Merging Workflows: A New Perspective on Connecting Business Processes

Shuang Sun[1], Akhil Kumar[2], John Yen[1]
School of Information Sciences and Technology[1], SMEAL College of Business[2]
Pennsylvania State University
University Park, PA 16802

Abstract

This paper describes the concept of workflow-merge and methods for merging business processes. Our research goal is to improve the responsiveness of corporations' business processes and fit them into dynamic business environments. We grouped merges in four categories according to the type of merge: *sequential, parallel, conditional,* and *iterative*. By analyzing these categories, we found that a merge cannot always yield a sound result. To avoid invalid merges, one should deliberately choose merge-points between which a sub-workflow, called a merge region, is well structured. These findings are important for guiding other workflow-merge research. We have identified various application areas where our techniques and results can be applied, such as: (1) simulations that can help decision makers to visualize merges for business processes; and (2) virtual enterprises that require flexibility in business operations. The paper discusses fundamental concepts, models, and methods of workflow merging. We leave issues of more complex merge problems, such as merge conflicts, semantic ambiguities and workflow splits for future research.

Keywords: Workflow management; workflow merge; workflow modeling; business process reengineering; sequential; parallel; conditional; iterative.

1. Introduction

For agile business operation, modern corporations must make frequent business process changes as well as organizational changes through mergers and acquisitions. In 2001, Hewlett-Packard Company and Compaq Computer Corporation announced a merger agreement to create an 87 billion dollar global technology leader. The merged company offers the most complete set of products and services in the IT industry with expected cost savings of approximately 2.5 billion dollar a year [10]. Many important issues arise in integrating the two giant organizations, one of them being how to integrate their business processes. Since frequent changes in business processes and operations are becoming increasingly common, both through internal reorganizations and through mergers and acquisitions, we have conducted preliminary research on

how workflows can be modified dynamically to adapt to such changes. In addition, our research provides support for complex process composition, i.e., creating complex workflow processes from simple ones.

A workflow may be modified at a *schema* level that defines a workflow process or at an *instance* level that represents a *specific instance* of an already defined process [12]. For example, a two-step workflow, "order" then "deliver", represents a simple business process at the schema level. Within this process there maybe orders or instances of process "order" for a particular customer. In this paper, we define the workflow-merge concept and methods focused at the schema level. This research topic is important and has not been fully addressed by previous research efforts. Process reengineering and evolution have been studied from different perspectives, such as business process integration [5, 17], using generic workflow modeling methods to ensure flexibility [2], describing methods for workflow evolution [12], improving workflow interoperability [4, 9, 11, 15], and enhancing exception handling capabilities [14, 19]. Dealing with more than one process makes a workflow-merge different from other problems that commonly assume a single process, and makes existing methods inefficient for addressing workflow merging issues. Therefore, this topic does not fit well into the existing research frameworks. For example, the classic process integration methods [5, 17] collaboratively bridge, adapt, and exchange information without actually modifying the processes of the business partners. By contrast, a workflow merge offers process level integration without considering how to connect the merged step pieces. Naturally, when company merges, both process integration and merger of processes are necessary for streamlining their operations.

In the remainder of the paper, we first introduce a workflow modeling method with Petri nets. Next, in sections 3 and 4, we introduce the workflow-merge concept and validate our approach. Then, in section 5, we discuss merge-point detection, and other issues such as conflicts, semantic ambiguities, and impact of merges on organizational roles and resources. Finally, Section 6 gives brief concluding remarks.

2. Workflow Modeling

Many research efforts have investigated methods for modeling workflow processes, which define the steps of business operations. Dumas and Hofstede tried to specify workflows with activity diagrams of the Unified Modeling Language (UML) [6]. They demonstrated that activity diagrams

can provide the expressive power that is required by most applications, and showed that an activity diagram is more powerful for express processes than most of the languages found in commercial workflow systems. A recent study by Aalst and Kumar has demonstrated that the Extensible Markup Language (XML) can be used to model inter-organizational workflows [4]. The main contribution of that research is to support process exchange through the internet. Aalst [1] has mapped the workflow concepts into *Petri nets*, giving a more formal way to represent and verify processes. In this paper, we also use *Petri nets* to specify workflows and related concepts. Dussart et al compared the workflow modeling methods including Petri-nets, WfMC, UML, ANSI, and EPC [7] on basis criteria such as formal basis, executability, ease of visualization, etc. Their study showed that Petri-nets satisfied most of the criteria, and were therefore desirable. However, the merge concept and algorithms are independent of modeling techniques, and not limited to Petri-nets. We choose Petri-nets mainly because they offer a formal basis that helps to determine soundness of a merge. In this section, we briefly introduce important Petri net concepts and how to use Petri nets to represent a workflow.

Definition 1 (Petri net)

A Petri net is a triple (P, T, F):

- *P is a finite set of places,*
- *T is a finite set of transitions* ($P \cap T = \phi$)
- $F \subseteq (P \times T) \cup (T \times P)$ *is a set of arcs (flow relation)*

A place *p* is called an *input place* of a transition *t* if and only if there exists a directed arc from *p* to *t* . Place *p* is called an *output place* of transition *t* if and only if there exists a directed arc from *t* to *p* . We use $\bullet t$ ($t\bullet$) to denote the set of input (output) places for a transition *t*, while $\bullet p$ ($p\bullet$) is the set of transitions sharing *p* as an input (output) place. Note that we restrict ourselves to arcs with weight 1. In the context of workflow procedures it makes no sense to have other weights, because places correspond to conditions.

Definition 2 (WF-net) [1]

A Petri net PN = (P, T, F) is a WF-net (WorkFlow net, or Workflow for short) if and only if

(i) PN has two special places: i and o. Place i is a source place: $\bullet i = \phi$; Place o is a sink place: $o\bullet = \phi$.

(ii) If we add a transition t to PN which connects place o with i, then the resulting Petri net is strongly connected.*

Thus, WF-nets are a special kind of Petri nets with the property that there is one source place and one sink place. Places in the set *P* represent nodes that contain tokens; transitions in the set T correspond to tasks or activities. By examining the locations of tokens, it is possible to determine the state of a Petri net, i.e., what tasks have been completed. Places can also represent conditions, as when one of multiple branches emerging from a place has to be chosen in an or-split to be discussed shortly. Further note that the requirements stated in Definition 2 are minimal requirements. Even if these requirements are satisfied, a workflow process may still cause potential deadlocks. Although a Petri-net does not have any notion of these constructs, we can use Petri-nets to model commonly used workflow constructs such as AND-splits, AND-joins, OR-splits and OR-joins. Figure 1 shows each of these constructs as Petri net representations. In a workflow context, an AND-split with *n* branches is supposed to represent parallelism among *n* activities; an OR-split with *n* parallel branches is supposed to represent a choice among *n* possible activities. An AND-split (OR-split) with *n* outgoing branches usually has a corresponding AND-join (OR-join) with *n* incoming branches. For example, when creating a sales order, a condition of whether a customer is a preferred one may lead to different branches for preferred and regular customers. This process may be represented as an OR-split with the customer type as the split condition. Moreover, after creating the order there may be two independent follow-up actions: sending an order confirmation to the customer and sending a delivery note to the warehouse. These parallel actions may be represented with an AND-split.

