WS-BPEL Business Process Abstraction and Concretisation

Xiaohui Zhao[1], Chengfei Liu[1], Wasim Sadiq[2], Marek Kowalkiewicz[2], and Sira Yongchareon[1]

[1] Centre for Complex Software Systems and Services
Swinburne University of Technology
Melbourne, Victoria, Australia
{xzhao, cliu}@groupwise.swin.edu.au,
575318X@student.swin.edu.au

[2] SAP Research
Brisbane, Australia
{wasim.sadiq, marek.kowalkiewicz}@sap.com

Abstract. Business process management is tightly coupled with service-oriented architecture, as business processes orchestrate services for business collaboration at logical level. Given the complexity of business processes and the variety of users, it is a sought-after feature to show a business process with different views, so as to cater for the diverse interests, authority levels, etc., of users. This paper presents a framework named *FlexView* to support process abstraction and concretisation. A novel model is proposed to characterise the structural components of a business process and describe the relations between these components. Two algorithms are developed to formally illustrate the realisation of process abstraction and concretisation in compliance with the defined consistency rules. A prototype is also implemented with WS-BPEL to prove the applicability of the approach.

1 Introduction

In service-oriented architecture (SOA), business processes are widely applied to organise service composition and service orchestration [1-4]. As one of the leading SOA advocators, the Web service community formally adopted business process technology in 2001 by establishing the Business Process Execution Language for Web Services (WS-BPEL) [5]. WS-BPEL supports the specification of both composition schemas and coordination protocols to fulfil complicated B2B interactions.

Reluctantly, most of current business process modelling languages, including WS-BPEL, stick to a fixed description of business processes. Although WS-BPEL can be used to define both abstract processes and executable processes, WS-BPEL is in lack of mechanisms to automatically represent a business process with different views on demand. The concept of “process view” has emerged recently to support for flexible views on business process representation, and thereby separate the process representation from executable business process models. This feature has been longed for in the practical business process application environment, for the purpose of authority control, privacy protection, process analysis, etc. [6, 7] For instance, users may prefer to see part of the process details at a time, due to the complexity of the business process. Users with different interests or different authority levels, may be interested to or be allowed to see different views of the same business process. For another instance, in a graphical displaying tool for business processes, the flexibility on showing a reduced version of business process at a time is highly expected, due to the limit of screen size. Similar functions can be found in other application areas. A good example is *google maps*, which allows users to zoom in or zoom out a map, while the displayed details on map automatically adapt to the scale level, for instance, streets and roads are shown on a large scale map, yet a small scale map only shows suburbs and towns.

To realise such "smart zooming" functions towards business process representation, this paper proposes a framework named *FlexView* to support flexible process abstraction and concretisation. With FlexView, users are allowed to define and switch among the different views for a business process. A comprehensive model defines the structural constructs of a business process and the relations between them. Two algorithms formally illustrate how to enforce the process abstraction and concretisation operations in compliance with structural consistency.

The remainder of this paper is organised as follows: Section 2 discusses the requirements for supporting flexible views with a motivating example; Section 3 introduces a process component model with a set of rules on structural consistency, and the algorithms for realising abstraction and concretisation; Section 4 addresses the incorporation of FlexView into WS-BPEL, and also introduces the implementation of a prototype; Section 5 reviews the related work and discusses the advantages of our framework; concluding remarks are given in Section 6 with an indication for the future work.

2 Motivating Example

Figure 1 (a) shows all details of the business process for a simplified sales management service, where the process starts from receiving purchase orders, and then handles the production, cost analysis and shipping planning concurrently, and finally terminates by sending the invoice. Each task may interact with proper Web service(s) to fulfil the assigned mission. The dashed arrows represent the synchronisation dependencies between tasks, for example, the arrow between "production" and "dispatch products" denotes that task "dispatch products" can only start after the completion of "production".

This business process mainly involves four departments, viz., sales department, workshop, accounting department, and distribution centre. For a user from the distribution centre, the user may only care about the shipping details, and thus the user may choose to "zoom out" the details for production and cost handling. The user can obtain the view for this business process as shown in Figure 1 (b). In this view, the details for production and cost handling are abstracted into two new tasks, i.e., "handle production at workshop" and "handle cost calculation at accounting department". These two tasks hide the details yet preserve the existence of the production and cost handling procedures. In this transformation, the related links are hidden automatically, as well as the synchronisation link from "schedule production" to "cost analysis". The synchronisation link from "production" to "dispatch products" is converted to connect "handle production at workshop" to "dispatch products", as these two tasks inherit the synchronisation dependency of the former one.

