

Zhao, X., & Liu, C. (2007). Version management in the business process change context.

Originally published in *Proceedings of the 5th International Conference on Business Process Management (BPM 2007), Brisbane, Australia, 24–28 September 2007*.

Lecture notes in computer science (Vol. 4714, pp. 198–213). Berlin: Springer.

Available from: [http://dx.doi.org/10.1007/978-3-540-75183-0\\_15](http://dx.doi.org/10.1007/978-3-540-75183-0_15)

Copyright © Springer-Verlag Berlin Heidelberg 2007.

This is the author's version of the work, posted here with the permission of the publisher for your personal use. No further distribution is permitted. You may also be able to access the published version from your library. The definitive version is available at <http://www.springerlink.com/>.

# Version Management in the Business Process Change Context

Xiaohui Zhao and Chengfei Liu

{xzhao, cliu}@ict.swin.edu.au  
Centre for Information Technology Research  
Faculty of Information and Communication Technologies  
Swinburne University of Technology  
Melbourne, Australia

**Abstract.** The current business endures a fast changing environment, which drives organisations to continuously adapt their business processes to new conditions. In this background, the workflow version control plays an important role for the change management of business processes. To better handle the versions of evolving workflow process definitions, a new versioning method is introduced in this paper. To capture the dynamics of the workflow evolution, we propose a novel version preserving directed graph model to represent the run time evolution of a workflow process, and devise a series of modification operations to characterise workflow updating on the fly. The extraction of workflow versions from a version preserving graph is also discussed with two different extraction strategies. Particularly, our method allows the execution of multiple workflow instances of different versions within a single graph, and supports the evolutions initiated by temporary changes.

## 1 Introduction

Current varying market opportunities are commented as “Change has become the only certainty.” [1] in nowadays business globalisation background. To stay efficient and effective in such a turbulent environment, organisations are required to adapt their structures and business processes to new conditions continuously [2, 3]. As a response, organisations are seeking for new facilitating technologies to manage their dynamic, expanding and changing business processes [4, 5].

Technically, this trend puts challenges to the issues such as business process updating, instance updating, version control etc. A frequently changing workflow process definition requires dynamic updating without suspending related running workflow instances. Further, a running workflow instance needs to keep up with the changed process definition by evolving to the latest version on the fly. In this scenario, some temporary and parallel changes may cause a lot of workflow variants which will result in various versions of workflow process definitions and their workflow instances. As such, some innovative version management mechanism is in great demand to harmonise the various versions of workflow processes and instances.

The previous work [6] done with other colleagues particularly focused on the handover of the running instances from an old workflow model to a new model, i.e., between only two versions. While, this paper concentrates on the version management in the context of multiple changes to business processes. A novel version preserving directed graph (VPG) model is proposed to represent the version evolution of a workflow process definition, and support the execution of workflow instances belonging to different versions within the same graph. A set of run time modification operations are developed for this VPG model, to support the dynamic updating to a workflow process definition. Two strategies for extracting a workflow process definition of a given version from a VPG are illustrated with formal algorithms, together with a performance analysis.

The remainder of this paper is organised as follows, Section 2 discusses the version issues in workflow process evolvments with a motivating example; a version preserving directed graph model is presented in Section 3, to support business process changes; strategies for dynamically extracting a workflow process definition of a given version are addressed in Section 4, together with a performance analysis on different strategies; Section 5 lists the work related to business process change management, and discusses the advantages of the proposed method; conclusion remarks and future work are given in Section 6.

## 2 Motivating Example

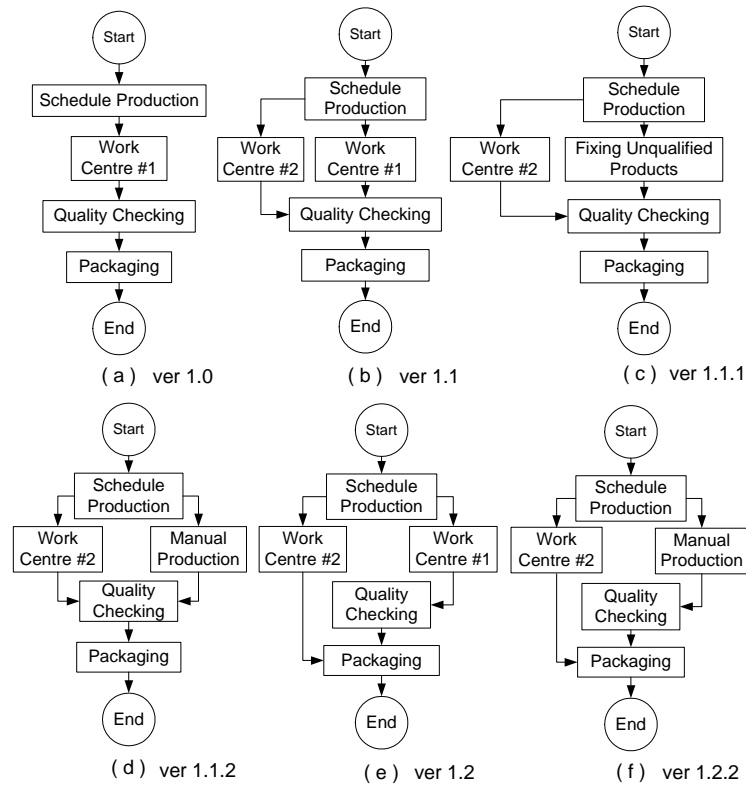
In this section, we use a production business process to demonstrate the process evolvment. The contextual scenario is that a factory owns several pipelines, and at the beginning, each pipeline follows the same workflow process shown in Figure 1 (a). Here, we see that the production process includes several activities: production scheduling, production using a work centre, i.e., work centre #1, quality checking and final packaging. To meet the soaring market demands, the factory may add a parallel work centre, for example work centre #2, to each pipeline for the purpose of increasing the production capability. In this case, the original workflow process upgrades to the one shown in Figure 1 (b).