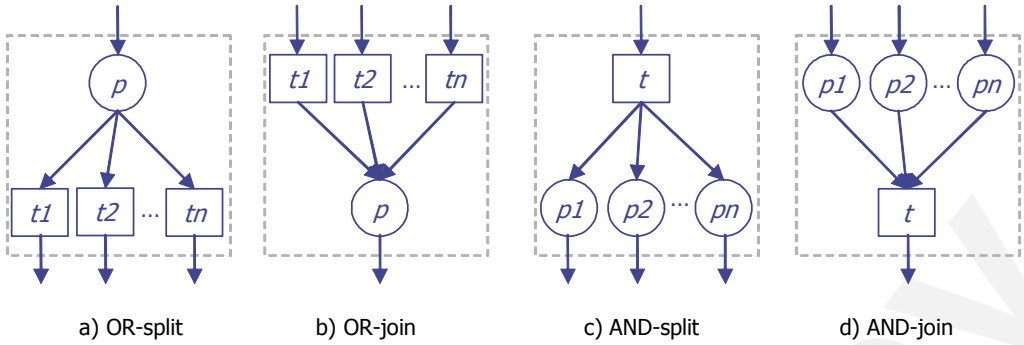

Figure 1: Petri-net representations of commonly used workflow constructs.

A structured workflow is a workflow where each OR-split has a matching OR-join and each AND-split has a matching AND-join, and moreover, if multiple splits and joins are present, they are properly nested inside each other. A simple algorithm can be used to check if a workflow is structured in this way. Our following definitions of structured WF-net and well-behaved WF-net are based on Kiepuszewski et al's work on structured workflow modeling [13].

Definition 3 (Structured WF-net)

A structured WF-net is a WF-net where:

1. A workflow consisting of a single activity is a SWF.

2. The concatenation of two SWF workflows, X and Y, where the final activity of X has a transition to the initial activity of Y, is a SWF. (see Figure 2a)

3. If X1, …, X n are SWFs, they can be combined in parallel by a pair of and-split node and and-join nodes. (see Figure 2b)

4. If X1, …, X n are SWFs, they can be combined in parallel by a pair of or-split and or-join nodes. An or-split is also called a choice or decision node. (see Figure 2c)

5. If X and Y are SWFs, and j(s) are two-way or-join (or-split) nodes, then the workflow with transition X between j and s, and transition Y between s and j, is also a SWF. (see Figure 2d)

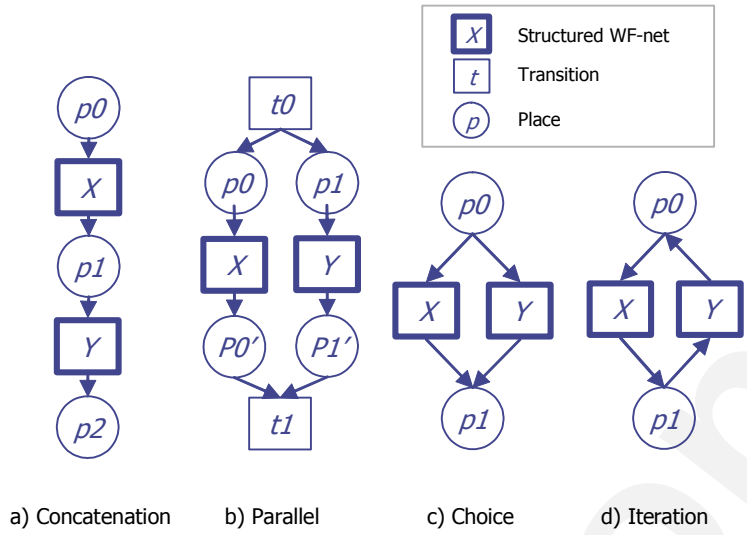

Figure 2: Combining structured WF-nets (X and Y) to create more complex structured WF-nets.

Definition 4 (Well-behaved WF-net)

A workflow model is well-behaved if it can never lead to deadlock nor can it result in multiple active instances of the same activity. Every structured workflow model is well-behaved [13].

3. Workflow-merge Concepts

In this section we introduce workflow merging concepts, taxonomy, and general merging algorithms.

3.1 Workflow Merging – Scenarios

Scenario One: Organizational Merge

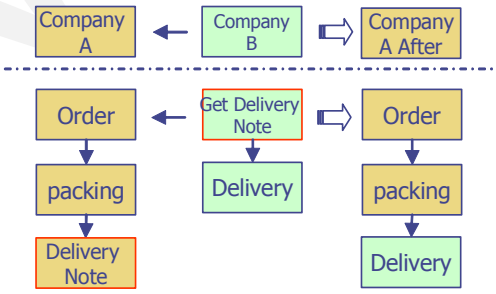

Figure 3: A scenario for merging two organizational processes.

Figure 3 shows the first scenario[1] in which a computer manufacturer (company A) merges with its logistics services provider (company B), which is responsible for delivery. Company A takes customer orders and packs computers, then it sends a delivery note to company B. Upon receiving the delivery note, B will ship the computers to customers. In an attempt to reduce costs, A decided to acquire company B and deliver computers by itself. The merged result has three tasks: order, packing, and delivery.

Scenario Two: Business Process Reengineering

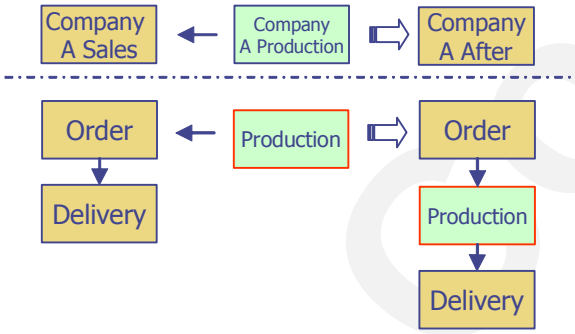

Figure 4: A business process reengineering scenario.

In the second scenario (Figure 4), a sales process is merged with a production process. Suppose company A is a computer manufacturer, and it has two functional departments, sales and production. The sales department takes customers' orders and ships finished products. The production department schedules factory production according to sales forecasts. In order to dynamically manage their production to meet market demand changes, company A decides to use a new production mode – make to order (MTO). In the MTO mode, the production department only produces after new sales orders arrive. In other words, the production process will have to be merged into the sales process.