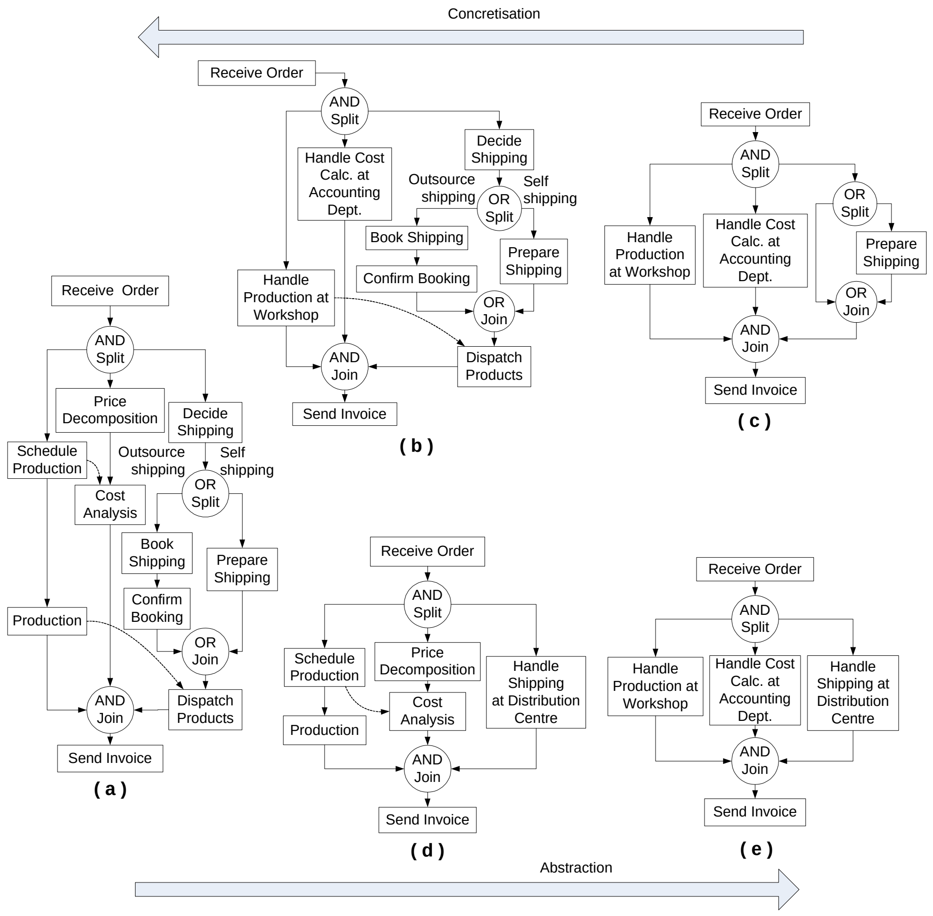


Fig. 1. Motivating example business process

Due to the screen size of the user's computer, the user may want to check the details of a single shipping option at a time. In this case, the process should change to the view shown in Figure 1 (c). In this view, the shipping procedure is represented as an Or-split/join structure, which contains a branch with task "prepare shipping", and an empty branch standing for the existence of an alternative shipping option. The user may later on select to "zoom in" this empty branch to see the details for the alternative option.

The users from other departments may not be authorised to see the shipping details. Such users may only see the view shown in Figure 1 (d), where all the shipping details are hidden in a new task “handle shipping at distribution centre”. The synchronisation link from “production” to “dispatch products” is also hidden, as the underlying synchronisation dependency is not effective for this view. Figure 1 (e) displays a further abstracted view of the business process, which only outlines the core part of the business process with three parallel tasks. The authorised users can choose to concretise the interested part to see more details. In either process concretisation (from the right to the left in Figure 1) or abstraction (from the left to the right), the structure of the new views keeps consistent with the previous one.

To support process abstraction and concretisation functions, new mechanisms are on demand to allow wrapping a sub process into a specific task or link, and releasing the sub process back from a task or link. In details, we list the following technical requirements:

– Maintain the relations between the hidden sub processes and the corresponding tasks/links.
– Preserve the structural information of a business process, such as split/join structures and synchronisation links, during process abstraction/concretisation.
– Support cascading abstraction/concretisation operations.
– Keep the structural consistency of process views during transformations.

Towards these requirements, our FlexView framework employs a process component model to describe the structure of process views and structural components, and maintain the relations between structural components. The algorithms are designed to enable the procedure of abstraction and concretisation. A set of defined rules regulate the structural consistency during the procedure. The framework is implemented with WS-BPEL as a proof-of-concept.

3 Framework of FlexView

3.1 Process Component Model

To well describe the structure of a process view and maintain the relations between structural components, we define a process component model. This model provides the foundation for process abstraction and concretisation functions, and particularly takes into account the characteristics of WS-BPEL.

Definition 1. (Gateway) Gateways are used to represent the structure of a control flow. Here we define five types of gateways, namely *Or-Split*, *Or-Join*, *And-Split*, *And-Join*, and *Loop*. Figure 2 shows the samples of these gateways, respectively. *Or-Split/Join* and *Loop* gateways may attach conditions to restrict the control flow.

Definition 2. (Synchronisation Link) In an *And-Split/Join* structure, synchronisation links are used to represent the synchronisation dependency between the tasks belonging to different branches. For example, in Figure 2 (b), the synchronisation link between t_i and t_j, represented as a dashed arrow, denotes that t_j can only start after the completion of t_i. In WS-BPEL, such a synchronisation dependency is supported with a <link> element.

Functions *ind*(*m*) and *outd*(*m*) define the number of edges which take *m* as the terminating node and the starting node, respectively. Note *ind* and *outd* only count the number of edges but not synchronisation links.

