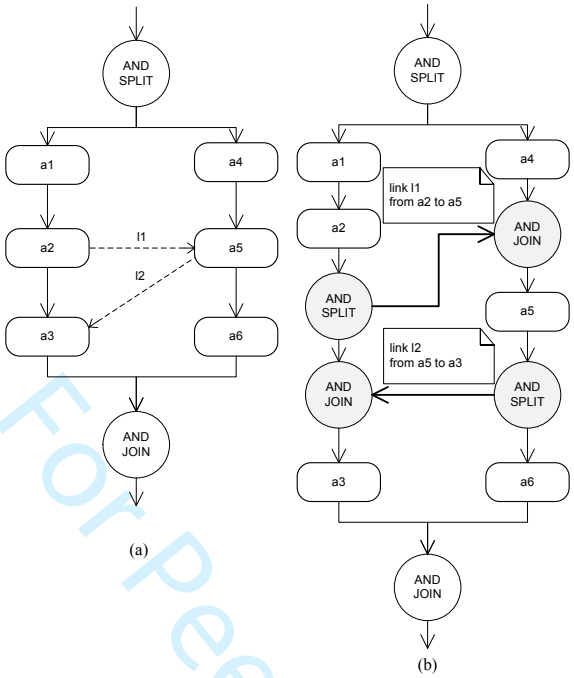


Figure 1. Synchronization link equivalent constructs

A synchronization link can be used within the AND-Split and AND-Join gateways, and between two structures nested inside such that gateways, as shown in Figure 2. A synchronization link is invalid if there is a link which is sourcing from an activity in a branch of an OR-Split gateway and targeting at an activity in another branch in AND-Split gateway in which both of them are nested inside the outer AND-Split/Join structure. In BPEL, uses of synchronization links are possible and can result in having a non-block-structure inside an AND-Split/Join structure but allowing more modeling flexibility. For example, a link sourcing from an activity in an AND-Split branch to a target activity in an OR-Split branch is considered valid in BPEL.

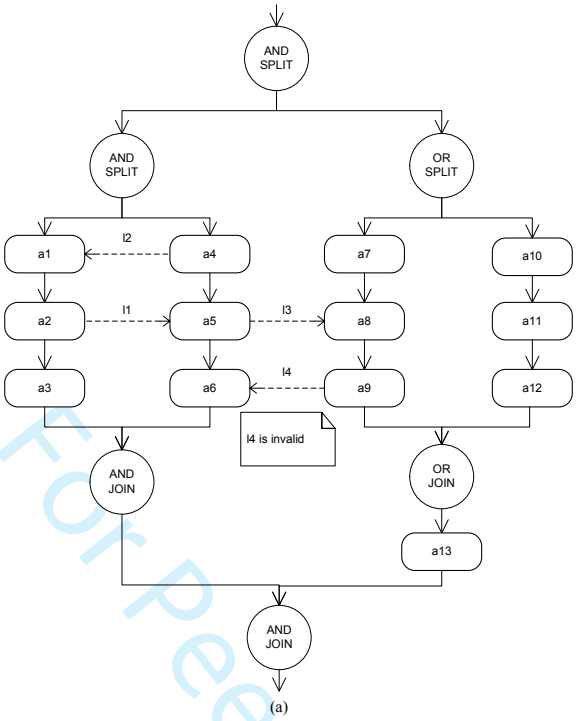


Figure 2. an example of invalid synchronization link between branches of split gateways

Process model functions

—A common split gateway predecessor (*CSP*), g, of a set of activities, $A_x \subseteq A$, denotes a split gateway such that g is a predecessor of each activity in A_x.

—Function $CSP(A_x)$ returns a set of common split gateway predecessors of all activities in $A_x \subseteq A$; otherwise returns null if there is no common split gateway predecessor.

—Function $LCSP(A_x)$ returns the least common split gateway predecessor of all activities in $A_x \subseteq A$; otherwise return null if not found.

—Function $LCJD(A_x)$ returns the least common join gateway descendent of all activities in $A_x \subseteq A$; otherwise return null if not found.

Note that the least common (split or join) gateway of activities is the closest gateway (if exists) that all the activities are structured under.

Next, we define a *process path* to represent a sequence of activities and gateways such that from each of its activities there is an edge to the next activity in which they are possibly executed from a start activity (predecessor activity) to an end activity (successor activity).

Definition 2: (Process Path). Let T denote a *process path* which is a set of paths and it is a tuple (m_s, m_e, β), where

—$m_s \in A \cup G$ represents a start activity of T,

—$m_e \in A \cup G$ represents an end activity of T,

—β denotes a set of all possible paths starting from m_s and ending with m_e,

Note that $T(m_s, m_e, \beta)$ is valid only if m_s exists as the first activity of all paths, and m_e exists at the last order of all paths. A process model p is valid if every activity in a process model exists in the process path T starting from m_s ending at m_e are in A.

In addition, we also define a function *activity_within_path* (p, a, a') which returns a set of nodes N' including all the activities and gateways lying between the paths leading from a start activity a to an end activity a' for a process model p. Correspondingly, $activity\_within\_path^{-1}(N') = (a, a')$ is an inverse function.

Next, we define a *structured activity* for representing an enclosed block structure of a set of atomic or compound activities consisting of one couple of gateways (in order to define a boundary of the structure and activities reside in it). There are four couples of gateways used to construct structured activities: *OR-Split* and *OR-Join*, *XOR-Split* and *XOR-Join*, *AND-Split* and *AND-Join*, *LOOP-Begin* and *LOOP-End*. *Structured activity* is well constructed by having a strictly enclosed block structure of at least one couple of gateways. It can contain nested structured activities.

Definition 3: (Structured activity). Let $\check{S}$ denote a *structured activity* in a process model p and it is a tuple $(A', G', E', L', m_s, m_e)$, where $A' \subseteq p.A$, $G' \subseteq p.G$, $E' \subseteq p.E$, and $L' \subseteq p.L$ are defined for a structured activity (as a set of activities), a set of gateways, a set of directed edges, and a set of synchronization links, respectively. m_s is defined for a start activity of the structure such that $m_s \in p.G$, $gateway\_type(m_s) \in \{$*AND-Split, OR-Split, XOR-Split*, *LOOP-Begin*$\}$, and m_e is defined for an end activity of the structure such that $m_e \in p.G$, $gateway\_type(m_e) \in \{$*AND-Join, OR-Join, XOR-Join*, *LOOP-End*$\}$.

In addition, we write $\check{S}(m_s, m_e)$ in short to denote a structured activity of a start gateway m_s and an end gateway m_e where $\check{S}.A' = activity\_within\_gateway(p, m_s, m_e)$, $\check{S}.G' = gateway\_within\_gateway(p, m_s, m_e)$. Note that any synchronization link in an AND-Split/Join structured activity can connect between two activities or gateways either within its structure or between two nested AND-Split/Join structures within the outer parent AND-Split/Join structure.

To check whether a structured activity is an *enclosed-block structure*, we define a function $isEnclosed\_structure(\check{S})$ which returns *true* only if all possible paths leading from the start activity cease at the end activity in $\check{S}$. If there is any path leading from m_s and it does not end at m_e, and any reversing paths starting from m_e and not ending at m_s, then the function returns *false*. In other words, if there is an edge that contains a dependency between two activities, in which any of them is not in $\check{S}.A \cup \check{S}.G'$, then $\check{S}$ is not enclosed. Nevertheless, this function does not guarantee that its inner structures of $\check{S}$ are valid as an enclosed block structure. It may comprise invalid enclosed block structures but altogether, considered as a whole structure, is valid.

3.2 Process View Model and Consistency Rules

A process view model represents a part of an actual business process model. Process owners can see the most detailed, or concreted, process model, in which all activities are revealed, while other process viewers may partially see an abstract process model of its base process model, called a process view model. As activities can be hidden or aggregated in a process view model.

Definition 4: (Process View Model). Let V denote a set of *process view models* defined for a corresponding base process model p, where $V = \{v_1, v_2, \ldots, v_x\}$, $v_i \in V (1 \le i \le x)$. The structure of v can be modelled as an extended directed graph in a form of a tuple $(A, G, E, L, \underline{m_s}, \underline{m_e})$, where A, G, E, L, m_s, and m_e are defined as a set of activities, a set

of gateways, a set of directed edges, and a set of synchronization links, a start activity, and an end activity, respectively.

To construct a process view, a structured activity or a set of activities (and their edges) in its original process is abstracted into an edge or a composite activity (*dummy activity*) in the view. We also call an edge a *dummy branch* if the edge is used as a branch in a Split/Join or Loop structure such that the branch contains no activity but only one edge. Only a single dummy branch can exist in a Split/Join structure. If there are two or more dummy branches, they are not distinguishable because they represent nothing but edges. Semantically, a dummy branch can be used in OR/XOR-Split/Join and Loop structures (not in AND-Split/Join structures) in order to represent an alternative way in a process view.

It is important to guarantee the consistency between a corresponding derived process view model and its underlying process model. According to Donghui Lin [9], because an organization may have different local process views defined, the incompatibility analysis is necessary for verifying an incompatibility of interaction protocols and business processes with the detection algorithms. Liu and Shen [11] proposed an order-preserving approach for deriving a structurally consistent process view from a base process. Add to that, Eshuis and Grefen [12] proposed a series of construction rules for validating the structural consistency. In our research, we define a set of consistency rules based on the order-preserving approach proposed in [11].

Given two process views v_1 and v_2 where v_2 is constructed based on v_1, we said that v_2 is a *consistent* view of v_1 if all the *view consistency rules* defined below are satisfied.

Consistency Rule 1: (Execution path preservation). An execution order of every pair of two activities that exist in both process views v_1 and v_2 are consistent, i.e.,

If $a_1, a_2 \in v_1.A \cap v_2.A$ and a process path $T(m_s, m_e, \beta)$ defines a set of possible paths β leading from a start activity m_s to an end activity m_e such that,

— $|\ activity\_within\_path(v_1, v_1.a_1, v_1.a_2)\ | \geq 2$, where $T(v_1.a_1, v_1.a_2, \beta_1)$ is valid, and

— $|\ activity\_within\_path(v_2, v_2.a_1, v_2.a_2)\ | \geq 2$, where $T(v_2.a_1, v_2.a_2, \beta_2)$ is valid,

where β_1 is a set of all possible paths that $v_1.a_2$ can be reached from $v_1.a_1$ and β_2 is a set of all possible paths that $v_2.a_2$ can be reached from $v_2.a_1$.

