

Ensuring Consistent Termination of Composite Web Services

An Liu^{1,2,3} and Qing Li^{2,3}

¹ Department of Computer Science and Technology, University of Science & Technology of China, Hefei, China

² Joint Research Lab of Excellence, CityU-USTC Advanced Research Institute, Suzhou, China

³ Department of Computer Science, City University of Hong Kong, Hong Kong, China

liuan@ustc.edu itqli@cityu.edu.hk

ABSTRACT

To reach a consistent termination state even in the presence of failures, composite web services require transactional support. Most current work on web services transactions is based on compensation. However, unlike in the traditional intra-organizational applications, the compensation is typically associated with temporal constraints in the context of web services. Such temporal constraints have been largely overlooked by the current research. In this paper, we propose a framework to ensure consistent termination of composite web services with temporal constraints. Given the process of a composite service and the temporal constraints of its component services, our framework not only assists the selection of services based on consistent termination requirements but also ensures the composite service can always terminate in a consistent state even in the presence of failures.

1. INTRODUCTION

Web services are emerging as a new paradigm for developing and deploying business processes within and across enterprises. One of the most promising ideas underlying web services is composition. In most cases, a single web service is simply incapable of fulfilling complex functional requirements. It is therefore necessary to provide new value added services (composite services) by assembling some pre-existing web services (component services).

In order to reach a consistent termination state even in the presence of failures, transactional support needs to be added into service composition. As composite services are often long-running, loosely coupled, and cross administrative boundaries, traditional ACID transaction model are inappropriate [7]. Instead, most current work on web services transaction [3, 4, 8, 9, 11, 13] is built on compensation. Web services provide compensation operations to semantically undo the effects they have done. A composite service can terminate in a consistent state by invoking the compensation operations of its component services when an error occurs. For instance, a travel agent arranges a trip using three independent services: a flight service, a hotel service, and a taxi

service. Assume the hotel service has completed and the taxi service is still running, but at this time the flight service fails, then travel agent needs to cancel the taxi service and invoke the compensation operation of the hotel service to cancel the reserved room. Finally, the travel agent terminates in a consistent state.

However, web services are typically provided by different organizations and thus are inherently autonomous. In addition, they run in an open environment where close trust is missing. Therefore, most services impose some kinds of constraints such as cost and time on their compensation operations [1]. For instance, the hotel service may declare following compensation policies for the deadline of reservation cancellation: two days prior to the check-in date for members and six hours before the check-in date for VIP users. As another example, an online purchase service may declare that cancellation of the reserved goods is only allowed within 15 minutes from the time when transaction completes.

An interesting problem is how these temporal constraints affect web services transaction. The consistency require all the completed component services be compensated for when a component services fails. However, these compensation operations may be unavailable at that time and thus the composite service will terminate in an inconsistent state. It therefore becomes necessary and urgent to take temporal constraints into account when providing transactional support for composition. However, these temporal constraints have been largely overlooked by the current research.

In this paper, we propose a framework to ensure consistent termination of composite web services. Given the process of a composite service and the temporal constraints of its component services, our framework not only assists the selection of services based on consistent termination requirements but also ensures the composite service can always terminate in a consistent state. In addition, our framework exploits a topological-sorting-based scheduling algorithm to decrease the possibility of inconsistent termination.

The rest of the paper is organized as follows. Section 2 provides a brief overview of transactional and temporal properties of web services. Section 3 formally states the targeted problem and introduces the methodology of our approach. We detail our framework and validate its effects in Section 4, and in Section 5, we refine our framework and further validate its effects. Section 6 discusses related work. Section 7 concludes the paper and sheds some light on future research.

2. PRELIMINARIES

Transactional and temporal properties of web services are the foundation of our approach. In this section, we first define trans-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Proceedings of SIGMOD2007 Ph.D. Workshop on Innovative Database Research 2007 (IDAR 2007), June 10, 2007, Beijing, China.

actional properties of web services and provide a brief introduction on transactional composition, then discuss temporal properties of web services.

2.1 Transactional Web Services Composition

The basic unit of web services world is elementary services which do not rely on other services. A composite service consists of multiple other elementary and composite services, whose execution is managed by a process. Generally, component services are used to refer to these services involved in a composition. The process contains a set of tasks performed by component services. The assignment of services to tasks (i.e., service selection) is not only based on functional requirements but also on non-functional requirements [14] such as execution price. In addition, there are two selection modes: static and dynamic selection. The former means component services to be used are selected at design time while the latter means component services are selected at run time.