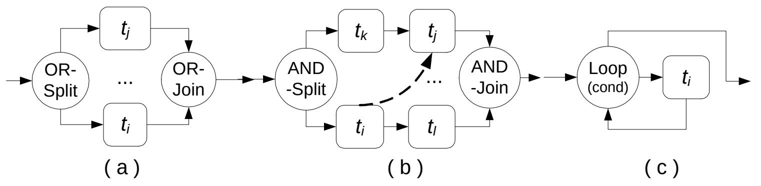


Fig. 2. Gateway samples

Function *type*: $G \rightarrow Type$ is used to specify gateway types, where $Type=\{Loop, And\text{-}Join, And\text{-}Split, Or\text{-}Join, Or\text{-}Split\}$. According to the natural characteristics of these gateways, we can define the following rules in terms of the incoming and outgoing degrees:

$$\text{if } type(g) = \text{“}Loop\text{”} \begin{cases} ind(g)=1,\ outd(g)=2 & \text{if } g \text{ is at the starting position;} \\ N/A & g \text{ is not allowed at the ending position;} \\ ind(g)=2,\ outd(g)=2 & \text{Otherwise.} \end{cases}$$

$$\text{if } type(g) = \text{“}And\text{-}Split\text{” or “}Or\text{-}Split\text{”} \begin{cases} ind(g)=0,\ outd(g)>1 & \text{if } g \text{ is at the starting position;} \\ N/A & g \text{ is not allowed at the ending position;} \\ ind(g)=1,\ outd(g)>1 & \text{Otherwise.} \end{cases}$$

$$\text{if } type(g) = \text{“}And\text{-}Join\text{” or “}Or\text{-}Join\text{”} \begin{cases} N/A & g \text{ is not allowed at the starting position;} \\ ind(g)>1,\ outd(g)=0 & \text{if } g \text{ is at the ending position} \\ ind(g)>1,\ outd(g)=1 & \text{Otherwise.} \end{cases}$$

Definition 3. (Sub Process) A sub process is a structural component of a business process, and it also maintains the necessary information for sub process composition. The structure of a sub process *s* can be modelled as an extended directed graph in the form of tuple $(N, G, E, L, m_s, m_t, L_0)$, where

- $N=\{n_1, n_2, \ldots, n_x\}$, $n_i \in N$ $(1 \le i \le x)$ represents a task of *s*.
- $G=\{g_1, g_2, \ldots, g_y\}$, $g_i \in G$ $(1 \le i \le y)$ represents a gateway of *s*.
- E is a set of directed edges. An edge $e=(m_1, m_2) \in E$ corresponds to the control dependency between m_1 and m_2, where $m_1 \in N \cup G$, $m_2 \in N \cup G$.
- L is a set of synchronisation links. A synchronisation link $l = (m_1, m_2) \in L$ corresponds to the synchronisation dependency between m_1 and m_2, where $m_1 \in N \cup G$, $m_2 \in N \cup G$.
- m_s is the starting node of *s*, which satisfies that $m_s \in N \cup G$ and $ind(m_s)=0$.
- m_t is the terminating node of *s*, which satisfies that $m_t \in N \cup G$ and $outd(m_t)=0$.

- $\forall n \in N \backslash \{ m_s, m_t \}$, $ind(n)=outd(n)=1$. This property is guaranteed by the usage of gateways.
- L_0 is a set of hidden synchronisation links, i.e., the synchronisation links that have a node not included in N or G. $\forall l=(m_1, m_2) \in L_0$, $(m_1 \in N \cup G) \wedge (m_2 \notin N \cup G)$ or $(m_1 \notin N \cup G) \wedge (m_2 \in N \cup G)$. The synchronisation links in L_0 are not displayable as they connect to foreign nodes, but such synchronisation link information is preserved for sub process composition.

Definition 4. (Sub Process Hierarchy) A sub process hierarchy $\Gamma(p)$ for business process p maintains all the related sub processes and mapping information for process representation. $\Gamma(p)$ can be represented as tuple (S, δ, γ), where

- S is a finite set of distinct sub processes.
  For a sub process $s \in S$,
  $$\forall l=(n_1, n_2) \in s.L \cup s.L_0 \cup s.G \quad \exists s_1 \in S \; (n_1 \in s_1.N \cup s_1.G) \wedge (n_2 \in s_1.N \cup s_1.G),$$
  $$\text{or } \exists s_1 \in S,\, s_2 \in S,\; (n_1 \in s_1.N \cup s_1.G) \wedge (n_2 \in s_2.N \cup s_2.G).$$
- $s_0 \in S$ is the root sub process, which shows the most abstracted view of p.
- δ: $E' \rightarrow S'$ ($E' \subseteq \bigcup_{s \in S} s.E$ and $S' \subseteq S \backslash \{s_0\}$) is a bijection describing the relations between edges and sub processes. Correspondingly, we have the inverse function δ^{-1}: $S' \rightarrow E'$.
- γ: $N' \rightarrow S'$ ($N' \subseteq \bigcup_{s \in S} s.N$ and $S' \subseteq S \backslash \{s_0\}$) is a bijection describing the relations between nodes and sub process. Correspondingly, we have the inverse function γ^{-1}: $S' \rightarrow N'$.
- A sub process can only occur in one of the two inverse functions. This denotes that $\forall s \in S \backslash \{s_0\}$, if $\delta^{-1}(s) \neq$null then $\gamma^{-1}(s)$=null; if $\gamma^{-1}(s) \neq$null then $\delta^{-1}(s)$=null.