Consistency Rule 2: (Branch preservation). For all activities belonging to process views v_1 and v_2, the branch subjection relationships of the activities are consistent, i.e.,

If $a_1, a_2 \in v_1.A \cap v_2.A$ and $g \in CSP(a_1, a_2)$ in v_1, $g \in CSP(a_1, a_2)$ in v_2 such that $\omega(g, a_1, a_2)$ in v_1, then $\omega(g, a_1, a_2)$ in v_2, where $\omega \in \{same\_branch, \neg same\_branch\}$.

Consistency Rule 3: (Synchronization dependency preservation). If a view operation involves any activity that contains synchronization links, the synchronization links may be rearranged, if applicable, in order to preserve the synchronization dependency within an And-Split/Join structure. This rule has three sub-rules: *Consistency Rules 3.1, 3.2* and *3.3*. The first one is used when an activity is hidden, the second one is used when a set of sequential activities is aggregated, and the last one is used for an AND-Split/Join structure.

Consistency Rule 3.1: When an activity is hidden

If an activity $a_x \in v_1.A$ is to be hidden (by deletion) and represented by a dummy edge in a process view v_2, then the following set of actions described below needs to apply.

—If a source activity a_x contains outgoing synchronization links, then the source activity of the links are moved up from the hidden activity to the direct predecessor activity if it exists and is not the least common split gateway predecessor (*LCSP*) of all activities referred in those links. Otherwise, if its direct predecessor is AND-Split which is the least common split gateway predecessor, then such those links are removed.

—If a target activity a_x contains incoming synchronization links, then the target activity of the links are moved down from the hidden activity to the direct successor activity if it exists and is not the least common join gateway descendant (*LCJD*) of all activities referred in those links. Otherwise, if its direct successor is AND-Split, which is the least common join gateway descendant, then those links are removed.

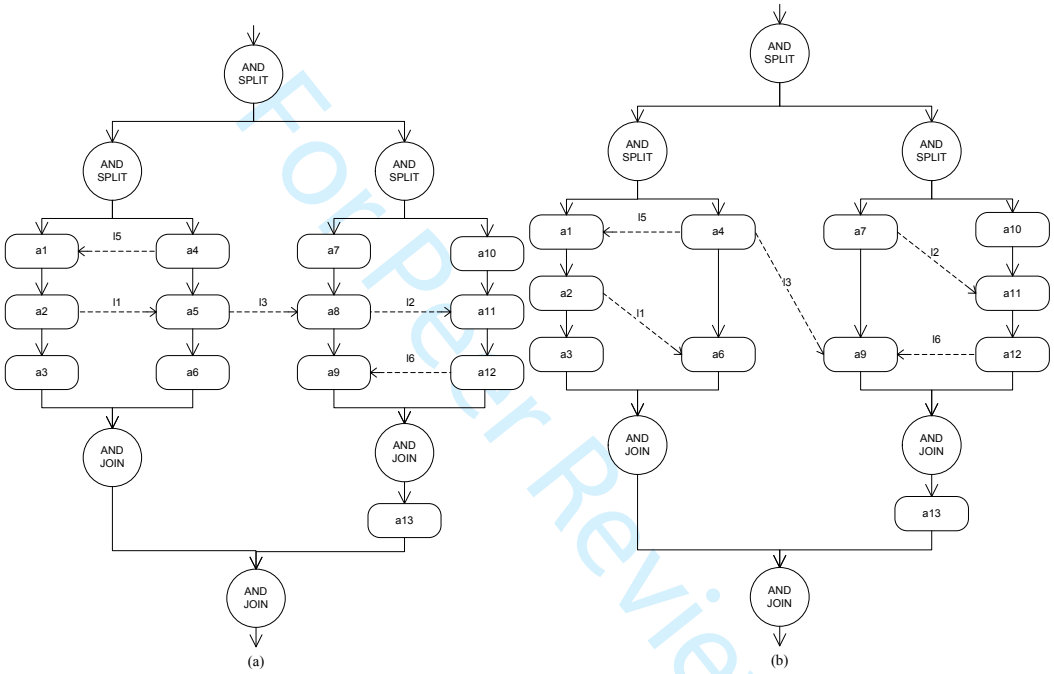


Figure 3. an example of synchronization links within, and between two AND-Split structures, and (b) the change when activities with links in and between AND-Split structure are hidden

Figure 3 (a) shows an example of a process model that consists of two inner AND-Split/Join structures nested in the outer AND-Split/Join structure. There are synchronization links inside inner AND-Split/Join structures as well as between two inner structures. There are five synchronization links l_1, l_2, l_5, and l_6 used to connect activities residing in inner AND-Split/Join structures, and l_3 is used to link between two activities from different AND-Split/Join structures.

Figure 3 (b) shows a change outcome after applying Consistency Rule 3.1 when activities a_5 and a_8 are hidden. It can be seen that the target activity of link l_1 is moved down from a_5 to the direct successor activity a_6 and the source activity of link l_2 is moved up from a_8 to the direct predecessor activity a_7. Consequently, the link l_3 between the two inner AND-Split/Join structures is taken into account; therefore, it is changed accordingly where the source of l_3 is moved up and its target is moved down.

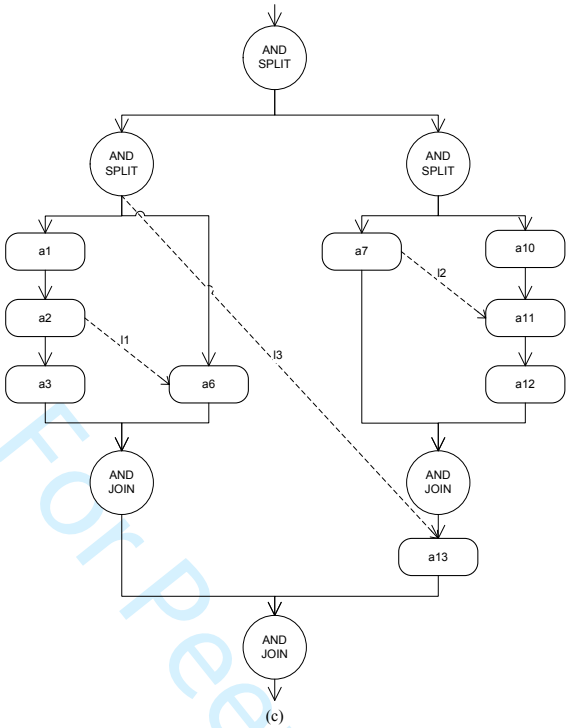


Figure 4. the change when a target activity in an inner AND-Split structure was hidden

Figure 4 (c) provides an example where activities a_4 and a_9 in Figure 3 (b) are hidden. The former contains two outgoing links targeting at its inner AND-Split/Join structure, l_5, and on the other AND-Split/Join structure l_3, while the latter contains two incoming links sourcing from its inner AND-Split/Join structure, l_6, and on the other AND-Split/Join structure l_3. The result of after having a_4 hidden shows that the link l_5 is removed since the direct predecessor activity is the least common split gateway predecessor of the activities of link l_5. Similarly, the result of removing a_9 also shows that the link l_6 is removed since the direct successor activity is the least common join gateway descendant of the activities of the link l_6. Apart from those two links, the link l_3 between two AND-Split/Join structures needs to be changed. The source activity of l_3 is moved up to AND-Split gateway, and since LCSP(a_4)≠ LCSP(a_9) so the link is removed. Similarly, the target activity of l_3 also needs to change from a_9 to its direct successor activity, which is its AND-Join gateway.

Consistency Rule 3.2: When a set of sequential activities are aggregated

When an aggregation of a set of sequential activities $A_g \in v_1.A$ in the view v_1 occurs, new dummy activity d is constructed in the resulted view v_2. To preserve the synchronization link consistency between the base model v_1 and the transformed model v_2 (with a set of synchronization links Lv_2), the consistency rules below are required to apply.

—All incoming synchronization links to the earliest executed activity of a set of sequential activities that are aggregated remain in the resulted process view where a dummy activity d is a target activity of such the links after the completion of aggregation. All outgoing links from such the activity are removed.

—All outgoing synchronization links from the latest executed activity of a set of sequential activities that are aggregated remain in the resulted process view where a dummy activity d is a source activity of such the links after the completion of aggregation. All incoming links to such the activity are removed.

—All outgoing or incoming synchronization links of activities that exist between the earliest and latest executed activities are removed.

Figure 5 shows an example of an aggregation of a set of sequential activities A_g={ a_1, a_2, a_3, a_4} with synchronization links L={l_1, l_2, l_3, l_4, l_5, l_6} to a new dummy activity d_1. After having Consistency Rule 3.2 applied, the resulted view has the new dummy activity d_1 with the only two remaining synchronizations. One is an incoming link l_1, and another one is an outgoing link l_6. Note that l_1 and l_6 in the base view are not identical to l_1 and l_6, respectively, in the resulted view after aggregation. This is because the source and target activities of them have changed.

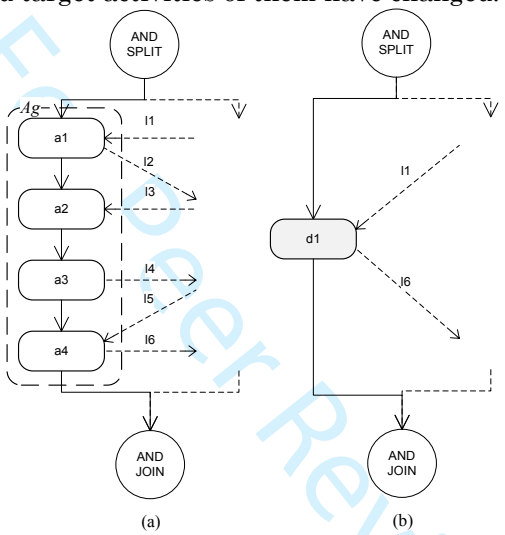


Figure 5. an aggregate with the remaining links

Figure 6 shows an example of a change of synchronization links of activities aggregation occurring in AND-Split/Join structure. Figure 6 (a) represents a process model p with an AND-Split gateway g. Dummy activities c_1 and c_2 in Figure 6 (b) are the result of an aggregation of a_1 with a_2 and a_3 with a_4 in Figure 6 (a), respectively. While Figure 6 (c) shows an alternative way of aggregation, activities a_1 and a_2 are aggregated and a new dummy activity c_1 is constructed. Similarly, a dummy activity c_3 is the result of aggregating a_7 and a_8. Figure 6 (d) demonstrates the result of both alternatives in Figure 6 (b) and Figure 6 (c), the dummy activity c_4 comes from an aggregation of activities a_5 and a_6 whether from the process model in Figure 6 (b) or in Figure 6 (c). Furthermore, both dummy activities c_2 and c_3 in Figure 6 (d) come from the result of aggregation in Figure 6 (b) and Figure 6 (c).

