Privacy and Recovery in Composite Web Service Transactions

Rattikorn Hewett[1] and Phongphun Kijsanayothin[2]
[1]*Department of Computer Science, Texas Tech University, USA*
[2]*Department of Electrical and Computer Engineering, Naresuan University, Thailand*

Abstract

Today's numerous online transactions are implemented as composite web services in various domains including business, healthcare, government and education. This paper addresses privacy and recovery issues in composite web service transactions. In particular, we propose an intelligent semi-automated privacy-aware approach to efficiently building an appropriate composite web service along with a recovery mechanism to provide protection against privacy and service execution failures in the transactions. Our proposed approach aims to satisfy service I/O requirements using a minimum number of services with minimal information leakage and maximal compliance with a customer's privacy preferences and trust (in service providers when available). The paper describes details of the proposed approach and illustrates its use that exploits generic knowledge about types and sensitivity levels of information, together with specific knowledge about customer privacy preferences and trusts on certain providers. By using privacy-aware contingency and various types of compensation, the recovery mechanism provides potentials to maximize forward recovery from failed transaction services.

1. Introduction

Advances in Internet and web service technology have made a significant impact on online transactions by enabling rapid development of new online applications from existing web services across the Internet. Web services (or services) are platform independent modular computing units prescribed by standard machine-readable protocols to support interoperable services of distributed applications [2, 20]. Software developers can automatically locate service providers, select and bind specified variables for invocation and execution of appropriate web services to perform the desired functions. When online services are complex and no single service is adequate, web service composition becomes necessary. Efficient automated composition of web services is challenging especially when there is an increasingly large number of available web services. Today's numerous online transactions are implemented as composite web services in various domains including business, healthcare, government and education.

One important aspect of secured online transactions is privacy protection. To engage in online transactions, customers are often required to provide personal information to service providers. This makes customers vulnerable for threats to privacy exploits and information security. The situations can get worse when a transaction involves multiple services where the transaction is conducted between a customer and a primary service provider that outsource services to other service providers. Consider a composite web service transaction where a primary online travel service provider *S* outsources flight services to two airline companies: *A* and *B* where *B* further outsources international flight services to two airline companies: *C* and *D*. Thus, a flight reservation transaction consisting of a chain of a domestic service followed by an international flight service has three options of providers: *A*, *B&C*, or *B&D*. The travel service *S* is a primary contact for the customer and responsible for the success of the transaction, but if the customer's personal information is passed to *A, B*, and *C* or *D*, it poses a threat to the customer by exposing potentially sensitive information to third party companies that do not have to comply with the privacy policies that are used by *S*. The customer is left vulnerable for not being able to control what happens to his/her information. In reality, some customer might trust (believe in) airlines *A* and *D* based on previous experiences but if *A* does not have an adequate privacy policy, the customer might prefer to use *B&D* even though it might be more expensive. The ability to take customer's trusts and privacy preferences into considerations for selecting appropriate service providers in a composite web service is crucial for gaining customer confidence and assurance of electronic transactions.

The problem of privacy preserving in online transactions has been addressed by providing the customer with explicit statements of terms-of-use of the data and their storage. Due to its verbose nature, this approach appears to be ineffective [24]. Recent

efforts in P3P (Platform for Privacy Preferences) [4] allow customers to check if their privacy preferences match with those of the service provider. While this is promising, customizing individual privacy preferences for multiple services with a very large number of potential service providers in a complex online transaction remains to be extremely difficult.

To provide sustainable and reliable composite service execution flows, the ability to provide automatic protection against service execution failures is another important aspect for secured online transactions. The concept of failure recovery is well studied in the context of data management [7, 5, 12] and transactional workflows [6, 13, 23]. For traditional data-oriented distributed transactions, the two-phase commit protocol to support atomicity, consistency, isolation, and durability (ACID) properties [12] are employed. In this approach, only complete executions are accepted and thus, exhibit all-or-nothing behavior where failed sub-transactions have to be undone. Because the ACID protocol requires locking resources during the entire execution period, it is not suitable for long running transactions (LRTs) [3]. Advanced transactional models such as nested and multi-level models have been proposed to relax the atomicity and the isolation of the ACID properties to better support LRTs [5, 7]. Therefore, a failure in sub-transactions does not necessarily lead to an abortion of the actual transaction.

Similar to web service composition, most approaches to failure recovery in web service research have focused on specifications and semantics of service recovery models (e.g., WS-BPL [1], WS-Transaction, a Business Transaction Protocol (BPT) (see details in [16]). Service recovery models are developed in the context of workflow transactions and business processes [14, 19]. Techniques for error handling and service recovery employ a compensation of each failed sub-transactions and execute them in reverse order [1, 9]. However, most current web service recovery mechanisms are inflexible [22]. They tend to rely on compensation handlers and contingency plans that are pre-determined. Web service discovery and composition depend on service availability that is constantly changing over time. It would be desirable to have a service recovery mechanism that can adapt to such changing environments in that it can dynamically find alternative services for failed services. Good progress has been made towards this goal, particularly in semantic web services [21, 22] but none of these approaches has taken a privacy issue into account.