To ensure composite services terminate in a consistent state even in the presence of failures, transactional support needs to be added into service composition. The first step towards such a mechanism is to identify transactional properties of elementary services. We use here the taxonomy of transactional properties introduced in [10]: compensatable (c), retrievable (r), and pivot (p). A service is said to be compensatable if it offers compensation policies to semantically undo its effects. A service is said to be retrievable if it is sure to complete successfully after a finite number of tries. A service is said to be pivot if it is neither compensatable nor retrievable. It should be noted that a service could be compensatable and retrievable at the same time (denoted as rc in this case). From an operation point of view, compensatable services have two kinds of operations: normal operations to fulfill functional requirements and compensation operations to semantically undo the effects of normal operations (for simplicity, we consider each kind has only one operation). Retriable and pivot services only have normal operations. The transactional property of a composite service depends on its component services. If all its component services are compensatable/retrievable, it is compensatable/retrievable; otherwise, it is pivot. We consider here the method of designing transactional composite services presented in [11], where designers deduce the required transactional properties of every task based on a set of predefined transactional requirements and then select component services according to those transactional properties. Therefore, the transactional properties of composite services could be predetermined.

After giving these definitions, some constraints and rules on component services and process structure of composite services could be deduced [11, 12]. For example, suppose s_c is the kind of composite service which contains at most one pivot service s_p . If s_c contains s_p , all predecessors of s_p should be compensatable and all successors of s_p should be retrievable. In addition, concurrent execution of s_p and any other service is forbidden. If s_c does not contain s_p , then any compensatable service in s_c must be predecessor of any retrievable service. Clearly, s_c can always terminate in a consistent state since all the completed services can always be compensated for when an error occurs during the execution of s_c .

Most compensatable services, however, have temporal constraints attached to their compensation operations. In particular, they become uncompensatable after a deadline elapses. If there is an error after the deadline, then s_c will terminate in an inconsistent state. For simplicity, we assume s_c is well-formed, namely, it can always terminate in a consistent state when temporal constraints are not taken into account.

2.2 Temporal Properties of Web Services

Web services and their operations have some inherent temporal properties, for example, execution time and available time. Execution time corresponds to the delay between the moment when a request is sent (i.e., an operation and by extension a web service is invoked) and the moment when the results are received. In the context of web services, execution time of normal operations may be very long, from minutes to hours even days. Available time indicates when an operation and by extension a web service can be invoked. For example, service providers may declare (the normal operations of) their services only work within business hours from 9am to 5am or the compensation operations of their services only available within 15 minutes from the time when their normal operations are completed successfully. As seen from those real examples introduced earlier, most services specify a deadline for compensation from a specific time such as when they complete. To facilitate later discussion, we assume services use *maximal compensatable time* (MCT) to specify the deadline. For example, a service may declare its MCT equals to 15 minutes which means it can only be compensated for within 15 minutes from the time when it completes. We also assume that individual service does not impose constraints on the available time of its normal operations since we focus on the compensation issue.

These temporal properties along with above transactional properties could be defined and published publicly by using multiple approaches, such as extended WSDL or service agreement so that service designers can obtain and make use of them. Because this is another topic outside the scope of our paper, we assume here service designers can directly obtain this non-functional information from service providers.

3. MODELING AND METHODOLOGY

3.1 Problem Modeling

For the sake of simplicity, we use a real number to represent a time and ignore the unit of time value. Therefore, the complex computations on time are transformed into the simple computations on real number.

Definition 1. A time interval (TI) is a 2-tuple(t_s , t_e) where t_s and t_e are real numbers denoting starting time and ending time.

Definition 2. A web service s is a 5-tuple (tp , o_n , o_c , t , sta), where

1. tp defines the transactional property of s , where $tp \in \{r, c, rc, p\}$.
2. o_n defines the normal operation of s .
3. o_c defines the compensation operation of s . For retrievable and pivot services, o_c equals to null.
4. t is an instance of TI, and $t.t_s/t.t_e$ records the starting/complete time of s .
5. sta records current state of s , where $sta \in \{\text{initial, active, completed, aborted, failed, cancelled, compensated}\}$.