As this workflow process is shared by multiple pipelines, the workflow process may have variants for different pipelines due to practical situations. For example, sometimes work centre #1 of a pipeline, say pipeline *A*, may come across a technical malfunction, and therefore has to be removed from the pipeline for maintenance. Here, we suppose that pipeline *A* attempts to keep the production output by fixing unqualified products at the absence of work centre #1. Therefore, the workflow process will evolve to the one shown in Figure 1 (c), accordingly.

While for other pipeline, for example pipeline *B*, its work centre #1 may also endure a temporary maintenance, yet it uses manual labour to replace its work centre #1. In this case, the workflow process will evolve to the one shown in Figure 1 (d). Afterwards, a technical upgrading to the work centre #2 of all pipelines may improve the product quality, and the products made by work centre #2 are thus not required to pass the quality checking. Due to the upgrading benefit, the workflow process for other pipelines, except pipeline *A* and *B*, will evolve from Figure 1 (b) to Figure 1 (e),

whilst for pipeline *B*, the workflow process will evolve from Figure 1 (d) to Figure 1 (f). Further, when work centre #1 of pipeline *B* comes back from maintenance, the workflow process for this pipeline evolves to the one shown in Figure 1 (e), as well.

Besides parallel evolvments, a workflow process may possibly go back to a previous copy. For example, after the workflow process for pipeline *A* evolves to Figure 1 (c) and before the upgrading to work centre #2, the workflow process may be changed back to Figure 1 (b) if work centre comes back to work.



**Fig. 1.** Workflow process evolution example

From this example, we see that a workflow process may not simply go along a linear evolution. In fact, the actual evolution is driven by many factors, such as unit replacement, technology shift, external changes, etc. Some of these factors, for example, unit replacement and external changes, may only cause temporary changes of a workflow process. While, some factors, such as technology shift etc., may cause permanent changes. Nevertheless, static workflow process definitions are not acceptable in such evolution scenarios, and more supports for dynamic evolution representation and description are highly needed. In such a business process change context, the workflow version is a direct indicator for the variants of an evolving workflow process, and therefore the representation and manipulation of workflow versions are of significant importance to assist business process changes. Here, we summarise the requirements for version supports as follows:

- Version representation. A version representation method is expected to clearly depict the evolution relation and dependency between versions of a workflow process definition.
- Version transformation. A set of modification operations are expected to transform a workflow process definition from one version to another on the fly.
- Version compatibility. For a workflow process definition, this denotes the compatibility that allows the workflow instances of different versions to be executed at the same time.
- Version extraction. For a workflow process definition, this stands for the process of dynamically deducing a specific version from a mass of version information during the execution period. This feature particularly helps the version change of workflow instances.

The whole lifecycle of business process changes comprises identifying tasks and links to replace, updating workflow processes, updating running workflow instances, together with version control and extraction, etc. Some work in adaptive workflow [7-9] and workflow evolution [10, 11] already addressed the issues of instance updating and process validation. Yet, to our best knowledge, few efforts have been put on the workflow version control in workflow process updating. This paper mainly targets at the process updating and the process version control. A particular versioning method is proposed to represent workflow process version evolution; and a version preserving directed graph model is established to support the dynamic modifications and transformations of workflow process versions, as well as version compatibility. Additionally, two version extraction strategies are discussed with performance analysis.

### **3 Version Control in Business Process Change Management**

This section discusses the version evolution of workflow process definitions. A versioning method is proposed for workflow process definitions, and a version preserving directed graph is used to represent the version evolution of workflow process definitions.

#### **3.1 Workflow process version evolution**

Once a workflow process definition is changed, a new version will be assigned to indicate the changed workflow process definition at this stage. In this way, a workflow process definition may own several versions during its lifecycle.

Currently, there are few standards specifying the versions of workflow process definitions. Some workflow products [12, 13] simply use the incremental numbering system that is widely used in the software development area, for versioning workflow process definitions. In such a versioning system, different numbers denote different versions, and dots separate main versions and sub versions [14]. Though this numbering system can well represent the inheritance relation in software

development, yet it fails to represent some sporadic changes of a workflow process, where the inheritance relation does not exist between versions.

In contrary to the software development, a workflow process definition may evolve along two axes, i.e., the process improvement and the temporary adaptation. The process improvement denotes the permanent evolvement of a workflow process definition driven by technology upgrading or strategy changes; while the temporary adaptation denotes the variation of the workflow process definition driven by unit replacements or other sporadic reasons.

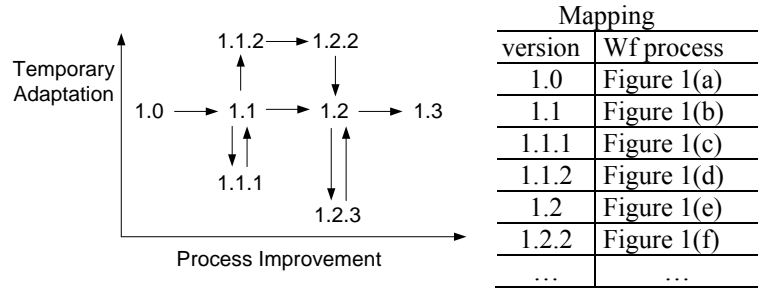
Here, we propose a new versioning method to represent the evolvement of a workflow process definition along the above two axes.

A version of a workflow process definition is defined as a three-digit string separated by dots,  $x.y.z$ , where,

- the first digit  $x$  denotes the major version of the workflow process definition;
- the second digit  $y$  denotes the minor version of the workflow process definition;
- the third digit  $z$  denotes the temporary variation of a workflow process definition.