Our research aims to alleviate privacy and recovery issues in composite web service transactions by providing an intelligent semi-automated privacy-aware approach to efficiently build an appropriate composite web service that (1) satisfies service functional requirements with (near) minimum number of services and information leakage, and (2) complies, as much as possible, with a customer's privacy preferences and trust (in service providers when available). This paper is an extended version of the work presented in [11] to include automated service recovery procedure in addition to the service composition procedure in [11]. In particular, we present a service recovery procedure that can dynamically compute alternative composite services for failed services. One unique aspect of our work is that both of the proposed approaches provide solutions to automatic service composition and recovery in the context of privacy protection.

The rest of the paper is organized as follows. Section 2 describes related work and Section 3 presents our main contribution including an overview of the proposed approach, algorithms for service composition and selection of service providers that best satisfy a customer's privacy preferences and trust. Section 4 illustrates the approach to service composition. Section 5 describes our proposed recovery mechanism along with an illustrated example followed by conclusions in Section 6.

2. Related work

Most current approaches to web service composition integrate discovery techniques into composite service structures that have been *pre-specified* (see [2, 20] for surveys). Our earlier work [10] presents an automated approach to efficiently composing web services but it does not address privacy issues.

Various approaches to privacy protection have been studied including sandboxing [8], an access control based technique that prevents information leakage at the entry service but loses privacy control once the information leaves the primary service provider. Privacy protection by anonymity [17] is not applicable to online transactions since a disclosure of customer information is necessary to obtain services.

Other work that addresses privacy issues in web services includes Rezgui et al. [18] and Xu et al. [24]. The former applies credential-based method and customer privacy preferences to prevent unauthorized access to information of both sides. However, it lacks the ability to control and reason through the flow of information in composite web service transactions as addressed by the latter. Our work is similar to Xu et al.'s approach in this aspect. In fact, Xu et al.'s approach focuses on checking whether a given composite web service (after service composition) can satisfy customer privacy preferences. Our approach, however, protects privacy even earlier during a web service composition stage by automatically adapting the service composition to minimize information leakage. This is useful for development of complex online transactions.

Work in failure recovery for composite service transactions are driven by language specifications and business transaction protocols. WS-BPEL [1]

exploits a compensation for error handling in each failed sub-transaction [9]. WS-transactions and its combination with WS-BPEL are also proposed [16, 22] to provide a two-phase protocol that allows more relaxed ACID properties in application control for failed sub-transactions to support long running transactions.

In transactional workflows, recovery mechanisms primarily focus on maintaining state consistency [6, 22]. When a failure occurs, the execution returns to the most recent consistent state and attempts to continue the execution to complete the workflow. Compensation and contingency are common recovery techniques [23]. The former provides a backward recovery to undo the affects of completed or partially service execution. The latter facilitates a forward recovery by performing an alternate execution path that will allow a service to continue. Because of the changing service environment, a compensation that requires a service that is no longer available can result in an inconsistent state of the transaction even though other services can be used as compensation for the failed service.

Recent work [21, 22] has attempted to provide a more flexible recovery technique by extending a well-known specification language OWL-S [15] with exception handling and basic recovery mechanisms. Weisner et al. [22] further show how semantic annotations of semantic web services and appropriate ontology can be exploited to facilitate dynamic discovery of alternative web services. Our proposed approach shares the same intent and basic concepts as Weisner et al.'s approach. However, our mechanisms are meant to be general and not to be restricted to semantic web services. Furthermore, we take privacy protection into consideration during the contingency plan to find appropriate alternative services to replace failed services.

3. Proposed framework

To accomplish our goal, Figure 1 shows a basic overview of our system with its corresponding processes. As shown in Figure 1, the proposed framework includes three basic algorithms: *build-composite-chain, prune-chain*, and *privacy-compliance*. Given a service repository that contains different web services, each of which has associated sets of input/output requirements and service providers. First, in a service discovery and composition step, the *build-composite-chain* algorithm takes input parameter requirements of a new online application to be developed as its input. It then searches for the "best" executable chain of services that produces the output parameters required by the new application using as few number of services and as little information leakage as possible (to be described later).

Next, the *prune-chain* algorithm is applied to eliminate unnecessary services (to further minimize the number of services) in a similar fashion as our previous work [10]. Finally, the *privacy-compliance* algorithm generates service providers that offer the required services and checks for compliance with "specific knowledge" about customer privacy preferences (based on the information to be supplied) and his/her trust on certain service providers (e.g., social security number is allowed for IRS, a government office but not for a credit card company). If they are satisfied, a transaction of the composite service constructed can be completed successfully. Otherwise, alternative providers or composite service chains are generated and tested. If no satisfactory providers can be found, a compromising alternative will be proposed to the customer, if it exists.

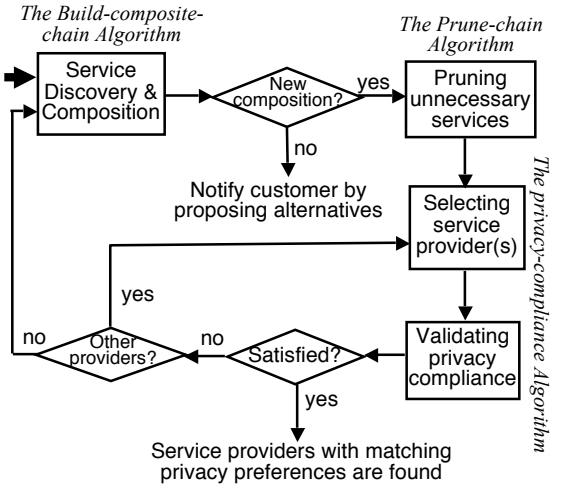


Figure 1. Overview of the proposed system.