The sub processes of a sub process hierarchy can be defined in a nested way. Figure 3 shows a sub process hierarchy example, where $subp_1$, …, $subp_5$ are five sub processes in this hierarchy, and $subp_1$ is the root sub process.

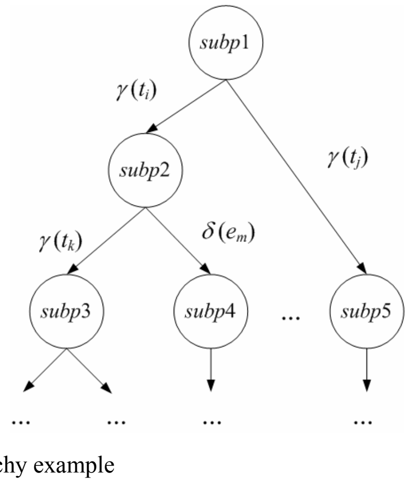


Fig. 3. Sub process hierarchy example

In this example, tasks t_i and t_j of sub process $subp_1$ can be mapped to sub processes $subp_2$ and $subp_5$ by functions $\gamma(t_i)$ and $\gamma(t_j)$, respectively. Further, task t_k and edge e_m

of $subp_2$ can be mapped to sub processes $subp_3$ and $subp_4$ by functions $\gamma(t_k)$ and $\delta(e_m)$, respectively. A *concretisation* operation denotes the extension by replacing a task or edge with the mapped sub process. Therefore, root sub process $subp_1$ can be concretised into combination $subp_1$+$subp_2$, $subp_1$+$subp_5$, or $subp_1$+$subp_2$+$subp_5$, where tasks t_i and t_j are replaced by the corresponding sub processes. Correspondingly, the *abstraction* operation can be realised by wrapping a sub process back into a task or edge with functions γ^{-1} and δ^{-1}. Each result combination denotes a partial view of the business process.

Such a sub process hierarchy is fully customisable for users, and thereby enables the adaptation to user-defined partitions and categorisations according to different levels of BPEL abstraction and concretisation.

Definition 5. (Process View) A process view represents the viewable part for a business process at a time. In the sub process hierarchy, each process view corresponds to a sub tree including the root sub process, where the mapped tasks/edges are concretised with corresponding sub processes. A fully concretised view, i.e., the view containing all the sub processes in this hierarchy, is equivalent to the base business process, and the view containing only the root sub process is the most abstracted view.

3.2 Consistency and Validity Rules

As explained in the motivating example, some rules are defined to guarantee the structural consistency of process views during view abstraction and concretisation. This section is to discuss the rules on preserving execution orders, branch subjection, synchronisation dependencies, and so on.

- *Preliminary*

– A *dummy branch* denotes a branch in a split/join structure such that the branch contains nothing but only one edge.

– A common split gateway predecessor (CSP), x, of a set of tasks, T, denotes a split gateway such that x is the predecessor of each task in T.

– *before*(t_1, t_2) denotes that task t_1 will be executed earlier than task t_2. This means that there exists a path from starting t_1 to t_2 in the corresponding directed graph, while the path does not contain any go-back edge of a loop structure. Apparently, *before* is a transitive binary relation.

– *CSP*(t_1, t_2) returns the set of common split gateway predecessors of t_1 and t_2, or returns null if the two tasks have no common split gateway predecessors.

– *branch*(g, t_1, t_2) is a boolean function, which returns true if t_1 and t_2 lie in the same branch led from split gateway g, otherwise returns false.

- *Structural Consistency and Validity Rules*

In regard to an abstraction/concretisation operation, the original process view v_1 and the result view v_2 are required to comply with the following rules:

Rule 1. (Order preservation) As for the tasks belonging to v_1 and v_2, the execution sequences of these tasks should be consistent, i.e., If t_1, $t_2 \in v_1.N \cap v_2.N$ such that $before(t_1, t_2)$ exists in v_1, then $before(t_1, t_2)$ also exists in v_2.

Rule 2. (Branch preservation) As for the tasks belonging to v_1 and v_2, the branch subjection relationship of these tasks should be consistent, i.e.,

If t_1, $t_2 \in v_1.N \cap v_2.N$ and $g \in CSP(t_1, t_2)$ in v_1, $g \in CSP(t_1, t_2)$ in v_2 such that $X(g, t_1, t_2)$ in v_1, then $X(g, t_1, t_2)$ in v_2, where $X \in \{branch, \neg branch\}$.

Rule 3. (Synchronisation dependency preservation) If an abstraction operation involves any tasks with synchronisation links, the synchronisation links should be rearranged to preserve the synchronisation dependency. Assume that sub process *s* comprising tasks t_1 and t_2 is to be abstracted into a compound task t_c as shown in Figure 4,

– for task $t_x \in s.N$ and t_x has an outgoing synchronisation link *l*,

If $\forall t \in s.N$, $before(t, t_x)$ then the source task of *l* should be changed to t_c, otherwise *l* should be hidden.

– for task $t_x \in s.N$ and t_x has an incoming synchronisation link *l*,

If $\forall t \in s.N$, $\neg before(t, t_x)$ then the target task of *l* should be changed to t_c, otherwise *l* should be hidden.