3.2 Concepts of Workflow Merge

We define workflow-merge as *the process of combining one workflow schema into another and removing redundant steps without losing necessary ones*. In the first scenario, the workflow of

[1] The steps or transitions that are used in examples do not necessarily correspond to atomic business operations found in real business situations. They may correspond to a function that is abstracted from a group of sub-steps. For example, "order", here, may correspond to a collection of transactions including request for quote, quotation, purchasing, and ordering. However viewing the steps as atomic steps or as abstract ones should not make any difference in understanding the concepts, algorithms, or other arguments presented in the paper.

company B is merged into the workflow of company A. After the merge, for example, company A keeps all of the *necessary* steps such as order, packing, and delivery, while *redundant* steps such as sending and receiving delivery notes are removed. We call the original workflows that are combined as *merging workflows,* and the resulting workflow the *merged workflow*.

Definition 5 (Workflow-merge)

When a WF-net $PN' = (P', T', F')$ *is combined with another WF-net* $PN = (P, T, F)$*, we call the process a workflow-merge if and only if*

(i) *the result is a new WF-net* $PN'' = (P'', T'', F'')$

(ii) $T'' \subseteq T \cup T'$

(iii) $P'' \subseteq P \cup P' \cup P_m$ *(where* P_m *are new merge-points)*

(iv) $F'' \subseteq F \cup F' \cup (T'' \times P_m) \cup (P_m \times T'')$

We call the merge function as $Merge(PN, PN')$ [2], and we call PN the *primary WF-net* and PN' the *secondary WF-net.*

According to condition *(ii)*, the merged workflow should not involve any new tasks that are not in the merging workflows; condition *(iii)* ensures that only result merge-points can include new conditions; condition *(iv)* states that dependencies in the merging workflows should be compliant with the ones in merged workflows. More importantly, a workflow-merge should not violate the soundness properties [1]. In section 4, we show how to achieve a sound merge by keeping properties of well-structureness in a merge function.

Definition 6 (Merge-point)

When a primary WF-net, $PN' = (P', T', F')$*, is merged with a secondary WF-net, PN = (P, T, F), and the merged workflow is a WF-net,* $PN'' = (P'', T'', F'')$*, a place node such as* $p \in PN$*,* $p' \in PN'$ *or* $p_m \in PN''$ *is called merge-point if and only if*

(i) *if* $p \in PN \bigcap PN''$, $\exists t(t \in (\bullet p \cup p\bullet)) \wedge (t \in PN')$*, or*

[2] We use the operator symbol $\rightarrow$ to represent merge in diagrams.

(ii) $if\ p' \in PN' \bigcap PN'',\ \exists t(t \in (\bullet p' \bigcup p' \bullet)) \wedge (t \in PN)$, *or*

(iii) $p_m \notin PN \bigcup PN'$

The place nodes where two merging workflows, say wf1 and wf2, are connected are called *merge-points*. Merge-points are always in pairs and they are noted as (p/p') which means that p' from *wf2* will merge with p from *wf1*. The result merge-points p_m may be p as in (i), p' as in (ii), or a new place as in (iii) which is different from p or p'. P_m, in Definition 5, is the set of result merge-points. A merging workflow contains at least one merge-point. When two merge-points (*p1* and *p2*) are specified in a merging workflow, if *p1* is prior to *p2*, *p1* is called beginning merge-point, and *p2* is called ending merge-point.

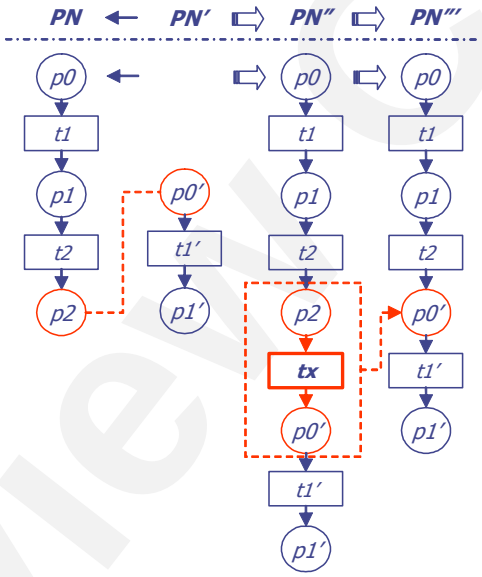

Figure 5: Workflow-merge example (a sequential merge).

For example, Figure 5 depicts a merge function—$Merge(PN, PN')$, and the merged workflow is PN''. We call *tx* (in PN'') an auxiliary transition node that helps to connect two places because, in a Petri net, two place nodes *(*such as *p2* and *p0'* *)* cannot be connected directly without a transition node in between. Unlike a normal WF-net node, an auxiliary node represents neither conditions nor tasks. By eliminating *redundant nodes* (*p2*) and *auxiliary node* (*tx*) we can reduce PN'' into PN'''. This is a workflow *simplification step.* Places *p2* and *p0'* are the merge-points

and the resulting merge-point is *p0'*. We will explain *auxiliary nodes and simplification* further in section *3.3.2*.

3.3 Workflow Merging Taxonomy

We group workflow-merges according to two dimensions: lossy/lossless and merge pattern. The first grouping method yields two types of merges: *lossy merge* or *lossless merge*. With the second dimension, we sort merges by their patterns: sequential, conditional, parallel, or iterative merges.

3.3.1 Workflow Merging Taxonomy – lossless or lossy

Definition 7 (Lossless merge)

When a WF-net $PN' = (P', T', F')$ *is merged with another WF-net PN= (P, T, F) and the result is a new WF-net* $PN'' = (P'', T'', F'')$*, we call the merge lossless if and only if* $T \bigcup T' \subseteq T''$. In a lossless merge, all tasks in the merging workflows are preserved.

Definition 8 (Lossy merge)

When a WF-net $PN' = (P', T', F')$ *is merging with another WF-net PN=(P,T,F) and the result is a new WF-net* $PN'' = (P'', T'', F'')$*, we call the merge lossy if and only if* $\exists t(t \in (T \bigcup T') \wedge t \notin T'')$. A lossy merge does not guarantee that all tasks are retained after merge.

A lossy merge is not necessarily bad because it can often result in improvement of a process by merging two tasks into one, and thus pruning *redundant* tasks. Determining what tasks in merging workflows are redundant requires more knowledge about the process and certainly is not a trivial problem that may easily to be automated. On the other hand, if a lost task is not redundant, the lossy merge yields an incomplete process, which is undesirable.

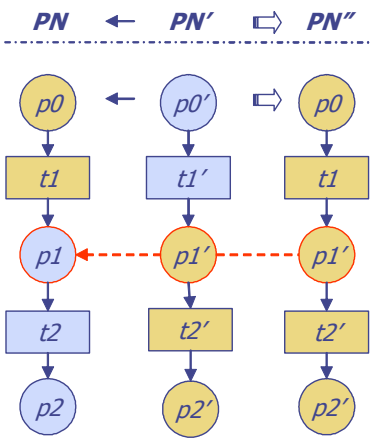

Figure 6: ***Merge(PN, PN').***

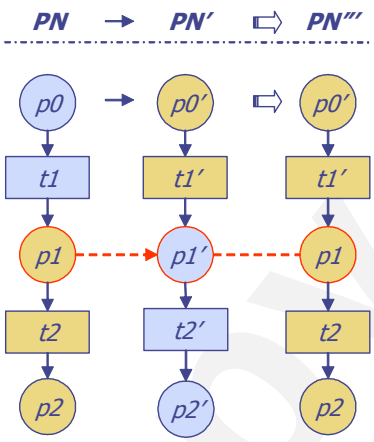

Figure 7: ***Merge(PN', PN).***

A *lossy* merge must specify the *merge direction* because it is important to note that *Merge (PN, PN') ≠ Merge (PN', PN).* In Figure 6, for example, the merge direction is specified as *Merge (PN, PN')*, and the result is *PN"*. In Figure 7, however, the merge direction is specified as *Merge (PN', PN)*, and the result is *PN'"*, which is different from *PN"*[3].

