To appear in *Proceedings of the 28th International Conference on Software Engineering (ICSE 2006), Shanghai, China 20-28, 2006.*

# Publishing and Composition of Atomicity-equivalent Services for B2B Collaboration*

Chunyang Ye
Department of Computer Science
Hong Kong University of Science and Technology
Hong Kong, China
cyye@cs.ust.hk

S.C. Cheung §
Department of Computer Science
Hong Kong University of Science and Technology
Hong Kong, China
scc@cs.ust.hk

W.K. Chan
Department of Computer Science
Hong Kong University of Science and Technology
Hong Kong, China
wkchan@cs.ust.hk

## ABSTRACT
Exception handling resolves inconsistency by backward or forward error recovery methods or both in Business-to-Business (B2B) process collaboration. To avoid committing irrevocable tasks followed by exceptions, B2B processes, which guarantee the atomicity sphere property, are attractive. While atomicity sphere ensures its outcomes to be either all or nothing, conflicting local recoveries may lead to global B2B inconsistencies. Existing (global) analysis techniques however mandate every process unveiling all individual tasks. Such an analysis is infeasible when some business parties refuse to disclose their process details for privacy or business reasons. To address this problem, we propose a process algebraic technique to prove, construct, and check atomicity-equivalent public views from B2B processes. By checking atomicity spheres in the composition of these public views, business parties can identify suitable services that respect their individual and overall atomicity requirements. An example based on a real-life multilateral supply chain process is included.

## Categories and Subject Descriptors
H.5.3 [**Information Interfaces and Presentation**]: Group and Organization Interfaces -*Theory and models, Web-based interaction*

## General Terms
Algorithms, Design, Theory, Verification.

## Keywords
Process collaboration, atomicity, process algebra, public view.

## 1. INTRODUCTION
Typical collaborative transactions in a Business-to-Business (B2B) environment involve multiple organizations. They prescribe business processes, which are published online via some service registries. These registries allow organizations to identify appropriate processes and services offered by their partners. To protect their own interests, organizations may want to publish views of their business processes as restrictive as possible, hiding the process details from its partners or competitors. Furthermore, accesses to these processes are granted only if right messages are passed to the processes via these public yet restrictive views.

The transactional support among collaborative processes is desirable because it helps guarantee application consistency. While B2B process transactions akin to distributive database transactions that may not share data, location or administration, B2B process transactions are typically long-running, loosely coupled and individually enacted by different organizations. In this setting, it is thorny to achieve both acceptable performance and enforcing the semantics of transactions with properties of atomicity, consistency, isolation and durability (ACID). This is because fine-grained lock controls and full trustworthiness are not generally applicable [7]. Weak consistency approaches, such as *exception handling*, are widely accepted as a suitable concession to compromise between these two aspects. In this paper, we restrict ourselves to exception handling for the atomicity property.

When an exception occurs, exception handling may recover the failures in the backward or the forward fashion [15]. For the backward recovery, it aborts the execution of the process and compensates all executed tasks it protected. Thus, it rewinds the state of the process to the state immediately before, as if none of its protected tasks were executed. For the forward recovery, it chooses an alternative activity of the process to continue the execution. However, if the execution of a process has passed the point of no return, process abortion would violate the atomicity property and lead the process to inconsistent states. To address this problem, Hagen and Alonso [14] propose a criterion to check whether or not a process fulfills the atomicity requirements. Their work is based on the notion of atomicity sphere.

A process is said to satisfy the *atomicity sphere* if it is guaranteed to terminate with the semantics of all or nothing. Satisfying this property, the process should be able either to proceed until termination or to compensate everything as if the tasks had not been executed [14]. *Compensability* and *retriability* are two boolean properties associated with any task of any process. A compensable task is a task that can be undone one way or the other whenever the process fails or aborts. A retriable task is a task that can, without cumulative effects, repeat itself internally if the latest internal trial fails, and finally succeed after a finite number of trials. A task is called a *pivot task* if it is not compensable [17], which means that the overhead or cost of compensating the task is unacceptable. Hence, for a B2B


*ICSE'06*, May 20–28, 2006, Shanghai, China.


* This research is supported by a grant from the Research Grants Council of Hong Kong (Project No. HKUST 6170/03E).

§ All correspondence should be addressed to Dr. S.C. Cheung at Department of Computer Science, The Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong, China. Tel: (+852) 2358-7016. Fax: (+852) 2358-1477. Email: scc@cs.ust.hk .

transaction, committing a pivot task means that the transaction is committed to completing the process; otherwise the consequence of process incompletion would be unacceptable.

Hagen and Alonso [14] propose a criterion for process collaboration to ensure the atomicity sphere property. Their approach safely determines the satisfaction of the atomicity sphere of every participating process if no task is hidden. Still, those processes, which have been individually verified by their approach, may still breach certain overall atomicity spheres when they collaborate. For example, an occurrence of an exception in a process may lead to the abortion of other collaborating processes. This obscures the strategy that aims at compensating all executed tasks if some processes have executed their pivot tasks. Moreover, their approach cannot handle processes with hidden tasks. Thus, although there are plenty techniques to alleviate the problem of state combinatorial explosion for collaborating processes by hiding unnecessary details such as [5], their approach [14] is incompatible to the latter kind.

Let us elaborate our discussion by an example. To ease readers to follow, we refer an individual process of a business party as a *private process*. Figure 1 depicts a collaboration of three private processes $p_1$, $p_2$, and $p_3$, in which tasks are represented by small black circles along the three swan-lines. Suppose that (*i*) task $c_1$ in $p_3$ is non-compensable and retriable, (*ii*) task $a_3$ in $p_1$ is compensable and non-retriable, and (*iii*) all the other tasks are compensable and retriable. It is ease to observe that every process satisfies its respective atomicity spheres.

When $a_3$ fails, it would abort $p_1$, $p_2$, and $p_3$. This is because $a_3$ being compensable and non-retriable, $p_1$ needs to rewind to its initial state to recover the failure. This triggers $p_2$ to rewind to its initial state also. However, before $a_3$ is able to execute, $p_3$ should have already committed its pivot task $c_1$ and is thus unable to compensate the process ($p_3$) without incurring heavy penalty. To avoid such a conflicting situation, a global analysis of the collaborative processes is desirable in the service discovery stage for process collaboration. However, such an analysis is infeasible when some business parties refuse to disclose their details for privacy or business reasons.

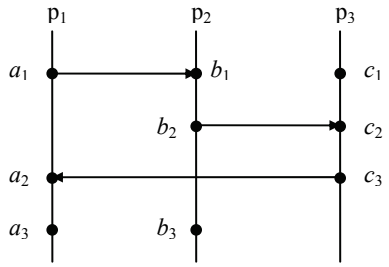


**Figure 1. Process collaboration with conflicts.**

This paper proposes a novel approach to addressing the atomicity requirements on exception handling mechanism. The main contributions of this paper are threefold. Firstly, a process algebraic model to derive atomicity-equivalent public views from private processes is proposed. Next, the satisfaction of atomicity sphere is proven to be verifiable based on the analysis of the public views. This exempts business parties from the needs to unveil complete details of their private processes in order to achieve collaborative atomicity sphere properties atop functional requirements. Lastly, algorithms are presented to construct and compose public views, and check their atomicity sphere.

The rest of this paper is organized as follows. Section 2 introduces the related work. Section 3 presents its process algebra, theoretical results and algorithms. Section 4 illustrates the proposal by an example based on a real-life multilateral supply chain process. Finally, there are evaluations followed by the conclusion.

## 2. RELATED WORK

In this section, we review the major techniques proposed by recent studies in the areas of advanced transaction model for long-running transactions, frameworks and protocol support for transactional process and web service collaboration, service publishing and discovery.