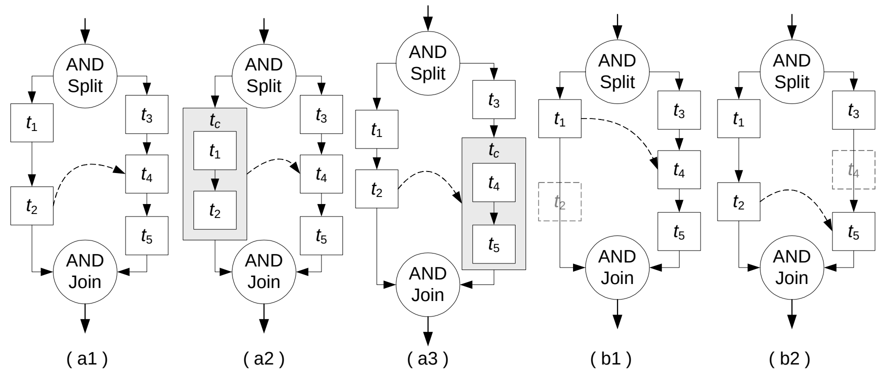


Fig. 4. Synchronisation link handling

In Figure 4, the transformation from (a1) to (a2), where t_1 and t_2 are hidden in task t_c, and the transformation from (a1) to (a3), where t_4 and t_5 are hidden in task t_c, illustrate the two mentioned scenarios, respectively.

In the case that a task involving a synchronisation link is abstracted into an edge, the re-arrangement of synchronisation links is subject to Rule 1. For example, Figure 4 (b1) and (b2) illustrate the re-arrangements in cases that t_2 and t_4 are hidden in edges.

Rule 4. (No empty Split/Join or Loop structures) If a loop structure contains no tasks, or if a split/join structure contains only dummy branches, then the loop or split/join structure should be hidden.

Rule 5. (No dummy or single branch in And-Split/Join structures) If an And-split/join structure contains both dummy and non-dummy branches, then the dummy branch(es) should be hidden. If the And-split/join structure contains only one non-dummy branch, then the And-split/join structure will be degraded into a sequential structure.

Rule 6. (Dummy branch in Or-Split/Join structures) If an Or-split/join structure contains a dummy branch, then this dummy branch should remain to indicate the existence of an alternative execution path. If an Or-split/join structure contains multiple dummy branches, these branches should merge into one dummy branch.

3.3 Process Abstraction and Concretisation

To realise the view abstraction and concretisation under the restriction of structural consistency, two algorithms are developed to formalise the procedures of process view transformation.

Given a sub process hierarchy $\Gamma(p)=(S, \delta, \gamma)$, the following functions are to be used in the algorithms: *addEdge*(s, e) inserts edge e into set E of sub process s. *addTask*(s, t) inserts task t into set N of sub process s. *addLink*(s, l) inserts synchronisation link l to set L of sub process s. *removeLink*(s, l) deletes synchronisation link l from set L of sub process s. *removeTask*(s, t) deletes task t from set N of sub process s. *removeEdge*(s, e) deletes edge e from set E of sub process s. *combineSubProc*(s_1, s_2) combines the constitute sets, i.e., N, G, E, L and L_0, of sub process s_2 into sub process s_1. *removeSubProc*(s_1, s_2) removes the constitute sets of sub process s_2 from sub process s_1. *toSequence*(s, g_1, g_2) flats a single branch split/join structure scoped by gateways g_1 and g_2 in sub process s into a sequence structure, i.e., removes the two gateways and re-connects the gateways' adjacent nodes to the single branch.

Algorithm 1.

taskZoomIn(s, t) transforms sub process s into a more concrete sub process s', by concretising task t.

```
$s'=s$; $subp=\gamma(t)$;
if $t=s'.m_s$ then $s.m_s=subp.m_s$;
if $t=s'.m_t$ then $s.m_t=subp.m_t$;
do while ($\exists e=(m_x, t)\in s'.E$)
   removeEdge($s'$, $e$); addEdge($s'$, ($m_x$, $subp.m_s$));
loop
do while ($\exists e=(t, m_y)\in s'.E$)
   removeEdge($s'$, $e$); addEdge($s'$, ($subp.m_t$, $m_y$));
loop
removeTask($s'$, $t$); combineSubProc($s'$, $subp$);
for each sync link $l=(m_1, m_2)\in s'.L_0$
   if $(m_1\in s'.N)\wedge(m_2\in s'.N)$ then
        addLink($s'$, $l$); $s'.L_0=s'.L_0\backslash\{l\}$;
   end if
for each sync link $l=(m_1, m_2)\in s'.L\backslash subp.L$
   if ($m_1=t$) or ($m_2=t$) then
     removeLink($s'$, $l$);
for each link $l=(m_1, m_2)\in s'.L_0\backslash subp.L_0$
    if ($m_1=t$) or ($m_2=t$) then $s'.L_0=s'.L_0\backslash\{l\}$;
```

```
20  return $s'$;
```

Lines 2-3 handle the connection in case that the task to concretise is the starting or ending node. Lines 4-9 connect edges according to Rule1 and Rule 2. Line 10 replaces task *t* with sub process *subp*. Lines 11-14 reveal the hidden synchronisation links if their source and target nodes are both visible during the concretisation. Lines 15-17 delete the synchronisation links that are involved with task *t*, because the newly revealed synchronisation links from *subp* will replaces these links. Lines 18-19 sort the hidden synchronisation links that are involved with task *t*.