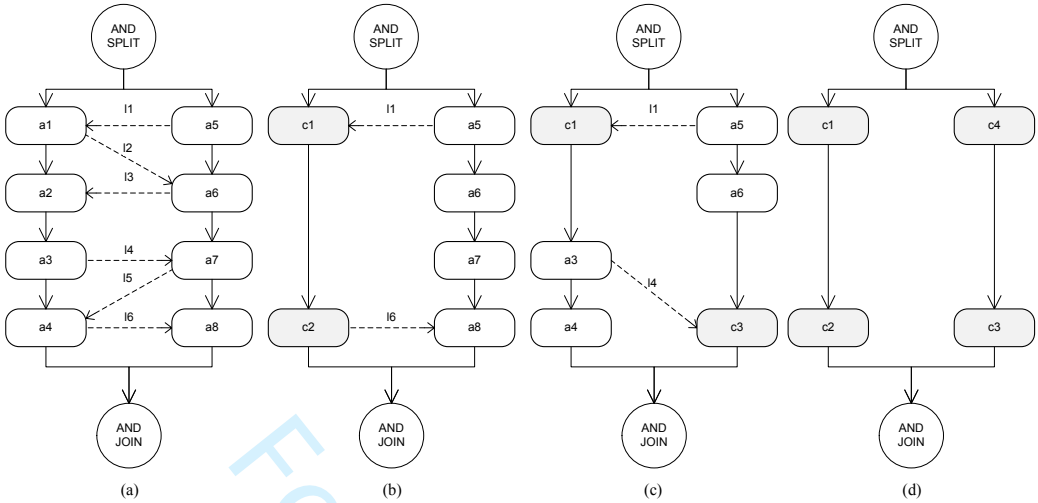


Figure 6. the changes when aggregating activities with links

Consistency Rule 3.3: When an AND-Split/Join structure is hidden or aggregated.

When a whole AND-Split/Join structured activity is hidden or aggregated, all the synchronization links contained within the structure are removed. In addition, if there is any link connecting between two nested AND-Join/Split structures within an outer parent structure and if one or both of the structures are hidden or aggregated, then such the link is removed.

Note that hiding a whole structure can be done differently. *Consistency Rule 3.1* is reused to hide every activity one by one in an AND-Split/Join structure. As a result, after having all the activities hidden, all the links in the structure are also removed.

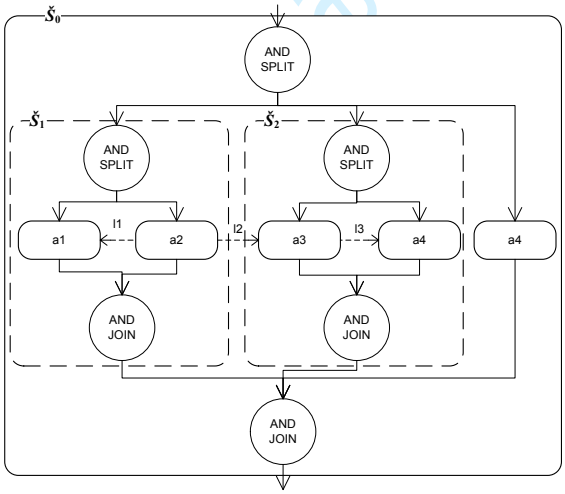


Figure 7. an example of synchronization link inside and between structures

Figure 7 shows an example of an AND-Split/Join structure $\check{S}_0$, which contains two nested AND-Split/Join structures $\check{S}_1$ and $\check{S}_2$. If the AND-Split/Join structure $\check{S}_1$ is hidden, then both l_1 and l_2 are removed. Similarly, both l_2 and l_3 are removed if the whole structure $\check{S}_2$ is aggregated.

Consistency Rule 4: (Dummy branch in OR/XOR-Split/Join structures). If an OR-Split/Join structure in a process view contains a dummy branch, then the dummy branch remains to indicate the existence of an alternative execution path. If an OR/XOR-Split/Join structure contains multiple dummy branches, then they are merged into a single dummy branch.

Figure 8 (a) shows an example of an OR-Split/Join structure in a base process model. When a set of activities $\{a_2, a_3, a_4\}$ of the middle branch are all hidden in a process view, the dummy branch b_1 is used to represent an empty branch in the structure, as shown in Figure 8 (b). Furthermore, in Figure 8 (c) if activities $\{a_7, a_8, a_9\}$ are hidden, the structure is supposed to have two dummy branches, b_1, b_2; by following the rule, these dummy branches are merged into one dummy branch b_3.

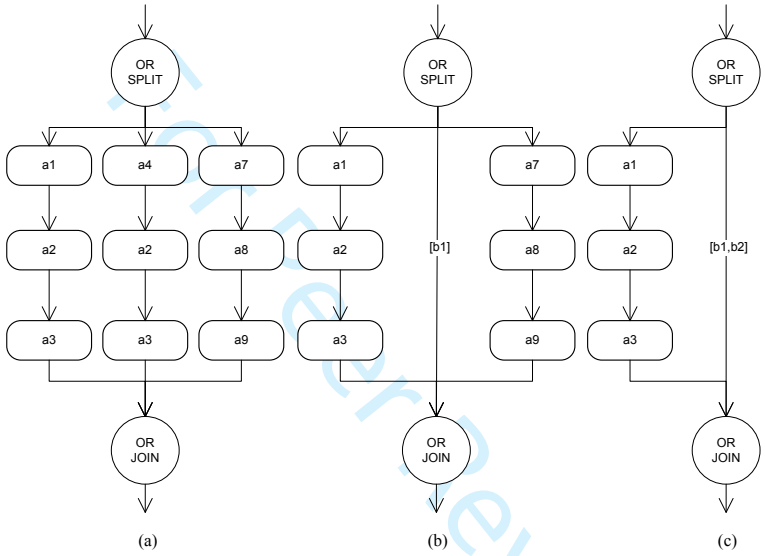


Figure 8. an example of Split/Join structure with dummy branch(es) and activities

Consistency Rule 5: (No empty Split/Join or Loop structures). If any kind of Split/Join or loop structure does not contain any activity or if it contains only dummy branches or edges, then the structure is collapsed and replaced with a new edge. As a result, the new edge links between the activity before the structure and the activity after the structure.

Figure 9 shows an example of empty OR-Split/Join and loop structures after having all activities residing in the structures hidden (see Figure 9 (c) and Figure 9 (e)). Figure 9 (b) shows a case when a set of activities $\{a_1, a_2, a_3\}$ in Figure 9 (a) are hidden. The dummy branch b_3 replaces those hidden activities. Moreover, because of *Consistency Rule 4*, all the dummy branches are merged into a single dummy branch. After that, *Consistency Rule 5* is applied so the entire structure is collapsed, as shown in Figure 9 (c). Similarly, Figure 9 (d) and Figure 9 (e) show that a loop structure is collapsed when all activities in the loop are hidden.

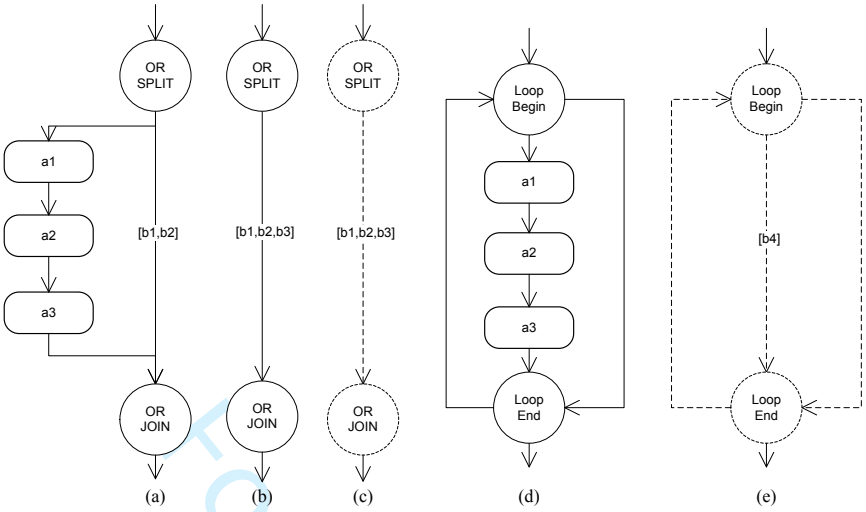


Figure 9. an example of empty OR-Split/Join and Loop structures

Consistency Rule 6: (No dummy branch in AND-Split/Join structures). If an AND-Split/Join structure contains dummy branches, then such the dummy branches are hidden.

Figure 10 represents a case of having a dummy branch b_1 in an AND-Split/Join structure (a) and the branch is hidden when this rule applies.

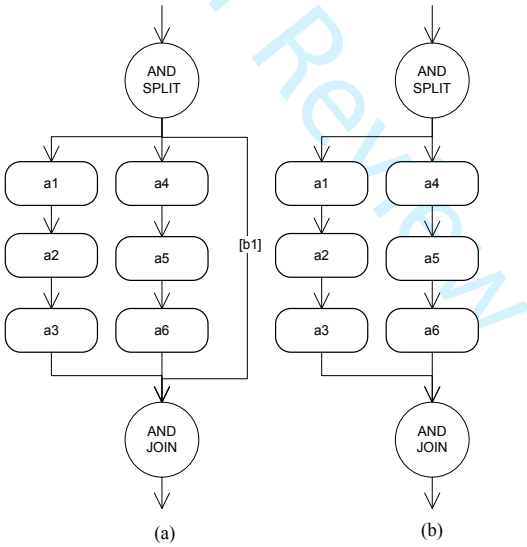


Figure 10. an example of a dummy branch in an AND-Split/Join structure

Consistency Rule 7: (No single branch in Split/Join structures). If any kind of Split/Join structure contains only one single branch, such the structure is collapsed to a single sequence. Consequently, both split and join gateways are removed from its structure.

