Atomicity Analysis of Service Composition across Organizations

Chunyang Ye, S.C. Cheung, *Senior Member*, *IEEE*, W.K. Chan, *Member*, *IEEE*, and Chang Xu

Abstract—Atomicity is a highly desirable property for achieving application consistency in service compositions. To achieve atomicity, a service composition should satisfy the atomicity sphere, a structural criterion for the backend processes of involved services. Existing analysis techniques for the atomicity sphere generally assume complete knowledge of all involved backend processes. Such an assumption is invalid when some service providers do not release all details of their backend processes to service consumers outside the organizations. To address this problem, we propose a process algebraic framework to publish atomicity-equivalent public views from the backend processes. These public views extract relevant task properties and reveal only partial process details that service providers need to expose. Our framework enables the analysis of the atomicity sphere for service compositions using these public views instead of their backend processes. This allows service consumers to choose suitable services such that their composition satisfies the atomicity sphere without disclosing the details of their backend processes. Based on the theoretical result, we present algorithms to construct atomicity-equivalent public views and to analyze the atomicity sphere for a service composition. Two case studies from the supply chain and insurance domains are given to evaluate our proposal and demonstrate the applicability of our approach.

Index Terms—Atomicity, privacy protection, process algebra, theory and models, Web services.

1 Introduction

Service orientation is an emerging software engineering paradigm for developing distributed Internet applications. In this paradigm, service providers from different business organizations develop their individual Web services and publish them in a public service registry (e.g., UDDI [61]). By looking up the service registries, service consumers can find suitable services and compose them to realize their business goals. Usually, each service is driven by a backend process, which describes its behavior. To protect privacy or business interests, service providers may hide certain details of their backend processes and expose only the restricted public views of these processes to the service consumers (such as the WSDL [69] interface of the service). As a result, a service composition is usually conducted based on restricted public views [1], [2], [11], [12], [22], [23], [24], [25], [49], [59].

Since Web services are often long running, loosely coupled, and enacted individually by different service providers, a service composition may meet with some undesirable situations (e.g., deadlock [35], [54], incompatibility [35], [37], [38], and the like). Thus, when a service consumer wants to compose some services from service providers, it may wish to analyze whether this service composition is feasible (e.g., exhibiting a deadlock-free property or similar). Therefore, a requirement analysis of the involved services is necessary. For example, recent studies on requirement analysis for service compositions include web service compatibility analysis [35], [37], [38], deadlock-free analysis [35], [54], synchronizability analysis [39], and QoS-awareness [66], [75].

One important requirement for a service composition is to check whether this service composition ensures application consistency [42], [55]. For example, in a typical supply chain application, the customer buys items from the suppliers by composing their sales services. In such an application, the scenario "*the customer has paid the bill whereas the supplier fails to deliver the items*" is inconsistent and undesirable. To avoid such inconsistent scenarios, transactional support for the service composition is attractive because of its all-or-nothing semantics. However, service consumers and service providers do not usually share data, location, or administration that collectively make fine-grained locking controls and full trustworthiness unachievable [27]. In addition, services are usually long running and heterogeneous. This makes it difficult to enforce conventional transactions with the property of atomicity, consistency, isolation, and durability (ACID) [41] in a service composition due to performance concern and other reasons (e.g., privacy). Instead, exception handling, a weak consistency approach, is often adopted to resolve such application inconsistency in a service composition [42], [47].

Exception handling is, informally, a flexible way to handle abnormal scenarios [18], [19], [20]. An execution of a service could be regarded as the execution of a series of tasks. For example, a typical procurement service executes task "*payment*" followed by task "*deliver items*." When such an execution fails to complete a task (e.g., "*deliver items*"),

- *C. Ye, S.C. Cheung, and C. Xu are with the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong, China. E-mail: {cyye, scc, changxu}@cse.ust.hk.*
- *W.K. Chan is with the Department of Computer Science, City University of Hong Kong, Tat Chee Avenue, Kowloon, Hong Kong, China. E-mail: wkchan@cs.cityu.edu.hk.*

Manuscript received 4 June 2007; revised 23 Aug. 2008; accepted 16 Sept. 2008; published online 24 Sept. 2008.
Recommended for acceptance by W. Emmerich.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-2007-06-0179.
Digital Object Identifier no. 10.1109/TSE.2008.86.

0098-5589/09/$25.00 © 2009 IEEE Published by the IEEE Computer Society

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 3, 2009 at 09:09 from IEEE Xplore. Restrictions apply.

the failure may cause an application inconsistency (i.e., a customer paid the money but did not receive any goods). Then, an exception will be thrown to activate an exception handler to deal with the abnormal scenario and resolve the inconsistency. There exists two basic ways to deal with the scenario. The exception handler may abort the execution of this service and compensate all the executed tasks (e.g., refunding the money for task "*payment*"). Alternatively, the exception handler may execute some other tasks to replace the failed one and continue the execution of this service (e.g., choosing another way to deliver the items). These two approaches are usually referred to as *backward error recovery* and *forward error recovery*, respectively [48]. The exception handling approach usually uses these two approaches together. If a failed task cannot be retried or find any alternative task, then the exception handler aborts the execution of this service and compensates all the executed tasks. However, if this service has already executed some task that cannot be compensated, then aborting the execution will fail to resolve the application inconsistency.

It is thus attractive if a service is *atomic.* This means that we could either complete the execution of this service successfully, or abort its execution but be able to compensate the effects of all executed tasks as if none of them had been executed.[1] A service exhibiting such an atomicity property could always resolve such application inconsistency by backward or forward error recovery.

To achieve such an atomicity property, Hagen and Alonso [44] proposed an approach based on the notion of *atomicity sphere.* This approach allows designers to express the atomicity concerns of a service and exclude at design time the irrecoverable situation of the service (e.g., the execution of a service is aborted, but some executed tasks cannot be compensated). A service satisfying the atomicity sphere implies that this service is atomic [44].

To check whether a service satisfies the atomicity sphere, each task is assigned two orthogonal attributes, that is, *compensability* and *retriability* [58]. A task is compensable if its execution effect is reversible. A retriable task will always eventually succeed by retrying itself or executing an alternative (e.g., another task in the exception handler) in the presence of failures. We mark a task as noncompensable if we are unable to compensate its effect, or the overhead or cost of compensating the effect is unacceptable. For example, if the payment is nonrefundable due to business policies, then the task "*payment*" is a noncompensable task. In a service, we can always compensate for the effects of any executed tasks[2] before committing any noncompensable tasks [44]. However, committing a noncompensable task of a service means the service provider should complete this service successfully; otherwise, aborting its execution would result in this task not being compensated (e.g., the money paid cannot be refunded). Thus, informally speaking, a service satisfies the atomicity sphere if and only if its execution can always complete successfully after committing any noncompensable tasks [44].

Although each service provider may design its own services to satisfy the atomicity sphere individually, a service composition (which is also a service) of these services may still violate the atomicity sphere. This is because an exception of one involved service may abort the execution of this service composition, and therefore all the involved services are required to compensate the effects of their executed tasks in this service composition. Full compensation is implausible when some service providers have already committed their noncompensable tasks, and thus this service composition violates the atomicity sphere.

Let us elaborate on our discussion by example. For ease of discussion, we represent a service by its backend process and refer to each backend process as a *private process.* Fig. 1a depicts a service composition scenario of three services: the retailer (e.g., WalMart), the shipper (e.g., Evergreen), and the supplier (e.g., Li & Fung). The retailer orders products from the supplier and then arranges the shipper to deliver the ordered products. From the perspective of the supplier, the task "*book order*" is noncompensable because any order is nonrefundable or only partially refundable due to its business policy. From the perspective of the shipper, the task "*schedule*" is nonretriable because sometimes the schedule may not be available. The task "*deliver*" is noncompensable because the delivery of some products might take an unacceptable amount of time and other resources (although the shipper could ship the products back to the supplier as compensation). The other tasks are both compensable and retriable in this scenario. Observe that all three services individually satisfy the atomicity sphere because they can all complete their execution successfully after committing any noncompensable tasks. However, a failure of the nonretriable "*schedule*" task would potentially abort the execution of the service composition, in which the supplier has already committed the noncompensable "*book order*" task. As mentioned earlier, canceling an order is nonrefundable or partially refundable. Therefore, aborting the execution of this service composition would lead to an atomicity sphere violation and application inconsistency. Obviously, in such a situation, aborting the execution of the service composition would render the retailer unable to get a refund, and this result is highly undesirable.

To avoid such undesirable situations, service consumers need to find services which have consistent atomicity specifications atop their composition (i.e., their composition, as a whole should satisfy the atomicity sphere). However, services may not always have a consistent atomicity specification since different service providers enact them individually. Therefore, conducting a global analysis of involved services in a service composition during the service discovery stage is helpful to identify whether their composition satisfies the atomicity sphere. For example, the three organizations in Fig. 1a could compose their services (i.e., regarding the three private processes holistically as a global process) and then check whether their composition (i.e., this global process) satisfies

1. Note that compensating the effects of executed tasks does not necessarily mean restoring the execution of a service to its original state. For example, a task draws money from an account and the compensation for this task is to deposit the same amount of money into this account. However, the money in this account after compensation may be not equal to the moment before the task is executed because some other services may also have updated this account.

2. Similarly to [58], we assume in this paper that a compensable task could always be compensated successfully.

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 3, 2009 at 09:09 from IEEE Xplore. Restrictions apply.

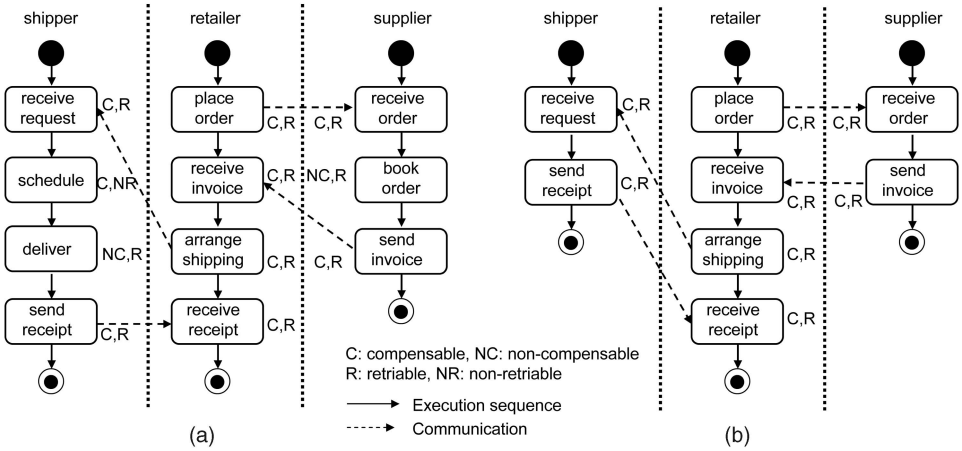

Fig. 1. (a) Service composition with conflict. (b) Their public views.

the atomicity sphere or not. If this global process satisfies the atomicity sphere, then their service composition satisfies the atomicity sphere. However, such global analysis is difficult in practice because some involved service providers may not release the details of their private processes to the service consumers and only make restricted public views available due to privacy or business reasons [59]. On the other hand, researchers have proposed many techniques on service composition based on public views [1], [2], [11], [12], [22], [23], [24], [25], [49], [59]. However, these works are unable to check the atomicity sphere in a service composition because the public views based on these techniques do not include all the atomicity-related information of tasks in private processes. For example, the public view of the supplier (as depicted in Fig. 1b) will miss the atomicity-related information of task "*book order*" (i.e., this task is noncompensable) if this task is not in the public view. Analyses missing such information will show that the composition satisfies the atomicity sphere. A service composition based on such misleading results may lead to undesirable results for service consumers or providers (e.g., the customer could not get the refund after aborting the collaboration).

The novel contribution of this paper is an adaptation of the operational semantics and deduction rules of conventional process algebraic models to construct atomicity-equivalent public views from private processes. Based on the semantics and deduction rules, we can conduct the global analysis of atomicity sphere directly from those public views instead of original private processes. This paper extends our previous work [72], which proposes a novel process algebraic model to publish the atomicity information of a private process in its public view. We call the public view an atomicity-equivalent public view. The aim is to provide service consumers an objective criterion to identify suitable service providers whose services can satisfy the atomicity requirements while not disclosing their entire private processes. This exempts service providers from the burden of having to release complete details of their private processes. Based on the previous results, we prove that we can derive the atomicity-equivalent public view of a composite service (i.e., a service composition) iteratively from the atomicity-equivalent public views of the composed services. This extends our previous work [72] to derive atomicity-equivalent public views for not only original private processes but also composite services. Further, based on the theoretical results, we present a more detailed explanation of the algorithms for deriving and composing public views and analyzing the atomicity sphere. Finally, we evaluate the applicability of our approach using two case studies.

The rest of this paper is organized as follows: Section 2 presents the process algebraic framework and corresponding algorithms. Section 3 illustrates this approach by using two case studies, and analyzes the complexity of our approach. Section 4 discusses the assumptions and limitations of our approach. This is followed by a comparison with related work in Section 5 and, finally, we present the conclusion in Section 6.

2 Methodology

2.1 Service Composition Model and Exception Handling

Fig. 2 describes a conceptual model summarizing the service composition introduced in Section 1. In this model, to protect privacy or business interests, the detailed behavior of the private process of each service is invisible outside the organization the service provider belongs to. The service providers reveal only restricted parts of their private processes in the public views. They compose their services through message exchanges between their public views. As mentioned in Section 1, to check whether services involved in a service composition have consistent atomicity specification, we propose including the atomicity-related information of tasks into public views. Then, we use these public views to analyze the atomicity sphere in a service composition instead of using their private processes. Before introducing our proposal, let us first refine the service models adopted in our proposal. Without loss of generality, we assume that the execution of any service should terminate (i.e., successfully complete or abort) [58].

We adopt the replacement exception model [74] for private processes (we will discuss other exception models

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 3, 2009 at 09:09 from IEEE Xplore. Restrictions apply.

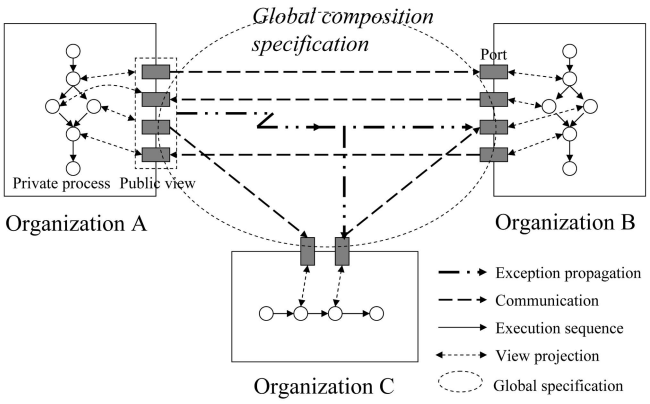

Fig. 2. Service composition model.

in Section 4). In this model, exception handlers semantically replace the failed tasks (or task units) and resume the execution of processes. If the execution of a process raises an exception without any predefined exception handler, it then invokes a default handler to stop this execution. This default handler will then try to remove the side effects of this execution by invoking the compensating tasks (if any exist) of all the executed tasks in the reverse order [44][3] (note that this is common practice in composition languages, such as BPEL [7]). In this service composition model, a process could propagate exceptions to another process by messages. For example, the bank may send an exception message to the customer if the payment is beyond the credit limit of the account of the customer. However, since service consumers and service providers are autonomous and they enact their processes individually, we assume that there is no predefined exception handler across these organizations.

Since exception handlers semantically replace the failed tasks in the replacement exception model, we therefore consider calculating the compensability and retriability properties of a task and those of its exception handlers as a whole. This can be done using the rules proposed by Hagen and Alonso [44]. According to these rules, a task and its exception handlers as a whole are compensable if and only if the task and all of its handlers are compensable. Similarly, a task and its exception handlers as a whole are retriable if and only if either the task or all of its exception handlers are retriable. For example, suppose that a is a compensable and nonretriable task associated with an exception handler b, which is noncompensable and retriable. The composite properties thus calculated are noncompensable and retriable. We then substitute the original properties of task a by the composite properties thus computed, and omit its exception handler b in the subsequent analysis. Such a treatment does not change the atomicity sphere analysis result [44]. In this way, we only need to focus on the tasks in the processes, and ignore the properties of exception handlers in our subsequent checking of the atomicity sphere.

According to the replacement exception model, if a task raises an exception or propagates an exception to other tasks in the process during the execution, the execution of this task is terminated and aborted (which means that this task may fail at some stage). As a result, we mark the retriability property of a task or an exception handler using the following rules: 1) If a task raises an exception, then we mark this task initially as nonretriable. 2) If an exception handler aborts the process or propagates the exception, then we mark this exception handler as nonretriable. Based on these marking rules and the simplification technique introduced above, we refine the atomicity sphere criterion as follows: A process in the replacement exception model satisfies the atomicity sphere if and only if no nonretriable tasks are executed after some noncompensable task in every possible execution of this process. This criterion is analogous to the one for guaranteed termination proposed in [58].[4] Note that if a process satisfies this condition, then the execution of this process should be able to complete successfully after commitment of any noncompensable tasks. This is because the execution of any task after a noncompensable task should be able to "retry" (i.e., this task could be retried or substituted by its retriable exception handler) and finally succeed in case of failure. On the other hand, if this condition is breached, then a nonretriable task occurs after a noncompensable task in some specific execution of this process. The failure of this nonretriable task will then abort this specific execution after commitment of some noncompensable task and thus violate the atomicity sphere.

To check the atomicity sphere in a service composition without disclosing the details of their private processes, we propose a novel framework in the following sections to derive atomicity-equivalent public views from private processes. By checking the atomicity sphere in the composition of these public views instead of those private processes, service consumers can identify suitable service providers such that the composition of their services satisfies the atomicity sphere. This feature is desirable as service consumers can thus learn whether the potential service composition is able to avoid nonatomic execution in advance, whereas service providers do not have to disclose their process details.

2.2 Process Definition

Note that we represent a service by its backend process. In this section, we first define the notion of *process* and its operators formally. Then, we introduce a predicate to check the atomicity sphere based on these definitions.

Definition 1. *A process space is a triple* (P, A, F)*, where* P *is a set of states,* A *is a set of actions, and* $F \subseteq P \times A \times P$ *is a ternary transition relation.*[5]

In this paper, an occurrence of an action refers to the commitment of an executing task. Hence, actions occur instantaneously.

3. Note that each compensable task has a compensating task which eliminates the effect of this compensable task [44].

4. The difference lies in that we use exception handlers to represent the alternative branches and calculate the compensability and retriability of a task and its exception handlers as a whole.

5. This definition is adapted from the Labelled Transition Systems definition adopted by van Glabbeek in [6].

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 3, 2009 at 09:09 from IEEE Xplore. Restrictions apply.

$$\frac{\forall p, q \in P, a \in A}{(a \cdot p, a, p) \in F_{a \cdot p}}$$

$$\frac{(p,a,p') \in F_p}{(p+q,a,p') \in F_{p+q}} \qquad \frac{(q,a,q') \in F_q}{(p+q,a,q') \in F_{p+q}}$$

$$\frac{(p,a,p') \in F_p, a \notin H_2}{(p_{H_1} \| q_{H_2}, a, p'_{H_1} \| q_{H_2}) \in F_{p_{H_1} \| q_{H_2}}} \qquad \frac{(q,a,q') \in F_q, a \notin H_1}{(p_{H_1} \| q_{H_2}, a, p_{H_1} \| q'_{H_2}) \in F_{p_{H_1} \| q_{H_2}}}$$

$$\frac{(p,a,p') \in F_p, (q,a,q') \in F_q, a \in H_1 \cap H_2}{(p_{H_1} \| q_{H_2}, a, p'_{H_1} \| q'_{H_2}) \in F_{p_{H_1} \| q_{H_2}}}$$

Fig. 3. Operational semantics of the three basic operators of PA^{a}.

Definition 2. *The reachability relation $R \subseteq P \times P$ is defined as the* smallest *relation satisfying, for any states $p, p', p'' \in P$, $a \in A$,*

1. *$p\ R\ p$ and*
2. *$p\ R\ p' \wedge (p', a, p'') \in F \vdash p\ R\ p''$.*

A state p' is said to be reachable from p if and only if $p\ R\ p'$. The set of all states reachable from p is denoted as p^*. A state p is a termination state if $\neg \exists a \in A,\ q \in P : (p, a, q) \in F$. Without loss of generality, let the constant 0 be the termination state. Note that $0 \in P$.