We can either explicitly specify the merge direction, or infer it from a semantic understanding of the individual processes that are merged. For example, in the first scenario of section 1, if company A merged with company B, one would infer that company B's process occurs after company A's because receiving a delivery note depends on the previous process of creating it. By inferring that B is after A, we can determine the merge direction as *Merge (A, B)*.

3.3.2 Workflow Merging Taxonomy – by merging result pattern

The second dimension of our taxonomy is based on the result patterns. Workflows have basic process patterns, such as *sequential, parallel, conditional,* and *iterative* [3]. In basic merge situations, we assume that two merging workflows contain a single pattern in the merged workflow and two pairs of merge-points: a pair of beginning merge-points and a pair of ending merge-points. In more complex situations, a merged workflow may contain multiple merge-points. However, a complex merge can be represented by combining simple merge patterns. In the rest of this section, we define the types of merge and give algorithms of merge functions. The merge

[3] Definition 9 and algorithm Merge_Seq explain how the merge results, *PN" and PN'"*, are obtained.

functions cannot guarantee that a merged workflow has a sound process. We will discuss the issue of soundness in detail in section 4.

Definition 9 (Sequential merge)

When two workflows, PN and PN', merge at merge-points (p1/ p1', p2/ p2')[4]*, if p1 is replaced by p1' and p2 by p2', it is a sequential merge. The processes of PN between p1 and p2 have been replaced by the processes in PN' between p1' and p2'. No new places are created in a sequential merge.*

In a sequential merge, parts of one workflow merge with another in a sequential manner. There are two types of sequential merges: *replacement merge* and *insertion merge*. Figure 8 shows a replacement— *Merge_Seq (PN, PN', p1, p0',p2, p1')* where *t2 is replaced by t1'*. Figure 9 shows an insertion— *Merge_Seq (PN, PN',p1, p1',p1, p2')* where place *p1*, in the primary workflow, is both the start place and the end place. Recall that in Figure 3, the "Delivery" process of Company B was put after the "Packing" process of Company A in the merged new workflow called "Company A After"; so the organizational merge in Figure 3 is also an example of sequential merge.

[4] *p1/ p1' and p2/ p2'* are merge-points: *p1/ p1' are beginning merge-points, and p2/ p2' are ending merge-points.*

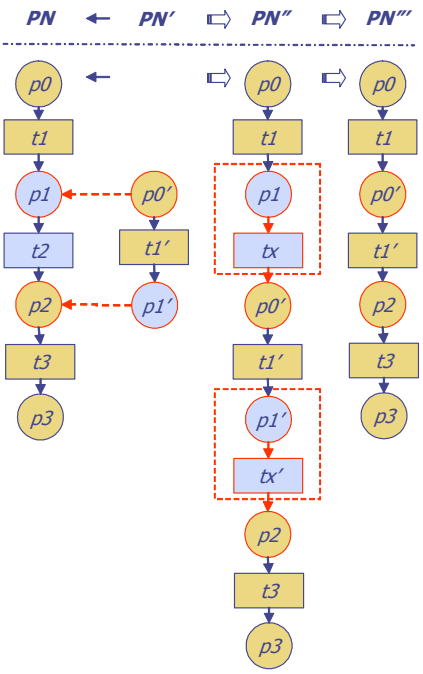

Figure 8: Replacement merge.

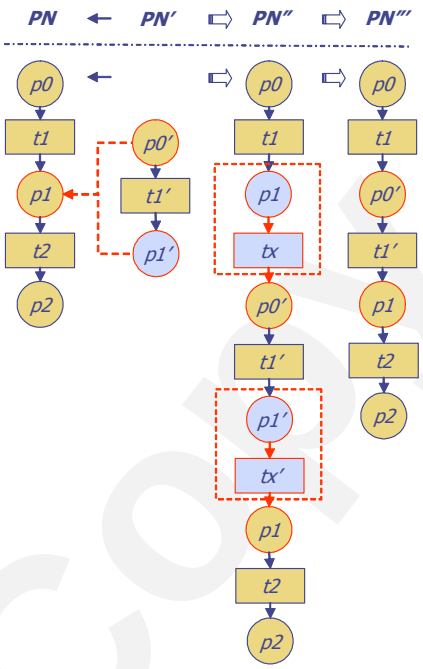

Figure 9: Insertion merge.

As mentioned before, a sequential merge involves two steps: *initial merge* and *simplification*. In the first step, merging workflows are combined through two pairs of auxiliary-node sets at the merging-points. In Figure 8, an auxiliary-node, *tx*, is connected to the merge-point, *p1*, while another auxiliary-node *tx'* is connected to *p2*. As discussed above, the auxiliary transition nodes are required to connect two place nodes in a Petri net. Between the auxiliary-node is the merge region of *PN'*. The use of the auxiliary-nodes in the merge guarantees a sequential relation between the merging workflows.

To simplify and make the workflow nets concise, we need to eliminate the redundant nodes and auxiliary transitions. In Figure 8, the merge begins with *p1*, which contains condition(s) for task *t2*. Task *t2* will be eliminated after the merge, therefore *p1* is redundant. In addition, *p1'* is also redundant because it is the post-condition of *t1'*, a deleted transition. Task *tx* and *tx'* are auxiliary transitions that indicate sequential merges and when fired, they will simply pass tokens to the output nodes. In both Figure 8 and 9, *PN"* represents the result of the initial merge and *PN"'* is the result after simplification. It is easy to find that *PN"* and *PN"'* is *equivalent* in the processes

they represent. Because the auxiliary nodes are not essential in the final merge result, we provide an algorithm that obtains the merged workflow without involving any auxiliary nodes.

The sequential merge (see Figure 8) algorithm *Merge_Seq (PN, PN',p1, p1',p2, p2') is defined by the following steps*:

1. *Remove p1, the arcs after p1, and the arcs before p1'*
2. *Connect • p1 to p1' (i.e., connect the input transitions of p1 to p1' with new arcs)*
3. *Remove • p2, arcs before p2, and arcs after p2'*
4. *Connect • p2' to p2 (i.e., connect all input transitions of place p2' to the place p2 with new arcs)*
5. *The process that contains p1' and p2 is the merged workflow.*

Definition 10 (Parallel merge)

When two workflows PN and PN' merge at merge-points (p1/ p1', p2/ p2')[5]*, if, after the merge, p1 and p1' construct an AND-split and p2 and p2' construct an AND-join, it is a parallel merge, i. e., PN and PN' have been connected at point p1/ p1' and p2/ p2' in parallel. No new places are created in a parallel merge.*

A parallel merge (see Figure 10) algorithm *Merge_Par (PN, PN',p1, p1',p2, p2')* is defined in the following steps:

1. *Remove the arcs before p1' and connect • p1 to p1'*
2. *Remove the arcs after p2' and connect p2' to p2•*

[5] *p1/ p1' and p2/ p2'* are merge-points: *p1/ p1'* are beginning merge-points, and *p2/ p2'* are ending merge-points.