To support transactional B2B collaborations, various transaction models and protocols have been proposed by extending conventional transaction models to support long-running, distributed and autonomic business processes and web services. Amongst many proposals, the Saga model [12] is a popular choice. Saga is a kind of long lived transactions (LLTs) that can be expressed as a sequence of transactions, of which each transaction can interleave with other transactions. By the problem nature, there is no general solution to the problem of low performance of LLTs, as well as a high rate of abortion and deadlock. Saga [12] alleviates this problem by relaxing the requirements for specific applications. In Saga, each sub-transaction preserves consistency in the database sense. However, transactions in a Saga are tightly related that they should be executed as a unit. An incomplete execution of a Saga is undesirable and should be compensated. To do this, each Saga transaction is paired with a compensating transaction, which semantically undoes every performed tasks of the incomplete transaction. A compensating transaction does not necessarily rewind the system to the state before the execution of the Saga.

Many LLT models have been proposed to extend the Saga model [12]. The global transaction model [13][19] facilitates an arbitrary distribution of and flexible rollback semantics of business processes over multiple workflow management systems. It supports the complete rollback semantics of [12] and extends the latter with the notion of partial rollback. Complete rollback undoes all the finished tasks; while partial rollback undoes some steps and rolls the process back to a step called a safe-point. A safe-point of a process allows the process to choose alternate courses for forward recovery. In addition, there are existing proposals to handle concurrent abortions of processes. Well-known works include flexible transactions [1], nested transaction model [8][9], the X-transaction model [20], multi-level transactions [21], to name a few.

There are other extended transaction models [11]. They support complex transaction structures and the relaxation of the ACID properties. However, they are database-centric instead of process-centric, and primarily aimed at preserving the consistency of shared data. Thus, they are generally inapplicable for applications comprising loosely coupled, web-based business services, in which the business process recovery is as important as the data recovery.

Besides the study on LLT models, many progresses have been made on the transactional protocols for process collaboration and web services. Business Transaction Protocol (BTP) [2] is proposed by the Organization for Advance Structured Information Systems (OASIS). It meets the requirements for long running collaborative business applications, at the same time, relaxes the traditional ACID properties in a controlled manner. They are

specific to the Web Services environment. To ensure the atomicity property amongst multiple participants, BTP uses a two-phase outcome protocol. This protocol supports two kinds of transaction, namely the atomic and the cohesive. The atomic type of transaction guarantees its outcome consistent by either accepting or rejecting the entire work all together. The cohesive type of transaction relaxes the atomicity requirements by permitting the final task of every process of a transaction to accept or reject its own part of work based on respective business rules. It facilitates every participant consistent by itself, and does not enforce the traditional atomicity property.

Web services coordination (WS-C) [23] is jointly proposed by IBM, Microsoft and BEA. Its aim is to define a generic framework for abstract modeling the coordination of entities. Concrete agents are required to concretize the abstract coordination protocols. WS-C can support a variety of coordination protocols such as WS-Transaction (WS-T) [24]. WS-T has two transaction models, namely atomic transactions and business activities. Atomic transactions assume the ACID semantics. Therefore, the acquired resources are locked during a transaction. Business activities are designed for long-running transactions and do not support the notion of two-phase commit. Any updates are committed immediately, reducing significantly the holding period of the available resources. It uses compensating tasks to bring the system into a consistent state.

Lastly, we review related work in the area of service publishing and discovery. Although there are many models, such as UDDI [18], to govern syntactic aspect of service collaborations, yet only a few research efforts have focused on service discovery with quality of services (QoS) constraints. Cardoso et al. [3] introduce several useful models for the measurement of QoS (such as time, cost, and reliability) aspects of workflow tasks. Based on these quality dimensions, they propose an approach to matching services for each QoS dimension according to fitness metrics [4]. Deora et al. [10] extend function-based approaches to service discovery by allowing service providers and consumers to express their QoS promises and requirements in service publishing and discovery. They use an ontology approach to specifying the required QoS attributes and their relations to the services. Wohlstadter et al. [22] tackle the QoS problem in service matchmaking by using a middleware based methodology. Their model allows services to use a meta-level protocol to negotiate with one another and exchange QoS policy information among them. This allows services to compute the agreeable common sets of QoS policies.

# 3. METHODOLOGY

## 3.1 B2B collaboration and exception handling

As pointed out in Section 1, business processes collaborate by message exchanges through their public views. In our model, we assume synchronous process communication. As depicted in Figure 2, the detailed behavior of a process is hidden from other processes, exposing only publicly accessible ports, useful silent actions, and their properties. A *port* is an action in private process to communicate with other processes. The other internal non-port actions are renamed as useful *silent actions* in the public views.

We adopt the replacement exception model [26] for private processes, where exception handler semantically replaces the failed task and resumes the execution of the process. Moreover, if an exception occurs without an accustomed exception handler, the default exception handler would abort the current execution and roll the process back by executing the compensating tasks of the executed tasks in the reverse order. As exceptions may propagate from one process to others through communication ports, to keep the autonomy of involved organizations, we assume that exception handlers are local to each organization.

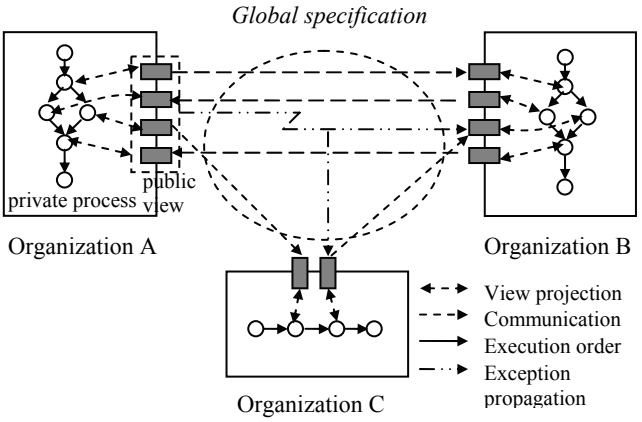


**Figure 2. B2B collaboration model.**

As explained, exception handlers semantically replace the failed tasks in the replacement exception model, we therefore consider the compensability and retriability properties of a task and its exception handlers as a composite unit. We then use the rules given by Hagen and Alonso [14] to compute the compensability and retriability properties of the composite unit. Next, we substitute the original properties of the task by the corresponding properties of the composite unit, and omit the exception handlers of the task in subsequent atomicity sphere analysis.

For example, suppose task *a* is a compensable and non-retriable task associated with an exception handler *b*, which is non-compensable and retriable. The composite properties thus computed would be non-compensable and retriable. The task *a* is therefore considered as non-compensable and retriable in the checking of atomicity sphere.

We note the following according to the replacement exception model: If an exception occurs internal to a task, the task is non-retriable. If an exception handler aborts the process or propagates the exception to another handler, then we mark the first exception handler as non-retriable. This simplification does not change the analysis result of atomicity sphere [14] but enables us to focus on the tasks in process flows, relieving us from considering the properties of exception handlers in our subsequent checking of atomicity sphere.

Hence, analogue to the criterion for guaranteed termination proposed in [17], a process in the above model satisfies the atomicity sphere property if and only if no non-retriable tasks are executed after some non-compensable task in every possible execution of this process. We note that if this condition is satisfied, then the process should be able to terminate successfully. Under this condition, in the case of any failure, the process should be able to proceed by either (*i*) internal retrials of any retriable task even though the process has executed a pivot task, or (*ii*) the compensation of all executed tasks prior to the commitment of any pivot task. On the other hand, if this condition is violated, then there would be a non-retriable task which is executed after some non-compensable task. The failure of this non-retriable task will abort the execution of the process. However, as certain non-compensable tasks have been executed, the process should already violate the semantics of all or nothing.

We introduce a type of process algebra in the next section to derive atomicity-equivalent public views from private processes. By checking atomicity sphere in the composition of these public views instead of the private processes, B2B organizations can identify whether or not the participating processes satisfy atomicity sphere when they collaborate.