The first two digits represent the progress of the process improvement; while the last digit denotes the temporary variation of a workflow process definition.

Figure 2 illustrates the version evolvement of the workflow process definition example discussed in Section 2, along the two axes.



**Fig. 2.** Workflow process evolution figure

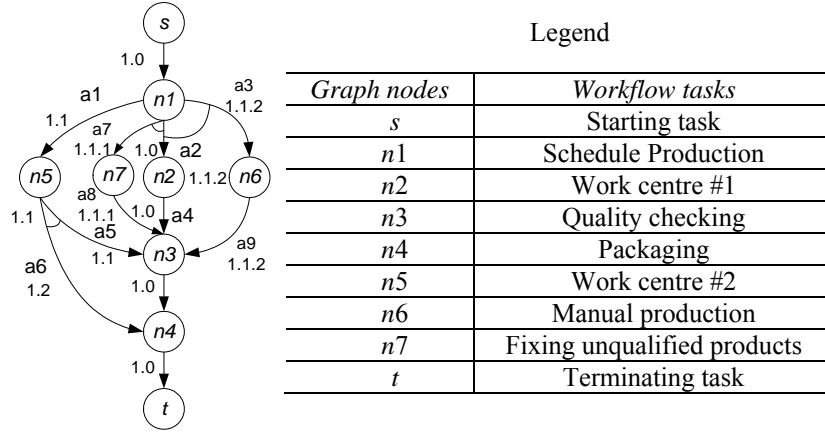
In Figure 2, the horizontal axis denotes the process improvement, and the vertical axis denotes the temporary adaptation. The mapping table illustrates the relation between versions and corresponding workflow process definitions. Here, we name the start version as the *base version* for all other versions. In Figure 1, version 1.0 is the base version. For version  $v$ , a version that contains a larger number for at least one of the first two digits and the other is no less than  $v$ 's counterpart is called  $v$ 's *subsequent version*. For examples, versions 1.2, 1.2.2 are subsequent versions of version 1.1

In Figure 2, each arrow represents an evolvement from one version to another. For any version, there always exists a path leads from the base version to this version in this evolution figure. Take version 1.2.2 as an example, the evolvement starts from base version, 1.0, to 1.1, then arrives to 1.1.2, and finally to 1.2.2. As discussed in the motivating example, a workflow process may have multiple evolution branches. A good example is that from version 1.1, it has three possible evolution branches, i.e.,

to version 1.2, to version 1.1.1 and to version 1.1.2. Choosing which route is dependant on the actual situation.

### 3.2 Version preserving directed graph

Although a workflow process definition may change frequently, we need to minimise the effect to the execution of its workflow instances. Our primary intention is to keep the nodes and arcs of all the versions belonging to the same workflow process definition in a single graph. Figure 3 shows an example of such a graph according to the production workflow process discussed in the previous section. In this graph, each node represents a workflow task, and the versions of nodes and arcs are marked aside as labels.



**Fig. 3.** An example of VPG

Obviously, there may exist exclusive branches between different versions. For example, from node  $n5$  in Figure 3, a workflow instance of version 1.1 is only allowed to go through arc  $a5$ , yet a workflow instance of version 1.2 or 1.2.2 can only go through arc  $a6$ . In this situation, we call that arc  $a5$  and arc  $a6$  are in an exclusive relation.

In this methodology, we extend the conventional directed graph with enhancements for version control to model workflow process definitions in the business process change context. In particular, a version set, a version mapping and a binary relation are designed for the version preservation purpose. We name the extended graph as a *version preserving directed graph (VPG)*.

A VPG for a workflow process  $p$ , can be defined as a tuple  $(\mathcal{N}, \mathcal{A}, \mathcal{V}, f, \mathcal{R})$ , where,

- $\mathcal{N}$  is the set of nodes, where each node  $v \in \mathcal{N}$ , represents a workflow task of  $p$ . Additionally, there must exist one and only one starting node  $s \in \mathcal{N}$ , whose incoming degree is 0; and one and only one terminating node  $t \in \mathcal{N}$ , whose outgoing degree is 0. This means that a workflow process must have one and only one starting task, and one and only one terminating task;

- $\mathcal{A}$  is the set of arcs, where each arc  $a \in \mathcal{A}$ , represents a link connecting two workflow tasks of  $p$ ;
- $\mathcal{V}$  is the set of version numbers, such as “1.1”, “1.2”, “1.2.2” etc.;
- $f: \mathcal{N} \cup \mathcal{A} \rightarrow \mathcal{V}$  is a mapping, which assigns proper version number to each node and arc in the graph;
- $\mathcal{R}$  is a binary relation  $\{ ( a1, a2 ) \mid a1, a2 \in \mathcal{A} \wedge a1 \text{ and } a2 \text{ are in the exclusive relation } \}$ . With this binary relation, the exclusive relation between the arcs in a VPG can be easily represented. Note, here  $\mathcal{R}$  only records the exclusive relation that are caused by versioning, not by business constraints.

In general, this graph keeps the version information in mapping  $f$ , stores the exclusive relation between arcs using relation  $\mathcal{R}$ , and represents the workflow structure with nodes and arcs.

Particularly, the VPG model tries to minimise the information to store, by dropping all deducible information. For example, arc  $a6$  marked version 1.2 in Figure 3, represents the evolvment triggered by work centre #2’s upgrading, which can be shared by the workflow processes of version 1.1.2 and version 1.1. Therefore, there is no need to create a new arc (exclusive to  $a6$ ) from  $n5$  to  $n4$  with version 1.2.2. We will see that the workflow process definition of version 1.2.2 is deducible from the information for versions 1.0, 1.1, 1.2 and 1.1.2.