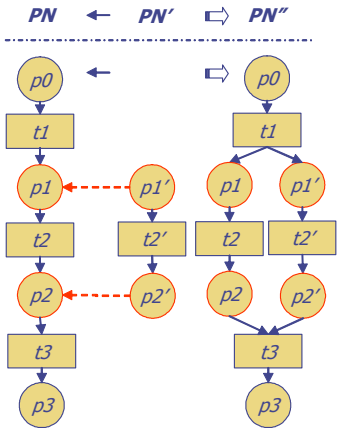

Figure 10: Parallel merge.

Parallel merges are used when the causal order between the merging workflows is not relevant. Figure 10 shows a parallel merge where the order of tasks *t2* and *t2'* is not relevant.

Definition 11 (Conditional merge)

When two workflows PN and PN' merge at merge-points (p1/ p1', p2/ p2')[6]*, if p1 and p1' construct an OR-split and p2 and p2' construct an OR-join, it is a conditional merge, i.e., PN and PN' have been connected at point p1/ p1' and p2/ p2' with additional conditions. A new place called a condition place will be created in a conditional merge.*

A conditional merge (see Figure 11) algorithm *Merge_Cond (PN, PN',p1, p1',p2, p2', C)* is defined by the following steps:

1. *Remove the arcs before p1', and connect p1'• to p1*
2. *Remove the arcs after p2', and connect • p2' to p2*
3. *Modify the conditions in p1 according to new choice conditions C and p1'*

[6] *p1/ p1' and p2/ p2'* are merge-points: *p1/ p1'* are beginning merge-points, and *p2/ p2'* are ending merge-points; *C* is the condition set that determines choices for the two processes.

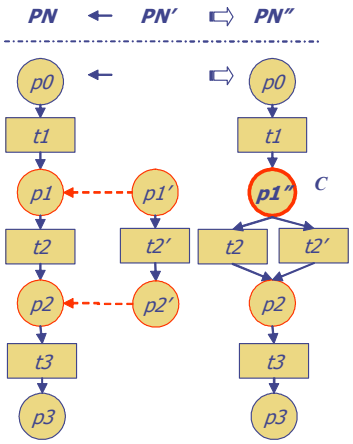

Figure 11: Conditional merge.

Figure 11 is an example of conditional merge, *Merge_Cond (PN, PN', p1, p1',p2, p2', C)*. In the merged workflow, *p1* and *p1'* have been merged into a new place called *p1"* that contains conditions for choosing between tasks *t2* and *t2'*. Because the transitions after place *p2'* are not included in the result, the conditions in *p2'* are useless in the merged workflow. Therefore, the merge-point of the secondary merging workflow is removed. In the real world, conditional merges can be used to model the combination of two processes that requires satisfying certain criteria, such as *C::= if (order_value > discount_qualify)*— to check whether a customer's order is big enough to qualify for a discount. If it qualifies then one branch out of the condition place will be taken, and otherwise, the other one will be taken.

Definition 12 (Iterative merge)

When two workflows PN and PN' merge at merge-points (p1/ p2', p2/ p1')[7]*, if p1 and p2' construct an OR-join and p2 and p1' construct an OR-split, it is an iterative merge, i.e., PN and PN' have been connected at point p1/ p2' and p2/ p1' with additional conditions. A new place will be created in an iterative merge.*

Iterative merge (refer to Figure 12) algorithm *Merge_Iterative (PN, PN',p1, p2' ,p2, p1', C)* is defined in the following steps:

1. *Remove the arcs before p1', and connect p2 to p1'* •

2. *Remove the arcs after p2', and connect • p2' to p1*

3. *Modify the conditions in p2 according to new choice conditions C and p2*

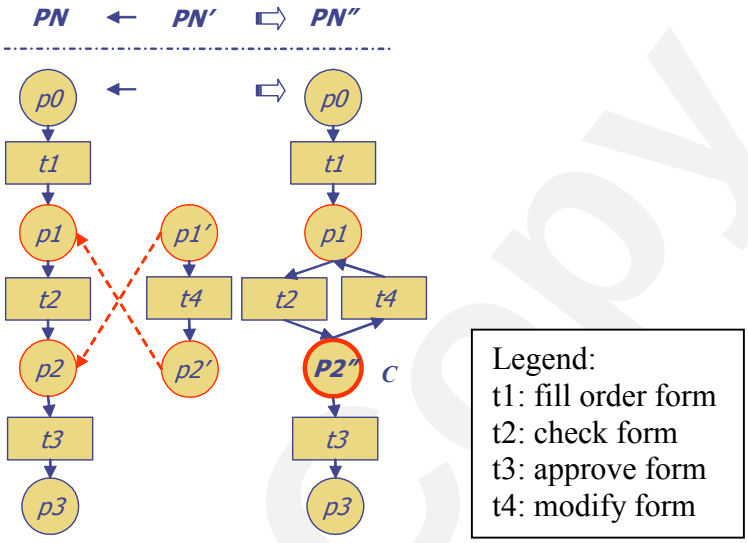

Figure 12: Iterative merge.

Figure 12 is an example of iterative merge, *Merge_Iterative (PN, PN', p1, p2' ,p2, p1', C)*. In the merged workflow, *p2"* is a new place that contains conditions for choosing between tasks *t3* and *t4*. To illustrate the need for an iterative merge, suppose *PN* is a purchase request process with the following steps: *t1* is filling the form (by an employee); *t2* is checking the form (by a secretary); *t3* is approving the purchase request (by a manager). Sometimes, however, after task *t2*, the form may need to be modified (if it is incorrect or some information is missing). Therefore, another step, in this case task *t4*, modifying the form, (by the employee) may be added as shown in Figure 12. In general, *t4* could be replaced by multiple tasks or a sub-workflow. This shows how a workflow may be modified dynamically, while still maintaining its correctness.

Definition 13 (Complex merge)

When two merging workflow PN and PN' merge at more than two pairs of merge-points, it is called a complex merge.

[7] *p1/ p2' and p2 / p1'* are merge-points: *p1and p1'* are beginning merge-points, and *p2 and p2'* are ending merge-points; *C* is the condition set that determines the ending conditions of the iterative processes.

A complex merge may involve multiple merge patterns. For example, Figure 13 shows that in a merge process, company A (primary workflow) has merged company B's "check availability process" sequentially, and it has also merged company B's "check stock" in parallel.