## 3.2 Atomicity-equivalent process algebra PA$^a$

In this section, we introduce a novel process algebraic model to derive atomicity-equivalent public views from private processes. Here we first introduce some basic definitions, and without the loss of generality, let constant *0* be the termination state.

***Definition 1***: A process space is a triple $(P, A, F)$, where $P$ is a set of states, $A$ is a set of actions, $F$ ($\subseteq P \times A \times P$) is a ternary transition relation. Note that $0 \in P$.

In this paper, an *action* refers to the commitment of an executing task. Actions occur instantaneously. 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 $a \cdot 0$.

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

$$p \; R \; p$$
$$p \; R \; p' \wedge (p', \alpha, p'') \in F \; \vdash p \; R \; p''$$

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 $p^*$.

***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. We also call the process as the process $p$. We assume that a B2B process always terminates.

Since our interest is to model and check the atomicity sphere property, we therefore associate every state with a derived predicate, *atomicity*, which is false if and only if the state has been confirmed to violate the atomicity sphere property. In this paper, we will formulate the method to derive the value of this predicate. Once a state of a process is confirmed to violate the atomicity sphere property, the state is denoted by $\phi$. The process would also be regarded as violating the atomicity sphere property. In the sequel, for brevity and without the loss of generality, we use $\phi$ instead of the actual state(s) violating the atomicity sphere property because we do not need to differentiate these states in the analysis of a process's atomicity sphere property.

***Definition 4***: Several basic operators are defined as follows that can be used on processes with the precedence "·", "||", "+".

"·": a prefix operator on processes
"+": a choice composition operator on processes
"||": a parallel composition operator on processes

In a process, an action is called *port action* if and only if this action is designed to communicate with other processes; otherwise it is called a *non-port action*. A process $p$ with a set of port actions $H$ is donated as $p_H$. For the sake of simplicity for the discussion purpose, we assume that two port actions from different processes with the same name represent that they communicate with each other synchronously. We also assume that any port action would not share the same name with any non-port action. In a public view, non-port actions are usually renamed as *silent actions* to encapsulate unnecessary details. There are four kinds of silent actions in our framework, $\tau_{c,r}$, $\tau_{nc,r}$, $\tau_{c,nr}$, $\tau_{nc,nr} \in A$. They cover all possible combinations of compensability and retriability properties (see the explanation in Section 3.3). Based on the above definitions, axioms of the algebra are given as follows (denoted as PA$^a$).

*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*) |
| $(a + b) \cdot z = a \cdot z + b \cdot z$ | (*A4*) | $\tau_{c,nr} \cdot \tau_{c,nr} \cdot x = \tau_{c,nr} \cdot x$ | (*R4*) |
| $x + 0 = x$ | (*A5*) | $\tau_{c,nr} \cdot \tau_{nc,nr} \cdot x = \tau_{nc,nr} \cdot x$ | (*R5*) |
| $x + \phi = \phi$ | (*A6*) | $\tau_{nc,nr} \cdot \tau_{nc,r} \cdot x = \tau_{nc,nr} \cdot x$ | (*R6*) |
| $x \,\|\| \, y = y \,\|\| \, x$ | (*M1*) | $\tau_{nc,r} \cdot \tau_{c,nr} \cdot x = \phi$ | (*R7*) |
| $(x+y) \,\|\| \, z = x\|\|z + y\|\|z$ | (*M2*) | $\tau_{nc,r} \cdot \tau_{nc,nr} \cdot x = \phi$ | (*R8*) |
| $0_{H1} \,\|\| \, 0_{H2} = 0_{H1 \cup H2}$ | (*M3*) | $\tau_{nc,nr} \cdot \tau_{c,nr} \cdot x = \phi$ | (*R9*) |
| $x \,\|\| \, \phi = \phi$ | (*M4*) | $\tau_{nc,nr} \cdot \tau_{nc,nr} \cdot x = \phi$ | (*R10*) |
| $a \cdot x_{H1} \| 0_{H2} = a \cdot (x_{H1} \| 0_{H2})$ $(a \notin H_1 \cap H_2)$ | (*M5*) | $\tau_{nc,r} \cdot \phi = \phi$ | (*R11*) |
| | | $\tau_{c,nr} \cdot \phi = \phi$ | (*R12*) |
| $a \cdot x_{H1} \| 0_{H2} = 0_{H1 \cup H2}$ $(a \in H_1 \cap H_2)$ | (*M6*) | $\tau_{nc,nr} \cdot \phi = \phi$ | (*R13*) |

$a \cdot x_{H1} \| b \cdot y_{H2} = a \cdot (x_{H1} \| b \cdot y_{H2}) + b \cdot (y_{H2} \| a \cdot x_{H1})$ $(a, b \notin H_1 \cap H_2)$ (*M7*)
$a \cdot x_{H1} \| b \cdot y_{H2} = a \cdot (x_{H1} \| b \cdot y_{H2})$ $(a \notin H_1 \cap H_2, b \in H_1 \cap H_2)$ (*M8*)
$a \cdot x_{H1} \| b \cdot y_{H2} = a \cdot (x_{H1} \| y_{H2})$ $(a, b \in H_1 \cap H_2, a = b)$ (*M9*)
$a \cdot x_{H1} \| b \cdot y_{H2} = 0_{H1 \cup H2}$ $(a, b \in H_1 \cap H_2, a \neq b)$ (*M10*)

Axioms A1-A5 define the behavior of the operators "+" and "·". Axioms M1-M10 (but M4) define the behavior of the operator ||. M6 states that the composite process would terminate if a process with a leading port action attempts to communicate with a terminated process. Axiom M7 states that if the leading actions of the two processes are not port actions having the same name, they would interleave. Axiom M8 represents that one process is waiting to communicate with another process. Axiom M9 represents that two processes are synchronized on their leading port actions. If the two port actions do not share the same name (Axiom M10), the composite process would terminate. Axioms R1-R13 represent the simplification rules for silent actions. Axioms A6 and M4 represent situations that violate the atomicity sphere property. In this algebra, we denote PA$^a$ $\vdash p_1 \rightarrow p_2$ if $p_1$ can be reduced to $p_2$ in the usual sense using axioms of PA$^a$.

***Definition 5***: A complete trace of a process is a sequence, possibly empty, of actions executed from its initial state to one of its end states. For a process $p$, *trace*($p$) denotes the set of all the complete traces of process $p$. For a trace $t$, $i$ is an integer larger than 0, $t[i]$ denotes the action in the $i^{th}$ position of $t$.

## 3.3 Semantics model $\mathcal{M}$

To explain the atomicity-equivalent process algebra, we introduce a semantics model of the algebra, denoted as $\mathcal{M}$. In this model, every action $a \in A$ has two essential properties: Compensability and Retriability. Predicate C($a$) is true if and only if the action $a$ is compensable, and predicate R($a$) is true if and only if the action $a$ is retriable. The properties of silent actions $\tau_{c,r}$, $\tau_{nc,r}$, $\tau_{c,nr}$ and $\tau_{nc,nr}$ are given in Table 1. They represent four kinds of silent actions with all possible combinations of the compensability and retriability properties.

**Table 1. Properties of actions $\tau_{c,r}$, $\tau_{nc,r}$, $\tau_{c,nr}$, $\tau_{nc,nr}$.**

| Property \ Action | $\tau_{c,r}$ | $\tau_{nc,r}$ | $\tau_{c,nr}$ | $\tau_{nc,nr}$ |
|---|---|---|---|---|
| Compensability | true | false | true | false |
| Retriability | true | true | false | false |

Based on this semantics model, we define the criterion for checking the satisfaction of atomicity sphere of processes in this algebra model formally. Informally, a process in this model

satisfies atomicity sphere if and only if it does not contain any $\phi$ state and no non-retriable tasks are executed after some non-compensable task in any of its complete execution traces. This is formalized as follows:

***Definition 6***: $\varphi$ is a predicate of a process. $\varphi(p)$ is true if and only if the following conditions hold:

$\phi \notin p^*$

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

Otherwise, $\varphi(p)$ is false. For any processes $p_1$, $p_2$, if $\varphi(p_1) = \varphi(p_2)$, we say that $p_1$ and $p_2$ are *atomicity-equivalent*. The following theorem shows that if a process can be reduced to another in $PA^a$, then they are atomicity-equivalent.

