To appear in *Proceedings of Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2007)*, in conjunction with the 29th International Conference on Software Engineering (ICSE 2007), Minneapolis, MN, USA, May 20-26, IEEE Computer Society Press, Los Alamitos, California, USA (2007).

Process Evolution with Atomicity Consistency

Chunyang Ye and S.C. Cheung
Department of Computer Science and Engineering
Hong Kong University of Science and Technology
Hong Kong, China
{cyye, scc}@cse.ust.hk

W.K. Chan
Department of Computer Science
City University of Hong Kong
Hong Kong, China
wkchan@cs.cityu.edu.hk

Abstract

The processes enacting cross organizational collaboration continually evolve to adapt to changing business environments. These processes may evolve at any time during collaboration when collaborative organizations should respond timely to critical environment changes. Atomicity is an important requirement for maintaining application consistency. The satisfaction of this requirement at the commencement of collaboration may no longer be held after process evolution. This paper analyzes three basic scenarios of process evolution in cross organizational collaboration, and proposes a collection of guidelines to govern atomicity-preserving evolution during collaboration.

Keywords: Process, evolution, atomicity

1. Introduction

The enacting processes of cross organizational collaboration continually evolve to adapt to changing business environments. These processes, owned by different organizations, are usually long-running, heterogeneous, and autonomous. Organizations may want to release to their partners the restricted views, called *public views*, of their processes to protect sensitive business information. This paper assumes cross-organizational collaborations over message passing services, such as Web services, based on released public views [7].

Due to the long-running, loosely coupled and distributed nature of processes in such environments, the application consistency of collaboration is a prime concern of business organizations [5]. Consider a scenario where a retailer has ordered hundreds of smart phones from a supplier. Should there be a problem occurring in the supplier's manufacturing process, the phone delivery can be delayed. This would induce value loss to the retailer if it has pre-sold these phones but fails to deliver them in time to its clients. To avoid such value loss scenario, atomicity spheres [6] were introduced as the structural criterion of processes for their collaboration to fulfill the atomicity property, in the sense that the collaboration either succeeds, or fails without any side effects. The satisfaction of atomicity sphere in a collaboration guarantees that the effects of all its executed tasks could be compensated in case this collaboration fails. In this paper, we restrict the atomicity requirement to the satisfaction of atomicity sphere.

To check whether a collaboration fulfills the atomicity requirement, our previous work [10] proposes an approach to deriving public views from organizations' business processes. The derived views encapsulate sensitive task details and remain atomicity-equivalent to their original processes. These views can be published at service registries so that potential collaborators may use them to check the atomicity of intended collaborations. In [11], we further propose an approach to checking atomicity in a distributed fashion.

However, the evolution of one process in an organization may breach the atomicity requirement in their collaboration. This is because the evolution of a process may alter its process structures or task properties. Let us illustrate this problem using an example, as depicted in Figure 1, where a retailer orders smart phones from a supplier. Originally, the retailer is allowed to abort the collaboration after the order is confirmed. In such case, the smart phones are returned to the supplier (if the smart phones have been shipped to the retailer) and the money is refunded to the retailer. However, if the supplier process evolves to make the payment non-refundable due to its new

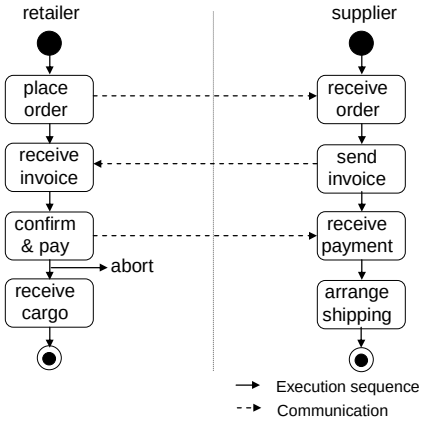

Figure 1. B2B process collaboration.

business policies, such evolution breaches the atomicity requirement in their collaboration based on the renewed supplier process, because the effects of the payment could not be compensated (that is, the money could not be refunded) if the retailer aborts the collaboration after confirming the orders.

Although organizations may choose to re-publish their updated atomicity-equivalent public views and to repeat the atomicity analysis after process evolution using our previous approaches, this strategy has limitations. For instance, the supplier may need to evolve its processes timely to respond to a critical market change. If these processes are enacting some collaboration, the adoption of the strategy would require either aborting the ongoing collaboration or waiting until the current collaboration finishes. However, the abortion of an ongoing collaboration may bring value loss to the participating organizations. On the other hand, delaying process evolution until the completion of collaboration would violate organizations' needs to adapt timely to environment changes.

As such, process evolution during collaboration is inevitable. We refer this to as *dynamic process evolution*. The problem of preserving atomicity in dynamic process evolution is challenging because this may involve multiple collaborating processes across different organizations. However, if these processes evolve simultaneously, it is intuitively hard to analyze the atomicity of their collaboration during the evolution. Furthermore, a process may have multiple collaborations (with possibly different sets of collaborators), it is challenging to find a way to retain all these collaborations to be atomic if adapting this process dynamically. These issues are not addressed by our previous approaches [10][11].

In this paper, we explore the process evolution scenarios in cross-organizational collaboration setting and propose some guidelines for organizations to conduct such evolutions, while addressing their concerns on whether the collaborations are atomic.