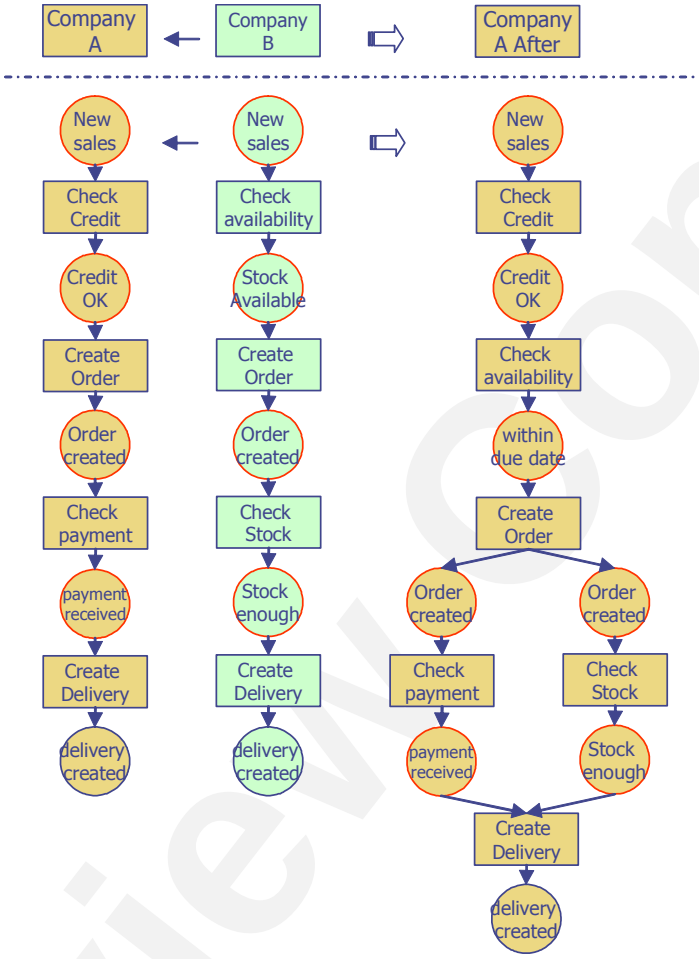

Figure 13: A complex merge scenario.

A complex merge could be expressed formally with the following merge function:

Merge_par(Merge_Seq (PNA, PNB, 'new sales', 'new sales' , 'credit OK', 'within due date'),PNB, 'order created', 'order created', 'payment received', 'stock enough').

This shows how the primitive merges discussed above can be combined to create more complex merges.

4. Workflow-merge Analysis

Without properly chosen merge-points and merge function, two merging workflows cannot yield a sound result, even a syntactically sound one. Figure 14, *Merge_Seq (PN, PN',p4, p6',p7, p1')*, gives an example of an unsound merged workflow because node *p5*, after the merge, becomes dangling, and the whole workflow, *PN",* is ill structured. This problem leads us to investigating the situations where a sound merge is not possible. In this section, we introduce the notions of *sound* and *unsound merges*, and analyze a workflow-merge at the structural level of a process.

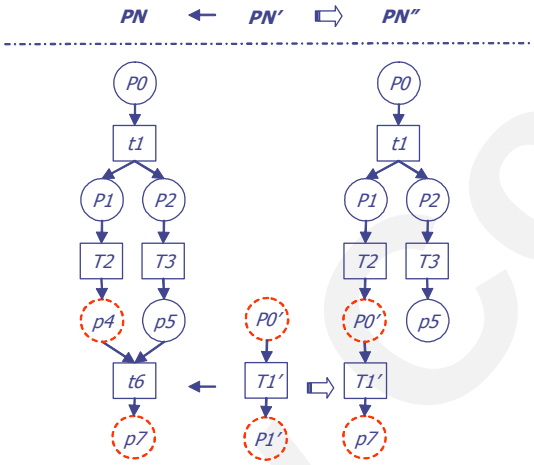

Figure 14: An unsound merged workflow.

4. 1 Sound Merge vs. Unsound Merge

If a workflow-merge yields a correct result, we call it a sound merge. On the other hand, in some situations, two workflows cannot be merged correctly, and we call such merges unsound. Figure 14, shows an unsound merge. Naturally, it is desirable to find rules that can ensure a sound merge. Therefore, we need some rules to distinguish between sound and unsound merges.

4.2 Process Structure Level of Analysis

From the process structure level of analysis, we obtained two theorems that can provide necessary conditions for sound merges. A process structural level of analysis studies a merge by treating the process between merge-points as a whole entity. We call the entity a merge region.

Definition 14 (Merge region)

When two workflow PN and PN' merge at merge-points (p1/ p1' , p2/ p2'), merge (PN, PN' ,p1, p1' ,p2, p2')[8]*, the sub-process between p1 and p2 or p1' and p2' is called merge region. Merge regions for merging workflows can be obtained through the following algorithm:*

1. *Remove • p1 and p2• (or • p1' and p2' •)*
2. *The process that contains merge-points p1 and p2 (or p1' and p2') is the merge region for the merging workflow PN (or PN')*

For example, the corresponding merge regions for *PN and PN' ,* shown in Figure 15, are *MR* and *MR'* respectively. It is easy to see that both these regions are structured, because they conform to the definition of a structured WF-net (see Definition 3).

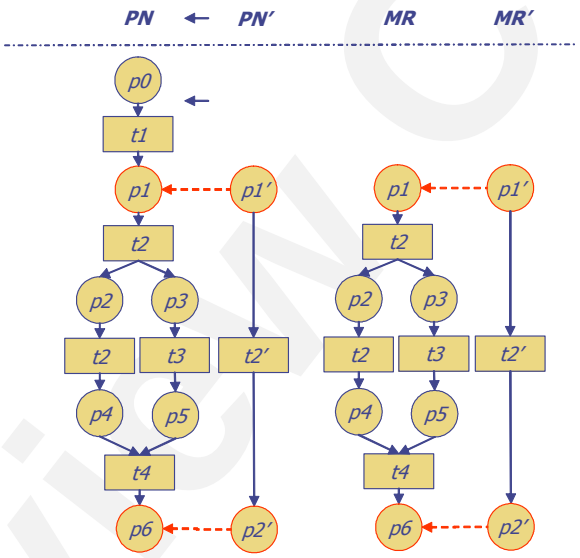

Figure 15: Merge regions (MR and MR' for PN and PN')

Theorem 1 *If the merge regions of two merging workflows are structured WF-nets, the merged workflow constructed with sequential, parallel, conditional, and iterative merge functions is structured too.*

Proof:

The proof is by construction and relies on Definition 3 where different forms of structured workflows are discussed. All the merging functions described above are based on combining

[8] *p1/ p1' and p2/ p2'* are merge-points: *p1/ p1'* are beginning merge-points, and *p2/ p2'* are ending merge-points.

structured workflows to create new workflows that are also structured. Thus, in performing a merge, we are taking a structured region from a workflow, and replacing it with another structured region. The proof follows from a case by case analysis.