Definition 3. *Let p be a state in P. The process defined by p is a quadruple $(p, p^*, A, F \cap (p^* \times A \times p^*))$. State p is the initial state of the process. For convenience, we also denote the process as p, and the set of its transitions as F_{P}.*

Definition 4. *Given a process p and a sequence of actions $a_1, a_2, \ldots,$ and a_{n}, if $\exists s_1, s_2, \ldots$ and $s_{\mathrm{n}} \in p^*$ such that $(p, a_1, s_1) \in F_{\mathrm{p}}, (s_1, a_2, s_2) \in F_{\mathrm{p}}, \ldots, (s_{\mathrm{n}-1}, a_{\mathrm{n}}, s_{\mathrm{n}}) \in F_{\mathrm{p}}$, then we call the sequence of actions $a_1, a_2, \ldots,$ and a_{n} a trace of this process. If s_{n} is a termination state of this process, then this trace is called a complete trace. Let $trace(p)$ denote the set of all the complete traces of process p. For a trace t and a positive integer i, $t[i]$ $(i > 0)$ denotes the ith action in the sequence of t if i is no more than the number of actions in the sequence of t.*

Definition 5. *Three basic operators are defined as follows for process composition with the precedence "$\cdot$," "$\|$," and "$+$," in descending order:*

- *"$\cdot$": a prefix operator,*
- *"$\|$": a parallel composition operator, and*
- *"$+$": a choice composition operator.*

For every action a, $a \cdot 0$ denotes the process that executes the action a and then terminates. We call a the leading action of the process $a \cdot 0$. Action a is called a *port action* if it is used to communicate with other processes. For ease of discussion, we restrict ourselves to the set of processes that communicate with one another synchronously.[6] We note that the process model supports recursion. For instance, we may define a process p to be $a \cdot b \cdot p$, which means that p first engages in performing tasks a and b, and then behaves like the process p.

A process p with a set of port actions H is usually denoted as p_{H}. For simplicity, we assume that two port actions from different processes with the same name mean that they communicate with each other synchronously. We also assume that the names of different port actions are distinct and that no nonport action shares the same name as a port action. The operational semantics of these three basic operators are given in Fig. 3.

For every action $a \in A$, we assign it two properties, *Compensability* and *Retriability*, because they are related to the atomicity sphere as discussed earlier. These two properties can be checked by two predicates: $\mathrm{C}(a)$ and $\mathrm{R}(a)$. Predicate $\mathrm{C}(a)$ is *true* if and only if the task represented by action a is compensable and predicate $\mathrm{R}(a)$ is *true* if and only if the task represented by action a is retriable. In our framework, nonport actions are renamed as *silent actions* in the public views to encapsulate process details that do not need to be exposed. In total, there are four kinds of silent actions according to all combinations of the two properties. These four silent actions are defined as follows:

Definition 6. *$\tau_{\mathrm{c,r}}$, $\tau_{\mathrm{nc,r}}$, $\tau_{\mathrm{c,nr}}$, and $\tau_{\mathrm{nc,nr}} \in A$ are special constants which represent four different kinds of silent actions with different combinations of compensability and retriability properties (see Table 1).*

For example, the action corresponding to the task "*schedule*" in the shipper process in Fig. 1 is a nonport action. It is compensable and nonretriable. So it is renamed as the silent action $\tau_{\mathrm{c,nr}}$ in the shipper's public view. Note that these four special actions are not involved in any

TABLE 1
Properties of Silent Actions

	$\tau_{c,r}$	$\tau_{nc,r}$	$\tau_{c,nr}$	$\tau_{nc,nr}$
Compensability	true	false	true	false
Retriability	true	true	false	false

6. Note that, in this paper, we refer to synchronous communication as the semantics of application level synchronizability, not the underlying communication protocol. We discuss the application level synchronizability and processes with asynchronous communication in Section 4.

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 3, 2009 at 09:09 from IEEE Xplore. Restrictions apply.

Let P be a set of processes. $\forall x, y, z \in P, a, b \in A$.

$x + y = y + x$	(A1)	$\tau_{c,r} \cdot x = x$	(R1)
$(x + y) + z = x + (y + z)$	(A2)	$\tau_{nc,r} \cdot \tau_{nc,r} \cdot x = \tau_{nc,r} \cdot x$	(R2)
$x + x = x$	(A3)	$\tau_{c,nr} \cdot \tau_{nc,r} \cdot x = \tau_{nc,nr} \cdot x$	(R3)
$x + 0 = x$	(A4)	$\tau_{c,nr} \cdot \tau_{c,nr} \cdot x = \tau_{c,nr} \cdot x$	(R4)
$a \cdot (x+ y) = a \cdot x + a \cdot y$	(A5)	$\tau_{c,nr} \cdot \tau_{nc,nr} \cdot x = \tau_{nc,nr} \cdot x$	(R5)
$x \,\|\| \, y = y \,\|\| \, x$	(M1)	$\tau_{nc,nr} \cdot \tau_{nc,r} \cdot x = \tau_{nc,nr} \cdot x$	(R6)
$(x + y)_H \,\|\| \, z = x_H \,\|\| \, z + y_H \,\|\| \, z$	(M2)	$\tau_{nc,r} \cdot \tau_{c,nr} \cdot x = \phi$	(R7)
$0_{H1} \,\|\| \, 0_{H2} = 0_{H1 \cup H2}$	(M3)	$\tau_{nc,r} \cdot \tau_{nc,nr} \cdot x = \phi$	(R8)
$a \cdot x_{H1} \,\|\| \, 0_{H2} = a \cdot (x_{H1} \,\|\| \, 0_{H2})$ $(a \notin H_1 \cap H_2)$	(M4)	$\tau_{nc,nr} \cdot \tau_{c,nr} \cdot x = \phi$	(R9)
$a \cdot x_{H1} \,\|\| \, 0_{H2} = 0_{H1 \cup H2}$ $(a \in H_1 \cap H_2)$	(M5)	$\tau_{nc,nr} \cdot \tau_{nc,nr} \cdot x = \phi$	(R10)
$a \cdot x_{H1} \,\|\| \, b \cdot y_{H2} = a \cdot (x_{H1} \,\|\| \, b \cdot y_{H2}) + b \cdot (y_{H2} \,\|\| \, a \cdot x_{H1})$ $(a, b \notin H_1 \cap H_2)$	(M6)	$\tau_{nc,r} \cdot \phi = \phi$	(R11)
$a \cdot x_{H1} \,\|\| \, b \cdot y_{H2} = a \cdot (x_{H1} \,\|\| \, b \cdot y_{H2})$ $(a \notin H_1 \cap H_2, b \in H_1 \cap H_2)$	(M7)	$\tau_{c,nr} \cdot \phi = \phi$	(R12)
$a \cdot x_{H1} \,\|\| \, b \cdot y_{H2} = a \cdot (x_{H1} \,\|\| \, y_{H2})$ $(a, b \in H_1 \cap H_2, a = b)$	(M8)	$\tau_{nc,nr} \cdot \phi = \phi$	(R13)
$a \cdot x_{H1} \,\|\| \, b \cdot y_{H2} = 0_{H1 \cup H2}$ $(a, b \in H_1 \cap H_2, a \neq b)$	(M9)	$x + \phi = \phi$	(R14)
		$x \,\|\| \, \phi = \phi$	(R15)

Fig. 4. Axioms of atomicity-equivalent process algebra.

communication synchronization since they are from nonport actions.

According to preceding discussions, a process violates the atomicity sphere if and only if there exists a nonretriable task executed after some noncompensable task in one possible execution of this process. Therefore, the constant processes with the format $p = \tau_{\mathrm{nc},x} \cdot \tau_{y,\mathrm{nr}} \cdot 0(\mathrm{x} \in \{r, nr\}, \mathrm{y} \in \{c, nc\})$ violate the atomicity sphere. Since our interest is to model and check the atomicity sphere through the public views, in order to simplify the derivation of public views and for ease of presentation, we introduce a constant abstract state ϕ to generalize such kinds of constant processes p (i.e., $p = \tau_{\mathrm{nc},x} \cdot \tau_{y,\mathrm{nr}} \cdot 0$, where $x \in \{r, nr\}$, $y \in \{c, nc\}$) that violate the atomicity sphere. In the sequel, for brevity and without loss of generality, we use state ϕ instead of these actual constant processes $p = \tau_{\mathrm{nc},x} \cdot \tau_{y,\mathrm{nr}} \cdot 0$ $(\mathrm{x} \in \{r, nr\}, \mathrm{y} \in \{c, nc\})$ because we do not need to differentiate them in the analysis of a process' atomicity sphere.

According to the above simplification technique and presentation notation, a process satisfies the atomicity sphere if and only if 1) no nonretriable tasks are executed after some noncompensable task in every possible execution of this process and 2) it does not contain any state ϕ. This is formalized as follows:

Definition 7. *φ is a predicate of a process. $\varphi(p)$ is* true *if and only if the following condition holds:*

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

The predicate $\varphi(p)$ is *true* whenever process p satisfies the atomicity sphere. For example, the supplier process in Fig. 1 has only one complete trace containing three actions (e.g., "*receive order*," "*book order*," and "*send invoice*"). Action "*book order*" is noncompensable and retriable, whereas the other two are compensable and retriable. Therefore, the predicate $\varphi(supplier)$ is *true*.

In the next section, we introduce our proposed process algebraic model to derive atomicity-equivalent public views from private processes. These public views will be used to check the atomicity sphere instead of the private processes.

2.3 Atomicity-Equivalent Process Algebra PA^a

Based on the preceding definitions, we present a process algebra called *atomicity-equivalent process algebra*, denoted as PA^{a}, to derive atomicity-equivalent public views from private processes. To do so, we introduce in this algebra the following axioms, as depicted in Fig. 4. These axioms describe the atomicity-equivalent relationship between processes. We later show how to use these axioms to derive atomicity-equivalent public views from private processes.

Note that the axioms (except A4 and A5[7]) in the left-hand column in Fig. 4 comply with the operational semantics of the three operators. In particular, Axioms A1-A3 represent the behaviors of the operator "+", whereas Axioms M1-M9 represent the behaviors of the operator "||" and "·". For example, Axiom M5 states that the composite process would terminate if a process with a leading port action attempts to communicate with a terminated process. Axiom M6 states that if the leading actions of the two processes are nonport actions, they would interleave in the sense that either of them can be executed first. If one leading action is a port action, whereas the other is not, then the process with the leading port action waits to communicate with another process, as represented in Axiom M7. Axiom M8 represents that two processes are synchronized on their leading port actions. If the two port actions do not share the same name (Axiom M9), the composite process would be terminated because of a deadlock.

The axioms in the right-hand column in Fig. 4 represent the simplification rules for silent actions (R1-R10) and the state ϕ (R11-R15). For example, R3 means that a compensable and nonretriable silent action $(\tau_{\mathrm{c,nr}})$ followed by a noncompensable and retriable silent action $(\tau_{\mathrm{nc,r}})$ can be simplified as a noncompensable and nonretriable silent

7. Axioms A4 and A5 represent that the processes on both sides have the same traces, but they do not comply to the operational semantics. We introduce these axioms into this algebra to facilitate the simplification of the atomicity-equivalent public views.

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 3, 2009 at 09:09 from IEEE Xplore. Restrictions apply.

action ($\tau_{\mathrm{nc,nr}}$). Such simplification is based on the observation that we are only interested in the calculation of atomicity-related properties, and thus any information other than compensability and retriability can be ignored. Axioms R14 and R15 represent that a process composing (in parallel or in choice) a state violating the atomicity sphere (ϕ) will also lead to the state violating the atomicity sphere. All the above axioms can be used to translate one process into another, especially for simplifying a private process into a public view. This is formalized as follows:

Definition 8. *A term of* PA^{a} *is defined as follows:*

1. *Variables* $x, y \in P$ *are terms.*
2. *Constant processes are terms, e.g.,* 0, ϕ, $\tau_{\mathrm{nc,r}} \cdot 0$.
3. *If* t_1, t_2 *are terms, then, for any* $a \in A$, $a \cdot t_1$, $t_1 + t_2$, $t_1 \| t_2$ *are also terms.*

A term that contains variables is called an *open term.* A term without variables is called a *close term*.

Definition 9. *A term* t_1 *can be reduced to another term* t_2 *using the axioms of* PA^{a}, *denoted as* $\mathrm{PA}^{\mathrm{a}} \vdash t_1 \rightarrow t_2$, *if and only if any of the following conditions holds:*

1. $t_1 = t_2$ *is an axiom of* PA^{a}.
2. *(Substitution)* $\exists$ *terms* $t(x_1, \ldots, x_{\mathrm{n}})$ *and*

$$s(x_1, \ldots, x_{\mathrm{n}}) : \mathrm{PA}^{\mathrm{a}} \vdash t(x_1, \ldots, x_{\mathrm{n}}) \rightarrow s(x_1, \ldots, x_{\mathrm{n}}) \wedge t_1 = t(p_1, \ldots, p_{\mathrm{n}}) \wedge t_2 = s(p_1, \ldots, p_{\mathrm{n}}),$$

   where $x_1, \ldots, x_{\mathrm{n}}$ *are variables and* $p_1, \ldots, p_{\mathrm{n}}$ *are close terms.*
3. *(Context)* $\exists t_1', t_2', a, p$:

$$\mathrm{PA}^{\mathrm{a}} \vdash t_1' \rightarrow t_2' \wedge ((t_1 = a \cdot t_1', t_2 = a \cdot t_2') \vee (t_1 = p + t_1', t_2 = p + t_2') \vee (t_1 = p \| t_{1\,\mathrm{H}}', t_2 = p \| t_{2\,\mathrm{H}}')).$$

Based on this definition, the *reduction* of a term t_1 to another term t_2 begins from the axioms of the algebra PA^{a} and then repeatedly applies the substitution rule or the context rule until t_1 is finally reduced into t_2. During the reduction, we call the application of each of the three rules in Definition 9 a *reduction step.* For example, the reduction of $\tau_{\mathrm{c,r}} \cdot a \cdot b \cdot 0 + c \cdot 0$ to $a \cdot b \cdot 0 + c \cdot 0$ has three reduction steps. First, $\tau_{\mathrm{c,r}} \cdot x \rightarrow x$ (the axiom of PA^{a}). Then, $\tau_{\mathrm{c,r}} \cdot a \cdot b \cdot 0 \rightarrow a \cdot b \cdot 0$ (substitution of x with $a \cdot b \cdot 0$). Finally, $\tau_{\mathrm{c,r}} \cdot a \cdot b \cdot 0 + c \cdot 0 \rightarrow a \cdot b \cdot 0 + c \cdot 0$ (context). The reduction relation is reflective and transitive. Therefore, we define a reduction relationship as follows:

Definition 10. $>$ *is a reduction relation satisfying:* $\forall t_1, t_2, t_3$,

1. $\mathrm{PA}^{\mathrm{a}} \vdash t_1 > t_1$,
2. $\mathrm{PA}^{\mathrm{a}} \vdash t_1 \rightarrow t_2 \Rightarrow \mathrm{PA}^{\mathrm{a}} \vdash t_1 > t_2$, *and*
3. $\mathrm{PA}^{\mathrm{a}} \vdash t_1 > t_2$, $\mathrm{PA}^{\mathrm{a}} \vdash t_2 > t_3 \Rightarrow \mathrm{PA}^{\mathrm{a}} \vdash t_1 > t_3$.

The reduction of a process into another using the axioms of PA^{a} will preserve the atomicity sphere of this process. This is guaranteed by the following theorem.

Theorem 1. *Let* p_1 *and* p_2 *be two processes. If* $\mathrm{PA}^{\mathrm{a}} \vdash p_1 \rightarrow p_2$, *then* $\varphi(p_1) = \varphi(p_2)$.

To prove this theorem, we first introduce an auxiliary function Ψ and prove several lemmas. To simplify the presentation, we use predicates $CT(t)$ and $RT(t)$ to indicate whether a trace t contains any noncompensable or nonretriable actions, that is, $CT(t) = true$ *iff* $\forall a = t[i]$, $i > 0$, $C(a) = true$; $RT(t) = true$ *iff* $\forall a = t[i]$, $i > 0$, $\mathrm{R}(a) = true$.

Definition 11. Ψ *is a function that partitions all the processes into five classes:* $\{violated, C\_R, NC\_R, C\_NR, NC\_NR\}$, *i.e.,* $\Psi(p) = violated$: *If* $\varphi(p) = false$; *otherwise,*

- C_R: *if* $p = 0$ *or* $\forall t \in trace(p)$: $CT(t) = true$, $RT(t) = true$,
- NC_R: *if* $\exists t \in trace(p) : CT(t) = false \wedge \forall t' \in trace(p)$: $RT(t') = true$,
- C_NR: *if* $\forall t \in trace(p) : CT(t) = true \wedge \exists t' \in trace(p)$: $RT(t') = false$, *and*
- NC_NR: *if* $\exists t, \quad t' \in trace(p) : CT(t) = false$, $RT(t') = false$.

Note that the partition covers all the situations. *violated* represents the class of processes that violate the atomicity sphere. C_R, NC_R, and C_NR represent the class of processes with no noncompensable and nonretriable tasks, the class of processes with certain noncompensable task(s) but no nonretriable task, and the class of processes with certain nonretriable task(s) but no noncompensable task, respectively. NC_NR describes the class of processes that have both noncompensable and nonretriable tasks but do not violate the atomicity sphere. For example, the value of function Ψ for the shipper process in Fig. 1 is equal to NC_NR, because the process satisfies the atomicity sphere, and the trace {"*receive request*," "*schedule*," "*deliver*," "*send receipt*"} has both noncompensable and nonretriable tasks. Obviously, for any two processes p_1 and p_2, $\Psi(p_1) = \Psi(p_2) \Rightarrow \varphi(p_1) = \varphi(p_2)$ (that is, $\Psi(p_1) = \Psi(p_2) = violated \Rightarrow \varphi(p_1) = \varphi(p_2) = false$; otherwise, $\varphi(p_1) = \varphi(p_2) = true$).

Lemma 1. *Let* p_1, p_2 *be two processes,* $\mathrm{PA}^{\mathrm{a}} \vdash p_1 \rightarrow p_2$ *and* $\Psi(p_1) = \Psi(p_2)$, *then* $\forall a \in A$, $q \in P$

1. $\Psi(a \cdot p_1) = \Psi(a \cdot p_2)$,
2. $\Psi(p_1 + q) = \Psi(p_2 + q)$, *and*
3. $\Psi(p_1 \| q) = \Psi(p_2 \| q)$.

Lemma 2. *Let* p_1, p_2 *be two processes,* $\mathrm{PA}^{\mathrm{a}} \vdash p_1 \rightarrow p_2 \Rightarrow \Psi(p_1) = \Psi(p_2)$.

The proofs of Lemmas 1 and 2 are given in the Appendix.

Since, for any two processes p_1, p_2, $\Psi(p_1) = \Psi(p_2) \Rightarrow \varphi(p_1) = \varphi(p_2)$. Based on Lemma 2, $\mathrm{PA}^{\mathrm{a}} \vdash p_1 \rightarrow p_2 \Rightarrow \Psi(p_1) = \Psi(p_2)$. Thus, we have $\mathrm{PA}^{\mathrm{a}} \vdash p_1 \rightarrow p_2 \Rightarrow \varphi(p_1) = \varphi(p_2)$, that is, Theorem 1 is satisfied. Theorem 1 shows that in the framework of PA^{a}, we can reduce one process into another and this reduction does not change their atomicity sphere properties. Based on Theorem 1, we have the following two corollaries:

Corollary 1. *Let* p_1, p_2 *be two processes. If* $\mathrm{PA}^{\mathrm{a}} \vdash p_1 > p_2$, *then* $\varphi(p_1) = \varphi(p_2)$. *We say that* p_1 *and* p_2 *are* atomicity-equivalent.

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 3, 2009 at 09:09 from IEEE Xplore. Restrictions apply.

Proof. Suppose the reduction of p_1 to p_2 needs m steps, that is, there exists $\mathrm{PA}^{\mathrm{a}} \vdash p_1 \rightarrow p_1^{(1)}, \mathrm{PA}^{\mathrm{a}} \vdash p_1^{(1)} \rightarrow p_1^{(2)}, \ldots, \mathrm{PA}^{\mathrm{a}} \vdash p_1^{(\mathrm{m}-1)} \rightarrow p_1^{(\mathrm{m})}$, and $p_1^{(\mathrm{m})} = p_2$. According to Theorem 1, $\varphi(p_1) = \varphi(p_1^{(1)}) = \varphi(p_1^{(2)}) = \cdots = \varphi(p_1^{(\mathrm{m})}) = \varphi(p_2)$. The result follows. □

Corollary 2.

$$\forall p_1, p_2, \ldots, p_{\mathrm{n}} \in P, \mathrm{PA}^{\mathrm{a}} \vdash p_1 > pv_1, \mathrm{PA}^{\mathrm{a}} \vdash p_2 > pv_2, \ldots, \mathrm{PA}^{\mathrm{a}} \vdash p_{\mathrm{n}} > pv_{\mathrm{n}} \Rightarrow \varphi(p_1 \| p_2 \| \cdots \| p_{\mathrm{n}}) = \varphi(pv_1 \| pv_2 \| \cdots \| pv_{\mathrm{n}}).$$

Proof. We first prove that $\varphi(p_1 \| p_2 \| \cdots \| p_{\mathrm{n}}) = \varphi(pv_1 \| p_2 \| \cdots \| p_{\mathrm{n}})$. The remaining proof steps are similar. Since $\mathrm{PA}^{\mathrm{a}} \vdash p_1 > pv_1$, there exists $p_1 \rightarrow p_1^{(1)}, p_1^{(1)} \rightarrow p_1^{(2)}, \ldots, p_1^{(\mathrm{m}-1)} \rightarrow p_1^{(\mathrm{m})}$ and $p_1^{(\mathrm{m})} = pv_1$. Therefore,

$$p_1 \| p_2 \| \cdots \| p_{\mathrm{n}} \rightarrow p_1^{(1)} \| p_2 \| \cdots \| p_{\mathrm{n}} \rightarrow \cdots \rightarrow pv_1 \| p_2 \| \cdots \| p_{\mathrm{n}}.$$

Hence, $\varphi(p_1 \| p_2 \| \cdots \| p_{\mathrm{n}}) = \varphi(p_1^{(1)} \| p_2 \| \cdots \| p_{\mathrm{n}}) = \cdots = \varphi(pv_1 \| p_2 \| \cdots \| p_{\mathrm{n}})$. The result follows. □

In the next section, we introduce a relabeling operator to derive atomicity-equivalent public views from private processes and a composition operator to integrate and compose web service in this process algebra model. We show that these public views can be used to check the atomicity sphere instead of using the private processes in the global analysis.