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

In order to prove this theorem, we introduce an auxiliary function $\psi$ to differentiate processes and partition the entire state space of any process into five types based on their complete traces.

***Definition 7***: $\psi$ is a function on a process $p$. Case (*i*): If $\varphi(p)$ = false, then $\psi(p) = -1$; otherwise, we have the following four additional cases. Case (*ii*): $\psi(p) = 0$ if, for every possible complete trace of $p$, there does not exist any non-compensable and non-retriable action. Case (*iii*): $\psi(p) = 1$ if some complete trace(s) of $p$ contains some non-compensable action(s), yet there does not exist any non-retriable action in any of its complete traces. Case (*iv*): $\psi(p) = 2$ if some complete trace(s) of $p$ contains some non-retriable action(s), yet there does not exist any non-compensable action in any of its complete traces. Case (*v*): $\psi(p)$ = 3 if both non-compensable and non-retriable actions are contained in its complete traces (not necessarily in the same trace), yet $p$ does not violate atomicity sphere.

Obviously, $\psi(p_1)=\psi(p_2) \Rightarrow \varphi(p_1)=\varphi(p_2)$. Hence, we only need to prove that if $PA^a \vdash p_1 \rightarrow p_2$ then $\mathcal{M} \vdash \psi(p_1)=\psi(p_2)$.

Here we outline the proof of this theorem; details of the proof can be found in [25]. We only need to prove that to reduce a given process $p$ to $p'$ by using every single axiom of $PA^a$, $\varphi(p) = \varphi(p')$. The proof contains three steps. In the first step, we prove that for each axiom "x=y" in this algebra, $\psi(x) = \psi(y)$. This is easy to verify. Then we prove that if $p \rightarrow p'$ and $\psi(p) = \psi(p')$, then for any action $a$ and process $q$, we have $\psi(a \cdot p) = \psi(a \cdot p')$, $\psi(q+p) = \psi(q+p')$, and $\psi(q||p) = \psi(q||p')$. Based on the conclusions of the first two steps, we prove that $PA^a \vdash p \rightarrow p' \Rightarrow \mathcal{M} \vdash \psi(p) = \psi(p')$ by the mathematical induction on the number of steps in process reduction. Hence the reduction using $PA^a$ preserves the outcome of $\varphi$. Suppose that the reduction of $p_1$ into $p_2$ uses $n$ axioms, then the reduction can be divided into $m$ sub-reductions, each of which uses a single axiom, that is, $p_1 \rightarrow p^{(1)}$, $p^{(1)} \rightarrow p^{(2)}$, …, $p^{(m-1)} \rightarrow p^{(m)}$, and $p^{(m)}=p_2$. Hence, $\varphi(p_1) = \varphi(p^{(1)}) = \ldots = \varphi(p^{(m)}) = \varphi(p_2)$.

***Corollary 1***: $\forall p_1, p_2, \ldots, p_n \in P$, $PA^a \vdash p_1 \rightarrow pv_1$, $PA^a \vdash p_2 \rightarrow pv_2, \ldots$, $PA^a \vdash p_n \rightarrow pv_n$, $\Rightarrow \varphi(p_1||p_2||\ldots||p_n) = \varphi(pv_1||pv_2||\ldots||pv_n)$.

This theorem shows that we can use axioms of $PA^a$ to simplify a process and preserve its atomicity sphere property. Based on the algebra and its semantics model, we introduce a renaming operator in next section to derive atomicity-equivalent public views from private processes. We will show that these public views can be used to check the satisfactory of atomicity sphere in the global analysis, instead of using the private processes.

## 3.4 Atomicity-equivalent public view and composition

In this section, we introduce two operators used to generate atomicity-equivalent public views from private processes, and compose them in an atomicity-equivalent way.

***Definition 8***: $\sigma_H$ is an operator that renames a process into another, where $H \subseteq A$ is the set of actions that keep intact in the renamed process. $\sigma_H$ is defined inductively as follows:

1) $\sigma_H(0) = 0$, $\sigma_H(\phi) = \phi$

2) $\forall a \in A$

$If\ a \in H$, $\sigma_H(a \cdot x) = a \cdot \sigma_H(x)$.

$If\ a \notin H$, $\sigma_H(a \cdot x) = \tau_{c,r} \cdot \sigma_H(x)$ *if* $C(a)=true \wedge R(a)=true$

$\tau_{nc,r} \cdot \sigma_H(x)$ *if* $C(a)=false \wedge R(a)=true$

$\tau_{c,nr} \cdot \sigma_H(x)$ *if* $C(a)= true \wedge R(a)=false$

$\tau_{nc,nr} \cdot \sigma_H(x)$ *if* $C(a)= false \wedge R(a)=false$

3) $\forall x, y \in P$, $\sigma_H(x + y) = \sigma_H(x) + \sigma_H(y)$

Operator $\sigma_H$ renames all the actions of a process into silent actions except the port actions defined in $H$. We use this operator to rename a process first, and then reduce the renamed process under the framework of $PA^a$ to generate the public view, that is, $PA^a \vdash \sigma_H(p) \rightarrow pv$, then $pv$ is the public view of $p$. The following theorem shows that the public views generated in this way preserves the atomicity sphere property, we called them *atomicity-equivalent* public views:

***Theorem 2***: Let $p$ be a process, and $PA^a \vdash \sigma_H(p) \rightarrow pv$, then $\mathcal{M} \vdash \varphi(p)=\varphi(pv)$.

***Proof***: Based on Theorem 1, $PA^a \vdash \sigma_H(p) \rightarrow pv \Rightarrow \mathcal{M} \vdash \varphi(\sigma_H(p))=\varphi(pv)$. Hence we only need to prove $\mathcal{M} \vdash \psi(p) = \psi(\sigma_H(p))$ to infer $\mathcal{M} \vdash \varphi(p) = \varphi(\sigma_H(p)) \Rightarrow \mathcal{M} \vdash \varphi(p) = \varphi(\sigma_H(p)) = \varphi(pv)$. The proof of $\psi(p) = \psi(\sigma_H(p))$ is simple because operator $\sigma_H$ only renames the actions but does not change their compensability and retriability properties. The result follows.

***Definition 9***: The global composition specification (GS) of collaborative processes is the set of all the pairs of port actions in the B2B model.

For every pair $(a, b) \in GS$, $a$ and $b$ are port actions of processes $p_1$ and $p_2$ respectively. It means that processes $p_1$ and $p_2$ communicate by synchronizing ports $a$ and $b$. Let $H_i$ be the set of port actions of the collaborative process $p_i$. GS is said to be well-formed if and only if $\forall a_1 \in H_i$: $(\exists a_t \in H_j\ (i \neq j)$: $((a_1, a_t) \in GS$ or $(a_t, a_1) \in GS) \wedge (\neg\exists a_k, a_k \neq a_t$: $(a_1, a_k) \in GS$ or $(a_k, a_1) \in GS))$.

In the remainder of this paper, we assume that GS is well-formed. Here is the formal definition of the composition operator for B2B collaboration.