To describe our approach, we define the following. Given a service repository W over a set of parameters P, where each web service $w \in W$, is described by its corresponding set of input (output) parameters $in(w)$ ($out(w)$) $\subseteq P$ and a set of providers $server(w)$. Let I (O) $\subseteq P$ be a set of input (output) parameter requirements of a composite service to be developed. Our service composition is a state space search where state s, of a composite service construction, is represented by $p(s)$, a set of *callable* (executable) parameters in s. The details of the proposed approach are described in the following subsections.

3.1. Service composition & privacy leakages

This section describes the *build-composite-chain* algorithm for finding an executable service chain that satisfies a given composite service I/O requirements with the aim to minimize information leakage. To quantify information leakage, we represent generic knowledge about different *types* of private information (e.g., personal Identification, financial information, contact address), and their levels of *sensitivity* (e.g., for identification, social security number, driver's license number, and an employee ID number has high, medium, and low sensitivity level, respec-

tively). Table 1 shows examples of various types and sensitivity levels of private information.

Table 1. Information types & sensitivity levels.

	Sensitivity Level		
Info Type	**High**	**Medium**	**Low**
ID	*SocSecNo*	*DriverLicNo*	*EmployeeID,UserID*
Personal	*Name, DoB*	*Email, PhoneNo*	*Address, Gender*
Financial	*CreditCardNo*	*BankAccount*	*BankName & Address*

Our service composition approach searches for a chain of services from an initial state represented by its callable set of parameters I to a final state with an associated callable set G where $O \subseteq G$. Each move from one state to another represents a selected service application in the composition chain, where web service w is *applicable* to state s, if $in(w) \subseteq p(s)$ (all of its input parameters are callable). The *build-composite* algorithm is a greedy search using two heuristics that prefer an applicable service that (1) involves a small number of parameters with a high sensitivity level (to minimize information leakage), and (2) produces a large number of output parameters (to maximize the chance of meeting the output parameter requirements faster, i.e., minimize the number of services). The former automatically captures possible leakage of sensitive information from service invocation. Basic steps of the *build-composite-chain* algorithm are shown in Figure 2.

Procedure *Build-composite-chain*(W, I, O)
Inputs: A web service repository W
A set of input (output) parameters I (O) required
Output: An executable service chain *seq* that satisfies I/O and aims to minimize information leakage
1 $seq \leftarrow nil$;
2 $C \leftarrow I$; a set of callable parameters of a current state
3 **while** $W \neq \emptyset$ **do**
4 $A \leftarrow \{w \mid in(w) \subseteq C\}$ a set of *applicable* services
5 $L \leftarrow \{w \in A \mid leakage(w, C) = min_{w \in A}\ leakage(w, C)\}$
6 **if** $|L| = 1$ **then** $service \leftarrow s \in L$
7 **else** $M \leftarrow \{s \in L \mid new(s, C) = max_{w \in L}\ new(w, C)\}$
8 ; define $new(w, C) = out(w) - C$
9 $service \leftarrow$ first $s \in M$ found
10 **Append** *service* to *seq*
11 $C \leftarrow C \cup new(service)$
12 **if** $O \subseteq C$ **then return** *seq* ; O is satisfied
13 **else** $W \leftarrow W - \{service\}$
14 ***return*** *nil* ; no sequence composition found

Figure 2. The *build-composite-chain* algorithm.

As shown in Figure 2, the algorithm first applies heuristic (1) to select among applicable services for services with minimum information leakage (Line 5). For a web service w to be applied to current state of a callable parameter set C, we measure information leakage by the amount of sensitive information being disclosed to w (i.e., $in(w)$) and the amount of *new* sensitive information being disclosed by w (i.e., $out(w) - C$). Thus, we implement the function *leakage*(w, C) to return a triple (h, m, l), where h, m and l is the number of *high, medium* and *low* sensitive parameters in $in(w) \cup (out(w) - C)$, respectively. To find a service with minimal leakage, we select a service with lowest h and if there is more than one of such services, we pick the one with the lowest m, and so on. If there is more than one of services with the lowest h, m and l, we apply heuristic (2) to pick a service that is likely to satisfy the output parameter requirements O quickly (Line 7). The process continues until O is satisfied or no more services to try (i.e., no solution can be found). Because of the monotone property of our state representation (i.e., as we apply more services along the path, the number of callable parameters increases), our search requires no backtracking and guarantees to find a composite service chain if it exists (see details in [4]). However, like other heuristic approaches, it cannot guarantee optimality of the number of services (i.e., solution path length) or the amount of leakage.

3.2. Pruning unnecessary services

Let $seq = \langle w_1, w_2, ..., w_k \rangle$ be a service chain obtained from Section 3.1. Our pruning mechanism determines which web service in the *seq* is necessary for the composition in a bottom up fashion. We refer to any element of O as a *desirable* parameter. Let N be set of necessary services found from the bottom of the path *seq* so far before. Conceptually, a web service is *necessary* if it is the only service (among itself and all services in N) that produces a new parameter that is either (1) desirable or (2) a required input parameter of some service in N such that it cannot be produced by any service in N.