2.4 Atomicity-Equivalent Public View and Service Composition

In this section, we introduce two operators to generate atomicity-equivalent public views from private processes, and to integrate and compose services into our process algebra model.

Definition 12. *σ_{H} is a relabeling operator that renames a process into another, where $H \subseteq A$ is the set of actions that keep unchanged in the renamed process. σ_{H} is defined recursively as follows:*

1. $\sigma_{\mathrm{H}}(0) = 0$, $\sigma_{\mathrm{H}}(\phi) = \phi$.
2. $\forall a \in A$,

$$\begin{array}{ll} \textit{If } a \in H, \ \sigma_{\mathrm{H}}(a \cdot x) = a \cdot \sigma_{\mathrm{H}}(x). & \\ \textit{If } a \notin H, \ \sigma_{\mathrm{H}}(a \cdot x) = & \\ \tau_{\mathrm{c,r}} \cdot \sigma_{\mathrm{H}}(x) & \textit{if } \mathrm{C}(a) = true \ \wedge \ \mathrm{R}(a) = true \\ \tau_{\mathrm{nc,r}} \cdot \sigma_{\mathrm{H}}(x) & \textit{if } \mathrm{C}(a) = false \ \wedge \ \mathrm{R}(a) = true \\ \tau_{\mathrm{c,nr}} \cdot \sigma_{\mathrm{H}}(x) & \textit{if } \mathrm{C}(a) = true \ \wedge \ \mathrm{R}(a) = false \\ \tau_{\mathrm{nc,nr}} \cdot \sigma_{\mathrm{H}}(x) & \textit{if } \mathrm{C}(a) = false \wedge \mathrm{R}(a) = false. \end{array}$$

3. $\forall x, y \in P$, $\sigma_{\mathrm{H}}(x + y) = \sigma_{\mathrm{H}}(x) + \sigma_{\mathrm{H}}(y)$.

Operator σ_{H} renames all nonport actions of a process into silent actions. We use this operator to rename a process first, and then reduce the renamed process under the framework of PA^{a} to generate its public view, that is, $\mathrm{PA}^{\mathrm{a}} \vdash \sigma_{\mathrm{H}}(p) > pv$, where pv is the public view of p. The following theorem shows that the public views generated in this way preserve the atomicity sphere. Therefore, we call them *atomicity-equivalent* public views.

Theorem 2. *Let p be a process, and $\mathrm{PA}^{\mathrm{a}} \vdash \sigma_{\mathrm{H}}(p) > pv$, then $\varphi(p) = \varphi(pv)$.*

Proof. Based on Theorem 1, $\mathrm{PA}^{\mathrm{a}} \vdash \sigma_{\mathrm{H}}(p) > pv \Rightarrow \varphi(\sigma_{\mathrm{H}}(p)) = \varphi(pv)$. Hence, we only need to prove $\Psi(p) = \Psi(\sigma_{\mathrm{H}}(p))$ in order to infer $\varphi(p) = \varphi(\sigma_{\mathrm{H}}(p)) \Rightarrow \varphi(p) = \varphi(\sigma_{\mathrm{H}}(p)) = \varphi(pv)$. The proof of $\Psi(p) = \Psi(\sigma_{\mathrm{H}}(p))$ is straightforward because operator σ_{H} only renames the actions but does not change their compensability and retriability properties. The result follows. □

In the process algebra model, we assume that synchronized communicating port actions have the same name. However, in a service composition, port actions in different services may have different names. To integrate and compose services using our model, we need a global composition specification to match the communicating port actions between different services. The acquirement of a global composition specification is discussed in Section 4.

Definition 13. *The global composition specification (GS) of services is a set of port action pairs in the service composition model. For every pair $(a, b) \in \mathrm{GS}$, a and b represent two port actions from processes p_1 and p_2, respectively. This pair means that two processes p_1 and p_2 communicate with each other by synchronizing port actions a and b. Let H_{i} be the set containing all port actions of process p_{i} in the collaboration. GS is said to be well formed if and only if $\forall a_{\mathrm{l}} \in H_{\mathrm{i}}$: $(\exists a_t \in H_{\mathrm{j}}, \quad (i \neq j)$: $((a_{\mathrm{l}}, a_{\mathrm{t}}) \in \mathrm{GS}$ or $(a_{\mathrm{t}}, a_{\mathrm{l}}) \in \mathrm{GS}) \wedge (\neg \exists a_{\mathrm{k}}, a_{\mathrm{k}} \neq a_{\mathrm{t}}$: $(a_{\mathrm{l}}, a_{\mathrm{k}}) \in \mathrm{GS}$ or $(a_{\mathrm{k}}, a_l) \in \mathrm{GS}))$.*

In the remainder of this paper, we always assume that GS is well formed. The acquisition of such a global composition specification is related to the compatibility of services. We further discuss the well-formedness issue in Section 4. Based on the global composition specification, services are composed by linking all communicating port action pairs. The formal definition of service composition is given as follows, which first renames all communicating port actions in involved services to the same names (so that we can represent them using the process algebra model PA^{a}) and then conducts parallel composition of these services in our model to form a composite service:

Definition 14. *$\oplus_{\mathrm{GS}}$ is a composition operator which composes n services into a composite service using the global composition specification GS. $\oplus_{\mathrm{GS}}(p_1, p_2, \ldots, p_{\mathrm{n}})$ is defined as follows:*

1. *Let $H_1, H_2, \ldots, H_{\mathrm{n}}$ be the sets of port actions of processes $p_1, p_2, \ldots, p_{\mathrm{n}}$, respectively. Mapping function γ_{GS} is defined as GS $\rightarrow$ A. If (a, b) or $(b, a) \in \mathrm{GS}$, then $\gamma_{\mathrm{GS}}(a, b)$ returns another action c, which is used to rename the actions a and b. We rename the actions of $p_1, p_2, \ldots, p_n$ to produce $p_1', p_2', \ldots, p_{\mathrm{n}}'$, respectively, by the following approach:*

   $\forall a \in H_{\mathrm{i}}$, if $\exists b \in H_{\mathrm{j}}$ such that $(a, b) \in \mathrm{GS}$, then we replace action a of p_{i} and action b of p_{j} with action $c = \gamma_{\mathrm{GS}}(a, b)$. The properties of c is defined as $\mathrm{C}(c) = \mathrm{C}(a) \& \mathrm{C}(b)$, $\mathrm{R}(c) = \mathrm{R}(a) \& \mathrm{R}(b)$, where "&"

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 3, 2009 at 09:09 from IEEE Xplore. Restrictions apply.

represents a logic "and." The names and properties of other actions remain unchanged.

2. *Let* $H'_1, H'_2, \ldots, H'_n$ *be the sets of port actions of* $p'_1, p'_2, \ldots, p'_n$, *respectively, that is,*

$$H'_i = \{c | (\exists a \in H_i, b \in H_j : (a,b) \text{ or } (b,a) \in \text{GS}, \\ c = \gamma_{\text{GS}}(a,b)) \text{ or } (c \in H_i, \neg\exists b \in H_j : (c,b) \text{ or } \\ (b,c) \in \text{GS}\}.$$

$\oplus_{\text{GS}}(p_1, p_2, \ldots, p_n)$ *is defined as* $p'_1 \| p'_2 \| \cdots \| p'_n$.

Operator $\oplus_{\text{GS}}$ is used to integrate services into a composite service in our process algebra model. Since a public view of a private process is also a process, $\oplus_{\text{GS}}$ can also be used to compose the public views of private processes. The following theorem shows that the composition of atomicity-equivalent public views is atomicity-equivalent to the composition of the corresponding private processes.

Theorem 3. *Let* $p_1, p_2, \ldots, p_n$ *be n private processes,* $pv_1, \ldots, pv_n$ *be their corresponding atomicity-equivalent public views, and GS be the global composition specification, that is,*

$$\text{PA}^a \vdash \sigma_{H1}(p_1) > pv_1, \text{PA}^a \vdash \sigma_{H2}(p_2) > pv_2, \ldots, \\ \text{PA}^a \vdash \sigma_{Hn}(p_n) > pv_n,$$

then $\varphi(\oplus_{\text{GS}}(p_1, p_2, \ldots, p_n)) = \varphi(\oplus_{\text{GS}}(pv_1, pv_2, \ldots, pv_n))$.

To prove this theorem, let us first introduce a lemma below.

Lemma 3. *Let p be a process, and H be the set of port actions of p, then for any process q,* $\varphi(\sigma_H(p) \| q) = \varphi(p \| q)$.

Proof. We know that $\sigma_H(p)$ only renames nonport actions. Thus, every complete trace of $p \| q$, after replacing each nonport action of p with its corresponding silent action $\tau_{x,y}(x \in \{c, nc\}, y \in \{r, nr\})$, is also a complete trace of $\sigma_H(p) \| q$, and vice versa. Moreover, each action's compensability and reliability properties in $\sigma_H(p) \| q$ keep the same as those in $p \| q$, therefore, we have $\varphi(\sigma_H(p) \| q) = \varphi(p \| q)$. □

Proof of Theorem 3. According to Definition 14, we have $\oplus_{\text{GS}}(p_1, p_2, \ldots, p_n) = p'_1 \| p'_2 \| \cdots \| p'_n$ and

$$\oplus_{\text{GS}}(pv_1, pv_2, \ldots, pv_n) = pv'_1 \| pv'_2 \| \cdots \| pv'_n.$$

Since $\text{PA}^a \vdash \sigma_{Hi}(p_i) > pv_i$, $(i = 1..n)$, and the operator $\oplus_{\text{GS}}$ renames only the communicating port actions to the same name with the same properties, we therefore have $\text{PA}^a \vdash \sigma_{Hi'}(p'_i) > pv'_i$, $(i = 1..n)$. According to Corollary 2, we have $\varphi(\sigma_{H1'}(p'_1) \| \sigma_{H2'}(p'_2) \| \cdots \| \sigma_{Hn'}(p'_n)) = \varphi(pv'_1 \| pv'_2 \| \cdots \| pv'_n)$. On the other hand, according to Lemma 3, we have

$$\varphi(\sigma_{H1'}(p'_1) \| \sigma_{H2'}(p'_2) \| \cdots \| \sigma_{Hn'}(p'_n)) = \varphi(p'_1 \| \sigma_{H2'}(p'_2) \| \cdots \\ \| \sigma_{Hn'}(p'_n)) = \varphi(p'_1 \| p'_2 \| \cdots \| \sigma_{Hn'}(p'_n)) = \cdots = \varphi(p'_1 \| p'_2 \| \cdots \| p'_n).$$

Hence, $\varphi(p'_1 \| p'_2 \| \cdots \| p'_n) = \varphi(pv'_1 \| pv'_2 \| \cdots \| pv'_n)$. The result follows. □

Theorem 3 shows that we can use atomicity-equivalent public views of private processes to check the atomicity sphere instead of using the private processes directly. Therefore, service providers only need to publish their atomicity-equivalent public views in the service registries for service matching.

Sometimes, a service may be composed of several other services. The following theorem shows that we can derive the atomicity-equivalent public view of this composite service directly from the atomicity-equivalent public views of these composed services instead of from their backend processes.

Theorem 4. *Let* p_1, p_2, p_3 *be three private processes.* $\text{PA}^a \vdash \sigma_{H1}(p_1) > pv_1$, $\text{PA}^a \vdash \sigma_{H2}(p_2) > pv_2$, $\text{PA}^a \vdash \sigma_{H3}(p_3) > pv_3$. *Service* p_{12} *is composed of service* p_1 *and* p_2 *with the specification GS, that is,* $p_{12} = \oplus_{\text{GS}}(p_1, p_2)$. *Let H be the set of port actions for* p_{12} *(H does not contain any nonport actions of* p_1, p_2*), and* $\text{PA}^a \vdash \sigma_H(\oplus_{\text{GS}}(pv_1, pv_2)) > pv_{12}$, *then we have* $\varphi(\oplus_{\text{GS}}(p_{12}, p_3)) = \varphi(\oplus_{\text{GS}}(pv_{12}, pv_3))$.

Proof. According to Definition 14, let p'_1, p'_2 be the processes of p_1, p_2 after renaming their communicating port action pairs in the global composition specification GS, and H'_1, H'_2 be the corresponding resulting port action sets, we have $p_{12} = \oplus_{\text{GS}}(p_1, p_2) = p'_1 \| p'_2$. On the other hand, since $\text{PA}^a \vdash \sigma_{H1}(p_1) > pv_1$, we have $\text{PA}^a \vdash \sigma_{H1'}(p'_1) > pv'_1$, where pv'_1 is taken from pv by renaming the corresponding port actions in the same way p_1 is renamed to p'_1 (since the reduction does not change the port actions). Similarly, we have $\text{PA}^a \vdash \sigma_{H2'}(p'_2) > pv'_2$, and $\oplus_{\text{GS}}(pv_1, pv_2) = pv'_1 \| pv'_2$. Since $\text{PA}^a \vdash \sigma_H(\oplus_{\text{GS}}(pv_1, pv_2)) > pv_{12}$ and $\oplus_{\text{GS}}(pv_1, pv_2) = pv'_1 \| pv'_2$, we have $\text{PA}^a \vdash \sigma_H(pv'_1 \| pv'_2) > pv_{12}$. Now, we further transform the process p'_1 to p''_1, p'_2 to p''_2, and p_3 to p'_3 by renaming the communicating port action pairs between p_1, p_3 and p_2, p_3 in the global composition specification GS, and get the corresponding port action set H''_1, H''_2, H'_3. Likewise, since we only rename the port actions of p'_1, p'_2, and p_3, we have $\text{PA}^a \vdash \sigma_{H1''}(p''_1) > pv''_1$, $\text{PA}^a \vdash \sigma_{H2''}(p''_2) > pv''_2$, $\text{PA}^a \vdash \sigma_{H3'}(p'_3) > pv'_3$, where pv''_1, pv''_2, and pv'_3 are taken, respectively, from pv'_1, pv'_2, and pv_3 by, respectively, renaming the corresponding port actions in the same way p'_1, p'_2, and p_3 are renamed to p''_1, p''_2, and p'_3. Similarly, since $\text{PA}^a \vdash \sigma_H(pv'_1 \| pv'_2) > pv_{12}$, we therefore have $\text{PA}^a \vdash \sigma_{H'}(pv''_1 \| pv''_2) > pv'_{12}$, where pv'_{12} is taken from pv_{12} by renaming the port actions in the same way pv'_1 and pv'_2 are renamed to pv''_1 and pv''_2, and H' is the resulting port action set from H. Thus, according to Definition 14, we have

$$\oplus_{\text{GS}}(p_{12}, p_3) = \oplus_{\text{GS}}(p'_1 \| p'_2, p_3) = p''_1 \| p''_2 \| p'_3, \quad (1)$$

$$\oplus_{\text{GS}}(pv_{12}, pv_3) = pv'_{12} \| pv'_3. \quad (2)$$

Since $\text{PA}^a \vdash \sigma_{H1''}(p''_1) > pv''_1$, $\text{PA}^a \vdash \sigma_{H2''}(p''_2) > pv''_2$, $\text{PA}^a \vdash \sigma_{H3'}(p'_3) > pv'_3$, according to Corollary 2, we have $\varphi(\sigma_{H1''}(p''_1) \| \sigma_{H2''}(p''_2) \| \sigma_{H3'}(p'_3)) = \varphi(pv''_1 \| pv''_2 \| pv'_3)$. On the other hand, according to Lemma 3,

$$\begin{aligned} \varphi(\sigma_{H1''}(p''_1) \| \sigma_{H2''}(p''_2) \| \sigma_{H3'}(p'_3)) &= \varphi(p''_1 \| \sigma_{H2''}(p''_2) \| \sigma_{H3'}(p'_3)) \\ &= \varphi(p''_1 \| p''_2 \| \sigma_{H3'}(p'_3)) \\ &= \varphi(p''_1 \| p''_2 \| p'_3). \end{aligned}$$

Hence, we have

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 3, 2009 at 09:09 from IEEE Xplore. Restrictions apply.

$$\begin{aligned}\varphi(p_1''\|p_2''\|p_3') &= \varphi(\sigma_{H1''}(p_1'')\|\sigma_{H2''}(p_2'')\|\sigma_{H3'}(p_3')) \\ &= \varphi(pv_1''\|pv_2''\|pv_3').\end{aligned} \quad (3)$$

Similarly, since $PA^a \vdash \sigma_{H'}(pv_1''\|pv_2'') > pv_{12}'$, according to the reduction rules in Definition 9, we have $PA^a \vdash \sigma_{H'}(pv_1''\|pv_2'')\|pv_3' > pv_{12}'\|pv_3'$. According to Corollary 1, we have $\varphi(pv_{12}'\|pv_3') = \varphi(\sigma_{H'}(pv_1''\|pv_2'')\|pv_3')$. Then, according to Lemma 3, $\varphi(\sigma_{H'}(pv_1''\|pv_2'')\|pv_3') = \varphi(pv_1''\|pv_2''\|pv_3')$. Hence, we have

$$\varphi(pv_{12}'\|pv_3') = \varphi(\sigma_{H'}(pv_1''\|pv_2'')\|pv_3') = \varphi(pv_1''\|pv_2''\|pv_3'). \quad (4)$$

Therefore, according to (1), (2), (3), and (4), we have $\varphi(\oplus_{GS}(p_{12}, p_3)) = \varphi(\oplus_{GS}(pv_{12}, pv_3))$. The conclusion follows. □

Theorem 4 shows that we can use the atomicity-equivalent public views of composed services to derive the atomicity-equivalent public view for the composite service. Thus, we have guaranteed the correctness of our approach of using derived public views to check the atomicity sphere instead of using original private processes. In the next section, we discuss the algorithms on how to construct and compose public views, as well as on how to check the atomicity sphere.

2.5 Algorithm

In this section, we present three algorithms to construct public views from private processes, compose several public views into one, and check the atomicity sphere for composed public views, respectively.

In Section 2.3, we have introduced some reduction rules to derive atomicity-equivalent public views from private processes. Service providers could choose to apply these rules to simplify their atomicity-equivalent public views based on their requirements (i.e., they may choose to simplify the parts they need using corresponding reduction rules). The following algorithm will implement all these rules to derive an atomicity-equivalent public view from a private process. Let $s_{10} = (s_{10}, S_1, A_1, F_1)$ be a private process, and H be the set of port actions used in this process. The algorithm for constructing an atomicity-equivalent public view $s_{20} = (s_{20}, S_2, A_2, F_2)$ from its original process s_{10} is given as follows. This algorithm contains two steps. First, all nonport actions in process s_{10} are renamed to silent actions based on the definition of σ_H (Lines 2-8). The *rename* function on Line 3 implements the operator σ_H. Then, the Breadth-First-Search (BFS) algorithm [26] is used to traverse the process from the termination state (and atomicity sphere violation state ϕ) back to its initial state. During the traversal, the algorithm simplifies the process using axioms of PA^a.

Algorithm 1. Construct an atomicity-equivalent public view from a private process.