Each operation is associated with two temporal properties: *execution time* (t_x) and *available time* (t_a) where t_x is a positive real number and t_a is an instance of TI. Generally, the execution time of s refers to the execution time of the normal operation of s .

Definition 3. The process of a composite service can be modeled as a directed graph $G(\underline{V}_t, \underline{V}_o, \underline{E})$, where

1. $\underline{V}_t = \{v_1, v_2, \dots, v_n\}$, $v_i \in \underline{V}_t$ ($1 \leq i \leq n$) represents a task in the process.

2. $\underline{V}_o = \{v_1, v_2, \dots, v_m\}$, $v_i \in \underline{V}_o$ ($1 \leq i \leq m$) represents a control-flow operator cfo, where $cfo \in \{\text{AND-split, AND-join, XOR-split, XOR-join, OR-split, OR-join, loop}\}$.
3. \underline{E} is a set of directed edges. Each edge $\underline{e} = (v_1, v_2) \in \underline{E}$ indicates the control dependency between v_1 and v_2 , where $v_1, v_2 \in \underline{V}_t \cup \underline{V}_o$.

Because a task can be performed by multiple services, we need to select an appropriate service for every task. Therefore, we have the following definition.

Definition 4. An instance of a process graph $\underline{G}(\underline{V}_t, \underline{V}_o, \underline{E})$ is defined as a directed graph $G(\underline{V}_s, \underline{V}_o, E)$, where

1. $\underline{V}_s = \{v_1, v_2, \dots, v_n\}$, $v_i \in \underline{V}_s$ ($1 \leq i \leq n$) represents a service defined in Def. 2 to perform the task v_i .
2. $\underline{V}_o = \underline{V}_o$.
3. E is a set of directed edges. For $\forall \underline{e} = (v_i, v_j) \in \underline{E}$, $\exists e = (v_i, v_j) \in E$.

We can now use G to describe a composite service s_c and \underline{G} to describe the process of s_c . Based on the above definitions we can define service compensatability from a temporal perspective.

Definition 5. An elementary service s_c ensures compensatability at time t if and only if $s_c.tp \in \{c, rc\}$ and $t \in s_c.o_c.t_a$.

When an error occurs during the execution of a composite service, all its completed component services must be compensated for to reach a consistent termination. Because of the autonomy and independence of component services, the composite service can simply inform the completed component services at the same time that they need compensation. Therefore, a composite service s_c ensures compensatability at time t if and only if all its completed component services ensure compensatability at time t .

Definition 6. A composite service s_c ensures compensatability at time t if and only if in its G , for $\forall v_i \in \underline{V}_s$ such that $v_i.tp \in \{c, rc\}$ and $v_i.sta = \text{"completed"}$, $t \in v_i.o_c.t_a$.

Note that by s_c ensures compensatability we do not imply s_c has the "compensatable" transactional property. If a composite service ensures compensatability in its lifecycle, then it can terminate in a consistent state. Hence, we have the following theorem.

Theorem 1. A composite service s_c ensures consistent termination if in its G , $v_i.tp \in \{c, rc\}$, $s_c.t.t_e \in v_i.o_c.t_a$ for $\forall v_i \in \underline{V}_s$.

Proof. Clearly, $s_c.t.t_e$ is the worst case when an error may occur during the execution of s_c . Since $s_c.t.t_e \in v_i.o_c.t_a$, all compensatable component services of s_c can be compensated for whenever an error occurs. Hence, s_c always terminates in a consistent state.

3.2 Methodology

The problem is how to ensure s_c can always terminate in a consistent state when temporal constraints are considered. By theorem 1, we only need to compute when s_c completes and the available time of compensation operation of each component service whose transactional property is c or rc . As mentioned in Section 2, service designers can obtain non-functional information from service providers, in particular, transactional property, execution time and MCT. Therefore, it seems that simple computation could solve part of the problem since the computation could tell us whether s_c can ensure compensatability before its execution. However, this is not an easy task. If we want to judge the condition in theorem 1, we must hold a global view of s_c before its execution, that is, we

must obtain the temporal information of every component service. Unfortunately, sometimes it is impossible. For instance, if we use dynamic selection, we can only obtain the required information of services which are about to start. Besides, if a component service itself is a composite service, it either can not provide such information until it completes or provides history information which may be inaccurate. In these cases, we are not able to hold a complete and accurate global view before execution of s_c .