Consistency Rule 8: (Structured activity as a whole is atomic). If an entire structured activity is hidden, then all its internal activities, edges, gateways and synchronization links are hidden. Similarly, if an entire structured activity is aggregated, all elements within it are aggregated.

It is also worth mentioning that based on our thorough observations and process model simulations based on varying complexity levels of process models (including random generations) ranging from basic structured BPEL/BPMN models to very complex process models including unstructured BPMN models, our framework can correctly and consistently generate process views from their underlying process models. While our simulations may not cover all possible complex scenarios, we believe that we had attempted as much as what we could potentially think of w.r.t our knowledge. Our consistency rules proposed can be considered complete based on the two following grounds. First, our rules are developed based on the extension of the base approach proposed [11] which claimed to be complete. Second, based on our model simulations, we did not find any case where the generated view is inconsistent.

4. VIEW DEFINITION LANGUAGE AND OPERATIONS

This section presents two mechanisms of how a process view can be created based on an underlying process model. We propose a *View Definition Language* (*VDL*) together with its set of operational functions which play a major role in a process view transformation process.

4.1 View Definition Language (VDL)

Aiming at providing a simple and semantic definition language for process view operations, our proposed process view definition language shall meet the below requirements.

—The language shall provide process modelers a simple structure and systematic method of defining process views using a primitive set of view operations.

—The language shall be generic and independent from specific process modeling standards including BPEL and BPMN (via the use of XPDL). This means that when changes to the standards should not make the language invalid.

—The language shall be extensible for additional view operations that might be added.

Here, we propose a View Expression Language (VDL) based on adopting the XML Path Language (XPath) standard [6]. The VDL is expressed in a slightly similar way to the XPath expression language in such a way that any node in an XML document can be selected by means of a hierarchic navigation path through the document tree via the XPath language. Similarly, an activity in a business process or view model could be determined and located by using the XPath approach. Although the XPath can be used only for the conformed data model created based on the XML standard [7], however, the VDL is not restricted by this limitation.

VDL Expressions

Based on the XPath language syntax and semantics, the VDL applies the XPath's concept of a location path, which is how a path in a structure can be navigated and an activity can be located using "/" or "//" notations. The former, when placed anywhere in a path, means that the activity after it must directly follow the activity before it, while the later represents that the activity after it must follow the activity before it in somewhere in the path.

A bracket notation "[ ]" is used to identify an activity name in a process model to be operated. The bracket can consist of a location path within it and this means every activity existing in that location path is selected and operated. Apart from that, the bracket can contain some available functions provided to facilitate a variety of ways of activity selection operations. The bracket with contents inside must follow by any of

the following mandatory notations (i.e., view operators): "-" for a deletion operator and ">" for an aggregation operator.

In order to concisely express the VDL, the Syntactic meta-language Extended BNF abbreviated EBNF [4] is used. A set of grammars used in VDL expressions is developed and expressed using the Extended Backus-Naur Form (EBNF), which is a meta-syntax notation used to express context-free grammars as defined in Table 1.

Table 1. VDL expression grammars in EBNF

```
path ::= {step};
step ::= axis, node;
axis ::= "/" | "//";
node ::= element name | operation;
operation ::= ("[", relative path del, "]-") | ("[", relative path agg, "]>", element name);
relative path ::= (element name, [{step}]);
relative path agg ::= relative path | element list | agg function;
relative path del ::= relative path | del function;
del function ::= "(",regular exp,")";
agg function ::= "(",starting gateway name, ",",[ending gateway name], element list, ")";
element list ::= "(",{element,[","]},")";
element ::= element name | "EMPTY";
```

In addition, a set of activities matched by Regular expression [3] can be used for selective deletion. This feature enables flexible activity selection in a path and provides an alternative way to identify activities to be deleted.

4.2 VDL Operation functions

The functions used in the VDL expression are classified by the purposes of operations which are deletion and aggregation.

Deletion functions

(1) *Single activity deletion function*
Syntax: [*a*] -
Result: Only an activity whose name exactly matched with *a* is to be deleted.

(2) *Sequential range deletion function*
Syntax: [a_x//a_y]-
Result: All the activities in the sequential range starting from a start activity a_x to an end activity a_y is to be deleted

(3) *Selective deletion function*
Syntax: [(*regular exp*)]-
Result: Delete all activities whose names matched with the regular expression, *regular exp,* in the parentheses.

Aggregation functions

(1) *Structure aggregation function*
Syntax: [g_0] >c
Result: Return a dummy activity *c*, which is a result of an aggregation of a structured activity having any Split-typed gateways or a LOOP-Begin gateway named g_0 as a start activity of the structure.
Condition: This function can be used only if a Split-typed gateway or LOOP-Begin gateway named g_0 has its couple join gateway so that a couple Join/LOOP-End gateway is not required to be identified in the function. Otherwise, if there is no couple gateway of g_0 (in some cases where g_0

and its structure is not well-constructed), then the branch aggregation function (3) is used.

(2) *Sequential range aggregation function*

Syntax: $[a_x//a_y]>c$

Result: Return a dummy activity *c*, which is a result of an aggregation of a set of activities and gateways starting from a start activity a_x to an end activity a_y. If there are any nested Split/Join structures in a sequential range, then they are collapsed and aggregated with other activities in the range.

(3) *Branch aggregation function*

Syntax 1: $[(g_0, (a_0, a_1, \ldots, a_n))]>c$

Result: Return a dummy activity *c*, which is a result of the aggregation of a set of branches starting from a split gateway g_0, and can be delegated by activity a_0, a_1,..., a_n. An EMPTY can be expressed in an activity list in the same way as an activity name in order to locate a dummy branch under a split gateway g_0. For example, [g, (a_0, a_1, EMPTY)]>*c* means that aggregation of branches having activity a_0 and a_1, and a dummy branch in a split gateway *g*.

Condition: This function can be used only if a split gateway g_0 has its couple join gateway so that a coupled Join gateway is not required to be identified in the function. Otherwise, if there is no couple gateway of g_0 (in some cases where g_0 and its structure is not well-constructed), then use branch aggregation Syntax 2 below.

Syntax 2: $[(g_0, g_1, (a_0, a_1, \ldots, a_n))]>c$

Result: Similar to Syntax 1, this function returns a dummy activity *c*, which is a result of the aggregation of a set of branches leading from a split gateway g_0 to a join gateway g_1 and can be delegated by activity a_0, a_1,..., a_n.

Syntax 3: $[(a_0, a_1, \ldots, a_n)]>c$

Result: Return a dummy activity *c*, which is a result of the aggregation of a set of branches of the least common split gateway predecessor (LCSP) and the least common join gateway descendent (LCJD) of all activities $\hat{A} = \{a_0, a_1, \ldots, a_n\}$. The function determines the $LCSP(\hat{A})$ and $LCJD(\hat{A})$, and then find a set of branches where each activity in $\hat{A}$ lies on. If $LCSP(\hat{A})$ and $LCJD(\hat{A})$ is located, then those branches are aggregated.

4.3 Use cases

To illustrate how the VDL expression can be used to construct view operations, we show some usage examples, as shown in Table 2.

Table 2. an example of VDL expressions

1. //[*a1//a3*]-
2. /[*a4//a6*]>c1
3. /*a10*/[(*a1.**[^*234*])]-/[*a36*]-/*a4*
4. /*a10*//[(*a7,a8,a23*)]>*a7810*
5. /*a1*//[(*switch1*,(*a12,EMPTY*))]>*aEMPTY*//*a12*
6. //[(*g1,g2*,(*a10,a11*))]>*dummy*//[*a7//a10*]>*a710*

The first statement contains "[*a1//a3*]-" which expresses a sequential range deletion function for all activities lying on the sequential range from *a1* to *a3* existing in anywhere in the process model because it begins with "//".

The second statement contains "/[*a4//a6*]>*c1*" which is a sequential aggregation function aggregating all activities lying on the sequential range from *a4* to *a6* in which a4 is the start activity of the process model. If this is true, then a new dummy activity named *c1* is constructed.

The third expression contains two operations: first one is "[(*a1.*[^234]*)]-" for deleting all activities where the name matched *a1* following by any number of characters except '2' or '3' or '4' where those activities must lie on the path between *a10* and *a36*, and the second one is "[*a36*]-" for deleting single activity *a36* which lies on the path between a set of activities *a1.*[^234]* and *a4*.

The fourth expression contains "[(*a7, a8, a23*)]>*a7810*" which shows the use of branch aggregation function. A branch where activity *a7* sitting in and a branch where an activity *a8* sitting in, and a branch where an activity *a23* sitting in, in which such branches exist anywhere after an activity *a10*, are all aggregated to a new dummy activity named *a7810*.

The fifth one contains "[(*switch1*, (*a12, EMPTY*))]>*aEMPTY*" which shows another branch aggregation function. Branches where an activity *a12* and a dummy branch (*EMPTY*) of a split gateway *switch1* sitting in are aggregated to a new dummy activity named *aEMPTY* only if the *switch1* exists in between the path of an activity *a1* to *a12*.

The last statement contains two aggregation operations in the path. One is "//[(*g1,g2,*(*a10,a11*))]>*dummy*". Two branches of a10 and a11 in a structured activity of a split gateway g1 and a join gateway g2 are aggregated into an activity named *dummy*. Second one is "[*a7//a10*]>*a710*". All activities existing in a path starting from an activity *a6* to an activity *a10* are aggregated into *a710*, but those activities in the path must exist after a structured activity *S(g1,g2)*.