The main contributions of this paper are twofold: First, we propose a scenario-based approach to capture the atomicity requirement in cross-organizational setting for process evolution. Next, guidelines for process evolution in each scenario are identified.

The rest of this paper is organized as follows: Section 2 introduces the fundamentals about process and atomicity-equivalent public views. Section 3 presents the evolution scenarios in cross-organizational collaboration, and their evolution guidelines. Section 4 compares our work with related works followed by some discussions in Section 5, and finally is the conclusion.

2. Background

In this section, we review the definition of process, atomicity-equivalent public view, and atomicity sphere. Unless stated otherwise, they are taken from [10].

Definition 1 [Process specification]: A process p is defined by a quadruple (p, P, A, F), where P is a set of states, A is a set of actions, $F \subseteq (P \times A \times P)$ is a ternary transition relation. We also call it a *process specification*.

For simplicity, we denote the initial state of a process p as p, and p^* as all the states reachable from p. Without loss of generality, we represent the termination state of a process as the constant 0. We assume that each process should terminate [9].

Processes are subject to several basic operators, that is, "$\cdot$", "$\|$", and "+", which represent prefix operator, parallel composition operator and choice composition operator, respectively [10]. In this paper, an *action* refers to the commitment of a task, which occurs instantaneously. To ease the presentation, the terms "*action*" and "*task*" are used interchangeably. In a process, *port actions* are designed to communicate with other processes, while *non-port actions* do not participate in communications. We assume in this paper that processes communicate with each other synchronously in the application level (although the underlying communication protocols may be asynchronous). For example, the customer process may send an order to the supplier process using asynchronous protocols (e.g., email), but it does not conduct next action until it receives the acknowledgement from the supplier. Therefore, the port action "*place order*" from the customer and the

port action "*receive order*" from the supplier communicate synchronously in the application level.

Definition 2 [Process instance]: A complete trace of a process is a sequence, possibly empty, of actions executed from its initial state to its termination state. For a process *p*, *trace*(*p*) denotes the set of all the complete traces of process *p*. For a trace *t*, *i* is an integer larger than 0, *t*[*i*] denotes the action in the i^{th} position of *t*. We also call it a *process instance*.

To achieve the atomicity property in a process, Hagen and Alonso [6] propose an approach based on the notion of *atomicity spheres*, which allow designers to express atomicity aspects in a process and exclude the irrecoverable situations at design time. A process satisfying atomicity sphere implies that this process is atomic [6]. To support the checking of whether a process satisfies atomicity sphere, every action $a \in A$ has two properties: *compensability* and *retriability*. A compensable action means that this task can be undone one way or the other after committing. An action is *non-compensable*, if the overhead or cost of compensating the committed task is unacceptable [6]. A retriable action can repeat itself internally without cumulative effects if the latest internal trial fails, and finally succeed after a finite number of trials; otherwise, the action is non-retriable. To ease the presentation, we use two predicates to describe these two properties: Predicate C(*a*) is *true* if and only if the action *a* is compensable; predicate R(*a*) is *true* if and only if the action *a* is retriable. Informally, a process satisfies atomicity sphere if and only if (i) it does not contain any state that violates atomicity sphere (denoted by ϕ), and (ii) no non-retriable task is executed after a non-compensable task in any of its complete execution traces [10]. This is formalized as follows:

Definition 3 [Atomicity sphere criterion]: φ is an atomicity sphere predicate of a process, $\varphi(p) = true$ if and only if the following condition holds:

$$\phi \notin p^* \wedge \neg\exists t \in trace(p)\ [(a=t[i],\ b=t[j],\ j>i>0) \wedge (C(a)=false,\ R(b)=false)]$$

To check the atomicity sphere criterion of a collaboration without disclosing their process details, atomicity-equivalent public views are derived from the original participating processes. A public view is *atomicity equivalent* if it satisfies the following property [10]:

Property (Atomicity-equivalence): If *pv* is an atomicity-equivalent public view of the process *p*, then

$\varphi(pv) = \varphi(p)$, and

$\varphi(pv \parallel q) = \varphi(p \parallel q)$ for any process *q*.

With this property, organizations could compose their atomicity-equivalent public views into a global process, and use this global process to check the atomicity sphere of their collaboration. The atomicity sphere property of this global process is equivalent to that of their collaboration [10].

3. Process Evolution

3.1. Evolution Scenarios

In cross organizational collaborations, organizations may have different evolution requirements in different scenarios. In this section, we explore and classify them in three selected basic scenarios.

Scenario 1*: An organization changes its process specification which has been involved in some contract.*

A *contract* is an agreed collaboration, which mandates all the involved organizations to promise their processes to be atomicity-equivalent to their public views during the collaboration.

In this scenario, an organization has published an atomicity-equivalent public view of its process, and used its public view to reach agreement of collaboration with other organizations. The organization already has a contract in effect. It needs to carry out processes according to their atomicity-equivalent views.

Therefore, the evolution requirement in this scenario is that the collaboration based on the evolved process specification should keep the original contract intact.