Procedure *Prune-sequence*(*seq*, O)
Inputs: O, a set of output parameters for a composite service;
$seq = \langle w_1, ..., w_k \rangle$ obtained by *build-composite-chain*
Output: *seq* without *unnecessary* services (defined above).
1 **for** $i \leftarrow k$ **to** 1
2 $N_i \leftarrow \{w_j \mid i < j \leq k\}$; necessary services *after* applying w_i
3 $O_i \leftarrow O_{i+1} \cup (O \cap out(w_{i+1}))$; associated desirable parameters
4 $I_i \leftarrow (I_{i+1} - out(w_{i+1})) \cup in(w_{i+1})$
5 ; input parameters required by all services in N_i s.t.
6 ; none can be created by N_i in the *seq* order.
7 $New \leftarrow p(s_i) - p(s_{i-1})$; a set of new parameters by w_i
8 **if** $New \cap (O - O_i) \neq \emptyset$; there is a new parameter that is
9 ; desirable but cannot be produced by N_i
10 or $New \cap I_i \neq \emptyset$; or that is a necessary input for N_i
11 **then** w_i is necessary
12 **else** *Eliminate* w_i;
13 **end for**

Figure 3. The *prune-sequence* algorithm.

To check if web service w_i that moves state s_{i-1} to state s_i on the *seq* path is necessary, we define (i) N_i,a set of all necessary services applied to *seq after* w_i, (ii) O_i, a set of desirable parameters produced by all service in N_i, and (iii) I_i, a set of input parameters required by all services in N_i such that they cannot be produced by any application of service in N_i in the *seq* order. Both O_i and I_i are used for testing the above condition (1) and (2), respectively, in order to determine if w_i is necessary or not. Figure 3 shows

our proposed pruning conditions are checked in Lines 8 and 10, respectively.

Note that a set of desirable parameters produced by all service in N_i (necessary services w_j for $j > i$) is equivalent to a set of desirable parameters produced by all service in N_{i+1} and those that produced by w_{i+1}, therefore we obtain Line 3 of Figure 3. Similarly, the necessary input parameters required by all services in N_i can be obtained by first finding the necessary input parameters required by all services in N_{i+1} that cannot be produced by w_{i+1} and then adding $in(w_{i+1})$ as shown in Line 4. By making use of the results in a previous iteration, our approach provides an efficient pruning mechanism. See details in [10].

3.3 Customer privacy preference and trust

This section describes the *privacy-compliance* algorithm that selects a service provider (if possible) that best complies with a customer privacy preference and trust on the providers for each service in the pruned service chain obtained in Section 3.2. Figure 4 shows basic steps of the algorithm.

Procedure *Privacy-compliance* (*seq, T*)
Inputs: An executable service chain *seq,* a set of service providers *server*(*w*) of service *w*, and a customer privacy preference *T*, a list of *T*(*i*)'s where *T*(*i*) represents a set of providers that the customer trusts to disclose parameter *i* to.
Output: A service provider, for each service in *seq*, that is in compliance with *T*.

```
1  Compliance ← server(w1), where w1 is the first service in seq
2  for each w in seq do
3    for each input parameter i of w do
4      if T(i) is specified in T
5        then if T(i) = ∅
6               then return("disclosure of i is disallowed")
7               else  Compliance(i) ← {s | s ∈ T(i) ∩ server(w)}
8                     if Compliance(i) = ∅ then return("fail on i")
9        else; assume all providers are trusted equally
10            Compliance(i) ← {s | s ∈ server(w)}
11     Compliance ← Compliance ∩ Compliance(i)
12   if Compliance=∅ then return("fail")
13   Select any provider in Compliance for service w
14   end for
15 end for
```

Figure 4. The *privacy-compliance* algorithm.

As shown in Figure 4, the approach searches for a provider for each service in the given chain. Our approach makes a distinction between specified disclosed parameters with no trusted providers (Line 4) and unspecified disclosed parameters (Line 8). The first case means that a customer specifies preference not to disclose certain parameters regardless of which providers offer the service. When the service requires any non-disclosed input parameter, the algorithm notifies (Line 5) this to the customer (who may change his privacy preference or requests for a different service chain). For the latter case, the customer does not provide any privacy or trust preference and therefore any service provider that offers the service can perform the service (Line 9). The *Compliance* set collects each provider that incrementally complies with each input parameter required by the service (Line 10). If it is empty then no provider is in compliance with all input parameters of the service and a failure is reported (Line 11). Although we report a failure of each input parameter (Line 7), Line 10 is necessary for unspecified parameters.

4. Illustrations

Consider the development of a composite web service for online ticket purchase transactions from a given service repository as shown in the top part of Table 2. For simplicity, we omit set brackets for the set notation where it is obvious. There are eight web services over 12 parameters and six providers. The bottom part of Table 2 shows a given generic knowledge base (KB) about service parameters and their corresponding sensitivity levels. For example, *CCNo* (credit card number) is personal financial information that is of high level of sensitivity. Note that not all parameters are necessarily relevant to this KB.

Table 2. Service repository & sensitivity KB.