4.4 VDL Limitations

As our VDL is built based on XPath's meta syntax and the use of regular expression semantics, some of the limitations inherit the XPath's limitations such as ordering and reoccurrence problems. Also, not all the standard XPath expressions are supported in the VDL, this includes a query with multi-valued attributes, and we find that this feature is not useful or beneficial to our work from our perspective. As previously discussed in Section 3, we performed a number of model simulations based on different complexity levels of BPEL and PBMN models (including unstructured BPMN models) to test our view framework w.r.t to the correctness and completeness of our consistency rules and the capability and potential limitations of the VDL (by defining various VDL patterns). Apart from those points discussed above, there is no evidence of other known limitations we found from our simulations. Furthermore, at the current stage, only BPMN (via XPDL) and BPEL modeling languages are supposed as these two languages are mostly used by both academics and industry in practice for defining a business process from both conceptual (by BPMN) and implementation (by BPEL) aspects. Other languages such as YAWL and EPC are less prominent to the industry however these languages can be translated to work on our framework (i.e., conversion between different process modeling languages are possible using existing model transformation techniques as briefly discussed in Section 2.2). All in all, we believe that these limitations do not significantly affect the usability and effectiveness of our framework and they can be addressed in our future work.

5. UNIFLEXVIEW SYSTEM

This section introduces and discusses our system architecture and implementation of the proposed process view framework.

5.1 UniFlexView's System Architecture

The critical part of *UniFlexView* system architecture is process-view transformation. The architecture provides an overview of the system, components required for the system, procedures, and rules for operating the system. The purposed system aims to generate a defined set of process views from a given original business process model. All the constructed process views are guaranteed *consistent* with their original process model and all their inherited process views. Process modelers can define a set of process view operations for each constructed process view. Defining process view operations can be done by expressing the operations in a *Process View Definition Language* (PVDL) document, which contains a set of views, and corresponding VDL expressions to be operated. Apart from that, we also use a *Process View Transformation Definition* (PVTD) document as an alternative to defining a low-level language that can be used by the system. However, a PVTD file is not used if a PVDL file is present. The structure of a *PVDL/PVTD* document is developed based on the XML markup language, which makes the file content simplified, readable, editable, and well-structured. The overall process view system architecture is depicted in Figure 11.

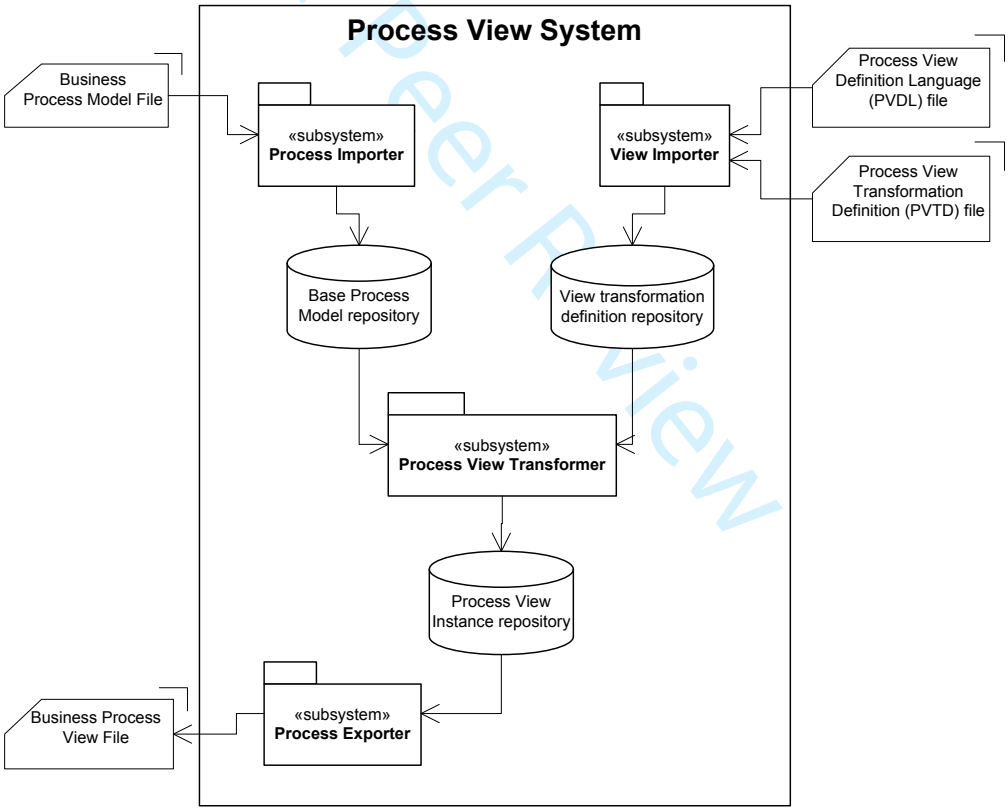


Figure 11. UniFlexView System Architecture

In a PVDL/PVTD document, a process modeler defines a set of view operations for a particular process model. When the system starts, the document is loaded and stored in the View Transformation Definition repository. Then, the transformation process is initiated by the Process View Transformer, which is the heart of the process view

system. After the transformation is completed, the resulted process view is generated and saved into a designated file by the Process Exporter system. The Process view Transformer applies the process view construction methodology and the concept of process view consistency altogether to produce a correct and consistent process view model.

5.1.1 Defining views in a PVDL document

The content of a PVDL document describes how process views are generated based on an input process model. In each process view model defined in a PVDL document, there is a set of view operations to construct such that the process view model. Table 3 illustrates the Document Type Definition (DTD) for a PVDL document.

Table 3. Process View Definition Language (PVDL) document Structure

```
<!DOCTYPE pvdl [
 <!ELEMENT pvdl (view+)>
 <!ELEMENT view (vdl+)>
 <!ELEMENT vdl EMPTY>
 <!ATTLIST pvtl
    name CDATA #REQUIRED
    schema (BPEL | XPDL) #REQUIRED
    processname CDATA #REQUIRED
    processfile CDATA #REQUIRED>
 <!ATTLIST view
    name CDATA #REQUIRED
    parentview CDATA #REQUIRED
    viewfile CDATA #IMPLIED
    aggtargetelement CDATA #REQUIRED>
 <!ATTLIST vdl
    exp CDATA #REQUIRED>
]>
```

Table 4. an example of a PVDL document

```
<pvdl name="PVDL1" schema="BPEL" processname="process1" processfile="process1.bpel">
  <view name="v1" parentview="Root-View" viewfile="process1-v1.bpel" aggtargetelement="empty">
    <vdl exp="/[a1//a3]- " />
    <vdl exp="/[a4//a6]>c1" />
  </view>
  <view name="v2" parentview="v1" viewfile="process1-v2.bpel" aggtargetelement="empty">
    <vdl exp="/a10/[(a1.*[^234])]-/[a36]-/a4" />
    <vdl exp="/a10/[(a1[234])]-/[a36]-/a4" />
  </view>
  <view name="v3" parentview="v2" viewfile="process1-v3.bpel" aggtargetelement="empty">
    <vdl exp="/[switch1]>aSwitch1" />
  </view>
  <view name="v4" parentview="v3" viewfile="process1-v4.bpel" aggtargetelement="empty">
    <vdl exp="/a10/[(a7,a8,a23)]>a7810" />
    <vdl exp="/a1/[(switch1,(a12,EMPTY))]>aEMPTY" />
  </view>
</pvdl>
```

From Table 4, the PVDL document defines four process views: *v1, v2, v3*, and *v4*. Each view has its own set of PVDL expressions, defined in a <vdl> tag, as required for a transformation from a base process model to a process view, or from an existing process view to a new process view.

A process view *v1* derived from the base process model contains two PVDL expressions. The first one is "[*a1//a3*]-" which expresses a sequential range deletion function for all activities lying on the sequential range from *a1* to *a3*. The second one is "/[*a4//a6*]>*c1*" which is a sequential aggregation function aggregating all activities lying on the sequential range from *a4* to *a6,* then it constructs a new dummy activity named *c1*.

A process view *v2* derived from the process view *v1* contains two PVDL expressions. The first expression contains two deletion operations. One is "[(*a1.*[^234]*)]-" for deleting all activities where their name match *a1* following by any number of characters except '2' or '3' or '4', another one is "[*a36*]-" for deleting single activity *a36*. The second expression contains two deletion operations, the first one is "[(*a1*[234]*)]-", for deleting all activities where their names match *a1* following by only single character '2' or '3', or '4', and the second one is "[*a36*]-"for deleting the activity *a36*. Since the activity *a36* is already deleted in the first expression, hence no action is taken place by the second expression.

A process view *v3* derived from the process view *v2* contains only one PVDL expression which is "/[*switch1*]>*aSwitch1*". It expresses a structure aggregation function used for aggregating a whole structure of a split gateway named *switch1* into a new dummy activity named *aSwitch*.

A process view *v4* derived from the process view *v3* contains two PVDL expressions. The first one is "[(*a7, a8, a23*)]>*a7810*" which shows the use of the branch aggregation function. A branch where an activity *a7* sitting in and a branch where an activity *a8* sitting in, and a branch where an activity *a23* sitting in are all aggregated to a new dummy activity named *a7810*. The second one is "[(*switch1*, (*a12, EMPTY*))]>*aEMPTY*" which shows another branch aggregation function. Branches where an activity *a12* and a dummy branch (*EMPTY*) of a split gateway *switch1* sitting in are aggregated to a new dummy activity named *aEMPTY*.

5.2 Case Studies

In this section, two real-world BPEL and BPMN business process models are used as case studies. There is a series of view operations in one VDL expression to demonstrate how the operations perform on activities and their outcomes. Our proposed view consistency rules are applied to ensure that every result process view model is consistent with its underlying process models.

5.2.1 A case study for BPEL process model

This section presents a case study using a customer service process model based on BPEL, as shown in Table 5 followed by demonstrating a variety of VDL expressions that are applied to the base process model and its generated views.