Therefore, a step by step methodology is taken where we first make some assumptions to predigest the problem and give a preliminary framework, then remove these assumptions one by one and finally solve the original problem. For a composite service s_c , we have four assumptions:

- A1. All its component services are elementary.
- A2. It uses static selection.
- A3. All its component services are compensatable.
- A4. There are only AND-split and AND-join operators in its process.

By assumptions A1 and A2, we can hold a global view of s_c before its execution. Consider G as illustrated in Figure 1, which is an instance of process graph of s_c . G has only AND-split and AND-join operators and thus follows assumption A4. Temporal information of component services is given in table 1. In the next section, we will present our approach to ensuring consistent termination of s_c through this numerical example.

Table 1. Execution time (T) and MCT per web service (S)

S	T	MCT	S	T	MCT	S	T	MCT
s ₁	8	30	s ₄	3	15	s ₇	3	10
s ₂	4	14	s ₅	5	25	s ₈	3	18
s ₃	6	14	s ₆	5	20	s ₉	2	10

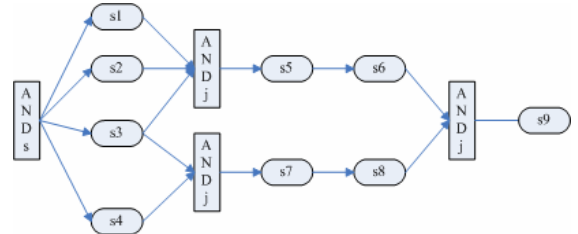


Figure 1. An instance of process graph

4. FRAMEWORK

An overview of our framework is illustrated in Figure 2. There are four main components: process pool, selector, consistency monitor (CM) and execution engine. When a new composite service s_c is designed, its process graph \underline{G} is saved in the process pool. Before execution of s_c , the selector selects services based on functional and non-functional requirements for each task in \underline{G} and sends G , an instance of \underline{G} , to CM in which consistent termination condition is checked by CEP (Consistency Evaluation Procedure). If the condition in theorem 1 holds, s_c can be executed; otherwise new component services are selected. If all available services are tried but s_c still violates consistent termination condition, further actions such as negotiation and redesign will be taken. During the execution of s_c , CM will assist the execution engine to ensure s_c to terminate in a consistent state.

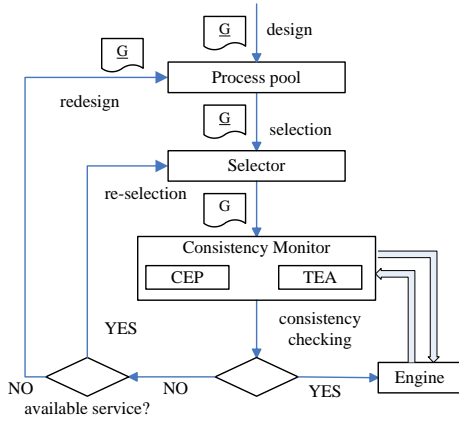


Figure 2. Overview of our framework

In the rest of this section, we detail how CEP checks consistent termination condition based on a global view of s_c (Section 4.1), and present how CM works (Section 4.2).

4.1 Static Evaluation

CEP checks consistent termination condition based on the temporal and transactional information obtained from service providers. Under the assumptions we have made in Section 3, CEP holds a global view of s_c and thus the computation is straightforward. In particular, we can use a topological-sorting-based function named $NCTime()$ to compute when s_c and its component services complete. The pseudo code of this function is omitted here due to limited space. Consider the example introduced earlier. Assume s_c starts at time 0, and a component service starts when all its predecessors complete. Using function $NCTime()$, we have $s_{2.o_c.t_a} = (4, 18)$, $s_{4.o_c.t_a} = (3, 18)$, $s_{7.o_c.t_a} = (9, 19)$ and $s_{c.t.t_e} = 20$. Since $s_{c.t.t_e} \notin s_{2.o_c.t_a}$, s_c does not ensure consistent termination.