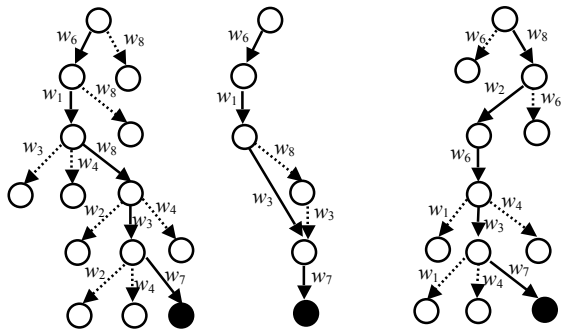
Service	Provider	Input parameters	Output parameters
Booking(w_1)	*A, B*	*TravelDates, Name, CCNo, BillingAdd*	*BookingID, Cost, SeatNo*
Booking(w_2)	*C*	*TravelDates, Email*	*BookingID, Cost*
Payment(w_3)	*BkA, BkB*	*CCNo, Cost, BookingID*	*TransID*
Verify(w_4)	*BkA, BkB*	*CCNo, Cost*	*VerifiedID*
Payment(w_5)	*BkA, BkB*	*VerifiedID, BookingID*	*TransID*
CrdInfo(w_6)	*D*	*UserID*	*CCNo, BillingAdd,Name*
Confirm(w_7)	*A, B, C*	*TransD*	*TicketNo*
CusInfo(w_8)	*D*	*UserID*	*Address, Email*

Sensitivity Level

Info Type	High	Medium	Low
Personal		*Name, Email*	*Address, UserID*
Financial	*CCNo*	*BillingAdd*	*VerifiedID*
Flight Trans		*TicketNo*	*BookingID, TransID, SeatNo*

To buy a flight ticket online, a customer has to sign on an online travel service system. Thus, the composite service to be developed requires I = {*TravelDates, User- ID*} and O = {*TicketNo*}. Since there is no single service in the repository that satisfies the given input and output specifications, a service composition is necessary.

4.1. Privacy-aware service composition



a) No privacy concerns b) After pruning c) Privacy concerns

Figure 5. Various composition strategies.

For comparison purposes, we first search for a service composition without privacy consideration (by applying the *build-composite-chain* algorithm without using the leakage information in *L,* Lines 5-6). The partial search tree is shown in Figure 5 a).

Starting from an initial state with a callable parameter set I, w_6 and w_8 are the only applicable services (since all of its input parameters are callable). However, based on heuristic (2) in Line 7 of Figure 3, w_6 is preferred to w_8 since $|out(w_4) - I| > |out(w_6) - I|$. After executing w_6, a new callable parameter set is $I \cup out(w_6)$ = {*TravelDates, UserID, CCNo, BillingAdd, Name*}. Similarly, new applicable services are w_1 and w_8, where w_1 is selected because it produces more new output parameters than w_8. As shown in Figure 5 a), the process repeats and we obtain a composite chain $seq = <w_6, w_1, w_8, w_3, w_7>$ that satisfies given sets I and O.

Next we apply the *prune-sequence* algorithm to the *seq* obtained. As a result shown in Figure 5 b), service w_6 is eliminated. Note that none of the outputs produced by w_8, which is {*Address, Email*}, is required by any of the following services in the chain (to be callable for input parameters). In other words, neither *Address* nor *Email* is in $in(w_3)$ or $in(w_7)$. Thus, w_8 is unnecessary. The reduced chain $seq^* = <w_6, w_1, w_3, w_7>$ is shown in Figure 5 b).

Now consider when we take information privacy and disclosure into consideration. By applying the *build-composite-chain* algorithm and the generic knowledge about sensitivity levels of certain service parameters, shown at the bottom part of Table 2, we obtain a partial search tree as shown in Figure 5 c). Starting from an initial state, we obtain two applicable services as before but this time w_8 is preferred to w_6 since w_6 leaks a high sensitive parameter, while w_8 does not. This is because $C = I$ and $in(w_8) \cup (out(w_8) - C)$ = {*Address, Email*} and thus, $leakage(w_8, C) = (0, 1, 1)$ (since Address and Email are of low and medium sensitivity levels, respectively). Similarly, $leakage(w_6, C) = (1, 2, 0)$, which means that w_6 leaks one high and two medium sensitive parameters, whereas w_8 leaks one medium and one low sensitive parameter. Thus, w_8 discloses less sensitive information and so it is selected for a composition chain. After applying w_8, we obtain a new callable parameter set $C = I \cup (out(w_8))$ = {*TravelDates, UserID, Address, Email*}. The only two new applicable services to C are w_2 and w_6. Since $leakage(w_2, C) = (0, 1, 1)$ while $leakage(w_6, C) = (1, 2, 1)$, therefore w_2 is selected. Figure 5 c) shows the resulting sequence $seq^p = <w_8, w_2, w_6, w_3, w_7>$. After applying the *prune-chain* algorithm, no service is eliminated and the same chain is obtained.

To assess seq^p, we compare it with seq^*. The length of seq^p is longer than that of seq^* in order to trade the number of services with privacy protection by avoiding the use of high sensitive information in the service chain until when it becomes necessary. For example, *CCNo* is first disclosed by w_6, the third service invoked in seq^p but the first in seq^*. To further evaluate seq^p, we count the number of times that each parameter is disclosed *to* (service input) and disclosed *by* (service output) each of the services in the chain. For example, the disclosure degree associated of a high sensitive *CCNo* is three in seq^* (from w_6, w_1, w_3), and two in seq^p (from w_6, w_3), which is denoted by (*CCNo*, 3, 2). Similarly, we obtain (*Name,* 2, 1), (*BillingAdd*, 2, 1), (*Email,* 0, 2), and (*TicketNo,* 1, 1) for medium sensitive parameters, and (*Address*, 0, 1), (*UserID*, 1, 2), (*BookingID,* 2, 2), (*TransID*, 2, 2), (*SeatNo*, 1, 0) for low sensitive parameters. In general, the degree of disclosure for higher sensitive parameters is mostly higher in *seq* than seq^p. This implies that the seq^p service composition has a higher degree of privacy protection as expected. We next verify if the service chain obtained complies with a customer privacy preferences.