Based on the VPG for a specific workflow process, we can determine different versions for different workflow instances at any time during the execution, by following three rules:

**Rule (1)** Version  $v$  cannot include the arcs and nodes with  $v$ ’s subsequent versions.

**Rule (2)** The arcs and nodes with the version in the form of  $x.y.z$  ( $z \neq \text{null}$ ) will not be included in the version in the form of  $u.v.w$ , where  $w \neq z$ .

For example, in Figure 3,  $n1$ ,  $a1$ ,  $a5$  and  $a7$  etc. are not includable in version 1.0, and  $a6$  is not includable in version 1.1, because of Rule 1.  $a7$ ,  $a8$  and  $n7$  are not includable in either version 1.1.2 or version 1.2.2, because of Rule 2.

**Rule (3)** The selection of an arc, with regards to the version in the form of  $x.y.z$ , from the set of arcs with an exclusive relation, is subject to the order of its version in the following priority list:

1. versions in the form of  $x.y.z$ ;
2. versions in the form of  $u.v.z$  ( $u \neq x$  or  $v \neq y$ );
3. versions in the form of  $x.y$ ;
4. versions in the form of  $x.v$  ( $v < y$ ,  $v$  is the closet to  $y$ );
5. versions in the form of  $u.v$  ( $u < x$ ,  $u$  is the closest to  $x$ , or  $v$  is the largest if  $u$  is the same).

The arcs with a version that is not listed in the priority list will not be considered.

For example, arcs  $a2$ ,  $a3$  and  $a7$  are in an exclusive relation in Figure 3. For version 1.2.2, the selection priority is  $a3 > a2$ , while  $a7$  is not considered.

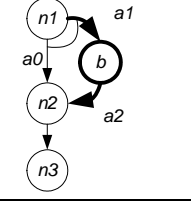
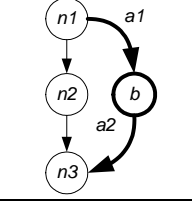
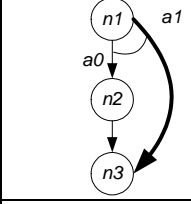
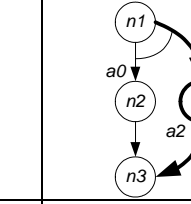


### 3.3 Run time operations

Obviously, it is unacceptable to suspend all running workflow instances for updating, thus all modifications are required to perform on the fly. Furthermore, such run time modifications are expected to be information preserved for previous versions. This is required to guarantee the consistency between workflow instances and the log information that is maintained by a workflow management system. The log records may have used the workflow process definition of previous versions, and the information about all these versions should be preserved. The loss of previous version information may disable the restoration of a workflow process back to a previous version after a temporary change.

In short, a run time modification operation should be *dynamic*, *information preserved* and *restorable*. In Table 1, we list the node modification operations, which satisfy all the three requirements.

Table 1. Node modification operations

Node modification operations			
Add a node		Remove a node	Replace a node
Sequential inserting	Parallel inserting		
			
$b \rightarrow \mathcal{N}$ ; <b>create arc <math>a1 = (n1, b)</math></b> , $a2 = (b, n2)$ ; $a1 \rightarrow \mathcal{A}$ ; $a2 \rightarrow \mathcal{A}$ ; $\text{"1.1"} \rightarrow \mathcal{V}$ ; $(a1, \text{"1.1"}) \rightarrow f$ ; $(b, \text{"1.1"}) \rightarrow f$ ; $(a2, \text{"1.1"}) \rightarrow f$ ; $(a0, a1) \rightarrow \mathcal{R}$	$b \rightarrow \mathcal{N}$ ; <b>create arc <math>a1 = (n1, b)</math></b> , $a2 = (b, n3)$ ; $a1 \rightarrow \mathcal{A}$ ; $a2 \rightarrow \mathcal{A}$ ; $\text{"1.1"} \rightarrow \mathcal{V}$ ; $(a1, \text{"1.1"}) \rightarrow f$ ; $(b, \text{"1.1"}) \rightarrow f$ ; $(a2, \text{"1.1"}) \rightarrow f$	<b>create arc <math>a1 = (n1, n3)</math></b> ; $a1 \rightarrow \mathcal{A}$ ; $\text{"1.1"} \rightarrow \mathcal{V}$ ; $(a1, \text{"1.1"}) \rightarrow f$ ; $(a0, a1) \rightarrow \mathcal{R}$	$b \rightarrow \mathcal{N}$ ; <b>create arc <math>a1 = (n1, b)</math></b> , $a2 = (b, n3)$ ; $a1 \rightarrow \mathcal{A}$ ; $a2 \rightarrow \mathcal{A}$ ; $\text{"1.1"} \rightarrow \mathcal{V}$ ; $(a1, \text{"1.1"}) \rightarrow f$ ; $(b, \text{"1.1"}) \rightarrow f$ ; $(a2, \text{"1.1"}) \rightarrow f$ ; $(a0, a1) \rightarrow \mathcal{R}$

In this table, each operation is illustrated by an example graph, where the boldfaced nodes or arcs denote the appended components of latest version, say 1.1 in this example, and an inclined angle between two arcs stands for the exclusive relation between these two arcs. Below each example graph, the corresponding codes are given for the modification operation.

Table 2 lists the arc modification operations.