***Definition 10***: $\oplus_{GS}$ is a composition operator which composes $n$ business processes into a composite process using the global specification GS. $\oplus_{GS}(p_1, p_2, \ldots, p_n)$ is defined as follows:

1) Let $H_1, H_2 \ldots, H_n$ be the sets of port actions of processes $p_1$, $p_2$, …, $p_n$ respectively. Mapping function $\gamma_{GS}$ is defined as $A \times A \rightarrow A$. $\gamma_{GS}(a, b)$ is defined if $(a, b)$ or $(b, a) \in GS$. We rename the actions of $p_1, p_2, \ldots, p_n$ to produce $p_1', p_2', \ldots, p_n'$ respectively by the following approach:

$\forall a \in H_i$, if $\exists b \in H_j$, $\gamma_{GS}(a, b)$ is defined, then we replace action $a$ of $p_i$ with action $c = \gamma_{GS}(a, b)$. The properties of $c$ is defined as $C(c) = C(a) \& C(b)$, $R(c) = R(a) \& R(b)$. The name and the properties of other actions remain unchanged.

2) Let $H_1'$, $H_2'$, …, $H_n'$ be the sets of port actions of $p_1'$, $p_2'$,…, $p_n'$ respectively, that is, $H_i' = \{c \mid (\exists a \in H_i, b \in H_j, c = \gamma_{GS}(a, b))$ or $(c \in H_i, \neg\exists b \in H_j, \gamma_{GS}(c, b)$ is defined)$\}$. $\oplus_{GS}(p_1, p_2, \ldots, p_n)$ is defined as $p_1' \parallel p_2' \parallel \ldots \parallel p_n'$.

Operator $\oplus_{GS}$ is used to compose business processes in the B2B environment. Since a public view of a private process is also a process, so $\oplus_{GS}$ 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 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 public views, and GS be the global specification, that is, $PA^a \vdash \sigma_{H1}(p_1) \rightarrow pv_1$, $PA^a \vdash \sigma_{H2}(p_2) \rightarrow pv_2, \ldots$, $PA^a \vdash \sigma_{Hn}(p_n) \rightarrow pv_n$ then $\mathcal{M} \vdash \phi(\oplus_{GS}(p_1, p_2, \ldots, p_n)) = \phi(\oplus_{GS}(pv_1, pv_2, \ldots, pv_n))$.

To prove this theorem, let us introduce a lemma.

***Lemma 1***: Let $p$ be a process, and $H$ be the set of port actions of $p$, then for any process $q$, $\phi(\sigma_H(p) \parallel q) = \phi(p \parallel q)$.

***Proof***: We only need to prove $\psi(\sigma_H(p) \parallel q) = \psi(p \parallel q)$. We know that $\sigma_H(p)$ does not rename those port actions. Thus, every complete trace of $p \| q$, after replacing those non-port actions of $p$ with the corresponding silent action $\tau_{x,y}$ ($x \in \{c, nc\}$, $y \in \{r, nr\}$) is also a complete trace of $\sigma_H(p) \parallel q$, and vice versa.

The proof of Theorem 3 follows directly from Lemma 1 and Corollary 1. Theorem 3 shows that we can use atomicity-equivalent public views of private processes to check atomicity sphere, instead of using the private processes directly. Therefore, service providers are flexible to publish their atomicity-equivalent public views in the service registries for service matching, encapsulating the details of the process. In the next section, we discuss the algorithms of constructing and composing public views, as well as the checking of their atomicity sphere properties.

## 3.5 Algorithms

This section introduces several algorithms to construct and compose public views, and check their atomicity sphere properties.

Let $pp = (s_{10}, S_1, A_1, F_1)$ be a private process, and $H$ be a set of port actions. The algorithm for constructing atomicity-equivalent public view $pv = (s_{20}, S_2, A_2, F_2)$ is given as follows. The algorithm has two steps. Firstly, all the actions of process $pp$ are renamed based on the definition of $\sigma_H$ (Lines 2-8). Then, the BFS algorithm [6] is used to traverse the process from the termination state to its initial state. During the traversal, the algorithm simplifies the process using Axioms of $PA^a$. (A purpose of traversing the process in the reverse direction is to prevent the repeated application of the axiom A4.) Lines 14-17 simplify the view being constructed by using axioms R1 and R11-R13. On 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. On line 16, the function *merge* combines states $cs$ and $ps$. If $cs$ is marked as $\phi$, then axiom A6 is used to merge $cs$ and $ps$. Line 17 back-tracks one step to guarantee complete reduction in the provision of a $\tau_{c,r}$ is in between two silent actions. Lines 18-29 simplify the process by using axiom A4. The function *canCombine* on line 20 gives true if and only if $c$ and $a$ can match the actions on the left hand side of an axiom of R2-R6; otherwise it gives false. The function *combine* on line 21 returns the action on the right hand side of the applied axiom. The function *violateAS* on line 22 gives true if and only if $c$ and $a$ can match the actions on the left hand side of any axiom of R7-R10; otherwise it gives false.

***Algorithm 1***: (Construct an atomicity-equivalent public view)

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. $A_2 = A_2 - \{a\} + \{b\}$
5. *for every* $(s_1, a, s_2) \in F_1$
6. $F_2 = F_2 - \{(s_1, a, s_2)\} + \{(s_1, b, s_2)\}$
7. *endfor*
8. *endfor*
9. $\forall s_i = 0$, *stack.push*$(s_i)$, $\forall s_i \in S$, *visited*$(s_i) \leftarrow$ *false*.
10. *while(not empty(stack))*
11. $cs \leftarrow$ *stack.pop()*
12. *visited(cs)* $\leftarrow$ *true*
13. *for every* $(ps, a, cs) \in F_2$
14. *if canSimplify(a, cs)*
15. $F_2 = F_2 - \{(ps, a, cs)\}$
16. *merge(ps, cs)*
17. *if* $a = \tau_{c,r} \wedge cs \neq \phi, \forall (cs, b, ns) \in F_2$, *stack.push(ns)*, *endif*
18. *else if* $\forall (ps, b, ns) \in F_2 \Rightarrow a = b \wedge ns = cs$
19. *for every* $(s, c, ps) \in F_2$
20. *if canCombine(c, a)*
21. $d \leftarrow$ *combine(c, a)*, $F_2 = F_2 - \{(s, c, ps)\} + \{(s, d, cs)\}$
22. *else if violateAS(c, a)*
23. $F_2 = F_2 - \{(s, c, ps)\}$, *replace*$(s, \phi)$
24. *endif*
25. *endif*
26. *endfor*
27. *if* $\neg\exists (s, c, ps) \in F_2$, $F_2 = F_2 - \{(ps, a, cs)\}$, *endif*
28. *endif*
29. *if visited(ps) = false*, *stack.push(ps)*, *endif*
30. *endfor*
31. *endwhile*

Given two processes $p_1=(s_{10}, S_1, A_1, F_1)$, and $p_2=(s_{20}, S_2, A_2, F_2)$. Let $H_1$ and $H_2$ be the sets of port actions of $p_1$ and $p_2$ respectively, and GS be the global specification. $p = \oplus_{GS}(p_1, p_2) = (s, S, A, F)$. The algorithm for constructing $p$ is shown in Algorithm 2. This algorithm constructs $p$ from its initial state, which is composed of the initial states of $p_1$ and $p_2$. For every state of $p$, which is composed of two states of $p_1$ and $p_2$, lines 6-19 calculate the new states created by a private transition from the current state of process $p_1$, and lines 20-26 calculate the new states created by the private transition from the current state of process $p_2$. If the transition represents a communication, then lines 7-13 create a new state by transferring current states of $p_1$ and $p_2$ to respectively target states. If the current states of $p_1$ and $p_2$ are both the termination state *0*, then the new state is also *0*. If the current state of either process is marked as $\phi$, then the new state is $\phi$ according to axiom M4.