Scenario 2*: An organization changes dynamically a process instance which is taking part in a contract.*

In this scenario, during their collaboration, one organization may want to modify its process instance to deal with certain changing business requirement. For instance, the supplier may temporarily use another shipping workflow to handle this particular shipment when the shipper just receives a large and urgent order that the benefits of meeting the order (say, using the standard process) out-rate the penalties of other business requirements. Continuing the execution of the ongoing process instances should be more desirable than aborting them.

Similarly, the atomicity requirement in this scenario is that, the dynamic change of a process instance in one organization should not violate the atomicity of its contract. We note that the modification is limited to the instance, and the process specification remains unchanged.

Scenario 3*: An organization modifies a process specification, and switches over the ongoing process instances of the original specification to the new ones.*

In this scenario, the ongoing process instances of the original process specification are required to continue the execution based on the new program specification. For example, during the supply chain collaboration, the customers and the suppliers may agree to change their

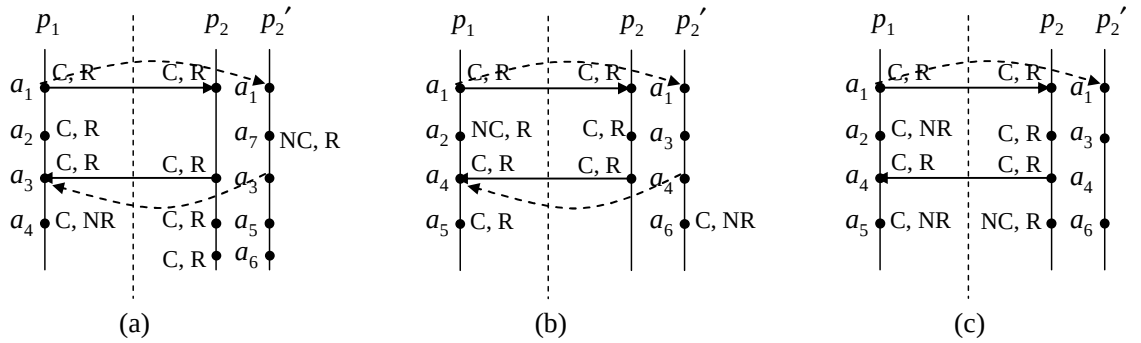

Figure 2. (a) Inserting a non-compensable task, (b) inserting a non-retriable task, (c) removing a port action.

contract due to some reason (e.g., detecting deadlock in the specification), and continue their collaboration based on a renewed contract. The requirement in this scenario is that the current process instances of original process specification should not violate its atomicity sphere property of the new contracts after migrating to the new specification.

Note that above scenarios describe basic service evolution requirements in cross organizational environment. One may combine these scenarios in one way or another to describe more complex evolution requirements. For example, Scenarios 1 and 3 could be used together to describe a scenario where an organization changes its process specification and migrates its ongoing instances to the new specification without scarifying the original contracts.

In the following sections, we will introduce some useful guidelines for dynamic process evolution in each scenario.

3.2. Evolution Guidelines for Scenario 1

In this scenario, organizations wish to evolve their process specifications that still meet the original contracts using the evolved process specification.

Let us consider two primitive operations for dynamic process evolution of process specification: *inserting an action*, and *removing an action*. We will show later how to utilize these two basic operations to semantically simulate complex specification evolutions.

Inserting an action. According to Definition 3, inserting a compensable and retriable action in a process does not affect the atomicity. We need only consider the cases of inserting non-compensable and those of inserting non-retriable actions.

We observe that an arbitrary insertion of a non-compensable or non-retriable action may not preserve the atomicity requirement. For example, as depicted in Figure 2, where *C, R, NC, NR* represents compensable, retriable, non-compensable and non-retriable, respectively, and p_1 communicates with p_2 using port actions (e.g., a_1, a_3 or a_4). According to Definition 3, the collaboration between process p_1 and p_2 in either Figure 2 (a) or (b) should meet the atomicity sphere criterion because there is only one non-retriable or non-compensable action, respectively, in each collaboration.

Suppose p_2' is the evolved process specification of p_2 in either case. Observe that the newly inserted non-compensable task a_7 of p_2' in Figure 1(a) will turn the process instance (representing the collaboration between p_1 and p_2') to have a non-compensable action (a_7) to be executed before a non-retriable action (a_4). It fails to meet the atomicity sphere criterion. Likewise, the collaboration between p_1 and p_2' in Figure 1(b) also does not meet the criterion.

We observe that the inserted action makes the evolved process specification inconsistent to the atomicity-equivalent public views in the original contract for the collaboration between p_1 and p_2.

To further present our guidelines, we firstly introduce the following notation.

Definition 4: Given two actions *a* and *b* in a trace of process *p*, we say *a CE-before b* if and only if at least one of the two conditions is satisfied:

C1: *a* is executed before *b* in this trace.

C2: *a* is executed after *b*, but any action between *b* and *a* in this trace (including *a, b*) is a non-port action.

The intuitive meaning of *a CE-before b* is that if an action from another process could be executed after *b* in their collaboration, then it could also be executed after *a*. For example, as depicted in Figure 2 (a), we have a_6 *CE-before* a_5 (of course, a_5 also *CE-before* a_6 in this case), because any action (e.g., a_4) that could be executed after a_5, it should also be executed after a_6. To ease the presentation, we call *b CE-after a* if *a CE-before b*. Note that if a process *p* satisfying atomicity sphere evolves to *p'*, *p'* should also satisfy atomicity sphere. This guideline can be checked by Definition 3.