**Table 2.** Arc modification operations

Arc modification operations		
Add an arc	Replace an arc	Remove an arc
$create\ arc\ a1=(n2, n3);$ $a1 \rightarrow \mathcal{A};$ $"1.1" \rightarrow \mathcal{V};$ $(a1, "1.1") \rightarrow f$	$create\ arc\ a1=(n1, n4);$ $a1 \rightarrow \mathcal{A};$ $"1.1" \rightarrow \mathcal{V};$ $(a1, "1.1") \rightarrow f;$ $(a0, a1) \rightarrow \mathcal{R}$	$create\ arc\ a1=(n2, n3);$ $A2 \rightarrow \mathcal{A};$ $"1.1" \rightarrow \mathcal{V};$ $(a1, "1.1") \rightarrow f;$ $(a0, a2) \rightarrow \mathcal{R}$ $(a1, a2) \rightarrow \mathcal{R}$

In operation “remove an arc”, as for the arc to remove, i.e.,  $a0$ , its starting node, i.e.,  $n2$ , must own more than one outgoing arc, to avoid a dead result node. In the example diagram, we see that  $a0$  is first replaced by a new arc  $a2$ , and  $a2$  in turn replaces an existing outgoing arc  $a1$ . Because  $a2$  also links  $n2$  to  $n3$ , it is equivalent to  $a1$ , but with different version. The result of this operation is equal to replace  $a0$  with an existing arc.

### 3.4 Updating a VPG

A VPG is updated without actual removal of any nodes or arcs, and it can preserve the information of previous versions in the same graph. There are two rules for updating a VPG with the discussed modification operations.

**Rule (4) Horizontal evolvment.** An evolvment from version  $x.y1.z$  to version  $x.y2.z$  ( $y1 \neq y2$ ) is projected to an evolvment from version  $x.y1$  to  $x.y2$ .

This rule means that all parallel horizontal evolvments can be represented by an evolvment along the process improvement axis, which indicates the permanent change to all parallel branch versions. This mechanism caters for the purpose of reusing versions.

**Rule (5) Vertical evolvment.** For two evolvments from version  $x.y$  to  $x.y.z$  and from  $x1.y1$  to  $x1.y1.z1$  ( $z, z1 \neq null, x \neq x1$  or  $y \neq y1$ ), respectively,  $z = z1$ , if the two evolvments are caused by the same temporary change; otherwise  $z \neq z1$ .

This rule means that the third digit of a version identifies the reason for the evolvment along temporary adaptation axis.

Consider the production workflow process discussed in Section 2. The initial workflow process can be represented as a VPG shown in Figure 4 (a), where all nodes and arcs are marked as version 1.0. When the workflow process evolves to version 1.1

as an additional work centre is inserted, the VPG will be updated to Figure 4 (b) with an “insert a parallel task” operation. The inserted arcs and nodes, i.e.,  $a1$ ,  $a5$  and  $n5$ , are marked with version 1.1, according to Rule 4. Following this, when the workflow process for a pipeline evolves to version 1.1.1 as task “work centre #1” is replaced by task “fixing unqualified products”, the VPG will be updated to Figure 4 (c) with an “replace a task” operation. The added arcs and nodes, i.e.,  $a7$ ,  $a8$  and  $n7$ , are marked with version 1.1.1. While, the workflow process for another pipeline may replace task “work centre #1” with “manual production”, and therefore evolves to version 1.1.2 with the VPG shown in Figure 4 (d). The added arcs  $a3$  and  $a9$  and node  $n6$  are marked with version 1.1.2. Thus, we see that these two versions are marked differently at digit  $z$ , because their evolvments are initiated by different temporary changes, according to Rule 5.

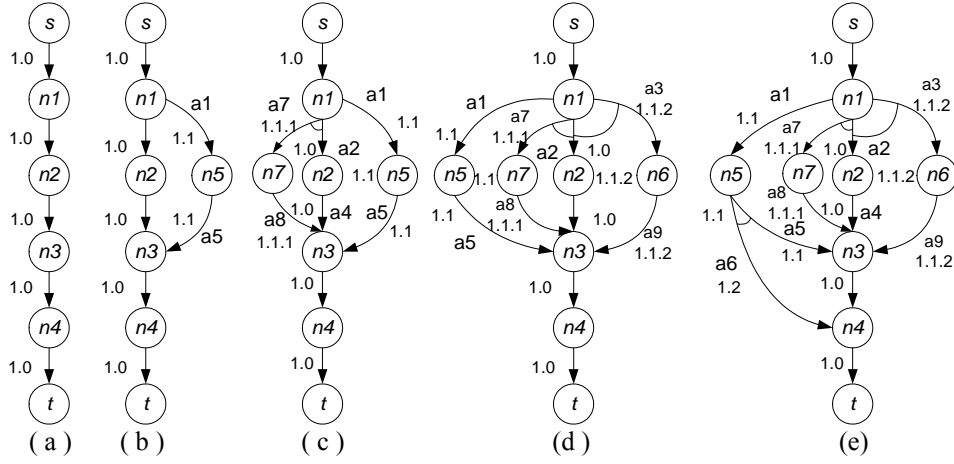


Fig. 4. VPG samples

When handling the evolvement to version 1.2.2, we need to note that arc  $a6$  is added by a “replace an arc” operation, as shown in Figure 4 (e). According to Rule 4,  $a6$  is marked with version 1.2 instead of 1.2.2. This denotes that the insertion of arc  $a6$  represents a permanent change rather than a temporary change. Though there are no nodes or arcs marked with version 1.2.2 in Figure 4 (e), version 1.2.2 can be obtained by aggregating the arcs and nodes of version 1.2 and 1.1.2. In this way, the stored information can be maximally reused, and in turn, a better space efficiency is obtained.

As a VPG records the key changes during evolvements of a workflow process, its arcs and nodes cover all possible combinations of workflow evolvements. For example, we mentioned that version 1.2.2 can be deduced from the VPG, even version 1.2.2 does not appear in the VPG at all. This feature reflects the strong expression ability of the VPG, as it can deduce any version that exists in the real evolvement situation. In addition, a VPG can also deduce any version that is achievable during evolvements of its workflow process definition. For example, version 1.2.1 is achievable so it can also be deduced from the VPG in Figure 4 (e).