Case (a): *concatenate, insert, replace*. In this case two workflows are merged by concatenation or insertion, or a structured workflow is replaced by another structured workflow. Since each workflow is structured, the resulting workflow is also structured (by Definition 3(2)).

Case (b): *parallel merge*. When two structured workflows are combined in parallel using AND-split and AND-join, the result workflow is also structured (by Definition 3(3)).

Case (c): *conditional merge*. When two structured workflows are combined in parallel using OR-split and OR-join, the result workflow is also structured (by Definition 3(4)).

Case (d): *iterative merge*. This is a variant of case (c), and here the OR-JOIN occurs first and it is followed by a matching OR-SPLIT. Since the two component workflows that are merged are structured, it follow that the resulting workflow is structured (by Definition 3(5)).

Thus, in all cases the workflow that results from the four types of merges is structured if the merge regions themselves are structured. ■

As discussed earlier, more complex merges can be created by combining these primitive merges as building blocks.

Theorem 2 *If the merge regions of two merging workflows are structured, the merged workflow constructed with sequential, parallel, conditional, and iterative merge functions is well-behaved.*

Proof:

This follows from Theorem 1 and Definition 4, which states that every structured workflow is well-behaved. Thus, we can conclude that Theorem 2 is true. ■

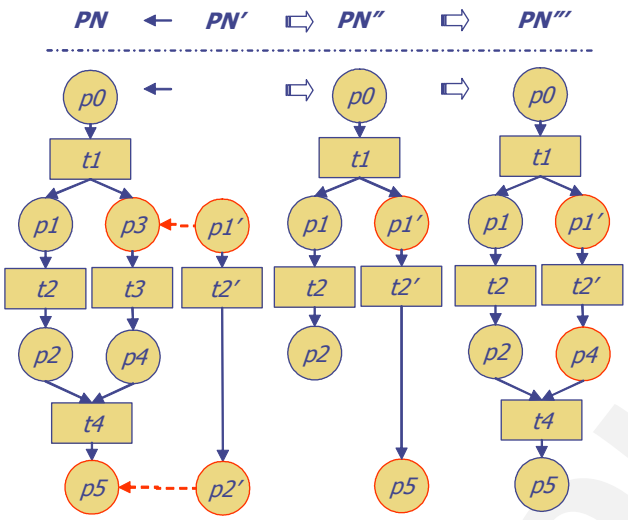

Figure 16: Example of choosing a proper merge-point.

By choosing a proper merge-point, we can change an unsound merge to a sound one. In Figure 16, *Merge_Seq (PN, PN', p3, p1',p5, p2')* result in *PN"*, which is not well-structured. If we change a merge-point of *PN* from *p5* to *p4*—thus, *Merge_Seq (PN, PN', p3, p1',p4, p2'),* the new result *PN'"* is well-structured and sound. Thus, unless two merging workflows have an inherent conflict in the sequencing of their activities, in most of the cases, one can achieve a sound merge by choosing merging points properly. In the next section, we will discuss the issue of suitable merge-point detection.

5. **Discussion**

We have defined workflow-merge concepts, categories of merges, and studied how routing and process structures affect a workflow-merge. Here, we will briefly discuss merge-point detection methods and other issues such as conflicts, semantic ambiguities, and impact of merges on organizational roles and resources.

5.1 Merge-point Detection

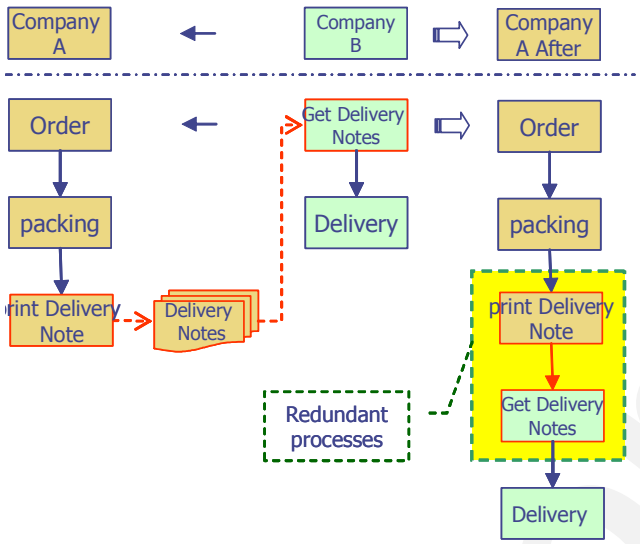

Figure 17: An example of merge point detection.

Automatically finding merge-points and applying merge functions can greatly simplify a workflow-merge task. The key step is to find out valid merge-points. The process is called *merge-point detection*. Our hypothesis is that merge-points could be detected through certain means such as reasoning about process dependencies. Because both merge-points' position and number of merge-points affects a merge result, a merge-point detection function can be evaluated by using completeness that specifies if the function has identified all the merge-points accurately and whether there are any invalid merge-points.

In Figure 17, delivery notes are generated in the step "print delivery note" and are fed into the step "get delivery notes", so "get delivery note" is dependent on "print delivery notes". Hence, we know where the merge points are. However, merging real workflows is much more complex than described here. Developing algorithms for automatic merge-point detection is a topic for future research.

5.2 Other Workflow-merge Issues

Workflow-merges are normally more complicated than the one described above. Ambiguities and conflicts cause errors. Multiple merge-points and constraints from organizational changes increase the complexity of a merge. These issues arise in real-world merge scenarios and should be addressed in a computational workflow-merge model. In this section, we will raise these issues,

but will not attempt to solve them. Moreover, the issues listed here are not an exhaustive list; we expect that other issues will appear as more research is conducted.

First, often merging workflows have conflicting dependencies between the same pair of tasks. For example, Figure 19 shows two such processes in two companies. Notice how the order of two processes, *Packing* and *Delivery*, in the two companies is reversed. Therefore, this conflict must be resolved manually before performing a merge.

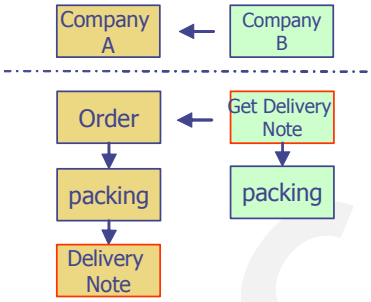

Figure 18: Merge conflicts.

Second, different companies may name workflow processes in their own way, so semantic meanings of the steps in the merging workflows should be clarified to avoid ambiguities. For example, a general term "*packing*" can represent packing for a particular part on a production line. It can also represent packing finished products into a container before shipment. If the difference in meaning is not noticed before the merge, the result will be wrong. We can either manually check the semantics or let merging workflows follow certain standards, such as RosettaNet [17, 18]. Moreover, two organizations may also use different workflow systems, which have very different ways of modeling business processes. In such cases, it becomes even harder to merge the two workflows unless one is converted into the modeling scheme of the other, or both are converted into a common scheme.