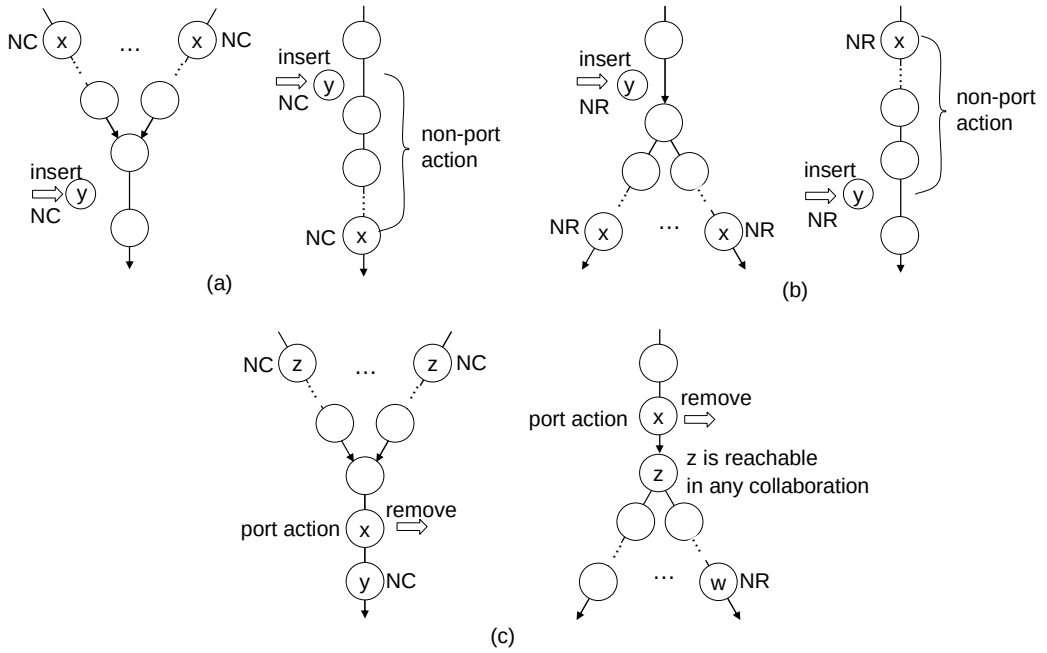

Figure 3. (a) Guideline for inserting a non-compensable task, (b) Guideline for inserting a non-retriable task, (c) Guideline for removing a port action.

In the rest of this paper, unless stated otherwise, we assume the evolved process satisfies atomicity sphere.

Therefore, we have the following guidelines, as depicted in Figure 3 (a) and (b):

Guideline 1*: a non-compensable action, y, can be inserted into a process, if the following condition is satisfied: For every trace of the evolved process that contains the inserted action y, there exists a non-compensable action x such that x CE-before y.*

Guideline 2*: a non-retriable action, y, could be into a process, if the following condition is satisfied: For every trace of the renewed process that contains the inserted action y, there exist a non-retriable action, x, such that x CE-after y.*

The intuitive meaning of the two guidelines is given as follows: Having non-compensable actions in the atomicity-equivalent public view means that its collaborators should not abort the collaboration if any of these non-compensable actions has been committed. Therefore, inserting one new non-compensable action *CE-before* any of the declared ones in the public view of a collaborator means that this organization breaks its promise (i.e., it pushes early the point after which aborting the collaboration by its collaborators is forbidden), and thus violates the contract. Similarly, inserting an additional non-retriable action after all the non-retriable actions in a trace (in the sense of *CE-after*) will postpone the abortion point of the collaboration, which however may not be agreed by its collaborators.

With these guidelines, inserting a new action into a process will not violate the atomicity requirement in any of their collaborations. This is guaranteed by the following theorem.

Theorem 1: Let p be a process, and pv be its atomicity-equivalent public view. Suppose p' is got by inserting a new action into p satisfying the Guideline 1 and 2, then for any process q, if $\varphi(pv \parallel q) = true$, then $\varphi(p' \parallel q) = true$.

Proof: According to Property, if $\varphi(pv \parallel q) = true$, then $\varphi(p \parallel q) = true$. Suppose $\varphi(p' \parallel q) = false$, and p' is got by inserting a non-compensable action, y, into p, then there exists a trace of the process $p' \parallel q$ such that a non-retriable action from q, z, is executed after the non-compensable action y in this trace. Since inserting y satisfying guideline 1, there exists another non-compensable action x in the process p, such that x CE-before y. According to Definition 4, z could be executed after x in some trace of the process $p' \parallel q$, so as in some trace of the process $p \parallel q$. This contradicts that $\varphi(p \parallel q) = true$. Similarly, if p' is got by inserting a non-retriable action, y, into p, then there exists a trace of the process $p' \parallel q$ such that a non-compensable action, z, is executed before the non-retriable action y in this trace. Since inserting y satisfying guideline 2, for every trace of p' that contains y, there exists another non-retriable action x in the process p, such that x CE-after y. Meanwhile, since every process could terminate, at least one such non-retriable action x is reachable in

the collaboration $p' \parallel q$. According to Definition 4, x could be executed after z in some trace of the process $p' \parallel q$, so as in some trace of the process $p \parallel q$. This contradicts that $\varphi(p \parallel q) = true$. Hence, the conclusion follows. ∎