Input: private process $s_{10} = (s_{10}, S_1, A_1, F_1)$

Output: public view $s_{20} = (s_{20}, S_2, A_2, F_2)$

1. $s_{20} \leftarrow s_{10}$, $S_2 \leftarrow S_1$, $A_2 \leftarrow A_1$, $F_2 \leftarrow F_1$
2. ***for*** *every* $a \in A_1 \wedge a \notin H$,
3. $b \leftarrow rename(a)$,
4. $A_2 \leftarrow A_2 - \{a\} + \{b\}$
5. ***for*** *every* $(s_1, a, s_2) \in F_1$
6. $F_2 \leftarrow F_2 - \{(s_1, a, s_2)\} + \{(s_1, b, s_2)\}$
7. ***endfor***
8. ***endfor***
9. $\forall s_i \in S_2$, $visited(s_i) \leftarrow false$.
10. $\forall s_i = 0$ *or* $s_i = \phi$, $queue.enqueue(s_i)$, $visited(s_i) \leftarrow true$.
11. ***while***(*not empty(queue)*)
12. $cs \leftarrow queue.dequeue()$
13. ***for*** *every* $(ps, a, cs) \in F_2$
14. ***if*** $canSimplify(a, cs)$
15. ***if*** $cs = \phi$,
16. $ps \leftarrow \phi$, $queue.enqueue(ps)$, $visited(ps) \leftarrow true$
17. ***for*** *every* $(ps, b, ns) \in F_2$, $removeTransition(ps, b, ns)$, ***endfor***
18. ***else***
19. ***for*** *every* $(cs, b, ns) \neq (ps, a, cs) \in F_2$, $F_2 \leftarrow F_2 + \{(ps, b, ns)\}$, ***endfor***
20. $removeTransition(ps, a, cs)$
21. ***endif***
22. ***else***
23. ***for*** *every* $(s, c, ps) \in F_2$
24. ***if*** $canCombine(c, a)$
25. ***if*** $num\_of\_outcomingedge(ps) = 1$,
26. $d \leftarrow combine(c, a)$, $F_2 \leftarrow F_2 + \{(s, d, cs)\}$, $removeTransition(s, c, ps)$
27. ***else***
28. ***for*** *every* $(ns, b, ps) \in F_2$, $F_2 \leftarrow F_2 + \{(ns, b, ps')\}$, ***endfor***
29. $S_2 \leftarrow S_2 + \{ps'\}$, $F_2 \leftarrow F_2 + \{(ps', a, cs)\}$, $removeTransition(ps, a, cs)$.
30. ***endif***
31. ***else***
32. ***if*** $violateAS(c, a)$
33. ***for*** *every* $(s, g, bs) \in F_2$, $removeTransition(s, g, bs)$, ***endfor***
34. $s \leftarrow \phi$, $queue.enqueue(s)$, $visited(s) \leftarrow true$
35. ***endif***
36. ***endif***
37. ***endfor***
38. ***endif***
39. ***if*** $ps \in S_2 \wedge visited(ps) = false$, $queue.enqueue(ps)$, $visited(ps) \leftarrow true$, ***endif***
40. ***endfor***
41. ***endwhile***

$removeTransition(ps, a, cs)$

42. $F_2 \leftarrow F_2 - \{(ps, a, cs)\}$
43. ***if*** $\neg\exists(ts, c, cs) \in F_2 \wedge cs \neq s_{20}$,
44. ***for*** *every* $(cs, b, ns) \in F_2$, $removeTransition(cs, b, ns)$, ***endfor***
45. $S_2 \leftarrow S_2 - \{cs\}$,
46. ***endif***

The reduction of a public view using axioms of PA^a can be classified into five situations, as depicted in Fig. 5, where a circle represents a state and the label of an edge represents an action and cs is the current state being visited in the traversal of the given process:

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 3, 2009 at 09:09 from IEEE Xplore. Restrictions apply.

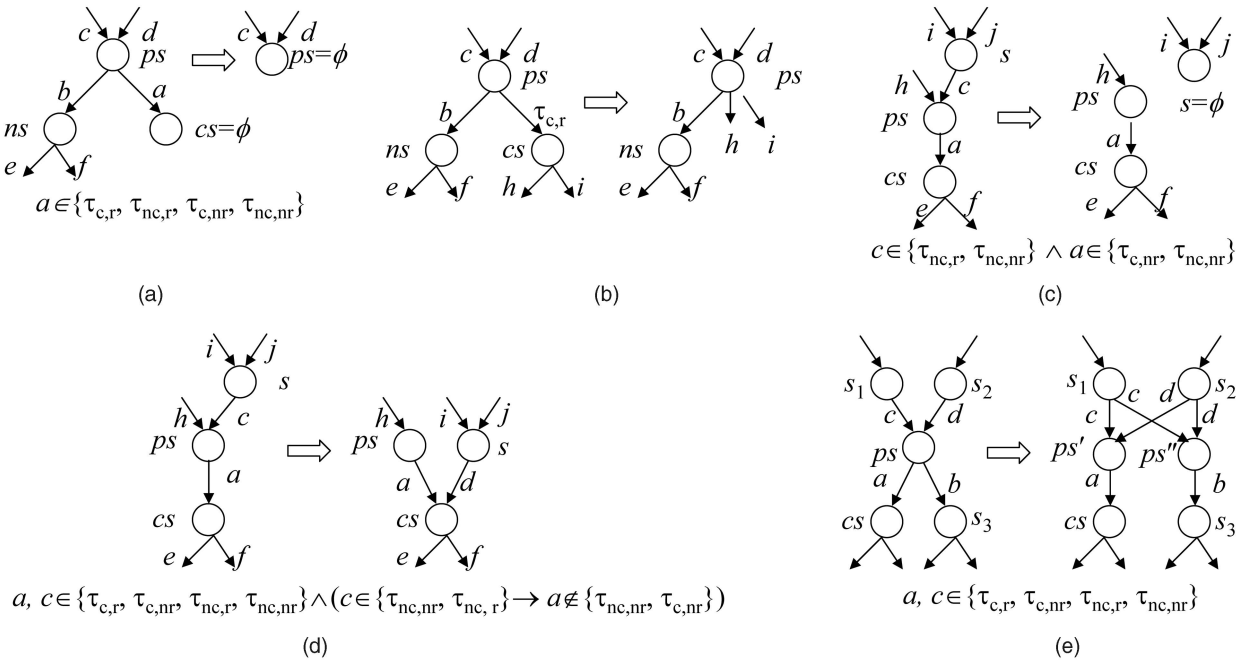

Fig. 5. Simplification of public views.

1. In Fig. 5a, $ps = b \cdot ns + a \cdot cs$. Since $cs = \phi$ and $a \in \{\tau_{c,r}, \tau_{nc,r}, \tau_{c,nr}, \tau_{nc,nr}\}$, based on R1 and R11-R13, $a \cdot cs > \phi$, so $ps > b \cdot ns + \phi$. According to R14, $b \cdot ns + \phi > \phi$. So, $ps > b \cdot ns + \phi > \phi$.
2. In Fig. 5b, $ps = b \cdot ns + \tau_{c,r} \cdot cs$. Based on R1, $ps = b \cdot ns + \tau_{c,r} \cdot cs > b \cdot ns + cs$. If $cs = a \cdot ps + q$ (i.e., a loop), then $ps > b \cdot ns + a \cdot ps + q$. During the traverse of the process, since state ps has already been visited, so the algorithm will not reduce the loop infinitely.
3. In Fig. 5c, $s = c \cdot ps = c \cdot a \cdot cs$. Since $c \in \{\tau_{nc,r}, \tau_{nc,nr}\}$ and $a \in \{\tau_{c,nr}, \tau_{nc,nr}\}$, so according to R7-R10, $s = c \cdot a \cdot cs > \phi$.
4. In Fig. 5d, $s = c \cdot ps = c \cdot a \cdot cs$. Since $c \in \{\tau_{c,r}, \tau_{nc,r}, \tau_{c,nr}, \tau_{nc,nr}\}$ and $a \in \{\tau_{c,r}, \tau_{nc,r}, \tau_{c,nr}, \tau_{nc,nr}\}$ and if $((c = \tau_{nc,nr})$ or $(c = \tau_{nc,r}))$, then $((a \neq \tau_{nc,nr})$ and $(a \neq \tau_{c,nr}))$, so, according to R1-R6, $s = c \cdot a \cdot cs > d \cdot cs$, where d is the resulting silent action by applying axiom of R1-R6.
5. In Fig. 5e, tasks a and c are both silent actions. $ps = a \cdot cs + b \cdot s_3$, $s_1 = c \cdot ps = c \cdot (a \cdot cs + b \cdot s_3)$. According to A5, $s_1 > c \cdot a \cdot cs + c \cdot b \cdot s_3$. Therefore, we split the original state ps into two states,[8] that is, $ps' = a \cdot cs$ and $ps'' = b \cdot s_3$. Then, we could apply the simplifications in Fig. 5a, Fig. 5b, Fig. 5c, and Fig. 5d to further simplify the public view.

8. Note that splitting a state into two may make the public view more complex than the original one (i.e., more states or more transitions). One alternative way is to split the state only when silent actions c and a can be reduced to ϕ, as illustrated in situation c. A related question is that whether the public views generated by Algorithm 1 are "optimal." Note that the criteria of "optimal" public views may vary in different contexts and is subjective, such as having a minimal number of states, a minimal number of transitions, or a simplest structure. Our solution in Algorithm 1 is to apply all the possible deductions to simplify the public views.

Lines 14-21 describe the first two situations, that is, a and b. At Line 14, the function *canSimply* gives *true* if and only if a and cs can match Axioms R1, R11, R12, or R13; otherwise, it gives *false*. If cs is marked as ϕ (situation a), then Axiom R14 is used to merge cs and ps (Lines 15-17); else (situation b), remove the silent action $\tau_{c,r}$ and merge state ps and cs (Lines 19-21). Lines 23-37 describe the last three situations. For every incoming edge c, the algorithm checks whether c and a can be reduced according to Axioms R1-R10. The function *canCombine* on Line 24 gives $true$ if and only if c and a can match the actions on the left-hand side of an axiom of R2-R6, or $c = \tau_{c,r}$ and $a \in \{\tau_{c,r}, \tau_{nc,r}, \tau_{c,nr}, \tau_{nc,nr}\}$. The function num_of_outcoming edge(ps) on Line 25 calculates the number of outcoming edges in state ps, that is, the number of transition $(ps, a, ns) \in F_2$. If the state ps has only one outcoming edge (ps, a, cs) (situation d), the function *combine* on Line 26 returns the action on the right-hand side of the applied axiom (R2-R6) or a if $c = \tau_{c,r}$, and the algorithm replaces the edge c with a new edge d. Otherwise (situation e), the algorithm splits the state ps into two states (Lines 28-29) to facilitate the simplification of c and a (Lines 25-26) in the next round of the loop (Lines 13-40). On the other hand, if c and a can match the actions on the left-hand side of any axiom of R7-R10, as indicated by the function *violateAS* on Line 32 (situation c), the state s is marked as ϕ (Line 34) and all its outcoming edges are removed (Line 33). During the construction of the public view, if a noninitial state is found to have no incoming edges, it is removed from the public view (Lines 43-45).

The second algorithm is to compose two processes (or public views since they are also processes) into one global process. Given two processes $s_{10} = (s_{10}, S_1, A_1, F_1)$ and $s_{20} = (s_{20}, S_2, A_2, F_2)$. Let H_1 and H_2 be the sets of port actions of s_{10} and s_{20}, respectively, and GS be the global composition specification. $s = \oplus_{GS}(s_{10}, s_{20}) = (s, S, A, F)$.

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 3, 2009 at 09:09 from IEEE Xplore. Restrictions apply.

The algorithm for constructing s is shown as follows. This algorithm constructs s from its initial state, which is composed of the initial states of s_{10} and s_{20}. For every state of s, which is composed of two states of processes s_{10} and s_{20}, Lines 9-23 calculate the new states created by a transition from the current state of process s_{10} and Lines 24-31 calculate the new states created by the transition from the current state of process s_{20}. If the transition represents a communication, then Lines 10-16 create a new state by transferring current states of processes s_{10} and s_{20} to their respective target states. If the current states of both processes are the termination state 0, then the new state is also 0. If the current state of either process is marked as ϕ, then the new state is ϕ according to Axiom R15.

Algorithm 2. Compose two processes.
Input: processes $s_{10} = (s_{10}, S_1, A_1, F_1)$ and
$s_{20} = (s_{20}, S_2, A_2, F_2)$
Output: $s = \oplus_{\mathrm{GS}}(s_{10}, s_{20}) = (s, S, A, F)$

1. $S \leftarrow \{\}$, $A \leftarrow \{\}$, $F \leftarrow \{\}$.
2. ***if*** $s_{1\mathrm{0}} = \phi \vee s_{20} = \phi$,
3. $s \leftarrow \phi$, $S \leftarrow S + \{s\}$, quit.
4. ***else***
5. $s \leftarrow (s_{10}, s_{20}).stack.push(s)$. $S \leftarrow S + \{s\}$.
6. ***endif***
7. ***while*** *(not empty(stack))*
8. $cs = (s_{1\mathrm{i}}, s_{2\mathrm{j}}) \leftarrow stack.pop()$.
9. ***for*** *every* $(s_{1\mathrm{i}}, a, s_{1\mathrm{t}}) \in F_1$,
10. ***if*** $a \in H_1 \wedge \exists b \in H_2$, (a, b) *or* $(b, a) \in GS$
11. ***for*** *every* $(s_{2\mathrm{j}}, b, s_{2\mathrm{k}}) \in F_2$
12. $ns \leftarrow (s_{1\mathrm{t}}, s_{2\mathrm{k}})$, $c \leftarrow \gamma_{\mathrm{GS}}(a, b)$, $A \leftarrow A + \{c\}$, $F \leftarrow F + \{(cs, c, ns)\}$.
13. ***if*** $s_{1\mathrm{t}} = 0 \wedge s_{2\mathrm{k}} = 0$, $ns \leftarrow 0$, $S \leftarrow S + \{ns\}$, ***endif***
14. ***if*** $s_{1\mathrm{t}} = \phi \vee s_{2\mathrm{k}} = \phi$, $ns \leftarrow \phi$, $S \leftarrow S + \{ns\}$, ***endif***
15. ***if*** $ns \notin S$, $stack.push(ns)$, $S \leftarrow S + \{ns\}$, ***endif***
16. ***endfor***
17. ***else***
18. $ns \leftarrow (s_{1\mathrm{t}}, s_{2\mathrm{j}})$, $A \leftarrow A + \{a\}$, $F \leftarrow F + \{(cs, a, ns)\}$
19. ***if*** $s_{1\mathrm{t}} = 0 \wedge s_{2\mathrm{j}} = 0$, $ns \leftarrow 0$, $S \leftarrow S + \{ns\}$, ***endif***
20. ***if*** $s_{1\mathrm{t}} = \phi \vee s_{2\mathrm{j}} = \phi$, $ns \leftarrow \phi$, $S \leftarrow S + \{ns\}$, ***endif***
21. ***if*** $ns \notin S$, $stack.push(ns)$, $S \leftarrow S + \{ns\}$, ***endif***
22. ***endif***
23. ***endfor***
24. ***for*** *every* $(s_{2\mathrm{j}}, b, s_{2\mathrm{k}}) \in F_2$,
25. ***if*** $b \notin H_2 \vee \neg\exists a \in H_1$, (a, b) *or* $(b, a) \in GS$,
26. $ns \leftarrow (s_{1\mathrm{i}}, s_{2\mathrm{k}})$, $A \leftarrow A + \{b\}$, $F \leftarrow F + \{(cs, b, ns)\}$
27. ***if*** $s_{1\mathrm{i}} = 0 \wedge s_{2\mathrm{k}} = 0$, $ns \leftarrow 0$, $S \leftarrow S + \{ns\}$, ***endif***
28. ***if*** $s_{1\mathrm{i}} = \phi \vee s_{2\mathrm{k}} = \phi$, $ns \leftarrow \phi$, $S \leftarrow S + \{ns\}$, ***endif***
29. ***if*** $ns \notin S$, $stack.push(ns)$, $S \leftarrow S + \{ns\}$, ***endif***
30. ***endif***
31. ***endfor***
32. ***endwhile***

With Algorithms 1 and 2, one can construct and compose the public views of services. Algorithm 3 is to check whether a process satisfies the atomicity sphere. Given a process $p = (p, S, A, F)$, we use Algorithm 3 to mark and traverse the process to check its atomicity sphere. This algorithm first marks all the states that are unable to be rolled back (that is, some noncompensable task has been executed in some trace from the initial state to this state), and then checks if there is any nonretriable transition that is reachable from those states. If one exists, then the process violates the atomicity sphere; otherwise, the process satisfies the atomicity sphere. For example, as depicted in Fig. 6, tasks b and c in this process are noncompensable and nonretriable, respectively, and the other tasks are compensable and retriable. The first step of this algorithm is to mark states s_1, s_2, s_3, and s_4 as nonrollbackable because they are reachable from the noncompensable task b. Note that these states also store action b as the reason of being nonrollbackable. Then, the algorithm searches all the transitions reachable from those nonrollbackable states and finds that a nonretriable task (task c) is reachable from the nonrollbackable state s_1. As a result, the algorithm reports the atomicity sphere violation for this process, and the potential violation-causing reasons (that is, the noncompensable and the nonretriable action pair). The detail of this algorithm is given as follows:

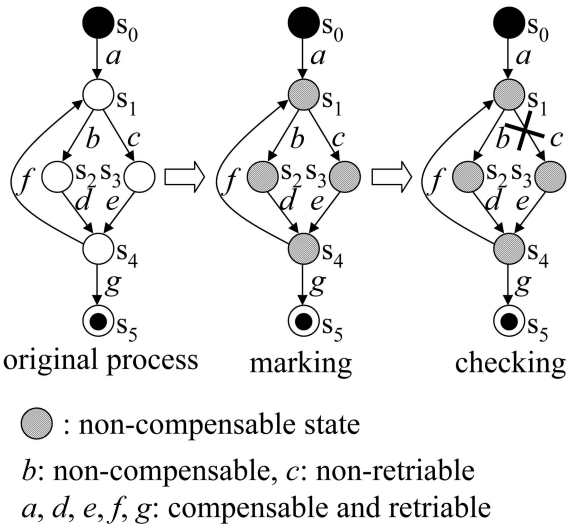

Fig. 6. Algorithm for checking the atomicity sphere (an example).

Algorithm 3. Check the atomicity sphere.
Input: process $p = (p, S, A, F)$
Output: true or false, and if false, also output the violation action pairs

1. $\forall s_{\mathrm{i}} \in S$, $rb(s_i) \leftarrow true$
2. $markrb()$
3. $\forall s_{\mathrm{i}} \in S$, $visited(s_{\mathrm{i}}) \leftarrow false$.
4. $check(p)$.

$markrb()$:

5. ***for*** *every* $a \in NC(A)$,
6. $\forall s_{\mathrm{i}} \in S$, $visited(s_{\mathrm{i}}) \leftarrow false$.
7. ***for*** *every* $(s_1, a, s_2) \in F$,
8. $rb(s_2) \leftarrow false$, $hashtable.put(s_2, a)$
9. ***if*** $visited(s_2) = false$
10. $markreachable(s_2, a)$.
11. ***endif***
12. ***endfor***
13. ***endfor***

$markreachable(s, b)$:

14. $visited(s) \leftarrow true$

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 3, 2009 at 09:09 from IEEE Xplore. Restrictions apply.

15. ***for*** *every* $(s, a, ns) \in F$,
16. $rb(ns) \leftarrow false$, $hashtable.put(ns, b)$
17. ***if*** $visited(ns) = false$
18. $markreachable(ns, b)$.
19. ***endif***
20. ***endfor***

$check(s)$:

21. $stack.push(s)$
22. ***while*** *(not empty(stack))*
23. $s \leftarrow stack.pop()$.
24. ***if*** $s = \phi$, *report violation (s), quit,* ***else*** $visited(s) \leftarrow true$ ***endif***
25. ***for*** *every* $(s, a, ns) \in F$,
26. ***if*** $rb(s) = false \wedge R(a) = false$,
27. ***for*** *every* $b = hashtable.get(s)$, *report violation(b, a).* ***endfor***
28. ***else***
29. ***if*** $visited(ns) = false$, *stack.push(ns),* ***endif***.
30. ***endif***
31. ***endfor***
32. ***endwhile***

In the algorithm, Line 1 initializes all states. $rb(s_i)$ is a predicate which returns *true* for rollbackable states (rb for short). This kind of state implies that all the executed tasks from the initial state to this state are compensable; otherwise, the predicate returns *false.* At Line 2, the function *markrb* marks all the states that are unable to be rolled back, and records the actions that make them that way. Line 3 resets the visited flag of all states. At Line 4, the function *check* checks whether 1) there is any nonretriable transition leading from any state (of the process), which is not rollbackable or 2) any state of the process has been marked as ϕ. If either condition is satisfied, then the process does not satisfy the atomicity sphere. Then, the algorithm reports the atomicity sphere violation as well as the potential reasons (Lines 24 and 27). Note that $NC(A)$ on Line 5 represents the set of noncompensable actions in action set A. The hashtable on Line 8 is used to store the actions that cause the state nonrollbackable. These actions are used to generate the reason for atomicity sphere violation (Line 27).

2.6 Deployment

As mentioned in Section 1, atomicity analysis for service composition across organizations is difficult when service providers are not willing to disclose the details of their private processes. Our approach provides a solution to this problem: by deriving atomicity-equivalent public views, organizations could check whether their service composition satisfies the atomicity sphere without disclosing the details of their private processes. In this section, we demonstrate how to apply our approach to this problem. The flow of applying our approach is depicted in Fig. 7.