The procedure of zooming in an edge is similar to Algorithm 1, and we here do not detail it due to space limit.

Algorithm 2.

zoomOut(*s*, *x*) transforms sub process *s* into a more abstract sub process *s'* by abstracting the part containing task or edge *x*.

```
1   $s'=s$; $\exists subp \in S$ such that $x$ belongs to $subp$.
2   if $\delta^{-1}(subp) \neq$ null then
3       $addEdge(s', \delta^{-1}(subp))$;
4   else if $\gamma^{-1}(subp) \neq$ null then $addTask(s', \gamma^{-1}(subp))$;
5   end if
6   for each sync link $l=(m_1, m_2) \in s'.L$
7       if $(m_1 \in subp.N \cup subp.G)$ and $(\forall t \in subp.N, before(t, m_1))$ then
8           if $\gamma^{-1}(subp) \neq$ null then
9               $addLink(s', (\gamma^{-1}(subp), m_2))$; $removeLink(s', l)$;
10          else if $\delta^{-1}(subp) \neq$ null then
11              $e= \delta^{-1}(subp)=(m_3, m_4)$; $addLink(s', (m_3, m_2))$; $removeLink(s', l)$;
12          end if
13      else if $(m_2 \in subp.N \cup subp.G)$ and $(\forall t \in subp.N, \neg before(t, m_2))$ then
14            if $\gamma^{-1}(subp) \neq$ null then
15                $addLink(s', (m_1, \gamma^{-1}(subp)))$; $removeLink(s', l)$;
16            else if $\delta^{-1}(subp) \neq$ null then
17                $e= \delta^{-1}(subp)=(m_3, m_4)$; $addLink(s', (m_1, m_4))$; $removeLink(s', l)$;
18            end if
19      end if
20  end for
21  $s'=removeSubProc(s', subp)$;
22  do
23      for each loop structure with loop gateway $g$ in $s'$
24          if $\exists e=(g, g) \in s'.E$ then $removeLoop(s', g)$; // remove empty loop structure
25      for each split/join structure scoped by split gateway $g_1$ and join gateway $g_2$, in $s'$
26          $flag=0$;
27          if $(outd(g_1)=ind(g_2)=1)$ and $(\exists e=(g_1, g_2) \in s'.E)$ then
28              $removeEdge(s', e)$; $toSequence(s', g_1, g_2)$; $flag=1$;
29          end if
30          if $(s'.type(g_1)=$And-split$)$ AND $(\exists e=(g_1, g_2) \in s'.E)$ then $removeEdge(s', e)$;
31          if $outd(g_1)=ind(g_2)=1$ then
32              $toSeqence(s', g_1, g_2)$; $flag=1$;
```

```
33        end if
34      end for
35  loop until ($flag$=0)
36  return $s'$;
```

Lines 2-5 replace the sub process to abstract with the mapped task or edge. Lines 6-20 handle the synchronisation links according to Rule 3. If *subp* has an outgoing link and the link leaves from the last node of *subp*, lines 7-12 rearrange the link to preserve the synchronisation dependency. Similarly, lines 13-19 do the arrangement, if *subp* has an incoming link and the link joins to the first node of *subp*.

Lines 22-35 iteratively check the structural consistency according to Rules 4-6, until no conflicts exist. According to Rule 4, lines 23-24 and lines 27-29 delete empty loop structures and empty split/join structures, respectively. According to Rules 5 and 6, line 30 deletes dummy branches in an And-split/join structure, and lines 31-33 flat any split/join structures with single branches into sequential structures. Note, due to the set definition, the dummy branches in an Or-split/join structure are already combined together.

The result sub process from these algorithms can be easily converted to a process view for representation, by discarding set L_0.