Removing an action. According to the atomicity sphere criterion, removing a non-port action does not affect the atomicity requirement of a collaboration. However, this is not the case for removing a port action. For example, as depicted in Figure 2 (c), according to Definition 3, one can observe that the collaboration of p_1 and p_2 in (c) satisfies atomicity. However, removing the port action a_4 from p_2 makes the non-compensable action a_6 to be executable before the execution of the non-retriable action a_2 in the collaboration of p_1 and p_2', and thus may lead to a violation of atomicity. To avoid such situations, we have the following guideline for removing a port action, as depicted in Figure 3 (c):

Guideline 3*: a port action, x, is allowed to remove from a process, if for every trace of this process that contains the port action, the following conditions are satisfied:*

C1: if there exists a non-compensable action, y, executed after x in this trace, then there must exist another non-compensable action, z, such that z is executed before x in this trace.

C2: if there exists a non-retriable action, w, executed after x in this trace, then x should not cause deadlock in any their collaboration.

Condition C1 precludes the situation that a non-compensable action, following the removed port action, executes before some non-retriable action in another collaborating process due to the loss of synchronization control between the two processes after removing this port action. Condition C2 reveals the situation that if the removed port action results in a deadlock in the original collaboration, removing such an action will make some unreachable and non-retriable action in the original collaboration able to execute and may thus violate the atomicity requirement. The correctness of this guideline is proven by the following theorem:

Theorem 2: Let p be a process, and pv be its atomicity-equivalent public view. Suppose p' is constructed from p by removing a port action, which satisfies the Guideline 3, then for any process q, if $\varphi(pv \parallel q) = true$, then $\varphi(p' \parallel q) = true$.

Proof: According to Property, if $\varphi(pv \parallel q) = true$, then $\varphi(p \parallel q) = true$. Suppose $\varphi(p' \parallel q) = false$, then there exists a trace, t, of $p' \parallel q$ such that a non-retriable action, x, is executed after a non-compensable action, y, in this trace. Since $\varphi(p \parallel q) = true$, x could not be executed after y in any trace of $p \parallel q$. Therefore, trace t of $p' \parallel q$ is got by removing a port action in p such that the synchronization between x and y is missing. On the one hand, if x is from q and y is from p, then y is executed after the removed port action. According to the condition C1 in Guideline 3, there exists another non-compensable action, z, executed before the removed port action. Hence, x is able to execute after the non-compensable action z in the process $p \parallel q$, which contradicts the fact that $\varphi(p \parallel q) = true$. On the other hand, if x is from p and y is from q, then x is executed after the removed port action in p. According to the condition C2 in Guideline 3, the removed port action does not cause deadlock in the process $p \parallel q$. Therefore, x is also able to execute after y in $p \parallel q$, which also contradicts the fact that $\varphi(p \parallel q) = true$. Hence, the conclusion follows. ∎

Based on the guidelines of these two operations, we can check the feasibility of other dynamic process evolutions by simulating them with these two operations.

Relocating an action from one position to another is regarded as inserting a new action followed by removing the existing one. For example, if we want to move a port action forward by one step (that is, exchanging the port action a with the action b right following it), we could regard it as inserting an action b before a and removing the original b. As a result, we can use the guidelines for inserting and removing actions to check its feasibility.

Changing the atomicity properties of an action has four cases, that is, changing an action from C (compensable) to NC (non-compensable), R (retriable) to NR (non-retriable), NC to C, or NR to R. According to Definition 4, only the first two cases may violate the atomicity requirement in their collaborations. We can simulate these two cases by inserting an action. For example, changing an action from C to NC could be regarded as inserting a new non-compensable and retriable non-port action after this action. Changing an action from R to NR could be simulated by inserting a non-retriable non-port action after this action, which shares the same compensability property.