Step 1 (extract processes). In this step, service providers model processes from the application specifications. Note that in practice, service providers may have modeled their processes in other formats (e.g., BPEL or UML activity diagrams). Then, they could transform the processes from other formats into our model. The process transformation from BPEL into our model is further discussed in Section 4.

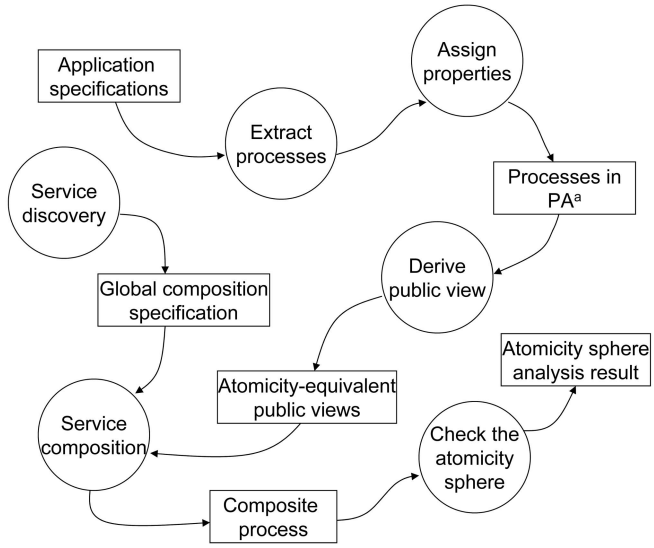

Fig. 7. Deployment flow of our approach.

Step 2 (assign properties). Service providers need to assign the compensability and retriability properties to each task in their processes. After that, services are modeled as processes in the PA^{a} framework. Note that assigning the two properties to tasks depends on many factors (e.g., business policies, service implementation, and so on), which are discussed in Section 4.

Step 3 (derive public view). Service providers apply Algorithm 1 in Section 2.5 to construct the atomicity-equivalent public view from the processes achieved in previous step. To facilitate the construction of atomicity-equivalent public views and the global analysis of atomicity sphere, we design a tool that implements Algorithms 1-3 in Section 2.5. (The details about how to use the tool could be found in [4].) Service providers can then publish their atomicity-equivalent public views to the public service registry.

Step 4 (service discovery). By looking up the service registries, service consumers identify potential services for composition, and make agreements with service providers on the global composition specifications. Note that the achievement of the global composition specifications could be done by keyword matching, service semantics, or negotiation between service providers and service consumers. Further discussions on the global composition specification are in Section 4.

Step 5 (service composition). With the global composition specification and atomicity-equivalent public views, service consumers could compose these public views into a composite process using Algorithm 2.

Step 6 (check the atomicity sphere). Finally, service consumers use Algorithm 3 to check whether this composite process satisfies the atomicity sphere or not. According to Theorem 3, the potential service composition satisfies the atomicity sphere if and only if this composite process satisfies the atomicity sphere.

In the next section, we will illustrate the deployment of our approach by using two case studies.

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 3, 2009 at 09:09 from IEEE Xplore. Restrictions apply.

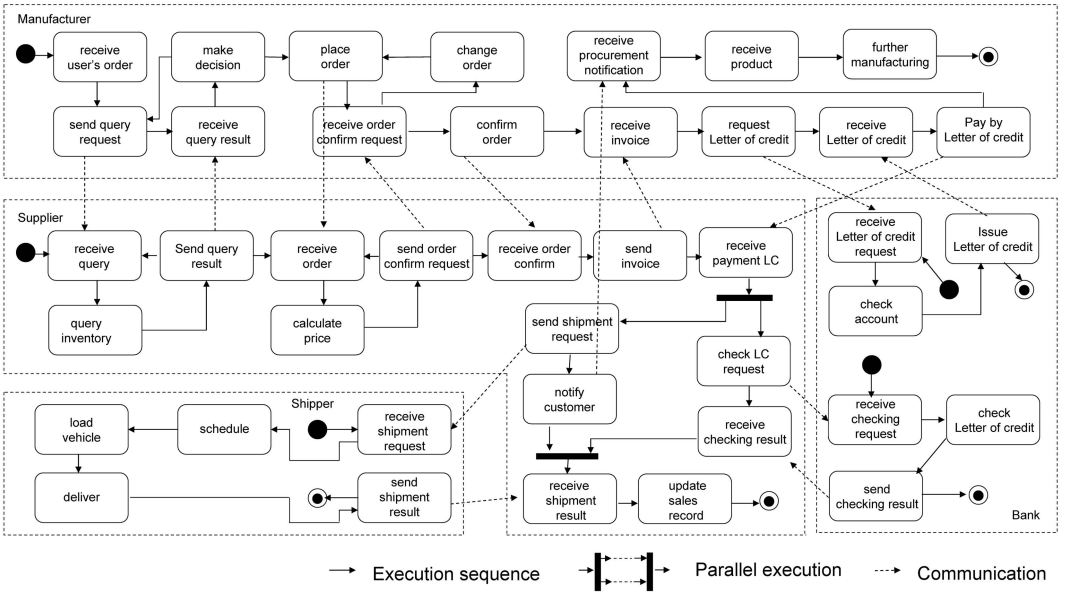

Fig. 8. Process collaboration in the supply chain application.

3 Evaluation

In this section, we illustrate our approach by two case studies. Then, we analyze the time complexity of the algorithms introduced in Section 2.5 to evaluate their feasibility in practice.

3.1 Case Studies

We have presented the process algebraic framework that derives the atomicity-equivalent public views from private processes in Section 2. In this section, we evaluate the applicability of our approach by two case studies. The first one is from a supply chain application domain and the second one is from an insurance application domain. Through the case studies, we aim at answering the following two research questions:

- R1. How to apply our approach to service-oriented applications?
- R2. To what degree our approach outperforms (in terms of accuracy) existing public view generation approaches in atomicity analysis for service compositions?

We use a replicated product design approach, which means that we replicate these two existing applications from their specifications, and then deploy these two applications to study these two questions.[9]

Our approach consists of setting up task properties from private processes, translating the task properties into the public views followed by the construction of a global composition specification, and checking the atomicity of the global service composition. To investigate (R1), we also study the following subquestions: (R1-1) How to extract the processes from the service specifications? (R1-2) How to assign the compensability and retriability properties to tasks? (R1-3) How to obtain a global composition specification? To investigate (R2), we summarize existing approaches to generating public views and compare the accuracy of atomicity analysis using these public views with that using our atomicity-equivalent public views.

We first present the application descriptions. After that, we revisit the problem setting in our framework and observe from the case studies to answer the above research questions.

3.1.1 Application Descriptions

Supply chain application. This application is modeled after an online service provided by a gift manufacturer [51]. The customers can use the service to design gifts with specific requirements and place orders for the designed gifts. Fig. 8 depicts the target business scenario.

To offer the service, the manufacturer needs to find three other collaborators, that is, a supplier, a bank, and a shipper, from an online e-commerce market (e.g., Alibaba [3]), where merchants publish their services for potential collaboration. The shipper is responsible to arrange the shipment of the OEM products from the supplier to the manufacturer for further processing. The payment for the purchase of OEM products and the shipping fee is made through the bank using the Letter of Credit (LC for short).

A normal service invocation is as follows: The manufacturer starts the above collaboration after it has received a customer's purchase order. The manufacturer first queries the supplier's stock availability and the prices of specified OEM products, and then places its order to the supplier. After calculating the prices of purchased products and shipping fees, the supplier sends an invoice to the manufacturer. The manufacturer thus requests an LC from the bank to settle the invoice. After receiving the LC from the manufacturer, the supplier checks with the bank to confirm the validity of the LC. Meanwhile, the supplier sends a request to the shipper to arrange the shipment of the products, and informs the manufacturer about the estimated arrival time of the products. On receiving the products, the manufacturer then

9. The reason we use a replicated product design approach is that the companies are not willing to provide the detailed implementation of their applications due to commercial confidentiality.

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 3, 2009 at 09:09 from IEEE Xplore. Restrictions apply.

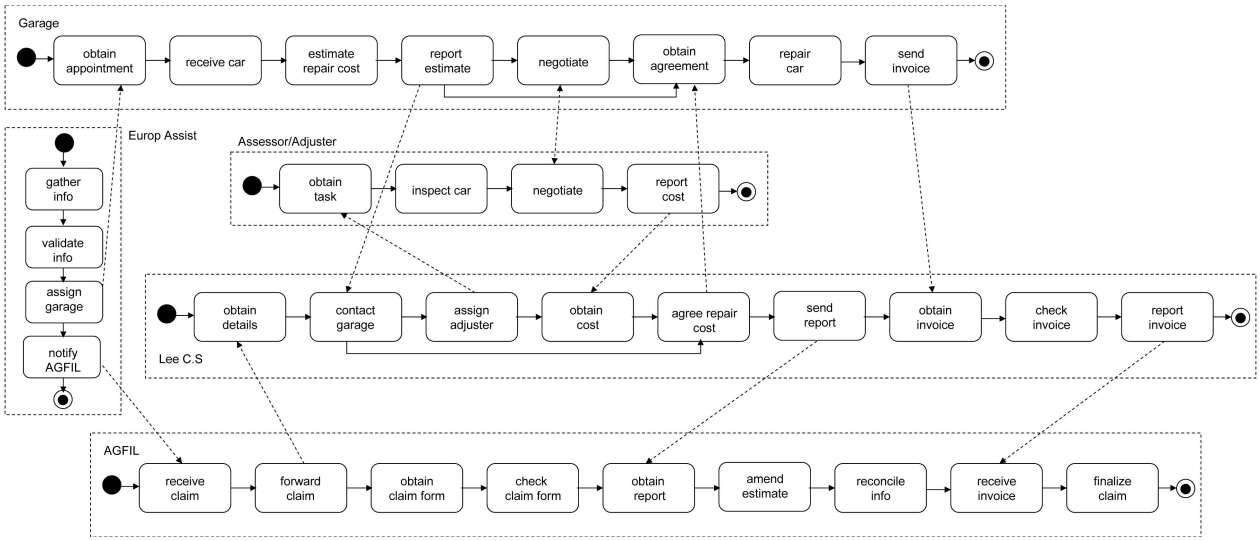

Fig. 9. Process collaboration for insurance.

processes the products to produce gifts according to the customer's specific requirements.

We note that the "Manufacturer," the "Supplier," the "Bank," and the "Shipper" are all Web services. Each of these services is driven by their corresponding backend process, which is nonobservable externally (and among collaborators).

Insurance application. This application is from a real-life insurance example (motor damage claims) [21]. AGF Irish Life Holding Plc. (AGFIL for short) is an insurance company operating within the Irish market providing domestic and commercial insurance. AGFIL collaborates with the other partners through a scheme known as the Emergency Service, under which motor policyholders could claim the repair costs of their insurance-covered motor vehicle from the partners of AGFIL. The objective is to provide efficient and cost-effective damage claim services for customers.

In this scenario, Europ Assist offers a 24-h high-quality emergency service hotline to individual motor vehicle policyholders. Lee C.S coordinates and manages the actual operations of the emergency service for AGFIL. It assigns Assessors/Adjusters to inspect the claimed damage to a motor vehicle and assess the repair price to ensure them not deviating from the industrial norms. Certain garages provide repair services to policyholders, who do not need to pay the repair fee directly. Instead, the garages forward the invoices to Lee C.S and receive the payment from AGFIL. AGFIL is responsible for underwriting the policies and covering the claimed losses, but does not take part in repair fee negotiations. At the same time, it has the final rights to decide whether a claim is faithful and whether any payment should be released to a claim. On identifying an invalid claim (e.g., the loss is not covered in the policy terms), AGFIL will stop the claim procedure immediately.

Their collaboration scenario is shown in Fig. 9. At first, a motor vehicle insurance policyholder calls Europ Assist to initiate a damage claim procedure. Europ Assist then submits the claim form to AGFIL, who also forwards a copy to Lee C.S. Upon receiving the claim form, Lee C.S appoints a loss assessor/adjuster to inspect the claimed vehicle and negotiate the repair price with the garage. Meanwhile, the AGFIL claim handler checks the claim manually. If Lee C.S approves the quoted repair price with a chosen garage, the garage will commence the repair work. Lee C.S will then send AGFIL an adjuster/assessor's report. Finally, AGFIL pays the involved garage and assessors/adjuster by a third-party payment system.

3.1.2 Problems Statement

Supply chain application. In our study, we observe that it is unsuitable to enforce traditional database-transaction approaches to this application to guarantee application consistency. This is because the services in this scenario are typically long running. Applying traditional database transactions to such application will compromise the performance of involved service providers. For example, to guarantee available products for this transaction, traditional database transactions add locks to the supplier process to prevent the products from being purchased by others. These locks may be in place from the moment the manufacturer makes the query until the transaction is completed or canceled. Since the manufacturer may hold the products pending for a purchase decision, others could not purchase the locked products from the supplier during this pending period. Therefore, this limits the potential concurrent execution rate of the supplier process as well as the interests of the merchant.

Exception handling is a way to trade off the inconsistency and performance problems in this application. If the products are out of stock during the collaboration, an exception handler can be invoked to abort their collaboration, or to suspend the transaction until the product has been replenished and then resume the transaction. In this way, locking of products can be avoided.

Nonetheless, we observe that in some scenarios, such an exception handling strategy may lead to undesirable consequences. For example, if the supplier finds that the LC received from the manufacturer is invalid, it will raise an exception to abort the collaboration (due to its business

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 3, 2009 at 09:09 from IEEE Xplore. Restrictions apply.

policy). However, the shipper may have already shipped the products. As such, the supplier or the manufacturer should bear the transportation cost for this aborted transaction. This may be undesirable to the supplier or manufacturer.

One solution is to detect and prevent such potentially undesirable scenarios in advance before their collaboration commences. This can be achieved by a global analysis of the atomicity sphere in the composition of these five services in the service look-up stage. For example, task "*deliver*" in the shipper process is noncompensable because of the high shipping fee. Task "*receive checking result*" in the supplier is nonretriable, because its exception handler will abort the process if the LC is invalid. From the perspective of the global process (regarding these five processes as a whole), the nonretriable task "*receive checking result*" is executed after the noncompensable task "*deliver*" in some complete trace of the global process. Therefore, the manufacturer identifies in advance that this service composition violates the atomicity sphere, and may look for collaborating with other services (e.g., choosing another shipping service) instead. However, from the perspective of the manufacturer, the details of the other four processes are nonobservable externally. Instead, these service providers release only restricted public views to their service consumers. Therefore, such a global atomicity analysis is difficult.

Insurance application. The composition of these services in this application provides an efficient damage claim service for policyholders. However, these organizations will suffer application inconsistency in some scenarios. For example, if the damage claim is invalid (e.g., the loss is not covered in the policy terms), AGFIL will notify Lee C.S and stop the service for the concerned policyholder. However, the garage may have already repaired the vehicle. In such a case, AGFIL or Lee C.S should pay the garage the repair charges, and thus incur losses.

We note that, although existing transactional protocols for web services (e.g., BTP [10]) may be applicable to solve the above problem by requiring all the involved services to commit tasks using a two-phase commit protocol, yet such a solution will not free a service until all services are committed, and thus compromises the efficiency of the claim process. This defies the objective of using a service-orientation approach in the application, which is to conduct the damage claim in a flexible, efficient, and cost-effective manner. Moreover, this also reveals the hidden details of the business processes to the partners, which may be unacceptable due to business reasons or privacy concerns.

Instead, exception handling is applied to this application to handle application inconsistencies [21]. In order to avoid undesirable consequences during the error recovery (e.g., AGFIL has to pay the garage for an invalid damage claim), the service composition should be atomic. A way is to conduct a global analysis of the atomicity sphere for their collaboration in advance to prevent such undesirable scenarios. However, service providers may not be willing to expose the details of their private processes to their customers. Therefore, such a global atomicity analysis is difficult.

3.1.3 Deployment of Our Approach

In this section, we report the deployment of our approach on these two applications.

Deployment instance 1 (supply chain application). In this section, we illustrate how to apply our approach to analyze the atomicity sphere of the service composition in this application. Let us take the shipper process as an example to explain the procedure for deriving its atomicity-equivalent public view.

Step 1 (extract processes, R1-1). In this step, the shipper models its process from the application specification, as depicted in Fig. 8. For ease of presentation, we make the following naming conventions: a_1, a_2, a_3, a_4, and a_5 represent tasks "*receive shipment request*," "*schedule*," "*load vehicle*," "*deliver*," and "*send shipment result*," respectively. The shipper process could then be represented as $a_1 \cdot a_2 \cdot a_3 \cdot a_4 \cdot a_5 \cdot 0$.

Step 2 (assign properties, R1-2). The shipper needs to assign the compensability and retriability properties to each task in its process. Note that the internal task "*schedule*" is assigned as compensable (because this task has no side effect) and nonretriable (since the shipper could not always guarantee available scheduling). Likewise, task "*deliver*" is marked as noncompensable (because the shipper will not refund the delivery fee) and retriable (since the shipper promises to deliver the products to the customers once confirming the schedule). All the other tasks in its process are marked as compensable and retriable, because these tasks have no side effects (or their side effects can be compensated) and can always be retried to guarantee successful execution.

Step 3 (derive public view). The shipper applies Algorithm 1 to construct the atomicity-equivalent public view. The shipper could use our provided tool [4] to facilitate the construction. We illustrate in the following how the public view is constructed. In the first step of Algorithm 1 (Lines 2-8), a_2, a_3, and a_4 are renamed as $\tau_{\mathrm{c,nr}}$, $\tau_{\mathrm{c,r}}$, and $\tau_{\mathrm{nc,r}}$, respectively. So the public view could be represented as $a_1 \cdot \tau_{\mathrm{c,nr}} \cdot \tau_{\mathrm{c,r}} \cdot \tau_{\mathrm{nc,r}} \cdot a_5 \cdot 0$. In the second step, Algorithm 1 simplifies this public view using the reduction rules: First, $\tau_{\mathrm{c,r}} \cdot \tau_{\mathrm{nc,r}} \cdot a_5 \cdot 0$ is reduced to $\tau_{\mathrm{nc,r}} \cdot a_5 \cdot 0$ (Axiom R1, Lines 19-20), and the public view is represented as $a_1 \cdot \tau_{\mathrm{c,nr}} \cdot \tau_{\mathrm{nc,r}} \cdot a_5 \cdot 0$. Next, $\tau_{\mathrm{c,nr}} \cdot \tau_{\mathrm{nc,r}} \cdot a_5 \cdot 0$ is reduced to $\tau_{\mathrm{nc,nr}} \cdot a_5 \cdot 0$ (Axiom R3, Lines 25-26), and we have the final public view as $a_1 \cdot \tau_{\mathrm{nc,nr}} \cdot a_5 \cdot 0$, as depicted in Fig. 10d.

Note that other service providers can construct their atomicity-equivalent public views in a similar way. These public views are depicted in Fig. 10. (Note that the state model of these public views and their composition is too large to be presented in this paper, we therefore use the simplified representation. Unless otherwise stated, other actions are assumed to be marked as compensable and retriable.)

Step 4 (service discovery, R1-3). By looking up the services from the service registries (e.g., Alibaba [3]), the manufacturer identifies the shipper, the bank, and the supplier as collaborators. Then, they make an agreement on the global specification of their service composition through negotiation. For example, the global specification defines that task "*place order*" in the manufacturer process communicates with task "*receive order*" in the supplier process. The

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 3, 2009 at 09:09 from IEEE Xplore. Restrictions apply.