***Algorithm 2***: (Compose two processes)

1. $S \leftarrow \{\}$, $A \leftarrow \{\}$, $F \leftarrow \{\}$.
2. *if* $s_{10} = \phi \vee s_{20} = \phi$, $s \leftarrow \phi$, *quit.*
3. *else* $s \leftarrow (s_{10}, s_{20})$. *stack.push(s)*. $S \leftarrow S + \{s\}$. *endif*
4. *while(not empty(stack))*
5. $cs = (s_{1i}, s_{2j}) \leftarrow$ *stack.pop()*.
6. *for every* $(s_{1i}, a, s_{1t}) \in F_1$,
7. *if* $a \in H_1 \wedge \exists b \in H_2$, *(a, b) or (b, a)* $\in GS$
8. *if* $(s_{2j}, b, s_{2k}) \in F_2$
9. $ns \leftarrow (s_{1t}, s_{2k})$, $S \leftarrow S + \{ns\}$, $c \leftarrow \gamma_{GS}(a, b)$,
10. $A \leftarrow A + \{c\}$, $F \leftarrow F + \{(cs, c, ns)\}$.
11. *if* $s_{1t} = 0 \wedge s_{2k} = 0$, $ns \leftarrow 0$ *endif*
12. *if* $s_{1t} = \phi \vee s_{2k} = \phi$, $ns \leftarrow \phi$, *else stack.push(ns) endif*
13. *endif*
14. *else*
15. $ns \leftarrow (s_{1t}, s_{2j})$, $S \leftarrow S + \{ns\}$, $A \leftarrow A + \{a\}$, $F \leftarrow F + \{(cs, a, ns)\}$

16. *if* $s_{1t}=0 \wedge s_{2k}=0$, *ns*←*0*, *endif*
17. *if* $s_{1t}=\phi \vee s_{2k}=\phi$, *ns*←$\phi$, *else stack.push(ns) endif*
18. *endif*
19. *endfor*
20. *for every* $(s_{2j}, b, s_{2k}) \in F_2$,
21. *if* $b \notin H_2 \vee \neg \exists\, a \in H_1$, *(a, b) or (b, a)*$\in GS$,
22. *ns* ←$(s_{1i}, s_{2k})$, $S \leftarrow S + \{ns\}$, $A \leftarrow A + \{b\}$, $F \leftarrow F + \{(cs, b, ns)\}$
23. *if* $s_{1t}=0 \wedge s_{2k}=0$, *ns*←*0*, *endif*
24. *if* $s_{1t}=\phi \vee s_{2k}=\phi$, *ns*←$\phi$, *else stack.push(ns) endif*
25. *endif*
26. *endfor*
27. *endwhile*

With algorithms 1 and 2, one can construct and compose the public views of collaborative processes. Given a process $p=(s, S, A, F)$, we use the following algorithm (Algorithm 3) to mark and traverse the process to check its atomicity sphere: Line 1 initializes all states. $rb(s_i)$ is a predicate which is marked as true for rollbackable states (*rb* for short), which means that all the executed tasks from the initial state to this state are compensable. Otherwise, it is false. On line 2, the function *markrb* marks all the states that are unable to roll back. Line 3 resets the visited flag of all states. On line 4, the function *check* checks either (*i*) all the states (which is not rollbackable) if there exists one non-retriable transition leading from the given state or (*ii*) there exists a state marked as $\phi$. If this condition is satisfied, then the process does not satisfy atomicity sphere. *NC(A)* on line 5 represents the set of non-compensable actions in *A*.

***Algorithm 3***: (Check atomicity sphere)

1. $\forall s_i \in S$, $rb(s_i) \leftarrow true$, $visited(s_i) \leftarrow false$
2. *markrb()*
3. $\forall s_i \in S$, $visited(s_i) \leftarrow false$.
4. *check(s)*.

*markrb():*

5. *for every* $a \in NC(A)$,
6. *for every* $(s_1, a, s_2) \in F$,
7. $rb(s_2) \leftarrow false$
8. *if* $visited(s_2)=false$
9. *markreachable*$(s_2)$.
10. *endif*
11. *endfor*
12. *endfor*

*markreachable(s):*

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

*check(s):*

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

# 4. CASE STUDY

In this section, we illustrate our approach by using a sample business transaction scenario depicted in Figure 3, in which the "Manufacturer", the "Supplier", the "Bank" and the "Shipper" are all Web services providers.

This example is taken from the online services provided by a gift manufacturer [16]. It allows users to design and order customizable gifts from their websites. To develop such an application, the manufacturer first looks up three services, i.e., supplier, bank and shipper, from an online e-commerce market, where merchants publish their services for collaboration. Each service is driven by an internal private process.

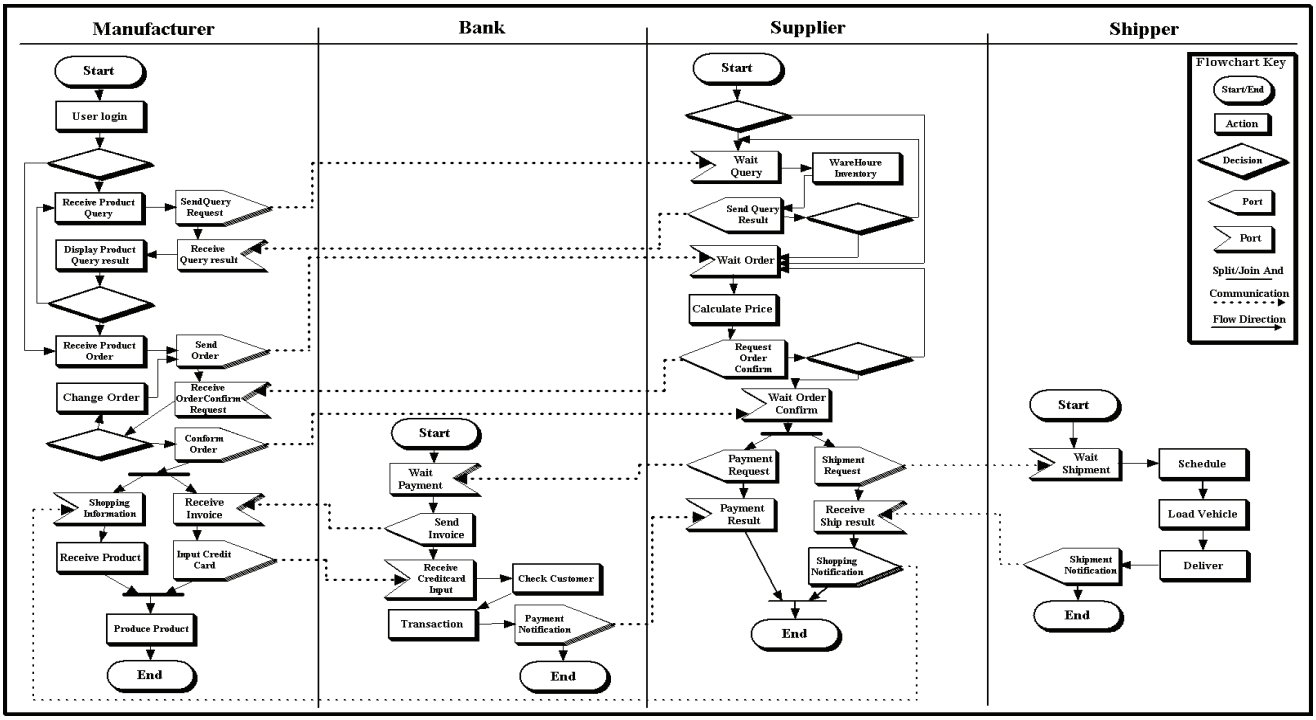


**Figure 3. Process collaboration amongst multi-organizations.**

Their collaboration is illustrated in Figure 3. On receiving a purchase order, the manufacturer first queries the supplier about the stock availability and the price of OEM products, and then places orders to the supplier if appropriate. After calculating the prices of purchased products and shipment fees, the supplier launches a payment transaction via a bank. The bank then sends an invoice to the manufacturer who pays the bills by its credit card. At the same time, the supplier sends a request to the shipper to arrange the shipment of the products. Once the products are aboard, the supplier notifies the manufacturer about the invoice of products and the estimated arrival date of the shipment. The manufacturer then receives the products and further processes these products based on the purchase order.