4.2. Privacy Compliance

After obtaining a privacy-aware composite chain, we need to select providers in the chain that best complies (if it is possible) with a customer privacy preference and trust as shown in Table 3. For example, the customer trusts to only disclose his credit card number to providers *A* and *BkB*.

Table 3. Customer privacy preference and trust.

Disclosed Information	Trusted Providers
Name, Email	*A, C, BkA, BkB*
Address, UserID	*D, C*
CCNo	*A, BkB*
BillingAdd	*A, BkA, BkB*
TicketNo	*A, B, C*
BookingID, TransID, SeatNo	*A, B, BkB*

Applying the *privacy-compliance* algorithm to $seq^p = <w_8, w_2, w_6, w_3, w_7>$, we first obtain provider *D* for service w_8 since (1) w_8 is provided by *D* (see the service repository in Table 2), (2) the customer trusts to disclose all input requirements of w_8 (i.e., *UserID*) to *D* as indicated by second row of Table 3. For w_2, provided by *C,* where $in(w_2)$ = {*TravelDates, Emails*}, the customer trusts to disclose his *Emails* to *C* but does not specify trust preference on providers for *TravelDates*. Thus, *C* is in compliance with customer privacy preference and trust for w_2. Similarly, we obtain providers *D, BkB*, and *A* for the rest of services in the chain, namely w_6, w_3 and w_7.

Note that our approach allows the customer to specify preferences on information not to be disclosed by simply assigning its corresponding trusted providers to an empty set. It is also flexible in that when no information on privacy preferences or trusted companies is available, the *privacy-compliance* algorithm selects appropriate providers by assuming that all of them are equally trusted.

5. Recovery Mechanisms

A service transaction can be viewed as a process model where its elementary unit is an atomic process that represents an individual service. Each atomic process (or service) defines its corresponding input/output parameters and compensation. Atomic processes can be combined into composite processes by using control constructs such as sequence or split. We assume that all processes (except the initial root process) and control constructs are nested to ensure that every process or control construct has a defined parent. Thus, in general, a service transaction can contain sub-transactions represented in the form of nested or multi-level chain of services.

To enable recovery of a failed service, the following *recovery actions* are defined based on the concepts introduced in [21, 22].

Compensation: For each process, compensation contains actions to undo the effects of the previously performed process. Compensation can be triggered by *Compensate*, which enables the compensation for a given process. Consequently, the execution will return to the most recent consistent state to allow future forward execution.

Retry: The execution of a process can be retried either for a specified number of times or until the specified time expires (*timeout*). This is useful for a communication failure to restore a normal execution flow after the failure occurs.

Skip: The execution of non-essential process can become unnecessary (e.g., recovery of execution of a process that does not involve any data update) and can be omitted as long as it has not been started yet.

Contingency: A failed process can be replaced by an alternative sequence of web services that satisfy the input/output requirements as well as privacy preference of the required information. Contingency is performed by combined execution of *Build-composite-chain* and *Privacy-compliance* (see details below).

Abort: All running processes are terminated and performed tasks on the same level are undone. The same is done for parent levels. The action is triggered by *abort-transaction*. Termination can be realized with or without compensation of the completed sub-processes. The latter is a strong abort where the process is terminated without waiting for current running activities to finish execution completely.

Service recovery mechanisms provide strategies for applying appropriate *recovery actions* when a service fails. For example, sometimes compensation may be required before the execution of contingency plans. Figure 6 shows basic steps of our proposed recovery procedure for failed service transactions. To recover a failed process w, the recovery procedure is called recursively to perform recovery from the failed process to its ancestors up until the root process.

As shown in Line 1 of Figure 6, when a failed process is a root, there is no way to rollback beyond the root process to compensate and undo the effect of the failed process and therefore the transaction needs to be aborted. Otherwise, if the execution of the failed process does not involve data changes or updates, the simplest recovery action is to retry the process until the execution is either successfully completed or *timeout.* In the latter case, next step is to find a contingency plan, i.e., a new sequence of services to replace the failed service. As shown in Line 5, this can be done by the *Find-contingency-plan* function that employs the *Build-composite-chain* algorithm that finds, from a current set of services (except the failed process w) available in the service repository, a service or a chain of services that has the same I/O requirements as those of w and that complies with a given privacy preference (by the Privacy-compliance function). If there is no alternative service (to replace w) that complies with a given privacy preference then we skip the process and recover its parent process. As shown in Line 18, the alternative chain of services to replace failed service is pruned to optimize number of services required and the *Privacy-compliance* minimizes information leakage in the same manner as their usages in service composition. The fact that both algorithms have proved to be efficient [10] makes the Find-contingency applicable for dynamic discovery of alternative services realistically feasible for online transaction failure recovery.