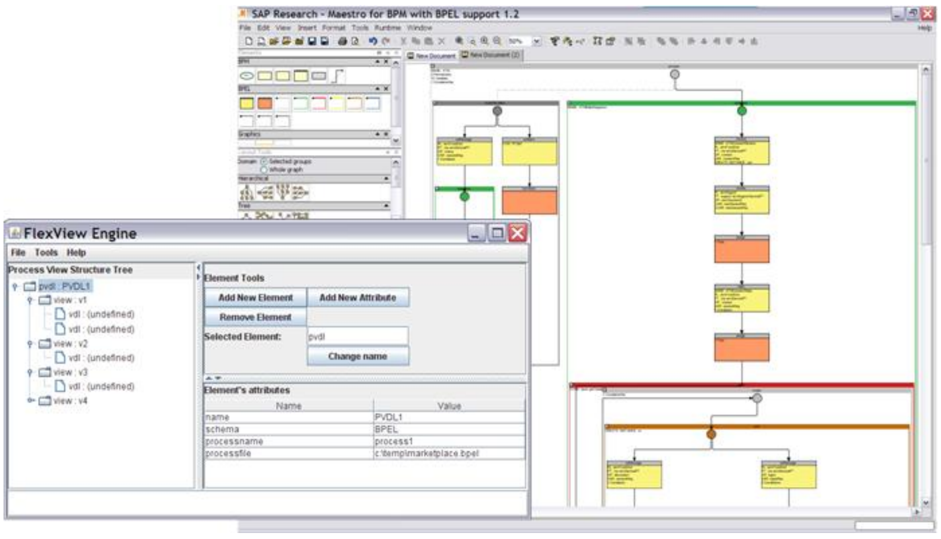
4 Incorporation into WS-BPEL

To enable abstraction and concretisation for WS-BPEL processes, we first need to incorporate the proposed model into WS-BPEL. As listed in Table 1, the main structural constructs of our model correspond to proper WS-BPEL elements. In WS-BPEL, every edge is implicitly represented, i.e., the execution sequence is determined by the occurrence sequence of elements nested in <sequence>, <pick>, <flow>, <while>, <switch> elements.

We have developed a prototype for the proof-of-concept purpose. This prototype is based on SAP Research Maestro for BPEL, with extension on process views. This prototype is purely programmed in Java, and utilises some packages from Tensegrity Software [8] for user interface design. Extensible Stylesheet Language Transformation (XSLT) [9] is selected as the technical tool to enforce process abstraction and concretisation. The FlexView engine is responsible for handling the generation of process views according to the user's requests and the pre-defined sub process hierarchy, while the Maestro is used as the displaying tool to represent process views graphically. Users send requests for "zooming in" or "zooming out" the representation of a business process through the FlexView engine, and see the result views in the Maestro. The user interfaces of FlexView and Meastro are given in Figure 5.

Table 1. WS-BPEL elements and our structural constructs

Structural construct	*WS-BPEL element*	*Description*
Task sequence	<sequence>	Allow for sequential execution of tasks.
A pair of *Or-Split/Join* gateways with conditions	<pick>	Perform the non-deterministic execution of one of several paths depending on an external event.
A *Loop* gateway with conditions	<while>	Perform a specific iterative task repeatedly until the given condition becomes false.
A pair of *Or-Split/Join* gateways with conditions	<switch>	Perform a conditional behaviour with a set of branches.
A pair of *And-Split/Join* gateways	<flow>	Perform parallel execution of a set of branches.
A synchronisation link	<link>	Support the synchronisation between tasks or gateways on the branches inside a <flow> element.
A sub process	<scope>	Originally used for defining the compensation scope for fault handling in WS-BPEL, yet here we use it to store the structural content and contextual information (such as variables and declarations), for sub processes.
A dummy branch of a split/join structure	<empty>	Originally used to denote a dummy task, yet here we use it to stand for a dummy branch of a split/join structure.



pvdl : PVDL1
view : v1
vdl : (undefined)
vdl : (undefined)
view : v2
vdl : (undefined)
view : v3
vdl : (undefined)
view : v4

Selected Element: pvdl

Name	Value
name	PVDL1
schema	BPEL
processname	process1
processfile	c:\temp\marketplace.bpel

Fig. 5. View generation system architecture

In current version, users have to define the sub process hierarchy in advance. However, we are developing necessary mining techniques to identify typical process patterns for defining sub processes. With such support, our FlexView system can automatically or semi-automatically create the sub process hierarchy.

5 Related Work and Discussion

Works on workflow/process views are related to ours. In regard to structural consistency during the process transformations, Liu and Shen [10] proposed an order-preserving approach for deriving a structurally consistent process view from a base process. In their approach, the generation of "virtual activities" (compound tasks) needs to follow their proposed membership rule, atomicity rule, and order preservation rule. Recently, Eshuis and Grefen [11] formalised the operations of task aggregation and process customisation, and they also proposed a series of construction rules for validating the structural consistency. Martens [12] discussed the verification on the structural consistency between a locally defined executable WS-BPEL process and a globally specified abstract process based on Petri net semantics. Compared with these work, first of all, our approach focused more on realising the process transformation at technical level rather than theoretical level. Secondly, in the mentioned works, the customisation process actually lost some tasks. Yet, our approach preserved the hidden tasks and necessary mapping relations, and thus supported both abstraction and concretisation operations. Finally, synchronisation links were considered in our approach.

To support process privacy and interoperability, many works targeted at applying workflow/process views in the inter-organisational collaboration environment. van der Aalst and Weske [13] proposed a "top-down" workflow modelling scheme in their public-to-private approach. Organisations first agree on a public workflow, and later each organisation refines the part it is involved in, and thereafter generates its private workflow. This work reflected a primitive idea of workflow view. In [14], Schulz and Orlowska focused on the cross-organisational interactions, and proposed to deploy coalition workflows to compose private workflows and workflow views together to enable interoperability. Issam, Dustdar et al. [15] extracted an abstract workflow view to describe the choreography of a collaboration scenario and compose individual workflows into a collaborative business process. By deploying workflow views in the workflow interconnection and cooperation stages, their approach allows partial visibility of workflows and resources. Our previous works [16, 17] also established a relative workflow model for collaborative business process modelling. A relative workflow for an organisation comprises the local workflow processes of the organisation and the filtered workflow process views from its partner organisations. In this way, this approach can provide a relative collaboration context for each participating organisation. Some follow-up work targeted at the instance correspondence [18] and the process evolvement [19] in collaborative business processes, as well as role-based process view derivation and composition [7]. In supplement to these works, our approach provided a practical implementation solution by incorporating the view concept into a popular standard business process modelling language. The abstraction and concretisation functions were naturally applicable to support privacy protection or perception control in the collaboration environment.