## 4 Runtime Version Management

A VPG contains the information of different versions of a workflow process definition. However, many workflow management operations, such as initiating workflow instances, reviewing workflow processes, etc., only refer to a workflow process definition of a specific version. This requires the capability of dynamically extracting a workflow process definition of a specific version from the changing VPG.

Basically, the extraction can be achieved with two different strategies, viz., backward assembling and top-down exploration. The following two sections are to discuss these two strategies, respectively.

### 4.1 Backward assembling strategy

The most direct strategy is to start with the nodes and arcs with the requested version, or the closest to the requested version if the requested one does not exist in the VPG. Then, it continues with searching and assembling the nodes and arcs with versions in the priority list discussed in Rule 3 of Section 3.2, in a descending order.

Before collecting the nodes and arcs of the next highest priority, we need to delete all arcs that are in an exclusive relation with any collected arc. This removal may result in some unreachable nodes, i.e., nodes with no incoming arcs. These unreachable nodes need to be deleted, and in turn, we need to remove those arcs connect to these nodes.

This collecting and removing process keeps running until all arcs or nodes with the versions on the priority list are handled. For example, suppose we extract a workflow process definition of version 1.2.2 from the VPG shown in Figure 4 (e). According to the priority sequence, version 1.1.2 holds the highest priority among all the versions contained in the VPG. Thus, we first collect all arcs and nodes with version 1.1.2, viz. arc  $a_3$ ,  $a_9$  and  $n_6$ . Afterwards, we find that arcs  $a_7$  and  $a_2$  are in the exclusive relation with a collected arc  $a_3$ , therefore  $a_2$  and  $a_7$  are removed from the VPG. Thereafter, nodes  $n_2$  and  $n_7$  are deleted as they become unreachable, and then arcs  $a_4$  and  $a_8$  are deleted too for the same reason. After that, arc  $a_6$  is collected, as version 1.2 holds the highest priority in the remaining VPG, while arc  $a_5$  is removed due to its exclusive relation with  $a_6$ . The workflow process definition of version 1.2.2 will be obtained after we handle the nodes and arcs of the base version, i.e., version 1.0. Algorithm 1 formalises the extraction procedure under this strategy.

In this algorithm, Function  $relatedArcs(\mathcal{G}, ASet)$  returns the set of arcs that are in an exclusive relation with the arcs in set  $ASet$  in VPG  $\mathcal{G}$ ; Function  $in(\mathcal{G}, n)$  returns the in-degree of node  $n$  in VPG  $\mathcal{G}$ ; Function  $outArcs(\mathcal{G}, n)$  returns a set of node  $n$ 's outgoing arcs in VPG  $\mathcal{G}$ ; Function  $linkedNode(\mathcal{G}, a)$ ; returns the node that arc  $a$  links to in VPG  $\mathcal{G}$ ; Function  $highestVer(\mathcal{G}, v)$  returns the version with the currently highest priority in VPG  $\mathcal{G}$  with version  $v$ .

## Algorithm 1. backward assembling

<b>Input</b>	$\mathcal{G}$	-	The VPG for a workflow process definition
	$v$	-	The requested version
<b>Output</b>	$\mathcal{G}'$	-	The graph for the workflow process definition of the requested version

---

<ol style="list-style-type: none"> <li>1.</li> <li>2.</li> <li>3.</li> <li>4.</li> <li>5.</li> <li>6.</li> <li>7.</li> <li>8.</li> <li>9.</li> <li>10.</li> <li>11.</li> <li>12.</li> <li>13.</li> <li>14.</li> <li>15.</li> <li>16.</li> <li>17.</li> <li>18.</li> </ol>	$ATarget = \emptyset; NTarget = \emptyset;$ <b>add</b> the nodes and arcs of version $highestVer(\mathcal{G}, v)$ <b>to</b> sets $NTemp$ and $ATemp$ , respectively; $A = relatedArcs(\mathcal{G}, ATemp);$ <b>while</b> ( $A \neq \emptyset$ ) <b>do</b> <b>pick</b> arc $a \in A$ $n = linkedNode(\mathcal{G}, a);$ <b>if</b> $in(n) = 1$ <b>then</b> $A = A \cup outArcs(n);$ <b>delete</b> $n$ <b>from</b> $\mathcal{G};$ <b>end if</b> <b>delete</b> $a$ <b>from</b> $A$ and $\mathcal{G};$ <b>end while</b> $NTarget = NTarget \cup NTemp;$ $ATarget = ATarget \cup ATemp;$ <b>delete</b> the arcs and nodes in $ATemp$ and $NTemp$ <b>from</b> $\mathcal{G};$ $ATemp = \emptyset; NTemp = \emptyset$ <b>if</b> $highestVer(\mathcal{G}, v)$ is not <i>null</i> <b>then goto</b> line 2; $\mathcal{G}' = (NTarget, ATarget);$
---	--

This algorithm uses sets  $NTemp$  and  $ATemp$  to keep the newly collected nodes and arcs, respectively. Set  $A$  temporarily stores the arcs to be deleted from the VPG. After picking an arc  $a$  in set  $A$ , the algorithm will check whether  $a$  is the only incoming arc to its linked node  $n$ . If so, node  $n$  will be deleted with  $a$  from the VPG, and the outgoing arcs of  $n$  will be inserted to set  $A$  for future checking. The collected nodes and arcs will be inserted to the result graph by moving the elements in  $NTemp$  and  $ATemp$  to  $NTarget$  and  $ATarget$ , respectively.

#### 4.2 Top-down exploration strategy

Another strategy is to search for the requested version from the top of a VPG. For each outgoing arc, if it has a version in the priority list with regard to the requested version and is not in an exclusive relation with any other arcs, it will be collected. As to the arcs in an exclusive relation, we need to select one proper arc that owns the highest priority among the exclusively coupled peers. The arcs and nodes with a version that is not in the priority list will not be considered at all.