Procedure *Privacy-aware-Recovery*(*w*)
Inputs: A web service repository *W,* a customer privacy preference *T*, a list of *T*(*i*)'s where *T*(*i*) represents a set of providers that the customer trusts to disclose parameter *i* to and a service *w* to be recovered.
Output: recovery of process *w*
1 **if not** *hasParent*(*w*) **then** *abort-transaction*
2 **else if** no data update in *w* **then**
3 **repeat** *retry*(*w*) **until** *succeed* or *timeout*
4 **if** *timeout* **then**
5 *new* ← *Find-contingency-plan*(*w*)
6 **if** *new* is *nil* **then**
7 *skip*(*w*)
8 *Privacy-aware-Recovery* (*parent*(*w*))
9 **else** ; there is data update
10 *compensate*(*w*)
11 *new* ← *Find-contingency-plan*(*w*)
12 **if** *new* is *nil* **then**
13 *Privacy-aware-Recovery*(*parent*(*w*))

14 **Function** *Find-contingency-plan*(*w*)
15 $S \leftarrow \{w\}$
16 **repeat**
17 *new* ← *Build-composite-chain*($W - S$, *in*(*w*), *out*(*w*))
18 *new* ← *Prune-sequence*(*new*, *out*(*w*))
19 $S \leftarrow S \cup \{ws \in W \mid ws$ appeared in *new*$\}$
20 **until** *Privacy-compliance*(*new*, *T*) **or** *new* is *nil*
21 **return** *new*
22 **End Function**

Figure 6. *Privacy-aware Recovery* algorithm.

On the other hand, if the execution of the failed process w involves data updates, compensation is needed to undo the effect of the execution so far and return to the most recent consistent state (before applying a contingency plan to find a replacement and continue recovering the parent process and so on). This is reflected in lines 10-13 of Figure 6.

The integration of dynamic discovery of a contingency that are in compliance with privacy preferences into the recovery procedure makes our recovery mechanism different from most current recovery approach. Next we illustrate the proposed *Privacy-aware-Recovery* algorithm as shown in Figure 7.

Consider the example scenario introduced in Section 5 where an online transaction requires execution of each process in the service chain $<w_8, w_2, w_6, w_3, w_7>$ as shown in Figure 7 (also Figure 5 c)). Suppose the execution fails at process w_3 during the data updating. To recover this failure, we need to first compensate w_3 (Line 10 of Figure 6) by applying actions to undo the effects of operations in w_3. As a result, the recovery action will return to State 4. Note that if the execution fails at process w_3 without data changes or updates, we can skip the failed process and simply rollback to State 4 without any need for compensation (as shown by a thick arrow in Figure 7).

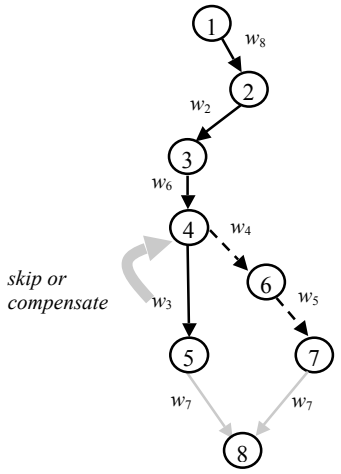


Figure 7. Recovery from a failed service.

Next, by applying contingency, we must find w_3's replacement that has the same I/O requirements as those of w_3. As shown in Figure 7, by applying the *Build-composite-chain* to all services except w_3 in a current web service repository (Line 17 of Figure 6), we obtained a composite of web services w_4 and w_5 as an alternative chain of services to replace w_3. Since each of w_4 and w_5 satisfies the customer privacy preference (e.g, w_4's privacy preference can be satisfied by *BkB* and w_5's privacy preference can be satisfied by *BkB* or *BkA*), we can recover the failure of w_3 by replacing $<w_4, w_5>$. The alternative services may lead to a different state than that led by w_3 (State 7 instead of State 5). However, both states are applicable to the application of service w_7 to complete the transaction.

Note that the implementation of contingency to find a replacement of the failed process has been refined in the following manner. The replacement does not have to have exactly the same I/O requirements as those of the failed process. In fact, the replacement only requires a set of input parameters that is a subset of the input parameters required by the failed process. Furthermore, the replacement only needs to produce a set of output parameters that are a superset of the output parameters required by the failed process. This makes the dynamic contingency for service recovery more viable. However, the latter may expose more information and thus, exhibit a tradeoff between recoverability and privacy protection.

6. Concluding Remarks

We present an analytical framework that takes privacy protection issues into consideration for developing web composite services, which are crucial elements of many existing online transactions. We propose a three-stage approach to semi-automatically compose the new web composite service by:

(1) constructing a privacy-aware composite service chain that aims to minimize information leakage from parameters disclosed to and by services in the composition,
(2) pruning the service chain to eliminate unnecessary services, and
(3) checking if the service chain can be offered by service providers that are in compliance with a customer privacy preferences and trusts in the providers when they are available.

The three stages are presented in the proposed three algorithms that exploit heuristics based on:

(i) knowledge about a problem domain, specifically service input and output requirements,
(ii) generic knowledge about parameters of different information types with corresponding sensitivity levels, and
(iii) specific knowledge about a customer privacy preferences and trusts.