3.3. Evolution Guidelines for Scenario 2

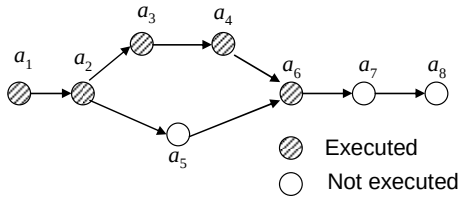

Figure 4. Process instance evolution using execution history

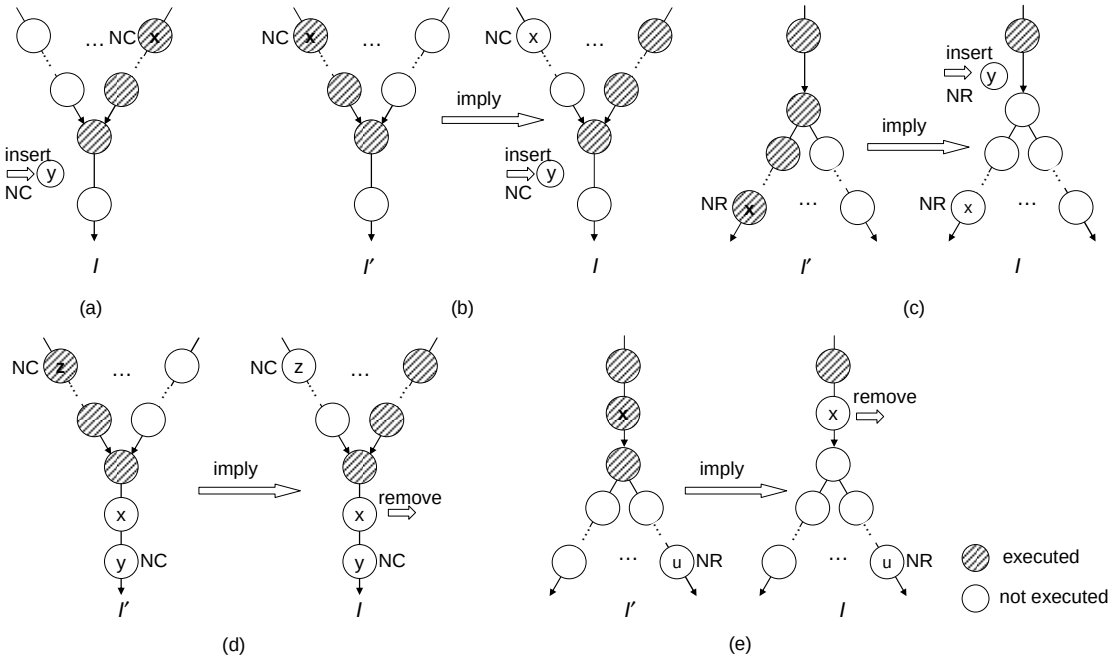

Figure 5. Evolution guidelines for instances. (a), (b) inserting a non-compensable task, (c) inserting a non-retriable task, (d), (e) removing a port action.

In this scenario, business organization changes its process instance dynamically. The guidelines proposed in Scenario 1 could be used to guide the modification of process instances. However, the difference is that the modification of a process instance affects only the ongoing collaboration, not others. So conditions in these guidelines could be relaxed by some special optimizations.

As a process instance belongs to one particular collaboration, we can use the passed execution information of this collaboration to improve the analysis accuracy. For example, as depicted in Figure 4, an organization wants to insert a non-compensable action following the action a_6 after the process instance has just committed a_6. If the instance has committed a non-compensable action (e.g., suppose a_3 is non-compensable), then the evolution is feasible. However, if this process instance has not committed any non-compensable action yet, but the passed execution information shows that some non-compensable action (e.g., a_5) in an alternative branch can be executed in this collaboration (which means that a_5 could be reachable in the collaboration), then this dynamic process evolution is also feasible. This is because the collaboration after a_6 is not allowed to abort based on the execution history. Therefore, we can revise Guideline 1 as follows to support Scenario 2, as depicted in Figure 5 (a) (b):

Guideline 1′: *a non-compensable action, x, can be inserted into a process instance, if any of the following conditions is satisfied:*

C1: The instance has committed at least one non-compensable action, or

C2: There exist another instance of the same collaboration which executes a non-compensable action, y, before the point inserting the new action x.

Likewise, given the instance execution information, the guidelines for inserting a non-retriable action and skipping a port action from current instance can be revised in a similar way, as depicted in Figure 5 (c), (d), (e) (note that Figure 5 shows only the parts different from Scenario 1):

Guideline 2′: *a non-retriable action, x, could be into a process instance, if any of the following condition is satisfied:*

C1: For every trace of the renewed process that contains the inserted action, there exists a non-retriable action, y, such that y CE-after x in this trace.

C2: There exists another instance of the same collaboration which executes a non-retriable action, z, after the point inserting the new action x.

Guideline 3′: *a port action, x, is allowed to skip from an instance, if for every potential trace of the current instance that contains the port action, the following conditions are satisfied:*

C1: if there exists a non-compensable action, y, executed after x in this trace, then there exist another

non-compensable action, z, such that z is executed before x in this trace, or there exist another instance of the same collaboration which executes a non-compensable action, w, before x.

C2: if there exists a non-retriable action, u, executed after x in this trace, then there exists another instance of the same collaboration which executes the skipped port action x.

The correctness of these revised guidelines is guaranteed by the following theorem (the proof is similar to that of Theorem 1 and 2, and thus omitted):

Theorem 3: Let *p* be a process instance, and *pv* be the atomicity-equivalent public view of *p*'s specification. Suppose the instance *p′* is constructed from *p* by inserting an action or removing a port action, satisfying the Guidelines 1′, 2′ and 3′, then for any process *q*, if $\varphi(pv \parallel q) = true$, then $\varphi(p' \parallel q) = true$.

3.4. Evolution Guidelines for Scenario 3

In this scenario, organizations need to migrate the existing process instances to the evolved process specification. However, such migration may violate atomicity of these instances. For example, if a process instance has committed a non-compensable action, executing a non-retriable action in the new process specification after migration will violate the atomicity sphere of this instance.