Table 5. Customer Service BPEL process code

```
<process name="CustomerService">
 <sequence>
  <invoke name="SendOrder" />
          <switch name="Cancel_order?">
           <case condition="not cancel">
                   <sequence>
                           <receive name="Receive_Confirmation"/>
                                   <switch name="re issue order?">
                                           <case condition="to re issue">
                                                   <invoke name="re-issue order"/>
                                           </case>
                                           <case condition="not re issue">
```

```
                                                    <sequence>
                                                        <invoke name="arrange payment"/>
                                                        <invoke name="pay order"/>
                                                    </sequence>
                                                </case>
                                            </switch>
                                        </sequence>
                    </case>
                    <case condition="to cancel">
                            <sequence>
                                    <invoke name="Send Cancellation"/>
                                    <receive name="Receive Confirmation"/>
                            </sequence>
                    </case>
  </switch>
  </sequence>
</process>
```

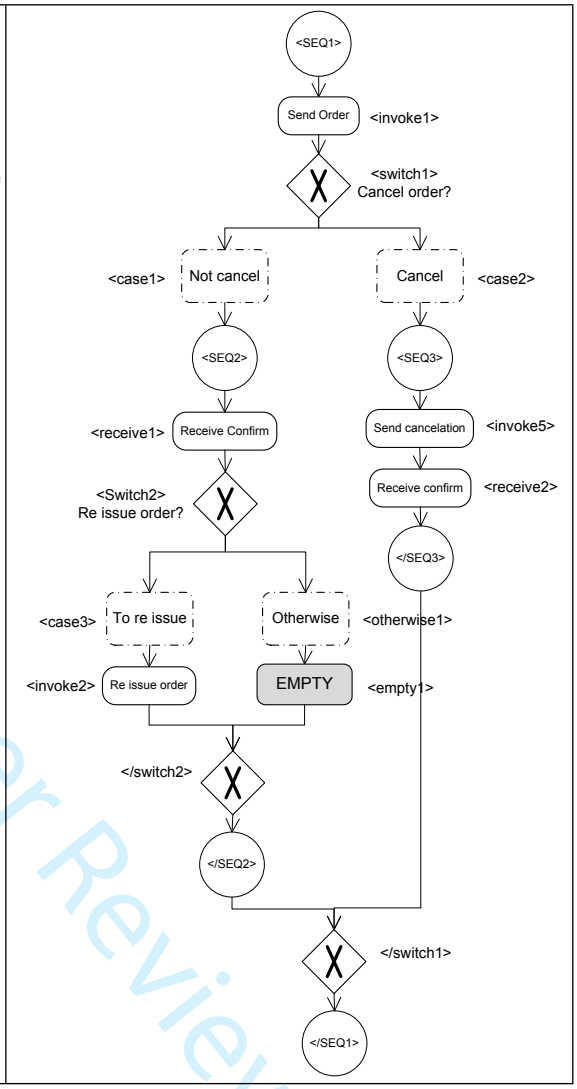
Table 6. An example of view construction for BPEL process

VDL expression and explanation	Transformed process view

24

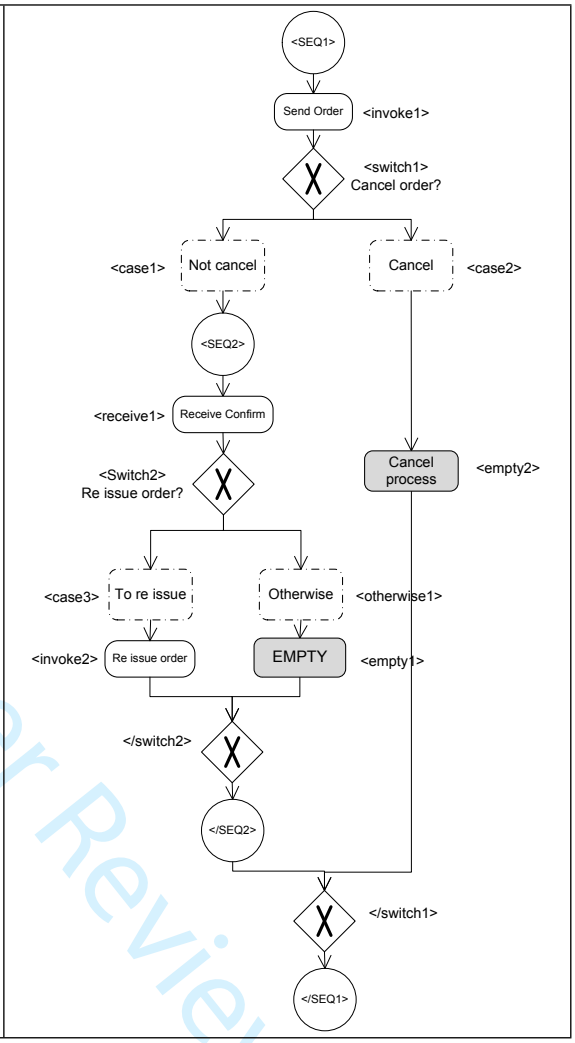
View 1： derives the base process
//[Arrange payment//Pay Order]-

Hide all activities in a sequential range starting at “Arrange payment” activity and ending at “Pay Order” activity.
Two activities are found in the range, so they are removed.
SEQ4 is collapsed.
CASE4 branch contains no activity, so it is to be collapsed, but one dummy branch must exist, so The new branch of OTHERWISE1 is created with an EMPTY activity in the branch.



View 2 : derives View 1
//[Send cancelation//Receive confirm]>Cancel process

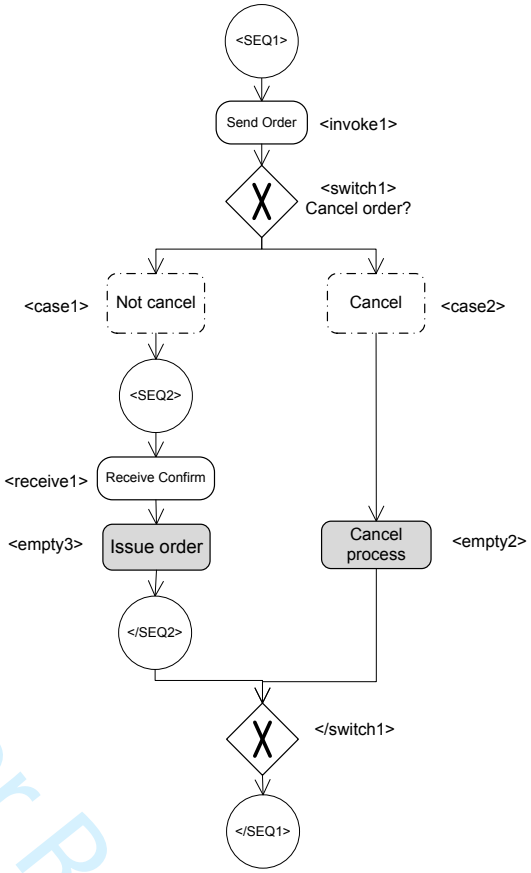
Aggregate all activities in a sequential range starting at “Send cancelation” activity and ending at “Receive confirm” activity, then create the dummy activity named “Cancel process”. Two activities are found in the range, so they are aggregated into the “Cancel process” activity. SEQ3 contains only one activity, so it is collapsed.



26

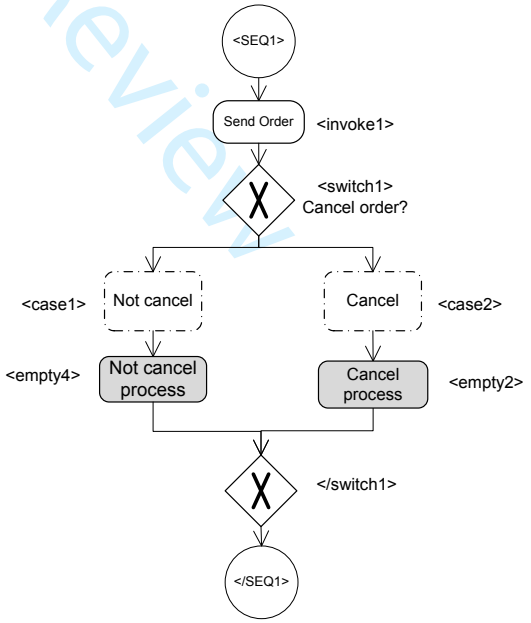
View 3 : derives View 2
//[(Switch2, (Reissue order, EMPTY)]>Issue order

Aggregate two branches of Switch2, one contains the "Reissue order" activity and another one is a dummy branch, into the new dummy activity named "Issue order".
Two branches are found in Switch2 and then they are aggregated.
Switch2 contains only one branch, so it is collapsed.



View 4 : derives View 3
//[SEQ2]>Not cancel process

Aggregate a whole structure of SEQ2 into a new dummy activity named "Not cancel process"
A sequence structure SEQ2 is found, so it is aggregated.



5.2.2 A case study for BPMN process models

This section presents a BPMN case study, which is a problem resolving process, shown in Figure 12. We define a group of VDL expressions for generating different process views for the process model and its constructed views.

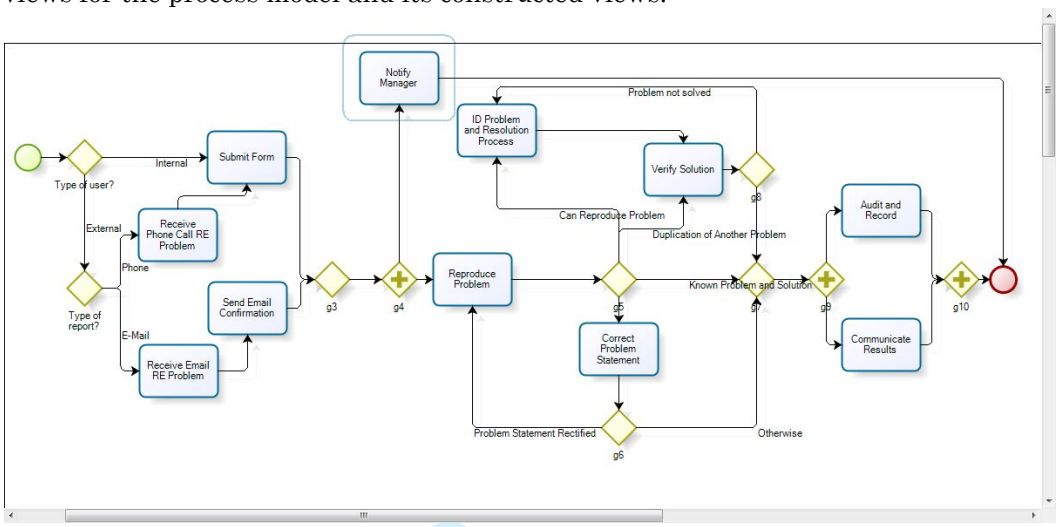


Figure 12. A problem resolver process model

Process View 1: derives from an original business process

VDL Expression: //[Receive Email RE Problem//Send Email Confirmation]-

Explanation: Hide all activities lying on the path between the start activity named “Receive Email RE Problem” and the end activity named “Send Email Confirmation”.