Proviado project [20] adopted process views for personalised visualisation of large business processes, and they allowed some trade-off between the structural consistency and the adequate visualisation. Our work firmly complied with the proposed structural consistency and validity rules, and supported bi-directional process view operations.

Our work is motivated by practical requirements from areas of process visualisation, process analysis, user friendly process representation, and so on. The work brings the process view concept to the technical level, and incorporates it into a standard process modelling language. In summary, this work contributes to the following aspects:

(1) Abstraction and concretisation functions towards process representations. With these two operations, users are allowed to choose and switch among different views of the same business process. In this way, our approach caters for the diversity of users' interests, authority levels, and so on. Although this paper chooses WS-BPEL as the candidate model to apply abstraction and concretisation functions, the essential idea of the proposed approach is applicable to most process models, like Business Process Modelling Notations (BPMN) [21], Petri net based workflow models, etc.

(2) Information preservation and structural consistency during transformation. The proposed model and developed algorithms guarantee that our process abstraction and concretisation are lossless in information and consistent in structure. Consequently, the two operations can be performed back and forth rather than one way only.

(3) Deployment in Web service domain using WS-BPEL language. The whole framework is completely incorporated into WS-BPEL via a prototype, which applies XSLT techniques and external repositories to realise WS-BPEL process abstraction and concretisation.

6 Conclusions and Future Work

This paper proposed a framework to support abstraction/concretisation functions towards flexible process view representation. A model was defined to describe the process components and their relations, while a set of algorithms were developed to enforce the abstraction/concretisation operations in compliance with the defined structural consistency rules. The whole framework was incorporated into WS-BPEL language, and a prototype was also developed for the proof-of-concept purpose.

Our future work is to refine the process component model, and further investigate the techniques to automat the generation of the sub process hierarchy. Besides, we plan to investigate similar use cases using BPMN as graphical representation.

Acknowledgement

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

7 References

1. Smith, H., Fingar, P.: Business Process Management - The Third Wave. Mehan-Kiffer Press (2003)
2. Khoshafian, S.: Service Oriented Enterprise. Auerbach Publisher (2006)
3. Papazoglou, M.: Web Services: Principles and Technology. Prentice Hall (2007)
4. Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web Services - Concepts, Architectures and Applications. Springer (2004)
5. Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S.: Business Process Execution Language for Web Services (BPEL4WS) 1.1. (2003)
6. Zhao, X., Liu, C., Li, Q.: Challenges and Opportunities in Collaborative Business Process Management. Information System Frontiers (2008)
7. Zhao, X., Liu, C., Sadiq, W., Kowalkiewicz, M.: Process View Derivation and Composition in a Dynamic Collaboration Environment. the 16th International Conference on Cooperative Information Systems, Monterrey, Mexico (2008) 82-99
8. Tensegrity Software (www.tensegrity-software.com).
9. XSLT (http://www.w3.org/TR/xslt).
10. Liu, D.-R., Shen, M.: Workflow Modeling for Virtual Processes: an Order-Preserving Process-View Approach. Information Systems **28** (2003) 505-532
11. Eshuis, R., Grefen, P.: Constructing Customized Process Views. Data & Knowledge Engineering **64** (2008) 419-438
12. Martens, A.: Consistency between Executable and Abstract Processes. the 7th IEEE International Conference on e-Technology, e-Commerce, and e-Services, Hong Kong, China (2005) 60-67
13. van der Aalst, W.M.P., Weske, M.: The P2P Approach to Interorganizational Workflows. International Conference on Advanced Information Systems Engineering (2001) 140-156
14. Schulz, K.A., Orlowska, M.E.: Facilitating Cross-organisational Workflows with a Workflow View Approach. Data & Knowledge Engineering **51** (2004) 109-147
15. Issam, C., Schahram, D., Samir, T.: The View-Based Approach to Dynamic Inter-Organizational Workflow Cooperation. Data & Knowledge Engineering **56** (2006) 139-173
16. Zhao, X., Liu, C., Yang, Y.: An Organisational Perspective on Collaborative Business Processes. the 3rd International Conference on Business Process Management. Lecture Notes in Computer Science, Nancy, France (2005) 17-31
17. Zhao, X., Liu, C.: Tracking over Collaborative Business Processes. the 4th International Conference on Business Process Management (2006) 33-48
18. Zhao, X., Liu, C., Yang, Y., Sadiq, W.: Handling Instance Correspondence in Inter-Organisational Workflows. the 19th International Conference on Advanced Information Systems Engineering, Trondheim, Norway (2007) 51-65
19. Zhao, X., Liu, C.: Version Management in the Business Process Change Context. the 5th International Conference on Business Process Management, Brisbane, Australia (2007) 198-213
20. Bobrik, R., Reichert, M., Bauer, T.: View-Based Process Visualization. the 5th International Conference on Business Process Management, Brisbane, Australia (2007) 88-95
21. OMG: Business Process Modeling Notation (BPMN 1.1). (2008)