Therefore, we have the following guideline for dynamic process evolution if all the instances of the original specification are required to migrate to the evolved one without violating their atomicity sphere, as depicted in Figure 6:

Guideline 4*: Let p be an original process specification and p´ be an intermediate specification during the evolution. An action b is allowed to insert into p´, if b is retriable, or for any trace t´ of p´ that contains b, the corresponding trace t of p does not contain any non-compensable action before the point b is inserted.*

This guideline prevents any process instance of the

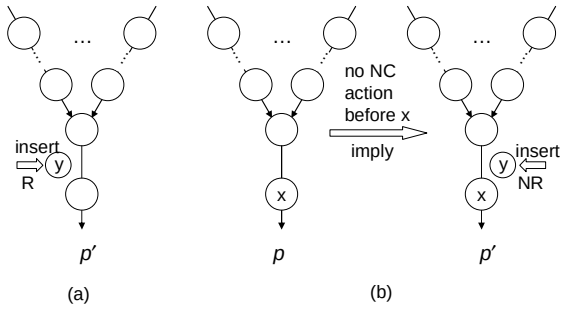

Figure 6. Evolution guidelines for migration.

original specification to reach a non-retriable action after migration if it has committed a non-compensable action. According to Definition 3, removing an action from the original specification will keep the atomicity of its process instance intact after migration, therefore, no guidelines are needed for this case. The correctness of this guideline is guaranteed by the following theorem:

Theorem 4: Let *p* be a process specification, and *p′* be its evolved specification whose evolution satisfies the Guideline 4, if *p* and *p′* both satisfy atomicity sphere, then for any instance *I* of *p*, the migration of *I* to *p′* does not violate the atomicity sphere of *I*.

Proof: Since removing an action from the specification does not violate the atomicity sphere of *I*, we consider only inserting a new non-retriable action or non-compensable action. Let us consider inserting a new non-retriable action first. If an instance *I* has passed the point inserting the new action, then inserting this action does not affect the atomicity sphere of *I*. If the instance *I* has not passed the point, inserting this action also does not affect its atomicity sphere because there does not exist any non-compensable action executed before this inserted action according to Guideline 4. Similarly, if the inserted action is non-compensable, it does not affect the atomicity sphere of any instance of original specification. This is because if an instance executes this inserted non-compensable action after migrating to the new specification, it will no longer execute any non-retriable action; otherwise, the new specification will violate the atomicity sphere, which contradicts the assumption. Therefore, the conclusion follows.

3.4. Application

In this section, we use the example in Figure 1 to illustrate how to use the guidelines proposed in this paper to keep atomicity consistency during the process evolution.

In this example, the supplier wants to evolve its business process by changing the property of the action "*receive payment*" from C to NC due to its new policies in which the payment is no longer refundable. Since the supplier process has been participating in some collaboration with the retailer, guidelines in Scenario 1 are used to check the feasibility of the evolution.

According to our proposed approach, changing the property of an action from C to NC could be regarded as inserting a non-compensable and retriable non-port action after this action, as depicted in Figure 7, where all the actions are originally compensable and retriable. According to Guideline 1, the supplier process is not

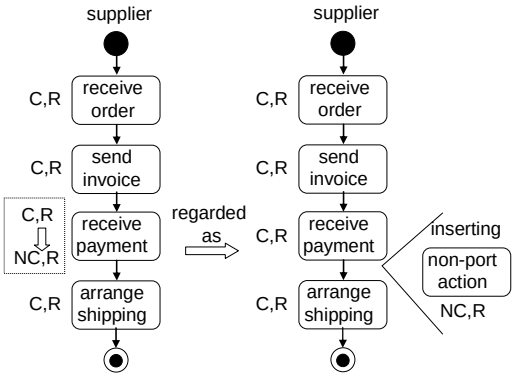

Figure 7. Changing task property.

allowed to insert a non-compensable action after the action "*receive payment*", because there exist no non-compensable action CE-before this inserted action in the supplier process. Therefore, the supplier process is forbidden from changing the property of action "*receive payment*" from C to NC; otherwise, the evolution might lead to violation of atomicity sphere and value loss in some of its collaboration.

4. Related Work

In this section, we review major recent works on dynamic process evolution and make a comparison with our work.

Casati et al. [3] address the process evolution from both a static and a dynamic points of view, that is, modifying the process description and managing ongoing process instances of a process specification whose description has been modified. A set of primitives are proposed to modify a process description or process instances. To guarantee the correctness of the process evolution, two consistency criteria are proposed, that is, structural consistency and behavioral consistency. Structural consistency ensures the application of any sequence of modifications to a legal schema will result in another legal schema. Behavioral consistency guarantees that the application of any set of primitives to a process instance will result in another process instance with a new schema and a legal state without any run-time errors.

Ellis et al. [4] also concern about dynamic changes within workflow systems, but they use a predictive approach based on Petri-net. Instance changes on a schema are carried out by substituting the marked sub-net of this schema (old change region) by another marked sub-net (new change region), which reflects the modifications set out by the change. The consistency criterion requires the instance of original schema should be able to resume in the new schema. This includes two situations: first, all firing sequences leading from the original process instance to the terminal marking on the new schema are either producible by this process instance on the original schema as well, or second, the firing sequence of the process instance of the existing process specification can be continued on the new schema after the same firing sequence.

Aalst and Basten [1][2] address the issue of process evolution using a different approach based on inheritance of workflow. The evolution abstracts or extends the original one by proposed rules, the correctness of which is guaranteed by four kinds of behavior consistency criteria. For example, to make the ongoing process instances executable in a new schema, the new schema should be protocol consistent with the original one, in the sense that whenever the original schema could be executed, the new one could too. This approach does not need any execution information of process instances, but it supports only some kinds of modification.