For example, suppose we also extract a workflow process definition of version 1.2.2 from the VPG shown in Figure 4 (e). The extraction process starts from the starting node  $s$ , and then comes to node  $n1$  which has four outgoing arcs, viz.,  $a1$ ,  $a2$ ,

$a3$  and  $a7$ . Here,  $a1$  is not in an exclusive relation with other three arcs, and version 1.1 meets the fourth requirement of the priority with regard to version 1.2.2 (please refer to Rule 3). Thus,  $a1$  will be first selected. As to the three exclusively coupled arcs,  $a3$  with version 1.1.2 holds a higher priority than the other two peers,  $a2$  with version 1.0 and  $a7$  with version 1.1.1. Thus, only  $a3$  is selected, while  $a2$  and  $a7$  are not considered. This process goes on as the trace flows along the collected arcs, and finally we can obtain the nodes and arcs for version 1.2.2 when the trace ends at the terminating node,  $t$ . Algorithm 2 formalises the extraction procedure under this strategy.

In this algorithm, Function  $outArcs(\mathcal{G}, n)$  returns a set of node  $n$ 's outgoing arcs in VPG  $\mathcal{G}$ ; Function  $coulpedArcs(\mathcal{G}, n)$  returns a set of node  $n$ 's outgoing arcs that are in the exclusive relation in VPG  $\mathcal{G}$ ; Function  $pickPriorityArc(ASet, v)$  returns the arc with the highest priority with regard to version  $v$  among set  $ASet$ ; Function  $checkNodes(\mathcal{G}, a)$  returns the set of nodes that arc  $a$  links to in VPG  $\mathcal{G}$ .

---

Algorithm 2. top-down exploration

---

<b>Input</b>	$\mathcal{G}$	-	The VPG for a workflow process definition
	$v$	-	The requested version
<b>Output</b>	$\mathcal{G}'$	-	The graph for the workflow process definition of the requested version

---

1.	$NTarget = \emptyset; ATarget = \emptyset;$
2.	$NTemp = \{ \mathcal{G}.s \};$
3.	<b>while</b> ( $NTemp \neq \emptyset$ ) <b>do</b>
4.	<b>for each</b> $n \in NTemp$
5.	$A = coulpedArcs(\mathcal{G}, n);$
6.	$ATemp = outArcs(\mathcal{G}, n) - A;$
7.	$ATemp = ATemp \cup \{ pickPriorityArc(A, v) \};$
8.	$NTarget = NTarget \cup \{ n \};$
9.	<b>end for</b>
10.	$NTemp = \emptyset;$
11.	<b>for each</b> $a \in ATemp$
12.	$NTemp = NTemp \cup ( checkNodes(\mathcal{G}, a) - NTarget );$
13.	<b>end for</b>
14.	$ATarget = ATarget \cup ATemp;$
15.	<b>end while</b>
16.	$\mathcal{G}' = (NTarget, ATarget);$

---

This algorithm starts searching from the starting node,  $s$ , and collects the includable nodes and arcs in sets  $NTarget$  and  $ATarget$ . When the search arrives to the outgoing arcs of the collected nodes, the algorithm (line 4 to line 9) checks whether the arcs are collectable by referring to the priority sequence and the exclusive relation. The search moves on to the nodes to which are linked from the newly collected arcs, and checks whether these nodes have been collected before to reduce potential redundant processing. Finally, the search terminates when it arrives to node  $t$ .

### 4.3 Strategy analysis

Under the backward assembling strategy, the algorithm needs to process most nodes and arcs. Whiling removing an arc that is in the exclusive relation with a collected arc, it may result in a chain reaction that may cause the linked node to be with no incoming arcs, and therefore the removal of this node and all its outgoing arcs. This process may cover a lot of nodes and arcs in the graph, no matter these nodes or arcs are really useful for the extraction or not. In fact, for version  $v$ , the arcs and nodes belonging to  $v$ 's subsequent versions have nothing to do with the extraction of version  $v$ , because these arcs and nodes only serve for the evolvments occurred after version  $v$ . Additionally, the arcs and nodes belonging to  $v$ 's parallel branch versions, offer no contributions, either. For example, the components for version 1.1.1 do not contribute to the extraction of version 1.1.2. However, the backward assembling strategy still processes the components of version 1.1.1 during the extraction of version 1.1.2.

In contrast, the top-down exploration strategy is more intelligent. When the top-down exploration strategy comes across a splitting structure, it leaves all irrelevant arcs untouched as long as they are not in the priority list with regard to the requested version. Thereby, this strategy pleasantly sidesteps the searching with irrelevant nodes or arcs, and in turn it outperforms the backward assembling strategy. As the version extraction is a frequent operation for a VPG, this improvement can lead to a considerable performance gain.

## 5 Related Work and Discussion

Workflow evolution is the most related to version management. Casati et al., [10] presented a workflow modification language (WFML) to support modifications of a workflow model. They also discussed the case evolution policies and devised three main policies to manage case evolution, viz., abort, flush and progressive. The proposed language contains declaration primitives and flow primitives for the changes of workflow variables and flow structures.

Work in adaptive workflows, addresses run time modifications for dynamic exception handling purpose. Hamadi and Benatallah [7] proposed a self-adaptive recovery net (SARN) to support workflow adaptability during unexpected failures. This net extends Petri net by deploying recovery tokens and recovery transitions to represent the dynamic adaptability of a workflow process, and a set of operations are used to modify the net structure.

In project ADEPT<sub>flex</sub> [15, 16], Rinderle, Reichert and Dadam did extensive studies on schema evolution in process management systems, which covered common workflow type and instance changes, as well as disjoint and overlapping process changes. Their work formally specified the change operations to both process schemas and workflow instances, as well as the related migration policies in handling potential conflicts.