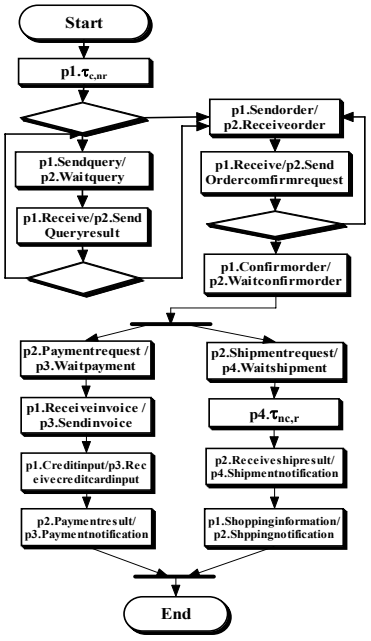


**Figure 4. Composition of public views.**

In this scenario, traditional transaction approaches are unsuitable to keep consistency for the sake of performance and autonomy, because these processes belong to different organizations and may take a long time to complete. For example, to ensure the products are available for this transaction, products would be locked from making a query until the transaction is completed or canceled. However, the manufacturer may take a long time to make a purchase decision. As a result, the locked resources in the supplier process reduce the degree of potential concurrent executions of the services. This is not in the interest of the merchant. Other advanced transaction models, such as Saga and global transaction models, relax the atomicity requirements, yet they are still incompetent for this scenario. This is because each sub-transaction in these models should have a compensating transaction, which may be impractical, since some tasks (e.g., task "deliver" in this scenario) may be non-compensable and thus have no compensating tasks.

Approaches based on exception handling do not incur such restrictions. For example, locks are not needed when the manufacturer queries. On receiving a purchase order, the supplier checks the stock. If the products are out of stock, then an exception will be resulted. An associated exception handler would be invoked to resolve the inconsistency by either aborting all the processes or waiting for incoming products. However, aborting processes in different organizations requires compensating all executed tasks. This does not guarantee consistent and desirable consequences. For example, during the payment transaction process, an exception may be raised if the drawing accounts has insufficient amount of credits. This exception would be propagated to the supplier process, which aborts all the processes due to its business policy. However, the products may have already been shipped by the shipper. The cost of compensating task "deliver" is thus expensive. As a result, the collaborators may oblige to bear the value loss for the shipper.

Transactional protocols like BTP tackle this problem by committing tasks in different organizations all together. For example, the task "transaction" in the bank process and the tasks "schedule", "load vehicle", and "deliver" in the shipper process should be committed all together by, for example, using the two phase outcome protocol. This guarantees either all of these tasks or none of them to be executed semantically. However, this approach sacrifices the autonomy of these services.

An alternative is to detect and prevent this kind of problem in advance. This may require a global analysis of atomicity sphere of the four participating processes in the service looking up stage. For example, task "deliver" in shipper is non-compensable because the cost of compensating the task is expensive. Task "payment result" is non-retriable because its exception handler will abort the process on receiving the exception "insufficient credits". From the perspective of the global process (by composing these four processes), the task "payment result" is non-retriable, which may be executed after the non-compensable task "deliver" in some trace of the global process. Therefore, this global process violates the atomicity sphere property.

However, the manufacturer does not know the details of the other three private processes. A global analysis based on all involved private processes is thus challenging. Our approach addresses this problem. Based on Theorem 3, the global analysis can be conducted using the composition of their atomicity-equivalent public views instead of the private processes. Algorithms introduced in Section 3.5 are used to derive public views from private processes. These public views could be published in the service registries. When looking up the three services in the service registries, the manufacturer can assess their public views and construct a global process by composing these public views, as depicted in Figure 4 (details can be found in [25]). Note that the properties of the task created by combining two communicating port actions (its name is derived from combining the names of the two actions. Prefix *p*i represents the id of the process. *p*1: manufacturer, *p*2: supplier, *p*3: bank, *p*4: shipper) are calculated by the logical-"and" of the properties of the two port actions. This process violates atomicity sphere because task "(*p*2.Payment result, *p*3.Paymentnotification)" is non-retriable (since "*p*2.Paymentresult" is non-retriable) and it may be executed after the non-compensatable task "$p4.\tau_{nc,r}$" in a trace of this process.

Based on the analysis result, the manufacturer can identify potential value-lost scenarios in advance and take measure to source other suppliers or shipping services, which does not incur such value-lost scenarios. This case study shows that our approach can prevent the selection of inappropriate services, which may lead to atomicity inconsistency and hence undesirable consequences. Compared to approaches using the conventional transactional models and protocols, our approach is more flexible in the sense that it can apply to not only, as in these approaches, situations that all details are exposed in the interfaces, but also situations some private actions are concealed (even they are known by other organizations).

# 5. EVALUATION

In this section, we analyze the time complexity of the algorithms introduced in Section 3.5 to evaluate the feasibility of our proposal. A comparison with related work is also presented. This is followed by discussions regarding the limitations and some possible extension of our work.

## 5.1 Complexity Analysis

### *5.1.1 Time cost for constructing public view*

Let $A$ be the set of actions of private process $pp$, $S$ be the set of states of private process $pp$, and $F$ be the set of transitions. Let $|S|=m$, $|F|=k$. In the first step (Lines 2-8), we want to rename the actions in all transitions of $pp$, so 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 algorithm from the termination state to the initial state, so all the states are visited at most twice (in case of the one-step backtracking), and thus the time cost is $O(2m)$. For each state of the view, we at most combine its every input transition with its output transition if the state has only one output transition (Lines 13-30). Hence the total number of combinations takes at most $k$ steps. So the time cost for the simplification step is $O(2m+k)$. As a result, the total time cost for constructing a public view is $O(2m+2k)$. Since in a process, $m \leq k+1$, so the time cost is $O(k)$.

### *5.1.2 Time cost for composing two processes*

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 $\oplus_{GS}(p_1, p_2)$ 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 $\oplus_{GS}(p_1, p_2)$, 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, time cost for calculating transitions of the new process (Lines 6-26) 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 $O(k_1 \times k_2)$.

### *5.1.3 Time cost for checking atomicity sphere*

Let $|S|=m$, and $|F|=k$. The time cost of step 1 is $O(m)$. In step 2, function *markrb* traverses all the states at most once, thus the time cost is $O(m)$. 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 $O(m+k)$. So, the total time cost is $O(k)$.

## 5.2 Comparison with related work

In contrast to the advanced transaction models and protocols, our work focuses on service publishing and discovery with atomicity sphere consistency. The aim is to provide business collaborators a criterion to find and choose suitable services. Our work is not based on these transaction models and protocols. Instead, exception handling mechanism is adopted to handle the inconsistency by performing backward recovery or forward recovery or both. This mechanism is similar to the Saga and the global transaction models. However, the Saga and global transaction models require each sub-transaction paired with a compensating transaction, which is hard to be satisfied in reality. Exception handling mechanism does not have such a constraint, allowing some tasks to be non-compensable.

On the other hand, although protocols like BTP can alleviate the conflicting rollback problem by using the two-phase outcome protocol to commit possible conflicting tasks all together so that either all or none of them can succeed semantically. However, these approaches will sacrifice the autonomy of the participating processes because all the involved processes have to commit their tasks all together. Other protocols like WS-C share similar problems. However, our work is more flexible because only public views of private processes are required, and the private processes do not have to conform to some specific protocols. As a result, the involved organizations are relatively more loosely coupled and their autonomies are respected to a greater extent.