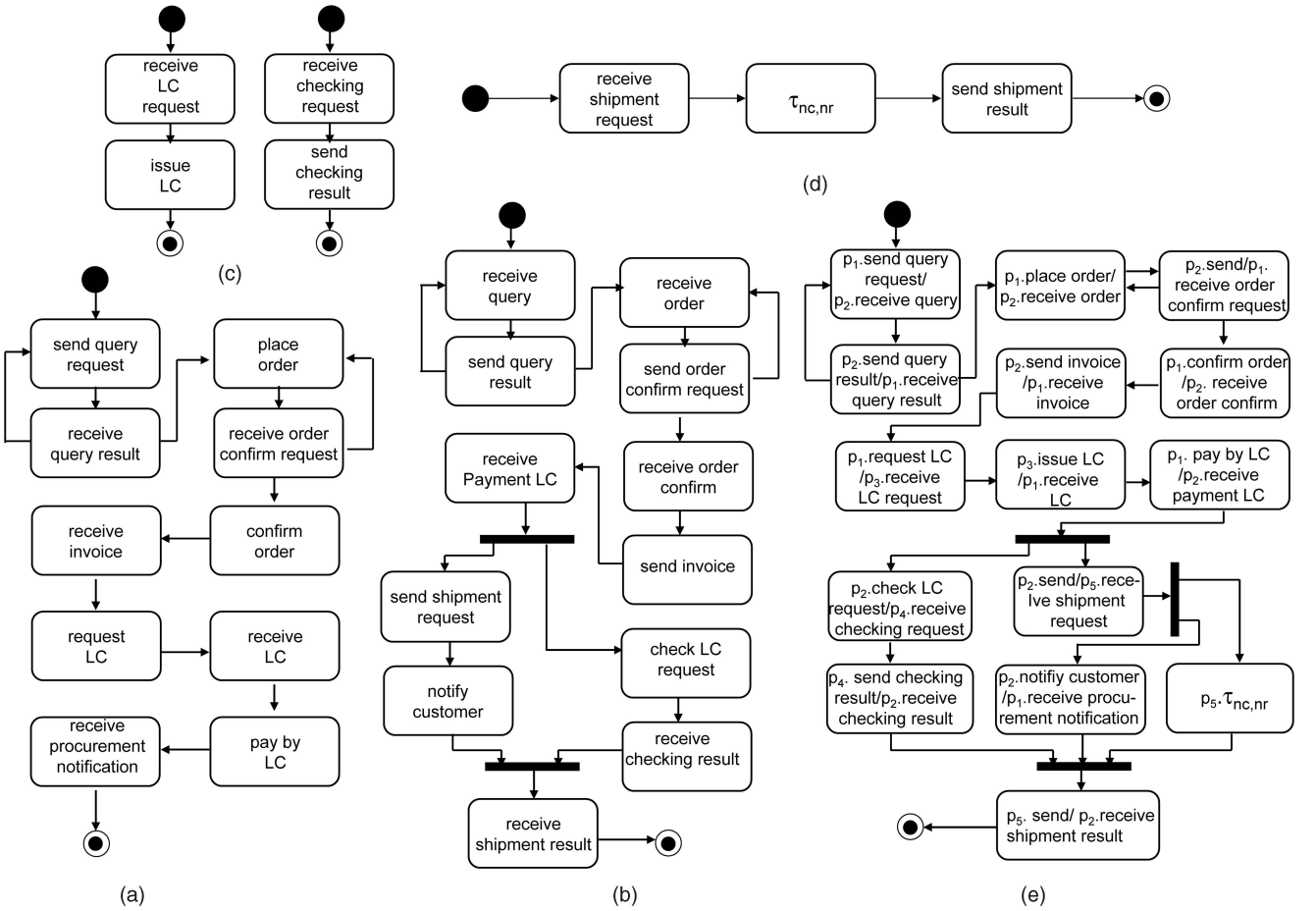

Fig. 10. Public views and their composition in the supply chain application. (a) Manufacturer. (b) Supplier. (c) Bank. (d) Shipper. (e) Composition.

other communicating port action pairs are listed in Fig. 8 and thus omitted here.

Step 5 (service composition). The manufacturer then composes the atomicity-equivalent public views of these services into a global process under the global specification using Algorithm 2. For ease of presentation, we make the following naming conventions in the global process composed of these public views: the synchronized port action is renamed to "*prefix.action/prefix.action*." Prefix p_i represents the ID of the process and p_1, p_2, p_3, p_4, and p_5 represent manufacturer, supplier, bank LC request, bank LC checking, and shipper processes, respectively. Meanwhile, according to the global composition operator $\oplus_{\mathrm{GS}}$, the compensability and retriability properties of the synchronized port actions are calculated by the logical "and" operation of the two properties from the corresponding communicating port actions. The final composed global process is shown in Fig. 10e.

Step 6 (check the atomicity sphere). By applying our tool (Algorithm 3), the manufacturer observes that this global process violates the atomicity sphere because the nonretriable task "p_4.*send checking result*/p_2.*receive checking result*" (since "p_2.*receive checking result*" is nonretriable) is executed after task "p_4. $\tau_{\mathrm{nc,nr}}$" in some complete trace of this process.

Based on the analysis result, the manufacturer identifies potential nonatomic scenarios in advance before commencing their collaboration. To avoid these scenarios, the manufacturer may try to find another supplier or shipping service that has no such nonatomic scenario. Alternatively, organizations may just modify their private processes to avoid such a scenario (e.g., the supplier may avoid aborting the collaboration after receiving the exception from the banker). Another approach would be adding extra synchronization to involved services to preserve the atomicity sphere.

Deployment instance 2 (insurance application). Similarly, to apply our approach, each organization needs to use Algorithm 1 to derive its atomicity-equivalent public view from its private process (e.g., Fig. 11a shows the public view of Garage) and then publishes its public view in some public registry. The detailed procedure of applying our approach to this example is similar to that in the supply chain application, and is thus outlined here.

Note that task "*check claim form*" in the AGFIL process is nonretriable because it raises an exception if the claim is invalid, and then aborts the process. Therefore, this task is renamed as the silent action $\tau_{\mathrm{c,nr}}$ in its public view. Task "*repair car*" in the Garage process is renamed as the silent action $\tau_{\mathrm{nc,r}}$ because it is noncompensable (since the repair of a car takes time and consumes other resources). The other tasks are all compensable and retriable. By applying our provided tool to compose their atomicity-equivalent public views into a global process and check the atomicity sphere of this global process, we observe that this global process violates the atomicity sphere because this particular nonretriable task $\tau_{\mathrm{c,nr}}$ (representing "*check claim form*") is

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 3, 2009 at 09:09 from IEEE Xplore. Restrictions apply.

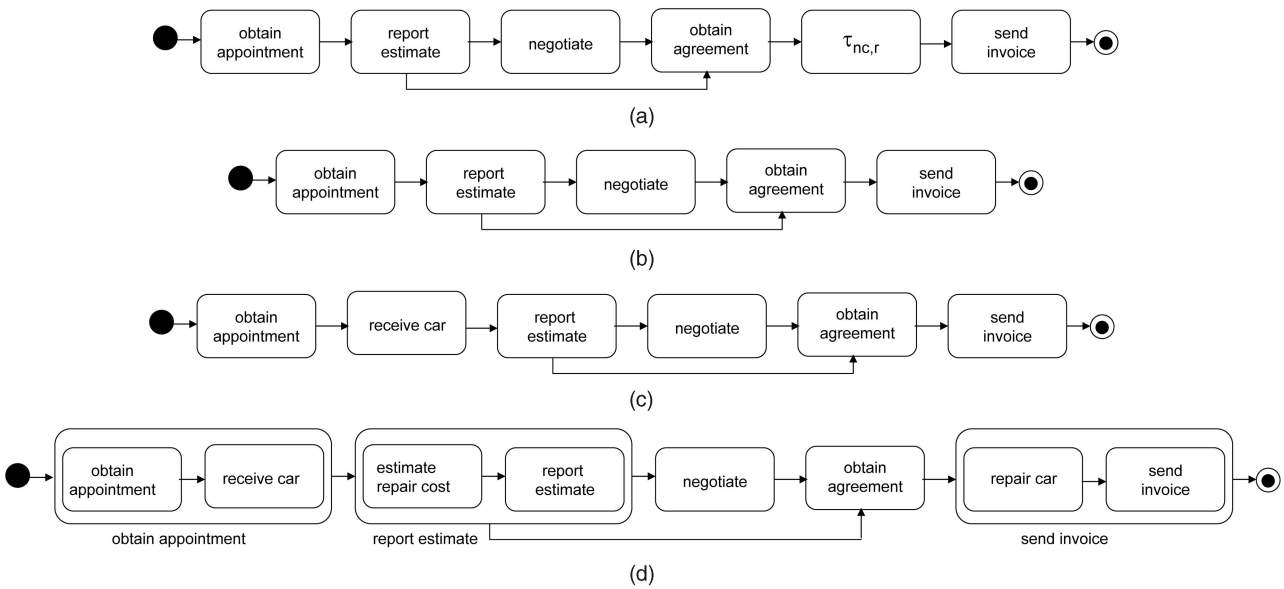

Fig. 11. Public views of Garage generated by different approaches. (a) Atomicity-equivalent public view of Garage. (b) Public view of Garage generated by projection approach. (c) Public view of Garage generated by projection approach with one internal action. (d) Public view of Garage generated by abstraction approach.

executed after the noncompensable task $\tau_{\mathrm{nc,r}}$ (representing "*repair car*"). Therefore, according to Theorem 3, this application violates the atomicity sphere.

3.1.4 Comparison with Existing Work

In practice, organizations would like to expose only restricted public views of their private processes to service consumers. Various approaches have been proposed to derive public views for service compositions. In this section, we summarize existing public view generation approaches and then apply them to generating public views from private processes. We compare the accuracy of atomicity analysis for service compositions using public views generated by these approaches with that of using our approach. The purpose is to answer the research question R2.

Roughly speaking, existing public view generation approaches [1], [2], [11], [12], [22], [23], [24], [25], [49], [59] could be classified into two categories: a projection approach and an abstraction approach. A projection approach hides internal actions and exposes only port actions in a public view. One representative projection approach is proposed by van der Aalst and Basten [2] which is based on the concept of inheritance. In contrast, an abstraction approach combines port actions and internal actions into composite actions in corresponding public views. One representative abstraction approach is proposed by Liu and Shen [49].

Projection approach. Let us first apply the projection approach to generating the public view for each private process. Public views generated by this approach contain only the port actions, and all the internal actions in the private processes are removed. Let us use the insurance application as an example. To generate the public view for Garage, internal actions "*receive car*," "*estimate repair cost*," and "*repair car*" are removed, and the public view keeps only port actions, as depicted in Fig. 11b. The public views of the other private processes could be derived in similar ways and are omitted here. In our study, we found that the atomicity analysis based on such public views is inaccurate: The analysis result shows that this application satisfies the atomicity sphere. However, this application violates the atomicity sphere instead.

The reason for such an approach to be inaccurate is that the public views generated in this way miss some information about internal actions. One may thus consider applying a simple solution by enhancing the public view generation approach with some internal actions inserted into the public views.[10]

For example, one may construct three different public views for the Garage process with one extra internal action in the public views. Fig. 11c shows the public view that exposes only the internal action "*receive car*." The other two public views could be derived similarly.

In this study, we also apply the above illustrated approach to constructing all the public views incrementally by hiding from 100 percent to 0 percent of internal actions from the public views. The numbers of public views generated for each private process with different percentages of internal action hidden from the public views in the two case studies are depicted in Tables 2 and 3. In both tables, the first row describes the percentage of internal actions to be hidden from the public views. Each column describes the number of public views that could be generated for different private processes with the percentage of internal actions hidden from the public views.

Then, we compose these public views to check the atomicity sphere for the application. We check the number of accurate analyses and calculate the accuracy rate (i.e., the number of accurate analyses against the total number of analyses). According to our investigation with the total details of private processes in these two applications, both of them violate the atomicity sphere. Therefore, such an

10. Note that such insertion should keep the public view consistent with the process under the inheritance relations [2].

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 3, 2009 at 09:09 from IEEE Xplore. Restrictions apply.

TABLE 2
Accuracy of Projection Approach for Supply Chain Application

	100%	80%	60%	40%	20%	0%
#Manufacturer	1	4	6	4	1	1
#Supplier	1	3	3	3	1	1
#Shipper	1	3	3	3	1	1
#Bank. S1	1	1	1	1	1	1
#Bank. S2	1	1	1	1	1	1
#Compositions	1	36	54	36	1	1
#Accurate Analyses	0	12	36	24	1	1
Accuracy Rate	0%	33.3%	66.7%	66.7%	100%	100%

TABLE 3
Accuracy of Projection Approach for Insurance Application

	100%	80%	60%	40%	20%	0%
#Europ Assist	1	2	2	1	1	1
#Garage	1	3	3	3	1	1
#Assessor/Adjuster	1	1	1	1	1	1
#Lee C.S	1	1	1	1	1	1
#AGFIL	1	5	10	10	5	1
#Compositions	1	30	60	30	5	1
#Accurate Analyses	0	2	16	12	4	1
Accuracy Rate	0%	6.7%	26.7%	40%	80%	100%

analysis is accurate if and only if the analysis result reports the atomicity sphere violation. From Tables 2 and 3, we observe that the more the amounts of internal actions are hidden from the public views, the less accurate the approach is. For the last column of both tables, the accuracy rate of the analyses is 100 percent, but the cost is to expose all the details of the private processes, which is unacceptable to service providers. In contrast, each public view generated by our approach in this study contains no more than one silent action for both applications, but the analysis result based on our approach is accurate. Therefore, our approach outperforms this simple approach in atomicity analysis for this application.

Abstraction approach. This approach derives public views from private processes by abstracting internal actions and port actions as composite actions in public views. Noting that such abstraction preserves the partial order of actions in the private processes (i.e., for any two actions a and b in a private process, let a' and b' be the resulting composite actions that abstract a and b in the public view, respectively. If a is executed before b in the private process in a trace, then b' should not be executed before a' in the corresponding trace in the public view). For example, as depicted in Fig. 11d, internal action "*receive car*" in the Garage process could be abstracted together with port action "*obtain appointment*," whereas internal action "*estimate repair cost*" is abstracted together with port action "*report estimate*." Another possible abstraction is to abstract both internal actions "*receive car*" and "*estimate repair cost*" together with port action "*report estimate*." Note that the abstraction to compose "*receive car*" and "*report estimate*" into a composite action and to compose "*estimate report cost*" and "*obtain appointment*" into another composite action is invalid because such an abstraction approach does not preserve the partial order of these two internal actions.

To check the atomicity sphere for service compositions using public views generated in this way, one may want to apply a simple solution by abstracting the compensability

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 3, 2009 at 09:09 from IEEE Xplore. Restrictions apply.

TABLE 4
Accuracy of Abstraction Approach for Supply Chain Application

#Manufacturer	#Supplier	#Shipper	#Bank S1	#Bank S2	#Compositions	#Accurate Analyses	Accuracy Rate
18	8	4	2	2	2304	576	25%

TABLE 5
Accuracy of Abstraction Approach for Insurance Application

#Europ Assist	#Garage	#Assessor/ Adjuster	#Lee C.S	#AGFIL	#Compositions	#Accurate Analyses	Accuracy Rate
3	6	2	2	18	1296	432	33.3%

and retriability properties of internal actions into the composite actions. The abstracted composite action is noncompensable (nonretriable) if and only if at least one of the internal actions for this composite action is noncompensable (nonretriable).

In this study, we generate all the possible public views using this approach for each private process. The numbers of public views generated for each private process for both applications are shown in Tables 4 and 5. We then compose these public views to check the atomicity sphere for both applications. For example, using the above procedure, we obtain 1,296 service compositions in total, but only 432 analysis results are accurate for the insurance application, as depicted in Table 5.

In contrast to this approach, our approach abstracts only useful atomicity information of internal actions and exposes them as silent actions in the public views. The deployment of our approach to both applications explores that both of them violate the atomicity sphere. Therefore, our approach outperforms such an abstraction approach in atomicity analysis for this application in terms of accuracy.

From this study, we observe that existing public view generation approaches are inapplicable because the analysis results based on these approaches are inaccurate and misleading.

3.1.5 Threats to Validity

We investigate the applicability of our approach by two case studies. One major concern is whether the conclusion drawn in our studies could be generalized to other service-oriented applications. Therefore, in our case studies, we choose two representative applications in service-oriented application domains. Similar applications are also adopted in the literature (e.g., [12], [13], [21], [22], [24], [34], [35], [42], [43], [52], [53]). Both applications share common characteristics with other service-oriented applications. That is, applications are composed of services from different service providers. These services are usually long running, distributed, heterogeneous (e.g., different service providers may use different infrastructures), and autonomous. Therefore, our case studies could be generalized to other service-oriented applications.

3.1.6 Experience

We learn from the studies that sometimes assigning the compensability and retriability properties to a task correctly may be not easy because it may depend on many factors (e.g., the implementation of a service, the business policies, and so on). On the other hand, the atomicity analysis result is sensitive to these two properties. For example, if the noncompensable task "*repair car*" is marked as compensable, the analysis shows that the service composition in the insurance application satisfies the atomicity sphere. Such result is misleading and organizations may thus incur some penalties in the collaboration due to the atomicity violation at runtime. Therefore, further efforts are needed to help service providers to assign and validate the properties correctly in a systematic way.

From the studies, we also find that the atomicity analysis based on the public views that combine internal actions with their nearby port actions does not always report accurate results. The reason is that such combination propagates the compensability and retriability properties of internal actions to the port actions in the public view, and may thus lose a partial order relationship between some tasks in the original private processes. Therefore, our process algebra framework PA^{a} does not introduce extra axioms to simplify the public views by combining the silent actions and port actions in this way. One may ask follow-up questions: When exposing such silent actions is unacceptable to a service provider, is it possible to analyze the atomicity sphere correctly? And how? The experience gained in the studies shows that the trivial property propagation from internal actions to nearby port actions does not work. We will investigate this issue and further generalize the applicability of our framework.

3.2 Complexity Analysis

3.2.1 Time Cost for Constructing a Public View (Algorithm 1)

Let A be the set of actions of private process p, S be the set of states of private process p, and F be the set of transitions. Let $|S| = m$, $|F| = k$. In the first step (Lines 2-8), since we rename the actions in all transitions of p, the time cost is $O(k)$. In the second step, we want to simplify the process. We only need to traverse the process using the BFS

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 3, 2009 at 09:09 from IEEE Xplore. Restrictions apply.

algorithm from the termination state back to the initial state, so all the states are visited once. At each state, we at most combine its every incoming transition with its outcome transition once to simplify the public view (Lines 11-41). Therefore, in the worst case, there are at most totally k^2 steps of simplification. As a result, the total time cost for constructing a public view is $O(k + k^2)$. Since $1 \leq k$, the time cost is $O(k^2)$.

3.2.2 Time Cost for Composing Two Processes (Algorithm 2)

Let $p_1 = (s_{10}, S_1, A_1, F_1)$, and $p_2 = (s_{20}, S_2, A_2, F_2)$. $|S_1| = m_1$, $|S_2| = m_2$, $|F_1| = k_1$, $|F_2| = k_2$. Since the process $\oplus_{\mathrm{GS}}(p_1, p_2)$ has at most $m_1 \times m_2$ states, the time cost for generating states is $O(m_1 \times m_2)$. In the newly constructed process $\oplus_{\mathrm{GS}}(p_1, p_2)$, each transition in process p_1 may exist at most m_2 times, and similarly each transition in process p_2 may exist at most m_1 times. Hence, the total number of transitions in the new process would be at most $m_1 \times k_2 + m_2 \times k_1$. As a result, the time cost for calculating transitions of the new process (Lines 9-31) is $O(m_1 \times k_2 + m_2 \times k_1)$. So, the total time cost for composition is $O(m_1 \times m_2 + m_1 \times k_2 + m_2 \times k_1)$. Since $m_1 \leq k_1 + 1$ and $m_2 \leq k_2 + 1$, so the time cost is $O(k_1 \times k_2)$.

3.2.3 Time Cost for Checking the Atomicity Sphere (Algorithm 3)

Let $|S| = m$, $|F| = k$, and $|A| = n$. The time cost of step 1 is $O(m)$. In step 2, function *markrb* traverses all the states at most n times, and thus the time cost is $O(nm)$. The time cost of step 3 is $O(m)$. In step 4, function *check* traverses all the states and transitions once, hence the time cost is $O(nm + k)$.

In addition, we have also conducted a simulation study to evaluate the time spent for these three algorithms. We randomly generate some processes, and apply our tool to calculate the required time for each algorithm. We run the simulation study on a PC equipped with two 2.13 GHz CPUs and 1 Gbyte memory. Our study shows that for processes with 1,000 transitions, the construction of an atomicity-equivalent public view takes only about 2 seconds, the checking of atomicity sphere takes about 0.3 seconds, and a service composition with 10,000 transitions takes about 10 seconds. Therefore, it is feasible to apply the algorithms in practice.

4 Discussion

In our model, we restrict our discussion to synchronous communication between port actions in different processes. Note that in this paper we refer to synchronous communication as the semantics of application-level synchronizability, not the underlying communication protocol. Let us take the processes in Fig. 1a as an example. After sending the order request to the supplier, the retailer must wait for the confirmation from the supplier process before it goes to the next task "*receive invoice*." We regard this as synchronous communication between port actions "*place order*" and "*receive order*," although they may use some underlying synchronous handshake protocols (e.g., HTTP) or asynchronous communication protocols (e.g., SMTP) to send the order and the acknowledgment information. Such kind of application-level synchronizability is often required in practice; otherwise, it may lead to application deadlocks (e.g., the supplier will not send the invoice to the retailer if the order message is lost). On the other hand, even if processes use bounded asynchronous communication, one may simulate it as synchronous communication by introducing buffer abstractions to decouple the message passing [60]. In this way, we could still apply our approach to these scenarios. One alternative is to use the approach proposed by Fu et al. [39] to analyze whether a composite web service is synchronized, in the sense that the conversation set is the same when asynchronous communication is replaced by synchronous communication. If a composite web service is synchronized, our model can still be applicable even with asynchronous communication. For the unbounded asynchronous protocols, the property analysis is undecidable [8].