In Sadiq et al.'s work on process constraints for flexible workflows [17], they proposed the concept of "pockets of flexibility" to allow ad hoc changes and/or building of workflows, for highly flexible processes.

Unfortunately, none of the above work mentions the versions of workflows. Therefore, they can hardly keep the trail about a series of evolutions and change-backs, or only support a kind of one-off modifications. The transformation between subsequent versions or sibling versions is not touched, let alone the compatibility of multiple workflow versions.

Kradolfer and Geppert [11] presented a framework for dynamic workflow schema evolution based on workflow type versioning and workflow migration. In their work, a version tree was proposed to represent the evolvement of a workflow schema, and to keep track of the resulting history. However, the version tree only provides primitive supports for version management. Typically, to re-assign a previous version to a running workflow instance, this method has to perform a series of inverse modification operations to achieve that version along the version tree. Yet, in our VPG approach, the version re-assignment can be easily realised by switching to the requested version according to the VPG.

In summary, our VPG approach has the following appealing features for business process change management:

- Dynamic updating

The proposed version preserving directed graph allows dynamic modifications without suspending running workflow instances. The defined modification operations preserve all the information during the modification on the fly. We can extract a workflow process definition of any version at any time. This feature enhances the flexibility of workflow technology at process level.

- Multiple version compatibility

A VPG allows the co-existence of workflow instances of different versions in a single graph. With the help of its strong expressive ability, this VPG provides enough navigation information for a workflow engine to execute these workflow instances. This feature enhances the flexibility of workflow technology at instance level.

- Compact model

Compared with other work, a VPG is a lightweight graph model for representing workflow evolvements. With the defined modification operations and proposed rules, a VPG preserves all information for existing versions, and it can derive a meaningful version that may not explicitly appear in the graph.

## 6 Conclusion and Future Work

This paper addressed the version control of workflow process definition in the business process change context. A versioning method was designed to represent the workflow evolvement along the axes for both temporary changes and permanent improvements. A novel version preserving directed graph, together with a series of run time modification operations, were proposed to update a workflow process definition on the fly. Strategies on extracting a workflow process definition of a given version from the corresponding version preserving directed graph were also discussed. Our future work is to incorporate the handover policies with our version control method, and provide a comprehensive solution for workflow version control.



## References

- [1] Smith, H. and Fingar, P.: *Business Process Management - The Third Wave*: Meghan-Kiffer Press, (2003).
- [2] van der Aalst, W.M.P., ter Hofstede, A.H.M., and Weske, M.: *Business Process Management: A Survey*, In *Proceedings of International Conference on Business Process Management*, pp.1-12, (2003).
- [3] Khoshafian, S.: *Service Oriented Enterprise*: Auerbach Publisher, (2006).
- [4] Kock, N.: *System Analysis & Design Fundamentals - A Business Process Redesign Approach*: Sage Publications, Inc., (2006).
- [5] Zhao, X., Liu, C., Yang, Y., and Sadiq, W.: *Handling Instance Correspondence in Inter-Organisational Workflows*, In *Proceedings of the 19th International Conference on Advanced Information Systems Engineering (CAiSE'07)*, Trondheim, Norway, pp.51-65, (2007).
- [6] Liu, C., Orłowska, M.E., and Li, H.: *Automating Handover in Dynamic Workflow Environments*, In *Proceedings of 10th International Conference on Advanced Information Systems Engineering*, Pisa, Italy, pp.159-171, (1998).
- [7] Hamadi, R. and Benatallah, B.: *Recovery Nets: Towards Self-Adaptive Workflow Systems*, In *Proceedings of the 5th International Conference on Web Information Systems Engineering*, Brisbane, Australia, pp.439-453, (2004).
- [8] Kammer, P.J., Bolcer, G.A., Taylor, R.N., Hitomi, A.S., and Bergman, M.: *Techniques for Supporting Dynamic and Adaptive Workflow*, *Computer Supported Cooperative Work*, vol. 9, pp.269-292, (2000).
- [9] Narendra, N.C.: *Flexible Support and Management of Adaptive Workflow Processes*, *Information Systems Frontiers*, vol. 6, pp.247-262, (2004).
- [10] Casati, F., Ceri, S., Pernici, B., and Pozzi, G.: *Workflow Evolution*, *Data & Knowledge Engineering*, vol. 24, pp.211-238, (1998).
- [11] Kradolfer, M. and Geppert, A.: *Dynamic Workflow Schema Evolution Based on Workflow Type Versioning and Workflow Migration*, In *Proceedings of International Conference on Cooperative Information Systems*, Edinburgh, Scotland, pp.104-114, (1999).
- [12] IBM: *IBM WebSphere Business Integration Handbook* (2005).
- [13] SAP: *SAP Business Workflow and WebFlow Documentation*.
- [14] Conradi, R. and Westfechtel, B.: *Version Models for Software Configuration Management*, *ACM Computing Surveys*, vol. 30(2), pp.232-282, (1998).
- [15] Reichert, M. and Dadam, P.: *ADEPTflex -Supporting Dynamic Changes of Workflows without Losing Control*, *Journal of Intelligent Information Systems*, vol. 10, pp.93-129, (1998).
- [16] Rinderle, S., Reichert, M., and Dadam, P.: *Disjoint and Overlapping Process Changes: Challenges, Solutions, Applications*, In *Proceedings of 12th International Conference on Cooperative Information Systems*, Agia Napa, Cyprus, pp.101-120, (2004).
- [17] Sadiq, S.W., Orłowska, M.E., and Sadiq, W.: *Specification and Validation of Process Constraints for Flexible Workflows*, *Information System*, vol. 30, pp.349-378, (2005).