Next, the analytical method adopted in this research demands validation and evaluation in a real world application setting or in controlled simulation experiments. As discussed earlier, how to further generalize the merge concepts and algorithms may also be tested in such real applications. In such a setting, the interaction of factors such as throughput, reliability, flexibility and quality can be studied. For example, throughput of a merged workflow relies on its flexibility, quality and reliability. Those problems are often hard to study with analytical models, while they can be tackled better by application or simulation. In addition to validating the research, such application

and experiments may also provide us with valuable insights into real problems that are encountered when workflows merge.

Last, but not least, workflow splitting is another interesting topic of study. A *workflow split* is the reverse of a workflow-merge. Business units often split, or outsource parts of their processes to other companies. In such situations, it is important to develop algorithms to split a process into two or more separate sub-processes. A process split requires finding split points (as opposed to merge-points) and ensuring that the splitting is correct. We expect that the research on the workflow split can benefit from some of our present results for workflow-merge.

6. Conclusion

This paper discussed fundamental concepts, models, and methods of various types of workflow-merge operations. We formally defined what a workflow-merge is and we also proposed merge methods. By the pattern of a merge result, we grouped merges in four categories: sequential, parallel, conditional, and iterative. More importantly, we showed the conditions under which a merge will yield a sound result. We believe this framework is very promising for developing applications to serve the business world. Potential benefits lie in: (1) performing simulations that can help decision makers visualize merges for business processes, (2) creating virtual enterprises that make flexible business operations possible, and (3) planning merges that allow software agents share process knowledge. It also offers a systematic approach for building complex workflows from simple ones by incorporating changes and new sub-processes into them in a correct way. Thus, this methodology also plays a useful role in workflow evolution.

Decision makers can gain valuable experience through simulating their upcoming workflow-merge on a computer with workflow-merge applications. The simulation results can provide crucial information to help them modify their merge plan and optimize working processes. Such an effort will eventually help to cut costs, increase throughput, and create more value.

The research on workflow-merge also promises more flexible business models, such as virtual enterprises, to address dynamic business environments. A virtual enterprise is a business model that dynamically organizes small companies into its business processes. A virtual enterprise can provide more services in a flexible manner and lead to more efficiencies as compared to a single enterprise providing multiple services. Moreover, such coalitions can disband when they are no

longer effective [16]. At present, coalition formation for virtual organizations is limited. In the real world, organizations define norms to regulate the behaviors to reach common goals. We anticipate that automation of coalition formation by using workflow-merge technologies will save both time and labor. In complex settings, workflow-merge techniques may also be more effective at finding better coalitions than other technologies or manual methods.

In a multi-agent environment, if a software agent's plan (process) is incomplete, other agents may be able to help with the other parts of the solution. Then the problem is how to assimilate such piecemeal solutions into the existing processes, which is equivalent to merging several plans[8]. As defining workflows and plans are both based on process specifications and can use similar process representations, we assume workflow-merge technologies can shed some light on plan merging too.

In summary, research on workflow-merge offers a new way to connect business processes. This paper has highlighted the importance of the workflow-merge problem and also raised many unanswered questions (such as resolution of merge conflicts, semantic considerations, workflow split, etc.) for further studies.

7. References

1. Aalst, W.M.P.v.d. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, *8* (1). 21-66.
2. Aalst, W.M.P.v.d., Generic Workflow Models: How to Handle Dynamic Change and Capture Management Information? in *Conference on Cooperative Information Systems*, (1999), 115-126.
3. Aalst, W.M.P.v.d., Hofstede, A.H.M.t., Kiepuszewski, B. and Barros, A.P. Workflow Patterns, Eindhoven University of Technology, Eindhoven, 2000, WP 47.
4. Aalst, W.M.P.v.d. and Kumar, A. XML Based Schema Definition for Support of Inter-organizational Workflow. *Information Systems Research*, *14* (1). 23-47.
5. Dan, A., Dias, D.M., Kearney, R., Lau, T.C., Nguyen, T.N., Parr, F.N., Sachs, M.W. and Shaikh, H.H. Business-to-business integration with tpaML and a business-to-business protocol framework. *IBM Systems Journal*, *40* (1). 68-90.
6. Dumas, M. and Hofstede, A.H.M.t., UML Activity Diagrams as a Workflow Specification Language. in *International Conference on the Unified Modeling Language (UML)*, (Toronto, Canada, 2001), Springer Verlag.
7. Dussart, A., Aubert, B.A. and Patry, M. An Evaluation of Inter-Organizational Workflow Modeling Formalisms. *Journal of Database Management*, *15* (2). 74-104.
8. Foulser, D.E., Li, M. and Yang, Q. Theory and Algorithms for Plan Merging. *Artificial Intelligence*, *57* (2-3). 143-181.
9. Gronemann, B., Joeris, G., Scheil, S., Steinfort, M. and Wache, H., Supporting Cross-Organizational Engineering Processes by Distributed Collaborative Workflow

Management - The MOKASSIN Approach. in *2nd Symposium on Concurrent Mulitdisciplinary Engineering (CME'99) / 3rd Int. Conf. on Global Engineering Networking (GEN'99)*, (Bremen, Germany, 1999).

10. Hewlett-Packard and Compaq. Hewlett-Packard and Compaq Agree to Merge, Creating $87 Billion Global Technology Leader, 2001.
11. Hoffner, Y., Ludwig, H., Gülcü, C. and Grefen, P., An Architecture for Cross-Organizational Business Processes. in *Second International Workshop on Advance Issues of E-Commerce and Web-Based Information Systems (WECWIS 2000)*, (Milpitas, California, 2000).
12. Joeris, G. and Herzog, O., Managing Evolving Workflow Specifications. in *the 3 rd Int. IFCIS Conf. on Cooperative Information Systems (CoopIS'98)*, (New York, 1998), 310-319.
13. Kiepuszewski, B., Hofstede, A.H.M.t. and Bussler, C., On structured workflow modelling. in *Int. Conference on Advanced Information Systems Engineering (CAiSE)*, (Stockholm, 2000), Springer Verlag.
14. Klein, M. and Dellarocas, C. A Knowledge-based Approach to Handling Exceptions in Workflow Systems. *Computer Supported Cooperative Work*, *9*. 399-412.
15. Lazcano, A., Alonso, G., Schuldt, H. and Schuler, C. The WISE approach to electronic commerce. *International Journal of Computer Systems Science and Engineering*, *15* (5). 345-357.
16. Luck, M., McBurney, P. and Preist, C. Agent Technology: Enabling Next Generation Computing - A Roadmap for Agent-Based Computing, AgentLink, 2003.
17. RosettaNet. http://www.rosettanet.org, 2003.
18. Sayal, M., Casati, F., Dayal, U. and Shan, M.-C. Integrating Workflow Management Systems with Business- to-Business Interaction Standards, 2001.
19. Strong, D.M. and Miller, S.M. Exceptions and exception handling in computerized information processes. *ACM Transactions on Information Systems (TOIS)*, *13* (2). 206 - 233.