In addition, some related works address process evolution with different behavior consistency criteria [8]. The difference between the work introduced above and our work is that we use atomicity as a correctness criterion, not the behavioral consistency criterion. Note that these two criteria are complementary. Our work complements the work introduced above to cover the atomicity concern during dynamic process evolution in cross organizational environment. Organizations could use both criteria to guide the evolution of their processes satisfying not only behavioral but also atomicity requirements.

5. Discussions

5.1. Implementation Issues

To check the guidelines proposed in this paper, we need to traverse only the local processes in one organization. Since the guidelines are to compare the execution orders between actions (including inserted action and removed action), the checking of each guideline needs traverse the process at most once. Let us take Guideline 1 as an example. If we insert a non-compensable action y into process p and get p', we need to check whether in every trace of p' that contains y, there exists another non-compensable action x from p such that x CE-before y. This can be done by traversing the process p' from the termination state to the initial state and marking suitable x. If a suitable x is found, stop the traverse of the current trace and continue the search from another trace. If there exists a trace that contains y but no suitable x, then stop the traverse and report that the guideline is not satisfied.

This procedure takes steps no more than the sum number of states and transitions in the process. Thus, checking the guidelines is feasible.

5.2. Limitations

In current work, we propose a collection of guidelines for two basic operations: action insertion and action removal. These two operations can be composed for more complex evolutions. If the guidelines of the basic operators are satisfied, then this complex evolution is feasible. The question is, if a complex evolution does not violate atomicity, then must these guidelines be followed? The answer to this question seems to find a minimal and complete set of evolution primitive operations such that their guidelines are sound and complete. We will investigate this issue in future.

Another limitation is that the current proposal does not support conditional branching. If the branch is conditional, then the optimization in the Scenario 2 is not available. One way to solve this problem is to add pre-conditions for the evolution. If the execution history satisfies the pre-conditions, then their execution information could be used to optimize the conditions in the guidelines.

The current work concerns mainly business processes in B2B collaboration, but it could be applied to software processes when atomicity is required. For example, software developers and testers from different organizations may collaborate in an open source project based on some pre-defined processes. When evolving some of such software processes, these guidelines could be used to keep the atomicity requirement.

6. Conclusion

Dynamic process evolution in cross organizational collaboration may violate atomicity as stated in their contracts, which are agreed collaborations among organizations. This paper explores three basic evolution scenarios and proposes guidelines for organizations to guide the dynamic process evolution in these scenarios without violating the atomicity requirement.

The present work presents a preliminary study of the dynamic process evolution guidelines. In the future, we will develop algorithms and tools to check these guidelines. In addition, evolution guidelines for more complex process structure, such as exception handling evolution, will be studied.

References

[1] WMP van der Aalst and T. Basten, "Inheritance of workflows: an approach to tackling problems related to change," *Theoretical Computer Science*, vol.270, no.1-2, 6 Jan. 2002, pp.125–203.

[2] T. Basten and WMP. van der Aalst, "Inheritance of behavior," *Journal of Logic & Algebraic Programming*, vol.47, no.2, Mar-Apr 2001, pp.47–145.

[3] F. Casati, S. Ceri, B. Pernici and G. Pozzi, "Workflow evolution," *Data & Knowledge Engineering*, vol.24, no.3, Jan. 1998, pp.211–238.

[4] C. Ellis, K. Keddara and G. Rozenberg, "Dynamic change within workflow systems," *Conference on Organizational Computing Systems*. ACM. 1995, pp.10–21.

[5] P. Greenfield, D. Kuo, S. Nepal, and A. Fekete, "Consistency for Web Services Applications", In *Proceedings of the 31st International Conference on Very Large Data Bases*, Trondheim, Norway, Aug-Sept. 2005, pp.1199–1203.

[6] C. Hagen and G. Alonso, "Exception handling in workflow management systems," *IEEE Transactions on Software Engineering*, vol.26, no.10, Oct. 2000, pp.943–958.

[7] D.R. Liu, and M.X. Shen, "Workflow modeling for virtual processes: an order-preserving process-view approach", *Information Systems*, vol.28, no.6, Sept. 2003, pp.505–532.

[8] S. Rinderle, M. Reichert, and P. Dadam, "Correctness criteria for dynamic changes in workflow systems-a survey," *Data & Knowledge Engineering*, vol.50, no.1, July 2004, pp. 9–34.

[9] H. Schuldt, G. Alonso, C. Beeri, and H.J. Schek, "Atomicity and isolation for transactional processes," *ACM Transactions on Database Systems*, vol.27, no.1, Mar. 2002, pp. 63–116.

[10] C.Y. Ye, S.C. Cheung, and W.K. Chan, "Publishing and composition of atomicity-equivalent services for B2B collaboration", In *Proceedings of the 28th International Conference on Software Engineering*, May. 2006, Shanghai, China, pp.351–360.

[11] C.Y. Ye, S.C. Cheung, W.K. Chan, and C. Xu, "Local analysis of atomicity sphere for B2B collaboration", In *Proceedings of the 14th ACM SIGSOFT Symposium on Foundation of Software Engineering*, Nov. 2006, Portland, Oregon, USA, pp.186–196.