Result: There are only two activities, “Receive Email RE Problem” and “Send Email Confirmation”, lying on that sequential path, thus those are deleted and a dummy branch is then created to link an XOR-Split gateway named “Type of report?” to an XOR-Join gateway named “g3”. The result is shown in Figure 13.

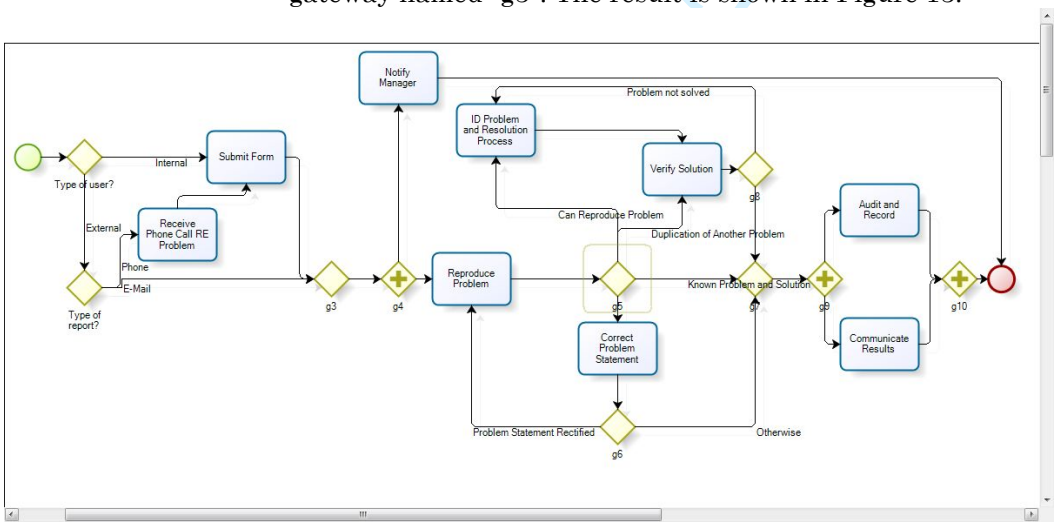


Figure 13. Process View 1

Process View 2: derives from the process view 1

VDL Expression: //[Reproduce Problem]-
Explanation: Hide one activity named “Reproduce Problem”.
Result: Although there is only a single activity named “Reproduce Problem” is to be hidden, however, this activity contains an embedded XOR-Join gateway, therefore the gateway is detached from the activity and then an activity can be deleted. The result is shown in Figure 14.

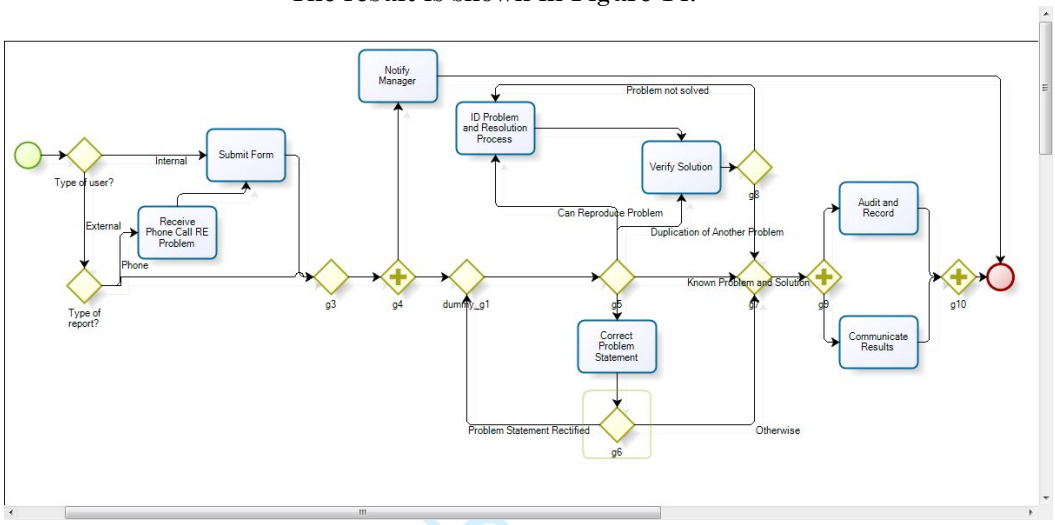


Figure 14. Process View 2

Process View 3: derives from the process view 2
VDL Expression: //[Type of user?//g3]>dummy_a1
Explanation: Aggregate all activities between the path starting at activity named “Type of user?” and ending at activity named “g3”.
Result: Both “Type of user?” and “g3” are XOR typed gateways, then the structure aggregation with the enclosed block verification process is to be taken place for this case. If it is valid, the new dummy activity named “dummy_a1” is created and placed to the process view. The result is shown in Figure 15.

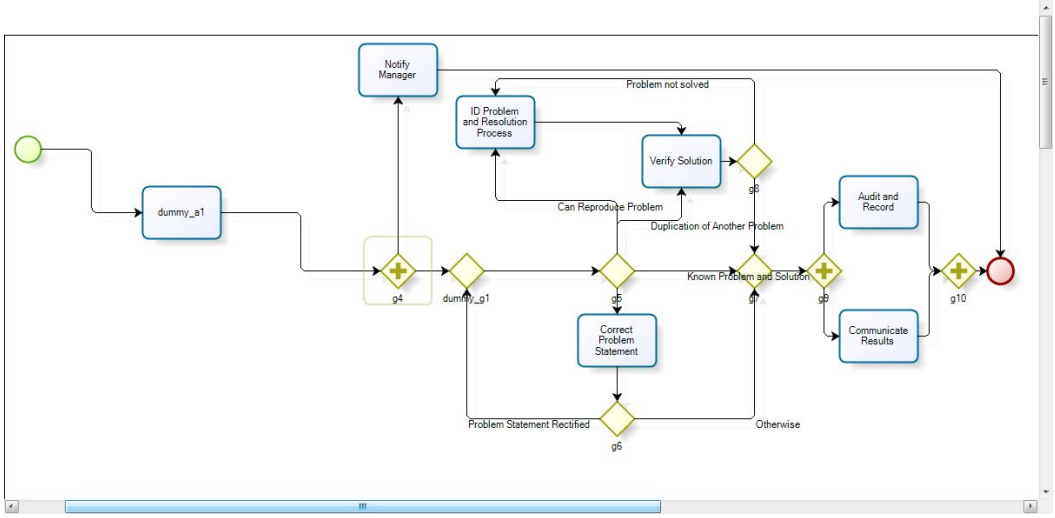


Figure 15. Process View 3

Process View 4: derives from the process view 3

VDL Expression: //[(g9,g10,(Audit and Record, Communicate Results))]>dummy_a2

Explanation: Aggregate two branches, containing activities named “Audit and Record” and “Communicate Results”, of a Split gateway named “g9” and a join gateway named “g10”.

Result: The branch having an activity named “Audit and Record” and the other branch containing an activity named “Communicate Results” are aggregated into a new dummy activity named “dummy_a2”. In this case, there is no other branch left after the aggregation, thus the structure of Split/Join gateways g9 and g10 is then collapsed. The result is shown in Figure 16.

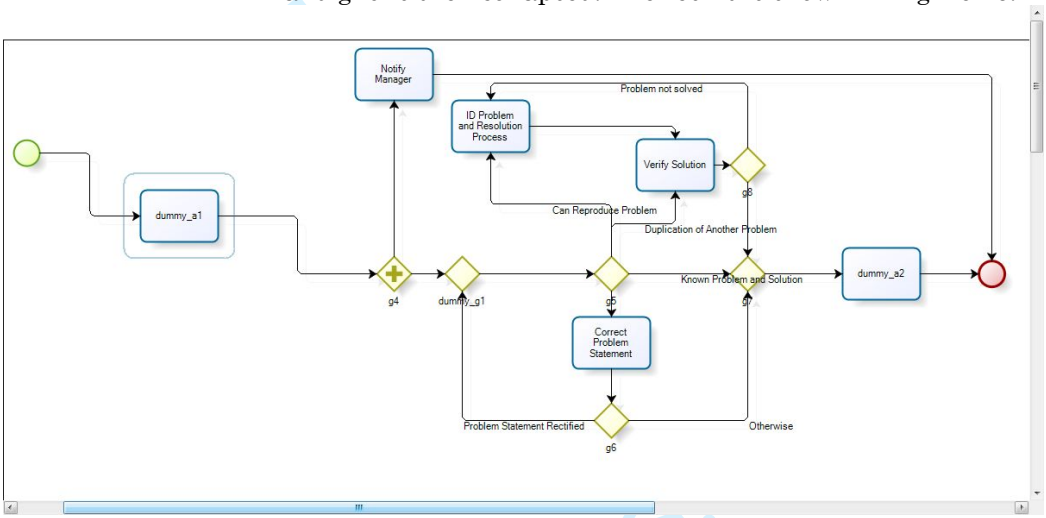


Figure 16. Process View 4

6. CONCLUSION AND FUTURE WORK

Our research addresses the unsolved issues in the area of business process view construction approach, i.e., there is a need for a unified process view framework on the two different process modeling standards: BPEL and BPMN. The research focuses on how process views can be automatically realized and deal with the different nature of the two standards in a consistency way. Our framework consists of a comprehensive set of process components and related functions to construct a process view, as well as necessary consistency rules required to guarantee the consistent structure between a process view and its base business process model. With our framework, process modelers exercise on a simplified language to defining process views regardless of modeling standards they use. In the future, further investigation is to be carried out to refine the organization-oriented view solution with an extended set of inter-organizational process view consistency rules and an extension to current process view operations and VDL expression.