Another assumption in our approach is that services in different organizations have fixed port action sets and their interaction is determined by the global composition specification. This is the way that many services are composed in practice. For example, the port actions of each web service are usually described in its publishable WSDL [69] document. The communication between port actions of different Web services is through the choreography of their WSDL interfaces, such as WSCI [68] and the like. These choreographies could be seen as the global composition specification in our model. In practice, service providers and service consumers could construct the global specification by identifying the communication ports in their WSDL interfaces through negotiation (e.g., as depicted in Fig. 1, the retailer and the supplier could collaboratively identify the actions "*place order*" and "*receive order*" as communicating port actions). This could also be done automatically with the aid of semantic web techniques such as the technique proposed by Medjahed et al. [53] and the like. In addition, some industry communities may predefine standard global specifications to which all the member services must conform. For example, Partner Interface Process (PIP) is defined in RosettaNet to describe the global specification that governs the collaboration protocol among supply-chain trading partners [56]. At the service lookup stage, service consumers may first apply existing work to identify compatible services (together with the global composition specification), and then apply our approach to identify whether the intended composition satisfies the atomicity sphere or not. Note that checking behavior compatibility is out of the scope of this paper. The compatibility issue has been widely studied in the literature, such as [35] and the like.

In our model, service providers need to decide the compensability and retriability properties of each task in their own private processes. The decision can be made during the design stage of a process based on the business policies in concern or contracts with others, or after its implementation based on the organization's backend support systems. Note that the decision of such properties may depend on various factors such as business policies, backend implementation, and so on. For example, a service provider may mark a task compensable if there is another compensating task to eliminate the side effects of the first task. Sometimes, service providers may assign different values to a task for the compensability property due to different business policies. For example, the supplier in Fig. 1 may treat the "*book order*" task as noncompensable for normal retailers but as compensable for VIP retailers due to

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 3, 2009 at 09:09 from IEEE Xplore. Restrictions apply.

its business policy. Note that such properties could be deemed as a kind of quality of the service such that service providers may adapt the values for different service consumers due to different quality requirements (e.g., the supplier allows VIP retailers to cancel the orders and refund them after booking orders). Likewise, the retriability property could be assigned in a similar way. For example, the service provider may mark a task as retriable if it is implemented by underlying fault-tolerant systems [44].

Many works and industrial standards have been proposed to support and facilitate web service development, including BPEL [7], UDDI [61], and so on. Although our work uses a process algebraic model, it could easily be adapted to industrial standards, such as BPEL and the like. For example, to apply the work to BPEL processes, we need to transform each BPEL process into our process model and then apply Algorithm 1 to construct the views. Currently, many works have been proposed to translate the BPEL process into process algebra [13], [33], [34], [52], finite state automata [37], [67], and Petri net [45]. We could reuse these works to help translate the BPEL process into our model. For example, Foster et al. [33], [34] proposed an approach to translate BPEL processes into an FSP [50] representation. By reusing their work, we only need to translate the FSP representation into our model. Such a translation is just a mechanical mapping which can be done in a systematic way. Thus, we could derive atomicity-equivalent public views from these BPEL processes. Those constructed atomicity-equivalent public views can then be published in the UDDI registry so that service consumers could use them to conduct the atomicity sphere check when seeking service providers.

In our approach, Algorithm 3 could report not only the atomicity sphere violation of the global process composed of the atomicity-equivalent public views but also counter examples for the violation. Such information is helpful for organizations to identify the services and tasks that cause the atomicity sphere violation. However, as the atomicity sphere violation may be caused by many services and many tasks, which ones should service consumers identify as being the "real" reason? For example, services 1 and 2 both execute one noncompensable task before one nonretriable task in service 3, which service(s), 1 and 2, or 3 should be replaced by alternative ones? Moreover, replacing one service may affect others (e.g., service 1 may only couple with service 2, therefore replacing service 1 requires replacing service 2 as well). One answer to this question is to establish a cost model for selecting the services, and rank the potential violations based on the cost of selecting their alternative services. We will investigate this issue in our future work.

Currently, our work still has some limitations. In our model, we assume no global exception handler across multiorganizations. All the exception handlers are local to private processes. If an exception affects several processes, this exception is propagated to the corresponding processes, and its exception fragments are then handled locally in these processes. In some scenarios, however, handling one exception may require the collaboration of several processes. For example, the exception handlers in different processes may interact with one another. In this case, the interference is nonlocal. Furthermore, organizations may adopt different exception models instead of the replacement exception model. As a result, combining a task with its exception handlers and calculation of their resultant properties are harder than the approach described in this paper. To analyze the atomicity sphere in these scenarios, we plan to extend the current process algebra model to describe the exception handling in different exception models as future work.

Another limitation of our approach is that our work supports implicit partial rollback instead of explicit partial rollback. The implicit partial rollback is realized by encapsulating the partial rollback tasks into a composite unit. The partial rollback of these tasks is thus through compensating the corresponding composite unit. In this way, the process resumes the execution by choosing an alternative task of this composite unit. In our model, we allow nested composite units, that is, a composite unit may contain other composite units. Note that a composite unit is compensable if and only if all the tasks in this unit are compensable; otherwise, this composite unit is noncompensable. Similarly, a composite unit is retriable if and only if all the tasks in this unit are retriable. Each such composite unit should also satisfy the atomicity sphere, and this can be checked from the innermost composite unit iteratively using Algorithm 3.

Our model also does not support conditional transitions. If a private process has conditional transitions, then our approach can still be applicable by assuming all the conditions are true for this private process. In this way, the final analysis results based on the thus derived atomicity-equivalent public view are conservative. This means that if the composition of these public views satisfies the atomicity sphere, then the collaboration of these private processes preserves atomicity; otherwise, their collaboration may violate the atomicity requirement. Another way to handle this problem is to build their reachability graph (this graph is finite since we assume all the processes terminate), and then use it to derive the atomicity-equivalent public view instead of the original private process. We consider supporting conditional transitions in our model as future work.

In addition, we assume no interference among internal actions of private processes in different organizations. In some scenarios, however, this kind of interference may exist and thus adds to the difficulty of analyzing the atomicity sphere. The current work has not addressed and analyzed these scenarios adequately. Further study of the interference among tasks in different processes and their effects on the atomicity sphere is necessary future work.

5 Related Work

Many studies have been carried out in the areas of advanced transaction models, frameworks, and protocol support for long-running transactional processes and Web service collaboration, service publishing and discovery, and so on. In this section, we summarize and classify the major techniques proposed by recent studies.

Transaction models and protocols for service composition. Various advanced transaction models and protocols have been proposed to extend conventional transaction models to support long-running, distributed, and autonomic business processes and web services. The Saga model [40] is a kind of long-lived transaction (LLTs) which can be expressed as a sequence of transactions interleaved with other transactions. There is no solution to the problem of

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 3, 2009 at 09:09 from IEEE Xplore. Restrictions apply.

LLT's low performance as well as a high rate of abortion and deadlock because of its long-running nature. Saga alleviates this problem by relaxing the ACID requirement for specific applications. In Saga, each subtransaction is a real transaction in the sense that it preserves database consistency. However, transactions in Saga are highly related so that they should be executed as a unit. Any partial executions of a Saga are undesirable and must be compensated for. To do so, each subtransaction in a Saga is paired with a compensating transaction, which semantically undoes the corresponding committed transaction of the incomplete Saga. However, this requirement is too restrictive in reality, because some tasks in a business process may be noncompensable due to the high compensation cost from the business perspective. The exception handling mechanism used in this paper is similar to the Saga model, but relaxes the constraints and allows tasks to be noncompensable and retriable.

Various long-lived transaction models have been proposed to extend the Saga model. The global transaction model [43], [62] extends its flexibility to support both complete and partial rollbacks. A complete rollback undoes all the finished steps, whereas a partial rollback undoes some steps and rolls the process back to a safe point, which is defined as a step in a process where forward recovery can be started. In addition, the global transaction model also provides approaches to handle concurrent abortions in this model. Other well-known extensions include the X-transaction model [63], nested transaction model [28], [29], flexible transactions [5], [76], multilevel transactions [65], and so on. Although the global transaction model provides a flexible rollback mechanism, the key problem remains unresolved, that is, the rollback of the global transaction may lead to inconsistency since some transactions cannot be compensated from the business perspective. Other models like the flexible transaction model proposed in [5] allow transactions to be noncompensable. The work in [76] further discusses the application of the flexible transaction model to achieve semiatomicity in multidatabase systems based on well-formed transactions, which is similar to the concept of the atomicity sphere in our work. However, their solution discloses details of their subtransactions and commit-dependency information. In contrast, using our approach, organizations only need to expose the atomicity-equivalent public view to their partners, whereas detailed private processes remain invisible from outside their organizations. In addition, the processes in service compositions are more complex than the database transactions.

There are other extended-transaction models [31] supporting complex transaction structures and the relaxation of the ACID properties, but they are mostly database-centric, and primarily aimed at preserving the consistency of the shared data. Thus, they are generally inapplicable to applications comprising loosely coupled, web-based business services in which process recovery is as important as data recovery.

Besides the modeling of long-running transactions, studies have been made on the transactional protocols for process collaboration and web services. Business Transaction Protocol (BTP) [10] is proposed by the Organization for Advance Structured Information Systems (OASIS) to meet the requirements for long-running collaborative business applications. It relaxes the traditional ACID properties for the web services environment in a controlled manner. To ensure atomicity between multiple participants, BTP uses a two-phase outcome protocol to solve the conflicting roll-back problem by committing together potential conflicting tasks from different organizations. In this way, all or none of the potential conflicting tasks are executed. However, this approach sacrifices the autonomy of the participating processes. Other transactional protocols like WS-C/WS-T [70], [71] use a similar mechanism to handle the inconsistency, and thus share similar problems. Our work is more flexible because only public views are required and the private processes do not have to conform to such specific protocols at runtime. Consequently, the involved organizations are relatively more loosely coupled and their autonomies are largely respected.

Public view-based cross-organizational process collaboration. Many works have been proposed using public views to choreograph web service composition and cross-organizational process collaboration. Liu and Shen [49] proposed an order-preserving process-view approach to model workflow for virtual processes. Chiu et al. also used workflow view to design E-Contracts for E-Services [24], Business-to-Business (B2B) information systems [25], and e-service collaboration [22], [23]. As well, van der Aalst and Basten [1], [2] proposed an approach based on inheritance of workflow to derive public views from private processes for cross-organizational process collaboration. Sadiq and Orlowska [57] proposed a conflict-preserving approach to reduce a process such that the verification of the deadlock and synchronous problem could be completed using the reduced process instead of the original. Other similar works include [11], [12], [59], and so on. The works introduced above are all based on the concept of public views, and some of them have addressed the issue of consistency between a view and a base process from which the view is derived. However, they are all based on execution or observation consistency, or conflict preserving, and none of them discusses the atomicity-equivalent relationship between public views and their original processes. Our work contributes to and complements the above works, providing an approach to derive atomicity-equivalent public views for a service composition. Our earlier work [73] also uses public views to analyze the atomicity sphere for a service composition. The difference is that our earlier work uses a local analysis approach and is useful when not all the public views are available, but is applicable to a restricted subset of processes (i.e., deadlock-free and all transitions are reachable). The approach in this paper uses a global analysis approach, and has no such restriction, but requires the knowledge of all collaborators' public views. These two approaches complement each other.

Service publishing and discovery with quality-of-service (QoS) constraints. There are many studies in the area of service publishing and discovery, such as UDDI [61], intelligent service discovery [64], and the like; however, only a few research efforts have focused on service discovery with QoS constraints. Jacobsen and Krämer [46] extended an interface description language to express in the interface the aspects of synchronization constraints, pre/post conditions, invariants, and QoS. Cardoso et al. [16] introduced several useful models for the measurement of QoS aspects of workflow tasks, including time, cost, and reliability. Based on these quality dimensions, an approach

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 3, 2009 at 09:09 from IEEE Xplore. Restrictions apply.

is introduced to match services satisfying each dimension of quality according to certain fitness metrics [17]. Deora et al. [30] extended current function-based approaches to service publishing and discovery, allowing service providers and consumers to express their QoS promises and requirements using ontologies. Wohlstadter et al. [66] tackled the QoS problem in service matchmaking by using a middleware-based approach. The middleware allows services to negotiate with one another and exchange QoS policy information so that they can compute an agreeable common set of QoS policies. Although the above works provide some QoS support in service publishing and discovery, few have considered the atomicity aspect. The collaborators may use the middleware-based approach [66] to negotiate their transactional configurations, but this work does not show how to do that. We consider this task nontrivial, because the atomicity sphere is different from other QoS aspects stated in [66] in the sense that the atomicity sphere may be affected by all involved processes. Our work contributes to the use of atomicity sphere as an aspect of QoS, and providing a systematic approach to checking the atomicity requirement in the composition of all involved services meanwhile protecting their privacy. This can be used to discover potential nonatomic executions in a service composition in advance. Therefore, our work complements the other works in this area.

Formal models for business processes and services. Some researchers have proposed using formal approaches to modeling long-running business processes and analyzing their properties. Butler et al. [13], [14], [15] adapted CSP to model long-running business processes and their precise compensation semantics. These works are similar to ours in that they also adopted a transactional model similar to the saga model, and use exception handling to support forward recovery. The difference is that their model describes the compensators and exception handlers explicitly to reflect their semantics, whereas ours does not. Instead, we use the compensability property to represent whether a task has a compensator. This representation is simple and competent enough for checking the atomicity sphere. Moreover, our work focuses on how to derive the atomicity-equivalent public views from private processes. This, however, is not addressed in their work. In addition, Foster et al. [33], [34], [36] also proposed to describe BPEL processes using FSP [50], and model checking service compositions. Broy et al. [9] proposed a formal model to describe services as a partial behavior. Fisher and Majumdar [32] also proposed a set consistency approach to verify correctness for long-running transactions, but the model does not address service interactions across organizations. Similarly, these works do not address how to derive atomicity-equivalent public views for checking the atomicity sphere.

6 Conclusion

This paper has presented a novel approach to publishing and discovering atomicity-equivalent services for service compositions. It provides a criterion for service consumers to choose suitable collaborators in the service look-up stage before a service composition. This approach is based on a process algebra model, which as we have proved, can be used to derive atomicity-equivalent public views from backend processes. By checking the atomicity sphere in the composition of these public views instead of the original backend processes, a global analysis of atomicity sphere can be conducted at an abstract level. In this way, service providers only need to release the atomicity-equivalent public views of their backend processes when they want to examine the atomicity property of their collaboration. Algorithms are also presented to derive corresponding public views from backend processes, to compose these public views and to check their atomicity sphere.

The present work has made a couple of assumptions such as static sets of port actions and well-formed processes. These assumptions might not necessarily hold in complex scenarios. In the future, we will relax these assumptions to widen the applicability of our approach. In addition, more real-life case studies will be conducted to investigate the usability and scalability of our approach in practice.

Appendix

Proof of Lemma 1. Numbers 1 and 2 follow directly from Definition 11. We only need to prove number 3. We prove it by induction on the number of steps used in deriving $p_1 \rightarrow p_2$. When $q = 0$, or ϕ, it is not difficult to verify that number 3 of the lemma is satisfied. Hence, in our proof, we assume that q is not equal to 0, or ϕ. Let n be the number of steps used in deriving $p_1 \rightarrow p_2$.

When $n = 1$, $p_1 = p_2$ is an axiom of PA^{a}:

1. For Axioms A1-A5, M1-M9, $p_1 \| q$ has the same trace as that of $p_2 \| q$. So they have the same value of function Ψ.
2. For R7-R15 (those axioms involving ϕ), $p_1 \| q$ will violate the atomicity sphere, and so does $p_2 \| q$.
3. For R1, the trace of $p_1 \| q$ is the same as those of $p_2 \| q$ except some trace of the former one has more silent actions $\tau_{\mathrm{c,r}}$, which however does not affect the value of function Ψ.
4. For R2, for every complete trace of $p_1 \| q$ with the format $<\tau_{\mathrm{nc,r}}, \tau_{\mathrm{nc,r}}, b_1, b_2, \ldots, b_{\mathrm{k}}, \ldots>$ or trace $<b_1, b_2, \ldots, b_{\mathrm{k}}, \tau_{\mathrm{nc,r}}, \tau_{\mathrm{nc,r}}, \ldots>$, where $b_1, b_2, \ldots, b_{\mathrm{k}}$ are actions from q, there must exist corresponding trace $<\tau_{\mathrm{nc,r}}, b_1, b_2, \ldots, b_{\mathrm{k}}, \ldots>$ or $<b_1, b_2, \ldots, b_{\mathrm{k}}, \tau_{\mathrm{nc,r}}, \ldots>$ belonging to $p_2 \| q$. If there exists a trace of $p_1 \| q$ with the format $<\tau_{\mathrm{nc,r}}, b_1, b_2, \ldots, b_{\mathrm{k}}, \tau_{\mathrm{nc,r}}, \ldots>$, where $b_1, b_2, \ldots, b_{\mathrm{k}}$ are actions from q, then there must exist a trace $<\tau_{\mathrm{nc,r}}, \tau_{\mathrm{nc,r}}, b_1, b_2, \ldots, b_k, \ldots>$ and another trace $<b_1, b_2, \ldots, b_{\mathrm{k}}, \tau_{\mathrm{nc,r}}, \tau_{\mathrm{nc,r}}, \ldots>$ belonging to $p_1 \| q$, and corresponding traces $<\tau_{\mathrm{nc,r}}, b_1, b_2, \ldots, b_{\mathrm{k}}, \ldots>$ and $<b_1, b_2, \ldots, b_{\mathrm{k}}, \tau_{\mathrm{nc,r}}, \ldots>$ belonging to $p_2 \| q$. Therefore, there is a mapping between the trace of $p_1 \| q$ and $p_2 \| q$, and obviously, they have the same value of function Ψ.
5. For R3-R6, the proof is similar to R2.

So the conclusion is satisfied when $n = 1$.

Suppose when $n < m$ $(m > 1)$, the conclusion is satisfied, i.e., $p_1^{(i)} \rightarrow p_2^{(i)}$, $\Psi(p_1^{(i)}) = \Psi(p_2^{(i)}) \Rightarrow \Psi(p_1^{(i)} \| q) =$

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 3, 2009 at 09:09 from IEEE Xplore. Restrictions apply.

$\Psi(p_2^{(i)}\|q)$, $i < m$, for any q, where $p_1^{(i)}$, $p_2^{(i)}$ represent the terms in the ith step reduction.

When $n = m$:

a. If the mth step in deriving $p_1^{(m)} \rightarrow p_2^{(m)}$ is a substitution, then the conclusion is satisfied because it is only a special case of $p_1^{(m-1)} \rightarrow p_2^{(m-1)}$.
b. If the mth step is a prefix context, i.e., $p_1^{(m)} = a \cdot p_1^{(m-1)}$, and $p_2^{(m)} = a \cdot p_2^{(m-1)}$, q may have three possible formats according to our operators:
   - When $q = q_1 \| q_2$, serialize q into $q_1' + q_2'$ first.
   - When $q = q_1 + q_2$, we only need to prove $\Psi(a \cdot p_1^{(m-1)}\|q_1) = \Psi(a \cdot p_2^{(m-1)}\|q_1)$. If $q_1 = q_{11} + q_{12}$, we continue this procedure until $q = b \cdot q'$.
   - When $q = b \cdot q'$, we have $p_1^{(m)}\|q = a \cdot p_1^{(m-1)}\|b \cdot q'$, $p_2^{(m)}\|q = a \cdot p_2^{(m-1)}\|b \cdot q'$. Let H_1 be the set of port actions of $a \cdot p_1^{(m-1)}$, $a \cdot p_2^{(m-1)}$, and H_2 be the set of port actions of q. If $a \notin H_2$ and $b \notin H_1$, $a \cdot p_1^{(m-1)}\|b \cdot q' = b \cdot (q'\|a \cdot p_1^{(m-1)}) + a \cdot (p_1^{(m-1)}\|b \cdot q')$ and
$$a \cdot p_2^{(m-1)}\|b \cdot q' = b \cdot (q'\|a \cdot p_2^{(m-1)}) + a \cdot (p_2^{(m-1)}\|b \cdot q').$$
   Based on the assumption, $\Psi(p_1^{(m-1)}\|b \cdot q') = \Psi(p_2^{(m-1)}\|b \cdot q')$, so $\Psi(a \cdot (p_1^{(m-1)}\|b \cdot q')) = \Psi(a \cdot (p_2^{(m-1)}\|b \cdot q'))$. We only need to prove $\Psi(q'\|a \cdot p_1^{(m-1)}) = \Psi(q'\|a \cdot p_2^{(m-1)})$. Using the same procedure recursively, we can finally serialize the process $q'\|a \cdot p_1^{(m-1)}$ into a summation (the "+" operator) of sequential processes, in which each sequential process has the form $d \cdot a \cdot p_1^{(m-1)}$, where d is a trace of q', and so do $q'\|a \cdot p_2^{(m-1)}$ into $d \cdot a \cdot p_2^{(m-1)}$, which have the same value of function Ψ. If ($a \in H_2$ *and* $b \notin H_1$) or ($a \notin H_2$ *and* $b \in H_1$) or ($a \in H_2$ *and* $b \in H_1$), the proof approach is similar.
c. If the mth step is a choice context, i.e., $p_1^{(m)} = p' + p_1^{(m-1)}$, and $p_2^{(m)} = p' + p_2^{(m-1)}$, then $p_1^{(m)}\|q = (p' + p_1^{(m-1)})\|q = p'\|q + p_1^{(m-1)}\|q$ and $p_2^{(m)}\|q = (p' + p_2^{(m-1)})\|q = p'\|q + p_2^{(m-1)}\|q$. Based on the assumption, $\Psi(p_1^{(m-1)}\|q) = \Psi(p_2^{(m-1)}\|q)$, so $\Psi(p'\|q + p_1^{(m-1)}\|q) = \Psi(p'\|q + p_2^{(m-1)}\|q)$.
d. If the mth step is a parallel context, i.e., $p_1^{(m)} = p_1^{(m-1)}\|p'$, and $p_2^{(m)} = p_2^{(m-1)}\|p'$, then $p_1^{(m)}\|q = p'\|p_1^{(m-1)}\|q = p_1^{(m-1)}\|(p'\|q)$ and $p_2^{(m)}\|q = p'\|p_2^{(m-1)}\|q = p_2^{(m-1)}\|(p'\|q)$ based on the assumption, $\Psi(p_1^{(m-1)}\|(p'\|q)) = \Psi(p_2^{(m-1)}\|(p'\|q))$.