This paper focuses on integrating privacy considerations into web service composition instead of other aspects of service composition including efficiency, semantics, quality of service, and optimality of the number of services in the composition. Though we did not show this in this paper, our previous experiments [10] on public benchmark data sets show that our modeling approach and the core of our service composition along with pruning can effi-

ciently construct service chains with (near) optimal number of services. To the best of our knowledge we are not aware of any work that exploits explicit privacy elements, or customer privacy preferences and trusts into automatic construction of service composition. We also present an algorithmic approach to verification of privacy compliance, which is rarely addressed. Unlike our previous work [11], we also present failure recovery mechanisms for composite web service to secure reliable execution of online transactions. In particular, we present a recovery algorithm that integrates privacy considerations into contingency to find alternative services for recovering from failed services. Future work includes fine-tuning of the proposed algorithms and applying them to a large-scale system that handle multiple online transactions.

7. References

[1] A. Alves, et al., "Web Services Business Process Execution Language Version 2.0," available at http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html, April, 2007.

[2] M. Beek, A. Bucchiarone and S. Gnesi, "Web Service Composition Approaches: From Industrial Standards to Formal Methods", in *Proc. of Conf. on ICIW*, IEEE Com. Soc. Press, Mauritius, 2007.

[3] A. Cichocki, A. Helal, M. Rusinkiewicz, and D. Woelk, *Workflow and Process Automation Concepts and Technology:* Kluwer Academic Publishers, 1998.

[4] L. Cranor, "The Platform for Privacy Preferences 1.1 (P3P 1.1) Specification", W3C working draft, 2004.

[5] R. de By, W. Klas, and J. Veijalainen, *Transaction Management Support for Cooperative Applications*: Kluwer Academic Publishers, 1998.

[6] J. Eder and W. Liebhart, "The Workflow Activity Model WAMO," in *Proc. of the 3rd Int. Conference on Cooperative Information Systems (CoopIs),* 1995.

[7] A. Elmagarmid, *Database Transaction Models for Advanced Applications*: Morgan Kaufmann, 1992.

[8] I. Goldberg, D. Wagner, R. Thomas and E. Brewer, "A secure environment for untrusted helper applications: confining the wily hacker", in *Proc. of USENIX Security Symposium,* 1996.

[9] P. Greenfield, A. Fekete, J. Jang, and D. Kuo, "Compensation is Not Enough", in *Proc. of the 7th International Enterprise Distributed Object Computing Conference (EDOC),* 2003.

[10] R. Hewett, P. Kijsanayothin, and B. Nguyen "Scalable Optimized Composition of Web Services with Complexity Analysis", in *Proc. of IEEE 7th ICWS*, 2009.

[11] R. Hewett and P. Kijsanayothin, "On securing privacy in composite web service transactions", in *Proc. of the 5th International Conference for Internet Technology and Secured Transactions (ICITST'09)*, London, 2009.

[12] M. Kifer, A. Bernstein, and P. M. Lewis, *Database systems: an Application-oriented approach*. 2nd ed: Pearson, 2006.

[13] F. Laymann, "Supporting business transactions via partial backward recovery in workflow management", in *Proc. of the GI-Fachtagung für Datenbanksysteme in Business, Technologie und Web (BTW'95)*, 1995.

[14] F. Lin, and H. Chang, "B2B E-commerce and enterprise Integration: the development and evaluation of exception handling mechanisms for order fulfillment process based on BPEL4WS", in *Proc. of the 7th IEEE Int. Conference on Electronic commerce*, 2005.

[15] D. Martin, M. Paolucci, S. McIlraith, M. Burstein, D. McDermott, D. McGuinness, B. Parsia, T. Payne, M. Sabou, M. Solanki, N. Srinivasan, and K. Sycara, "Bringing Semantics to Web Services: The OWL-S Approach", in *Procs. of the First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, San Diego, California, USA, 2004.

[16] M. Papazoglou, "Web Services and Business Transactions", *World Wide Web* 6(1), 49–91, 2003.

[17] M. Reiter and A. Rubin, "Anonymous web transactions with crowds", *CACM,* 42(2), 1999.

[18] A. Rezgui, M. Ouzzani, A. Bouguettaya and B. Medjahed, "Preserving privacy in web services", in *Proc. of Workshop on information and data management,* 2002.

[19] P. Sauter, P. and I. Melzer, I., "A Comparison of WS-Business Activity and BPEL4WS Long-Running Transaction", in *Kommunikation in Verteilten Systemen (KiVS), ser., Informatik Aktuell,* pp. 115–125, Springer, Heidelberg, 2005.

[20] B. Srivastava and J. Koehler, "Web Service Composition - Current Solutions and Open Problems", *ICAPS 2003 Workshop on Planning for Web Services,* 2003.

[21] Vacul´ın, R., Wiesner, K., Sycara, K., "Exception handling and recovery of semantic web services", in *Proc. of the Fourth International Conference on Networking and Services,* IEEE Computer Society, Los Alamitos, 2008.

[22] K. Wiesner, R. Vaculín, M. Kollingbaum and K. Sycara, "Recovery Mechanisms for Semantic Web Services", in *Proc. of the 8th International Conference of Distributed Applications and Interoperable Systems (DAIS 2008)*, pp. 100-105, Oslo, Norway, 2008.

[23] D. Worah, and A. Sheth, "Transactions in Transactional Workflows," *Advanced TransactionModels and Architectures*, edited by S. Jajodia and L. Kershberg: Springer, 1997.

[24] W. Xu, V. Venkatakrishnan, R. Sekar and I. Ramakrsihnan, "A framework for building privacy-conscious composite web services", in *Proc. of IEEE International Conference on Web Service*, 2006.