REFERENCES

1. ‘OASIS Web services Business Process Execution Language (BPEL) version 2.0, 2007, *OASIS*, April, http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf

2. 'Business Process Execution Language for Web Services Version 1.1', S. Thatte, et al., *BEA, IBM, Microsoft, SAP and Siebel*, May 2003. http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf
3. 'Regular Expression, The Single UNIX ® Specification, Version 2', 1997, *The Open Group*, http://www.opengroup.org/onlinepubs/007908799/xbd/re.html
4. ISO/IEC 14977:1996, Information technology - Syntactic metalanguage - Extended BNF
5. 'WFMC XML Process Definition Language (XPDL) version 2.0', 2005, *Workflow Management Coalition*, October, WFMC-TC-1025, http://www.wfmc.org/standards/documents/TC-1025_xpdl_2_2005-10-03.pdf
6. 'XML Path Language (XPath) 2.0', 2007, *World Wide Web Consortium*, January, http://www.w3.org/TR/xpath20/
7. 'Extensible Markup Language (XML) 1.0 (Fourth Edition)', 2006, *World Wide Web Consortium*, August, http://www.w3.org/TR/2006/REC-xml-20060816/
8. BizAgi Process Modeler, 2008, *BizAgi Ltd*, July, http://www.bizagi.com/eng/
9. Lin, D, "Compatibility Analysis of Local Process Views in Interorganizational Workflow", 2007, The 9th IEEE International Conference on E-Commerce Technology and The 4th IEEE International on Enterprise Computing, *IEEE computer Society,* July, pp.149-156.
10. van der Aalst, W.M.P., 'Interorganizational Workflows: An Approach based on Message Sequence Charts and Petri Nets', 1999, *Systems Analysis - Modelling - Simulation*, vol. 34, pp.335-367.
11. Liu, D.-R., Shen, M, 'Workflow modeling for Virtual Processes: An Order-Preserving Process-View Approach', 2003, *Information systems*, vol. 28, pp. 505-532.
12. Eshuis, R., Grefen, P., 'Constructing Customized Process Views', 2008, *Data & Knowledge Engineering*, vol. 64, pp. 419-438.
13. van der Aalst, W.M.P., 'Inheritance of interorganizational workflows to enable business-to-business e-commerce', 2002, *Electronic Commerce Research*, vol. 2(3), pp. 195-231.
14. Zha, H, Yang, Y, Wang, J, and Wen, L, 'Transforming XPDL to Petri Nets', 2008, *BPM 2007 Workshop*, LNCS 4928, pp. 197-2007.
15. Dun, H, Xu, H, and Wang, L, 'Transformation of BPEL Processes to Petri Nets', 2008, 2'nd IFIP/IEEE International Symposium on Theoretical Aspects of Software Engineering, *IEEE computer Society*, June, pp. 166-173.
16. van der Aalst, W.M.P., Lassen, K.B., 'Translating Unstructured Workflow Processes to Readable BPEL: Theory and Implementation', 2008, *Information and Software Technology*, vol. 50, no.3, February, pp. 131-159.
17. van der Aalst, W.M.P., Ouyang, C, Dumas, M, and ter Hofstede, A.H.M., 'Pattern-Based Translation of BPMN Process Models to BPEL Web Services', 2007, International Journal of Web Services Research.
18. van der Aalst, W.M.P., Ouyang, C, Dumas, M, and ter Hofstede, A.H.M., 'From BPMN Process Models to BPEL Web Services', 2006, Proceeding of the 4th International Conference on Web Services, *IEEE computer Society*, September, pp.285-292.
19. Yuan, P, Jin, H, Yuan, S, Cao, W, and Jiang, L, 'WFTXB: A Tool for translating Between XPDL and BPEL', 2008, The 10th IEEE International Conference on High Performance computing and Communications, *IEEE computer Society,* September, pp. 647-652.
20. Ouyang, C, Verbeek, E, van der Aalst, W.M.P., Breutel S, Dumas, M, and ter Hofstede, A.H.M., 'Formal Semantics and Analysis of Control Flow in WS-BPEL', 2007, *Science of Computer Programming*, vol.67, no. 2-3, July, pp. 162-198.
21. Eshuis, R, Grefen, P, 'Structural matching of BPEL Processes', 2007, Fifth European Conference on Web services, Nov, *IEEE computer Society*, Nov, pp.171-180.
22. Zhao, X., Liu, C., and Yang, Y., 'An Organisational Perspective on Collaborative Business Processes', 2005, In Proceedings of the 3rd International Conference on Business Process Management, Nancy, pp.17-31.
23. 'Business Process Modeling Notation, V1.1', 2008, *Object Management Group*, January, http://www.omg.org/spec/BPMN/1.1/PDF

24. Mendling, J, Lassen, K.B., and Zdum, U, 'Transformation Strategies between Block-Oriented and Graph-Oriented Process Modelling Languages', 2006, *Multikonferenz Wirtschaftsinformatik 2006. Band 2*, pp.297-312.
25. White, S, 'Using BPMN to Model a BPEL Process', 2005, *BPTrends*, vol.3 (3), pp.1-18.
26. Shapiro, R.M., 'XPDL 2.0: Integrating Process Interchange and BPMN', 2006, 2006 Workflow handbook including business process management, *Future Strategies Inc*, USA, pp.183-194.
27. van der Aalst, W.M.P., 'Patterns and XPDL: A Critical Evaluation of the XML Process Definition Language', 2003, QUT Technical report FIT-TR-2003-06, *Queensland University of Technology*, Brisbane, Australia.
28. Hornung, T, Koschmider A, and Mendling, J, 'Integration of heterogeneous BPM Schemas: The Case of XPDL and BPEL', 2006, Technical Report JM-2006-03-10, *Vienna University of Economics and Business Administration*.
29. Van Dongen, B.F., Mendling, J, van der Aalst, W.M.P., 'Structural Patterns for Soundness of Business Process Models', 2006, Proceedings - IEEE International Enterprise Distributed Object Computing Workshop, *EDOC 2006*, pp.116-125.
30. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B, and Barros, A.P., 'Workflow Patterns', 2003, *Distributed and Parallel Databases*, vol.14(1), pp. 5-51.
31. van der Aalst, W.M.P., Lassen, K.B., 'WorkflowNet2BPEL4WS: A tool for translating unstructured workflow processes to readable BPEL', 2006, *Lecture Notes in Computer Science*, vol. 4275, pp. 127-144.
32. Chiu, D.K.W., Cheung, S.C., Till, S, Karlapalem, K, Li, Q, and Kafeza, E, 'Workflow View Driven Cross-Organizational Interoperability in a Web Service Environment', 2004, *Information Technology and Management*, Kluwer Academic Publishers, vol.5, pp. 221-250.
33. Schulz, L.A.,Orlowska, M.E., 'Facilitating cross-organisational workflows with a workflow view approach', 2004, *Data & Knowledge Engineering*, vol.51, pp.109-147.
34. Zhao, J.L., 'Workflow Management in the Age of e-Business (Tutorial)', 2002, In Proceedings of International Conference on System Sciences.
35. Perrin, O., and Godart, C., 'A Model to Support Collaborative Work in Virtual Enterprises', 2004, *Data & Knowledge Engineering*, vol. 50, pp.63-86.
36. Liu, D, Shen, M, 'Modeling Workflows with a Process-View Approach', 2001, Database Systems for Advanced Applications, *IEEE computer Society,* April, pp.260-267.
37. Chebbi, I, Dustdar, S, and Tata, S, 'The view-based approach to dynamic inter-organizational workflow cooperation', 2006, *Data & Knowledge Engineering*, vol. 56, pp.139-173.
38. Shapiro, R, 'A Technical Comparison of XPDL, BPML and BPEL4WS', 2002, *Cape Vision Inc.*, December, viewed 13 November 2008, http://www.businessprocesstrends.com/deliver_file.cfm?fileType=publication&fileName=Comparison%20of%20XPDL%20and%20BPML_BPEL%2012-8-02111.pdf.pdf
39. Zhao, X, Liu, C, Sadiq, W, Kowalkiewicz, M, and Yongchareon, S, 'On Supporting Abstraction and Concretisation for WS-BPEL Business Processes', 2008*, Centre for Information Technologies Research*, Swinburne University of Technologies, Melbourne, and *SAP Research Centre*, Brisbane.
40. Yongchareon, S, Zhao, X, 'Technical Report of FlexView Manual – Support for Business View Operations', 2008, *Centre for Information Technologies Research*, Swinburne University of Technologies, Melbourne, and *SAP Research Centre*, Brisbane.
41. Jiang, P., Shao, X., Gao, L., Qiu, H., and Li, P.: A process-view approach for cross-organizational workflows management, *Advanced Engineering Informatics*, 2010, vol. 24, pp. 229–240
42. Eshuis, R., Norta, A., Kopp, O., and Pitkänen, E.: Service Outsourcing with Process Views, *IEEE Transactions On Services Computing*, 2015, vol. 8, no. 1, pp. 136-154.

43. X. Zhao, C. Liu, W. Sadiq, M. Kowalkiewicz, and S. Yongchareon.: Implementing process views in the web service environment. *World Wide Web*, 2011, vol. 14, no. 1, pp. 27–52.
44. X. Zhao, C. Liu, S. Yongchareon, M. Kowalkiewicz, and W. Sadiq. 2015. Role-based process view derivation and composition. *ACM Transactions on Management Information Systems* 6, 2 (2015), 7:1–7:24.
45. Polyvyanyy, A., García-Bañuelos, L., Dumas, M.: Structuring acyclic process models, Information Systems, Volume 37, Issue 6, September 2012, Pages 518-538
46. Yongchareon S., Liu C., Zhao X., Kowalkiewicz M. (2010) BPMN Process Views Construction. In: Kitagawa H., Ishikawa Y., Li Q., Watanabe C. (eds) Database Systems for Advanced Applications. DASFAA 2010. Lecture Notes in Computer Science, vol 5981. Springer, Berlin, Heidelberg
47. J. Vanhatalo, H. Volzer, J. Koehler, The refined process structure tree, Data Knowl. Eng. 68 (9) (2009), pp. 793–818
48. R. Eshuis, A. Norta, R. Roulauxa, Evolving process views, Information and Software Technology, 2016, vol. 80, pp. 20-35.
49. J. Küster, H. Völzer, C. Favre, M.C. Branco, K. Czarnecki, Supporting Different Process Views through a Shared Process Model, in the 9th European Conference on Modelling Foundations and Applications, 2013, LNCS 7949, pp. 20–36.