So the conclusion is also satisfied when $n = m$. Hence, Lemma 1 is satisfied. □

Proof of Lemma 2. We prove this lemma by induction on the number of steps used in deriving $p_1 \rightarrow p_2$. Let n be the number of steps used in deriving $p_1 \rightarrow p_2$. We use $p_1^{(i)} \rightarrow p_2^{(i)}$ to represent the ith step during the reduction, where $p_1^{(i)}$, $p_2^{(i)}$ represent the terms in the ith step reduction.

When $n = 1$, for every axiom $t_1 = t_2$ of PA^a, obviously we have $\Psi(t_1) = \Psi(t_2)$.

Suppose $n < m$ $(m > 1)$ the conclusion is satisfied, i.e., $p_1^{(i)} \rightarrow p_2^{(i)} \Rightarrow \Psi(p_1^{(i)}) = \Psi(p_2^{(i)})$.

When $n = m$:

a. If the mth step in deriving $p_1^{(m)} \rightarrow p_2^{(m)}$ is a substitution, then the conclusion is satisfied because it is only a special case of $p_1^{(m-1)} \rightarrow p_2^{(m-1)}$.
b. If the mth step is a prefix context, i.e., $p_1^{(m)} = a \cdot p_1^{(m-1)}$, and $p_2^{(m)} = a \cdot p_2^{(m-1)}$, then based on Lemma 1 and the assumption, we have $\Psi(a \cdot p_1^{(m-1)}) = \Psi(a \cdot p_2^{(m-1)})$.
c. If the mth step is choice context, i.e., $p_1^{(m)} = p' + p_1^{(m-1)}$, and $p_2^{(m)} = p' + p_2^{(m-1)}$, then based on Lemma 1 and the assumption, we have $\Psi(p' + p_1^{(m-1)}) = \Psi(p' + p_2^{(m-1)})$.
d. If the mth step is parallel context, i.e., $p_1^{(m)} = p_1^{(m-1)}\|p'$, and $p_2^{(m)} = p_2^{(m-1)}\|p'$, then based on Lemma 1 and the assumption, we have $\Psi(p_1^{(m-1)}\|p') = \Psi(p_2^{(m-1)}\|p')$.

So the conclusion is also satisfied when $n = m$. Lemma 2 is thus satisfied. □

ACKNOWLEDGMENTS

This research was partially supported by the Research Grants Council of Hong Kong under grant nos. 6167/04E, 612306, 111107, 717308, and HKBU 1/05C, the National Science Foundation of China under grants no 60736015, and National Basic Research of China 973 under grant no. 2006CB303000.

REFERENCES

[1] W.M.P. van der Aalst, "Inheritance of Interorganizational Workflows to Enable Business-to-Business E-Commerce," *Electronic Commerce Research*, vol. 2, no. 3, pp. 195-231, 2002.
[2] W.M.P. 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, pp. 125-203, Jan. 2002.
[3] Alibaba Website, http://alibaba.com/, Aug. 2008.
[4] *Atomicity Analysis Tool*, http://ihome.ust.hk/~cyye/tools/intro.html., 2008.
[5] M. Ansari, L. Ness, M. Rusinkiewicz, and A. Sheth, "Using Flexible Transactions to Support Multi-System Telecommunication Applications," *Proc. 18th Int'l Conf. Very Large Data Bases*, pp. 65-76, 1992.
[6] J.A. Bergstra, A. Ponse, and S.A. Smolka, "The Linear Time—Branching Time Spectrum I," *Handbook of Process Algebra*, R.J. van Glabbeek, ed., Elsevier, 2001.
[7] BPEL, http://www-128.ibm.com/developerworks/library/specification/ws-bpel/, Aug. 2008.
[8] D. Brand and P. Zafiropulo, "On Communicating Finite-State Machines," *J. ACM*, vol. 30, no. 2, pp. 323-342, Apr. 1983.
[9] M. Broy, I.H. Krüger, and M. Meisinger, "A Formal Model of Services," *ACM Trans. Software Eng. and Methodology*, vol. 16, no. 1, pp. 1-40, Feb. 2007.
[10] BTP, http://www.oasis-open.org/committees/download.php/1184/2002-06-03.BTP_cttee_spec_1.0.pdf, Aug. 2008.
[11] C. Bussler, "Behavior Abstraction in Semantic B2B Integration," *Conceptual Modeling for New Information Systems Technologies, ER 2001 Workshops. HUMACS, DASWIS, ECOMO, and DAMA*, revised papers, pp. 377-389, 2002.

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 3, 2009 at 09:09 from IEEE Xplore. Restrictions apply.

[12] C. Bussler, "Public Process Inheritance for Business-to-Business Integration," *Proc. Third Int'l Workshop Technologies for E-Services,* pp. 19-28, 2002.
[13] M. Butler, C. Ferreira, and M.Y. Ng, "Precise Modelling of Compensating Business Transactions and Its Application to BPEL," *J. Universal Computer Science,* vol. 11, no. 5, pp. 712-743, 2005.
[14] M. Butler and C. Ferreira, "An Operational Semantics for StAC, A Language for Modelling Long-Running Business Transactions," *Proc. Sixth Int'l Conf. Coordination Models and Languages*, pp. 87-104, 2004.
[15] M. Butler, C.A.R. Hoare, and C. Ferreira, "A Trace Semantics for Long-Running Transactions," *Proc. 25 Years of CSP*, pp. 133-150, 2005.
[16] J. Cardoso, A. Sheth, and J. Miller, "Workflow Quality of Service," *Proc. 12th Int'l Conf. Enterprise Integration and Modeling Technology*, pp. 303-311, 2003.
[17] J. Cardoso and A. Sheth, "Semantic E-Workflow Composition," *J. Intelligent Information Systems,* vol. 21, no. 3, pp. 191-225, Nov. 2003.
[18] F. Casati and G. Cugola, "Error Handling in Process Support Systems," *Advances in Exception Handling Techniques,* pp. 251-270, Springer-Verlag, 2001.
[19] F. Casati, S. Castano, M.G. Fugini, I. Mirbel, and B. Pernici, "Using Patterns to Design Rules in Workflows," *IEEE Trans. Software Eng.,* vol. 26, no. 8, pp. 760-785, Aug. 2000.
[20] F. Casati, S. Ceri, S. Paraboschi, and G. Pozzi, "Specification and Implementation of Exceptions in Workflow Management Systems," *ACM Trans. Database Systems,* vol. 24, no. 3, pp. 405-451, Sept. 1999.
[21] CrossFlow Consortium/AGFIL. Insurance Requirements, CrossFlow deliverable: D1.b. La Gaude, http://www.crossflow.org/public/pubdel/D1b.pdf (accessed on 5 May 2006), Mar. 1999.
[22] D.K.W. Chiu, S.C. Cheung, S. Till, K. Karlapalem, Q. Li, and E. Kafeza, "Workflow View Driven Cross-Organizational Interoperability in a Web Service Environment," *Information Technology and Management,* vol. 5, no. 3/4, pp. 221-250, July-Oct. 2004.
[23] D.K.W. Chiu, K. Karlapalem, and Q. Li, "Views for Inter-Organization Workflow in an E-Commerce Environment," *Proc. IFIP TC2/WG2.6 Ninth Working Conf. Database Semantics: Semantic Issues in E-Commerce Systems,* pp. 137-151, 2003.
[24] D.K.W. Chiu, K. Karlapalem, Q. Li, and E. Kafeza, "Workflow View Based E-Contracts in a Cross-Organizational E-Services Environment," *Distributed and Parallel Databases,* vol. 12, nos. 2/3, pp. 193-216, Sept.-Nov. 2002.
[25] D.K.W. Chiu, Z. Shan, P.C.K. Hung, and Q. Li, "Designing Workflow Views with Flows for Large-Scale Business-to-Business Information Systems," *Proc. Fifth Int'l Workshop Technologies for E-Services,* revised selected papers, pp. 107-121, 2004.
[26] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms,* pp. 449-457. MIT Press, 1990.
[27] S. Dalal, S. Temel, M. Little, M. Potts, and J. Webber, "Coordinating Business Transactions on the Web," *IEEE Internet Computing,* vol. 7, no. 1, pp. 30-39, Jan./Feb. 2003.
[28] U. Dayal, M. Hsu, and R. Ladin, "Organizing Long-Running Activities with Triggers and Transactions," *SIGMOD Record,* vol. 19, no. 2, pp. 204-214, June 1990.
[29] U. Dayal, M. Hsu, and R. Ladin, "A Transactional Model for Long-Running Activities," *Proc. 17th Int'l Conf. Very Large Data Bases,* pp. 113-122, 1991.
[30] V. Deora, J. Shao, G. Shercliff, P.J. Stockreisser, W.A. Gray, and N.J. Fiddian, "Incorporating QoS Specifications in Service Discovery," *Proc. Fifth Int'l Conf. Web Information Systems Eng.,* pp. 252-263, 2004.
[31] A.K. Elmagarmid, *Database Transaction Models for Advanced Applications.* Morgan Kaufmann, 1992.
[32] J. Fischer and R. Majumdar, "Ensuring Consistency in Long Running Transactions," *Proc. 22nd IEEE/ACM Int'l Conf. Automated Software Eng.,* pp. 54-63, 2007.
[33] H. Foster, "A Rigorous Approach to Engineering Web Service Compositions," PhD thesis, Imperial College London, Jan. 2006.
[34] H. Foster, S. Uchitel, J. Magee, and J. Kramer, "Model-Based Verification of Web Service Compositions," *Proc. 18th IEEE Int'l Conf. Automated Software Eng.,* pp. 152-161, 2003.
[35] H. Foster, S. Uchitel, J. Magee, and J. Kramer, "Compatibility Verification for Web Service Choreography," *Proc. Third IEEE Int'l Conf. Web Services,* pp. 738-741, 2004.
[36] H. Foster, W. Emmerich, J. Kramer, J. Magee, D. Rosenblum, and S. Uchitel, "Model Checking Service Compositions under Resource Constraints," *Proc. Sixth Joint Meeting of the European Software Eng. Conf. and the ACM SIGSOFT Symp. Foundations of Software Eng.,* pp. 225-234, 2007.
[37] X. Fu, T. Bultan, and J.W. Su, "Analysis of Interacting BPEL Web Services," *Proc. Third IEEE Int'l Conf. Web Services,* pp. 621-630, 2004.
[38] X. Fu, T. Bultan, and J.W. Su, "Conversation Protocols: A Formalism for Specification and Verification of Reactive Electronic Services," *Theoretical Computer Science,* vol. 328, nos. 1-2, pp. 19-37, Nov. 2004.
[39] X. Fu, T. Bultan, and J.W. Su, "Synchronizability of Conversations among Web Services," *IEEE Trans. Software Eng.,* vol. 31, no. 12, pp. 1042-1055, Dec. 2005.
[40] H. Garcia-Molina and K. Salem, "SAGAS," *SIGMOD Record,* vol. 16, no. 3, pp. 249-259, Dec. 1987.
[41] J. Gray, *Notes on Data Base Operating Systems,* pp. 393-481, Springer-Verlag, 1978.
[42] P. Greenfield, D. Kuo, S. Nepal, and A. Fekete, "Consistency for Web Services Applications," *Proc. 31st Int'l Conf. Very Large Data Bases,* pp. 1199-1203, Aug./Sept. 2005.
[43] P. Grefen, J. Vonk, and P. Apers, "Global Transaction Support for Workflow Management Systems: From Formal Specification to Practical Implementation," *The VLDB J.,* vol. 10, no. 4, pp. 316-333, Dec. 2001.
[44] C. Hagen and G. Alonso, "Exception Handling in Workflow Management Systems," *IEEE Trans. Software Eng.,* vol. 26, no. 10, pp. 943-958, Oct. 2000.
[45] S. Hinz, K. Schmidt, and C. Stahl, "Transforming BPEL to Petri Nets," *Proc. Third Int'l Conf. Business Process Management,* pp. 220-235, 2005.
[46] H.A. Jacobsen and B.J. Krämer, "Modeling Interface Definition Language Extensions," *Proc. 37th Int'l Conf. Technology of Object-Oriented Languages and Systems,* pp. 242-252, 2000.
[47] A. Kumar and J.L. Zhao, "Workflow Support for Electronic Commerce Applications," *Decision Support Systems,* vol. 32, no. 3, pp. 265-278, 2002.
[48] P.A. Lee and T. Anderson, *Fault Tolerance: Principles and Practice,* pp. 143-185. Springer-Verlag, 1990.
[49] D.-R. Liu and M. Shen, "Workflow Modelling for Virtual Processes: An Order-Preserving Process-View Approach," *Information Systems,* vol. 28, no. 6, pp. 505-532, Sept. 2003.
[50] J. Magee and J. Kramer, *Concurrency—State Models and Java Programs.* John Wiley & Sons, 1999.
[51] Manufacturer, http://shop35602377.taobao.com/, Aug. 2008.
[52] M. Mazzara and S. Govoni, "A Case Study of Web Services Orchestration," *Proc. Seventh Int'l Conf. Coordination Models and Languages,* pp. 1-16, 2005.
[53] B. Medjahed, A. Bouguettaya, and A.K. Elmagarmid, "Composing Web Services on the Semantic Web," *The VLDB J.,* vol. 12, no. 4, pp. 333-351, Nov. 2003.
[54] S. Nakajima, "Model-Checking of Safety and Security Aspects in Web Service Flows," *Proc. Fourth Int'l Conf. Web Eng.,* pp. 488-501, 2004.
[55] M.P. Papazoglou, P. Traverso, S. Dustdar, F. Leymann, and B.J. Krämer, "Service-Oriented Computing: A Research Roadmap," *Proc. Service Oriented Computing,* 2006.
[56] RosettaNet, http://portal.rosettanet.org/cms/sites/RosettaNet/index.html, Aug. 2008.
[57] W. Sadiq and M.E. Orlowska, "Analyzing Process Models Using Graph Reduction Techniques," *Information Systems,* vol. 25, no. 2, pp. 117-134, 2000.
[58] H. Schuldt, G. Alonso, C. Beeri, and H.-J. Schek, "Atomicity and Isolation for Transactional Processes," *ACM Trans. Database Systems,* vol. 27, no. 1, pp. 63-116, Mar. 2002.
[59] K. Schultz and M.E. Orlowska, "Facilitating Cross-Organisational Workflows with a Workflow View Approach," *Data and Knowledge Eng.,* vol. 51, no. 1, pp. 109-147, Oct. 2004.
[60] S. Uchitel, J. Kramer, and J. Magee, "Incremental Elaboration of Scenario-Based Specifications and Behavior Models Using Implied Scenarios," *ACM Trans. Software Eng. and Methodology,* vol. 13, no. 1, pp. 37-85, Jan. 2004.
[61] UDDI, http://www.oasis-open.org/specs/index.php#uddiv3.0.2, Aug. 2008.

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 3, 2009 at 09:09 from IEEE Xplore. Restrictions apply.

[62] J. Vonk, P. Grefen, E. Boertjes, and P. Apers, "Distributed Global Transaction Support for Workflow Management Applications," *Proc. 10th Int'l Conf. Database and Expert System Applications,* pp. 942-951, 1999.
[63] J. Vonk and P. Grefen, "Cross-Organizational Transaction Support for E-Services in Virtual Enterprises," *Distributed and Parallel Databases,* vol. 14, no. 2, pp. 137-172, Sept. 2003.
[64] X. Wang, B.J. Krämer, Y. Zhao, and W.A. Halang, "Representation and Discovery of Intelligent E-Services," *E-Service Intelligence.* Springer, 2007.
[65] G. Weikum, "Principles and Realization Strategies of Multilevel Transaction Management," *ACM Trans. Database Systems,* vol. 16, no. 1, pp. 132-180, Mar. 1991.
[66] E. Wohlstadter, S. Tai, T. Mikalsen, I. Rouvellou, and P. Devanbu, "GlueQoS: Middleware to Sweeten Quality-of-Service Policy Interactions," *Proc. 26th Int'l Conf. Software Eng.,* pp. 189-199, 2004.
[67] A. Wombacher, P. Fankhauser, and E. Neuhold, "Transforming BPEL into Annotated Deterministic Finite State Automata for Service Discovery," *Proc. Third IEEE Int'l Conf. Web Services,* pp. 316-323, 2004.
[68] WSCI, http://www.w3.org/TR/wsci/, Aug. 2008.
[69] WSDL, http://www.w3.org/TR/wsdl, Aug. 2008.
[70] WS-C, http://docs.oasis-open.org/ws-tx/wscoor/2006/06, Aug. 2008.
[71] WS-T, http://www.ibm.com/developerworks/library/specification/ws-tx/ Aug. 2008.
[72] C.Y. Ye, S.C. Cheung, and W.K. Chan, "Publishing and Composition of Atomicity-Equivalent Services for B2B Collaboration," *Proc. 28th Int'l Conf. Software Eng.,* pp. 351-360, 2006.
[73] C.Y. Ye, S.C. Cheung, W.K. Chan, and C. Xu, "Local Analysis of Atomicity Sphere for B2B Collaboration," *Proc. 14th ACM SIGSOFT Int'l Symp. Foundations of Software Eng.,* pp. 186-196, Nov. 2006.
[74] S. Yemini and D.M. Berry, "A Modular Verifiable Exception-Handling Mechanism," *ACM Trans. Programming Languages and Systems,* vol. 7, no. 2, pp. 214-243, Apr. 1985.
[75] L.Z. Zeng, B. Benatallah, A.H.H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-Aware Middleware for Web Services Composition," *IEEE Trans. Software Eng.,* vol. 30, no. 5, pp. 311-327, May 2004.
[76] A. Zhang, M. Nodine, B. Bhargava, and O. Bukhres, "Ensuring Relaxed Atomicity for Flexible Transactions in Multidatabase Systems," *SIGMOD Record,* vol. 23, no. 2, pp. 67-78, June 1994.

Chunyang Ye received the BEng degree from the University of Science and Technology of China in 2000, the MEng degree from the Chinese Academy of Sciences in 2003, and the PhD degree from the Hong Kong University of Science and Technology in 2008. His research interests include software engineering issues on service engineering, workflow, exception handling, and dependable computing.

S.C. Cheung received the MSc and PhD degrees in computing from Imperial College London. He is an associate professor in the Department of Computer Science and Engineering at the Hong Kong University of Science and Technology. His research interests include context-aware computing, service-oriented computing, software testing, fault localization, RFID, and wireless sensor network systems. He participates actively in the research communities of software engineering and service-oriented computing. He serves on the executive committee of ACM SIGSOFT, the editorial board of the *IEEE Transactions on Software Engineering*, and the editorial board of the *Journal of Computer Science and Technology*. He is a senior member of the IEEE.

W.K. Chan received the BEng degree in computer engineering, the MPhil degree in software engineering, and the PhD degree in software engineering from the University of Hong Kong in 1993, 1995, and 2004, respectively. He has more than 10 years of industrial software development and management experience of enterprise systems. From 2005 to 2006, he was a postdoctoral research fellow at the Hong Kong University of Science and Technology. Since 2006, he has been a lecturer in the Department of Computer Science at City University of Hong Kong. His current research interests include issues in software testing and analysis, service engineering, and development of context-aware applications and wireless sensor network applications. He is currently on the editorial board of *Journal of Systems and Software* and on the technical program committees of many international conferences. He is a member of the IEEE.

Chang Xu received the BEng degree from the University of Science and Technology of China in 2000, the MEng degree from the Chinese Academy of Sciences in 2003, and the PhD degree from the Hong Kong University of Science and Technology in 2008. Since September 2008, he has been a research assistant professor in the Department of Computer Science and Engineering at the Hong Kong University of Science and Technology. His research interests include context management for pervasive computing, RFID benchmarking, semantic Web services, and software engineering.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.

Authorized licensed use limited to: IEEE Xplore. Downloaded on March 3, 2009 at 09:09 from IEEE Xplore. Restrictions apply.