In the field of service discovery with QoS constraints, to our best knowledge, few works have taken the atomicity aspect into account. Although the middleware based methodology proposed by Wohlstadter et al. [22] may be used by the collaborators to negotiate their transactional configurations, their work does not show the method to do so. We view that it is non-trivial, because atomicity sphere is different from the QoS aspects stated in [22] in the sense that atomicity sphere may be affected by all the involved processes. Our work contributes to using atomicity sphere as an aspect of QoS, and providing an approach to checking the atomicity sphere of all involved processes while preserving their privacies. This can discover potential value-loss in advance. Our work complements the works in this area.

## 5.3 Discussions

In our approach, we assume that there is no global exception handler across multi-organizations. All the exception handlers are local to private processes. If an exception affects several processes, this exception is propagated to corresponding processes. They then handle its exception fragment locally. In some scenarios, however, the handling of an exception may require the collaboration of several processes. The exception handlers in different processes may interact with one another. The interference is thus non-local. As a result, the composition of 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 property in these processes, we plan to extend current process algebra to describe exceptions and the handlers as a future work.

Another limitation of our approach is that our work supports implicit partial rollback. This is realized by encapsulating the tasks needing partial rollback into a composite unit. The partial rollback of these tasks is through the compensation of the corresponding composite unit, and the process resumes the execution by choosing an alternate task.

In addition, we assume no interference amongst internal actions of private processes. However, in some scenarios, these kinds of interference may exist and thus add to the difficulty in analyzing atomicity sphere. The present paper has not addressed analyze these scenarios adequately. We will study the interference amongst tasks in different processes and their effects on atomicity sphere as a future work.

# 6. CONCLUSIONS

This paper presents a novel approach to publishing and discovering services with atomicity sphere consistency for B2B collaboration. It provides a criterion for service requester to choose suitable collaborators in service looking up stage during their collaboration. This method is based on a novel process algebra, which, as we have proved mathematically, can be used to construct atomicity-equivalent public views from private processes. By checking whether the satisfactory of the atomicity sphere property in the composition of these public views is fulfilled, a global analysis of atomicity sphere can be conducted

even through some actions are deliberately concealed behind the public views. In this way, service providers need only release the public views of private processes, leaving business details and privacy invisible to others. Algorithms are also provided to derive public views from private processes, to compose them and to check their atomicity sphere.

The present work makes certain assumptions that might not hold in some complex scenarios. In future, we will relax these assumptions to improve the applicability of our approach. In addition, other real life case studies will be performed to investigate the usability and scalability in practice.

## 7. ACKNOWLEDGEMENTS

The authors would like to thank Prof. Jun Wei, Prof. Xinxin Liu and anonymous reviewers for their useful comments and suggestions. Thanks are also given to Sam Ng, Chang Xu, Xinming Wang, Du Li and other members of the software engineering group in The Hong Kong University of Science and Technology for their informative discussions.

## 8. REFERENCES

[1] Ansari, M., Ness, L., Rusinkiewicz, M., and Sheth, A. Using flexible transactions to support multi-system telecommunication applications. In *Proceedings of 18th International Conference on Very Large Data Bases.* Morgan Kaufmannn. 1992, pp.65–76. San Mateo, CA, USA.

[2] BTP. Available at: http://www.oasis-open.org/committees /download.php/9836/business_transaction-btp-1.1-spec-wd-04.pdf (accessed on 8 Jan 2006).

[3] Cardoso, J., Sheth, A., and Miller, J. Workflow quality of service. In *Proceedings of 12th International Conference on Enterprise Integration and Modeling Technology*. Kluwer Academic Publishers. 2002, pp.303–311. Norwell, MA, USA.

[4] Cardoso, J. and Sheth, A. Semantic e-workflow composition. *Journal of Intelligent Information Systems,* vol.21, no.3, Nov. 2003, pp.191–225.

[5] Cheung, S.C., and Kramer, J. Context Constraints for Compositional Reachability Analysis. *ACM Transactions on Software Engineering and Methodology*, vol. 5, no.4, 1996, pp.334–377.

[6] Cormen, T.H., Leiserson, C.E., and Rivest, R.L. *Introduction to algorithms*, MIT Press, 1990, pp.449–457.

[7] Dalal, S., Temel, S., Little, M., Potts, M., and Webber, J. Coordinating business transactions on the web. *IEEE Internet Computing*, vol.7, no.1, Jan.-Feb. 2003, pp.30–39.

[8] Dayal, U., Hsu, M., and Ladin, R. Organizing long-running activities with triggers and transactions. *Sigmod Record (Acm Special Interest Group on Management of Data)*, vol.19, no.2, Jun. 1990, pp.204–214.

[9] Dayal, U., Hsu, M., and Ladin, R. A transactional model for long-running activities. In *Proceedings of 17th International Conference on Very Large Data Bases*. Morgan Kaufmannn.1991, pp.113–122. San Mateo, CA, USA.

[10] Deora, V., Shao, J., Shercliff, G., Stockreisser, P.J., Gray, W.A., and Fiddian, N.J. Incorporating QoS specifications in service discovery. *Web Information Systems - WISE 2004 International Workshops (Lecture Notes in Computer Science Vol.3307)*. Springer-Verlag. 2004, pp.252–263. Berlin, Germany.

[11] Elmagarmid, A.K. *Database Transaction Models for Advanced Applications*. Morgan Kaufmann publishers, 1992. San Mateo, CA.

[12] Garcia-Molina, H., and Salem, K. SAGAS. *Sigmod Record (ACM Special Interest Group on Management of Data)*, vol.16, no.3, Dec. 1987, pp.249–259.

[13] Grefen, P., Vonk, J., and Apers, P. Global transaction support for workflow management systems: from formal specification to practical implementation. *VLDB Journal*, vol.10, no.4, Dec. 2001, pp.316–333.

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

[15] Lee, P.A., and Anderson, T. *Fault tolerance: Principles and practice*, pp.143-185. Springer-Verlag 1990.

[16] Manufacturer website, http://bjqad.com/yawen/mall/index.asp (accessed on 8 Jan 2006).

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

[18] UDDI. Available at: http://www.oasis-open.org/committees /uddi-spec/doc/tcspecs.htm#uddiv3 (accessed on 8 Jan 2006).

[19] Vonk, J., Grefen, P., Boertjes, E., and Apers, P. Distributed global transaction support for workflow management applications. In *Proceedings of 10th International Conference on Database and Expert System Applications*. Springer-Verlag. 1999, pp.942–951. Florence, Italy.

[20] Vonk, J., and Grefen, P. Cross-organizational transaction support for e-services in virtual enterprises. *Distributed & Parallel Databases*, vol.14, no.2, Sept. 2003, pp.137–172.

[21] Weikum, G. Principles and realization strategies of multilevel transaction management. *ACM Transactions on Database Systems*, vol.16, no.1, Mar. 1991, pp.132–180.

[22] Wohlstadter, E., Tai, S., Mikalsen, T., Rouvellou, I., and Devanbu, P. GlueQoS: middleware to sweeten quality-of-service policy interactions. In *Proceedings of 26th International Conference on Software Engineering*. IEEE Comput. Soc. 2004, pp.189–199. Los Alamitos, CA, USA

[23] WS-C. Available at: http://msdn.microsoft.com/library/en-us/ dnglobspec/html/ws-coordination.asp (accessed on 8 Jan 2006).

[24] WS-T. Available at: http://msdn.microsoft.com/ws/2002/08 /wstx/ (accessed on 8 Jan 2006).

[25] Ye, C.Y., and Cheung, S.C. *On Atomicity Consistent Services for B2B Collaboration.* Technical report, No.HKUST-CS-05-16, Department of Computer Science, The Hong Kong University of Science & Technology, 2005.

[26] Yemini, S., and Berry, D.M. A modular verifiable exception-handling mechanism. *ACM Transactions on Programming Languages & Systems*, vol.7, no.2, Apr. 1985, pp.214–243.