We have seen how to make use of a-priori information to check consistent termination condition. By running multiple iterations of computation and re-selection, sometime even negotiation or re-design, we can always achieve our goal. However, the results from function $NCTime()$ sometimes lead to unnecessary re-selections, negotiations and re-design. Due to the deadline characteristics of compensation, we should start the execution of component services as late as possible. In particular, when several services need synchronization through the AND-join operator, we can selectively delay the starting time of them. Two criteria should be considered. First, this delay should result in the overall compensatability of composite services. Second, this delay should not affect the complete time of composite services. Therefore, we need to compute the minimal and maximal delay of every component service. The minimal delay of service s $t_{\min}(s)$ can be directly deduced from the result of function $NCTime()$: $t_{\min}(s) = \max\{0, s_{c.t.t_e} - s.o_n.t_x - s.MCT\}$. The maximal delay $t_{\max}(s)$ of a component service s could be obtained by function $DCTime()$ which, as illustrated in Figure 3, also computes when s_c and its component services can complete at the latest.

Consider again our numerical example, after using function $DCTime()$, we have $t_{\max}(s_2) = 4$, $t_{\max}(s_4) = 9$, $t_{\max}(s_6) = 6$. We have also $t_{\min}(s_2) = 2$, $t_{\min}(s_4) = 2$ and $t_{\min}(s_6) = 1$. If we use the maximal delay value, we have $s_{2.o_c.t_a} = (8, 22)$, $s_{4.o_c.t_a} = (12, 27)$ and $s_{7.o_c.t_a} = (15, 25)$. If we use the minimal delay value, we have $s_{2.o_c.t_a} = (6, 20)$, $s_{4.o_c.t_a} = (5, 20)$ and $s_{7.o_c.t_a} = (10, 20)$. By theorem 1, s_c ensures consistent termination in both cases.

Function: $DCTime(g)$ // g is an instance of process graph

Begin

```

1 NCTime(g);
2 for (int i = 1; i ≤ n; i++) {
3   td[j] = v[j].t.t_e; v[j].t.t_e = s_c.t.t_e;
4   if (count[i] == 0) st.push(i); //count records vertices' out-degree
5 }
6 while (!st.isEmpty()) { //st is a stack
7   int j = st.pop();
8   v[j].t.t_s = v[j].t.t_e - v[j].o_n.t_x;
9   Edge e = v[j].adj; //the edge whose tail is v[j]
10  while (e) {
11   int k = e.head; // the head of edge e
12   if (--count[k] == 0) st.push(k);
13   v[k].t.t_e = min(v[k].t.t_e, v[j].t.t_s);
14   e = e.link; //the next edge whose head is v[j]
15  }
16  for (int i=0; i ≤ n; i++)
17   td[i] = v[i].t.t_e - td[i]; //td records the maximal delay
18 }

```

End

Figure 3. Pseudo Code of Function $DCTime()$

We can select any value from the range specified by $t_{\min}(s)$ and $t_{\max}(s)$ for service s if the temporal properties are strictly followed. In the web services world, however, this is unrealistic. For example, the execution time of a service depends on many factors such as system workloads and thus may increase or decrease in a small range against its declared value. This problem is denoted as UET (Uncertainty of Execution Time). UET may lead to inconsistent termination of composite services since the global view held by CEP is inaccurate. For example, consider that we select the minimal delay for s_2 , s_4 and s_7 , and then put s_c into execution. Assume the execution time of s_1 changes from 8 to 9. In this case, s_c may terminate in an inconsistent state since an error may occur at time 21. Similar cases may happen if we select the maximal delay.

We now consider this problem from a qualitative point of view. Actually, there is a critical path in G , an instance of process graph of s_c . The increase of the execution time of any service in the critical path will lead to a delayed completion of s_c . Therefore, if we select the minimal delay, inconsistent termination occurs if the execution time of any service in the critical path increases. On the other hand, if we select the maximal delay for service s , then s will become a service in the critical path and the increase of its execution time may in turn lead to an inconsistent termination. Therefore, a better value may be the mean of minimal and maximal delay. If we want to obtain an optimal value, we need to model more parameters such as the probability function of execution time, and do more complex deduction. The quantitative analysis is one direction of our future work. It should be noted that even we have obtained the optimal value, inconsistent termination may still happen since this value only decreases the possibility of inconsistent termination.

To check the consistent termination condition, we need traverse graph G . Since functions $NCTime()$ and $DCTime()$ are based on topological sorting, the time complexity of checking procedure is $O(n+e)$ where n and e are the number of vertices and the number of edges in G , respectively.

4.2 Dynamic Monitoring

The objective of dynamic monitoring is to ensure consistent termination of composite services even in the presence of failures and uncertainty of execution time of component services. The consistency monitor (CM) of Figure 2 is used to achieve this goal, and the working process of CM is detailed in Figure 4. The main components of CM are CEP and time exception alarm (TEA). CEP checks consistent termination condition. TEA records the time when the compensation operation of some services becomes unavailable, and sends an exception message (step 8) to execution engine before that time (e.g., 5 minutes in advance). Note that by how much time in advance to send such an exception is application-specific and can be decided by designers. When the execution engine receives this message, it aborts the composite service.

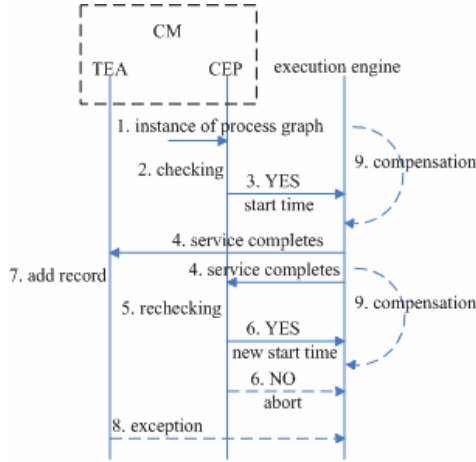


Figure 4. CM working process

As illustrated in Figure 4, for a composite service s_c , CM first uses CEP to check whether it violates consistent termination condition (step 2), and meanwhile obtains additional temporal information about its component services (such as starting time and complete time). If the consistent termination condition is true, it is sent to the execution engine to execute (step 3). When a component service completes (step 4), CM adds a record in TEA to indicate when the compensation operation of this service will be unavailable (step 7), and uses CEP to recheck consistent termination condition according to the up-to-date temporal information (step 5). Based on the result, CM tells the execution engine either to abort the composite service or when to invoke other component services (step 6), thereby ensuring the compensation operations of every component service to be always available during the execution of composite services. When the execution engine detects a failure, it tells every component service to take compensation actions (step 9).

If the execution time of a component service s becomes longer, the execution time of the composite service may inevitably increase. If this is acceptable, then we need only consider the consistent termination condition: before s returns the result, TEA can ensure that those completed component services can be compensated for by alarming in advance; after s returns the result, CM can use CEP to recheck the consistent termination condition. If the execution time of service s becomes shorter, there will be no obvious problem since CM will use CEP to recheck the consistent termination condition. Hence, CM ensures consistent termination of composite web services even in the presence of uncertainty of execution time. In other words, CM can handle UET problem.

Because CEP needs to recheck consistent termination condition whenever a component service completes, the time complexity of dynamic monitoring is $O(n^2+ne)$ where n and e are the number of vertices and the number of edges in G , respectively.

5. REMOVING ASSUMPTIONS

Up to now, we have presented a preliminary framework to ensure consistent termination of composite services under four assumptions. In this section, we refine our framework by removing assumptions one by one.

Since our approach is based on the global view of composite services, we first consider a kind of problem named “information missing” (IM), which means we can not obtain some information of some component services until they complete. In particular, we consider two sub problems:

IM1: The information of execution time is missing;

IM2: The information of available time of compensation operation is missing.

For *IM1* sub problem, we can assume the missing execution time equals to 0. Then, *IM1* becomes a particular UET problem with an increased execution time, which has been discussed in Section 4.2. Therefore, our approach can also handle *IM1*. For *IM2* sub problem, the missing information will be obtained when the corresponding service completes. At that time, CM will use CEP to recheck consistent termination condition and in the worst case the composite service will be aborted, but consistent termination could still be achieved. Hence, our approach can also handle *IM2*.

We are now in a position to remove our four assumptions. Consider first assumption A4. For the loop control-flow operator, its effects on a service s correspond to increased execution time of s from a temporal perspective. Since the number of loops often can not be predetermined, the increase of execution time is also uncertain. This is just the UET problem with an increased execution time, but our approach needs a little modification: for each instance of s , CM will add a record in TEA when the instance completes. When the XOR/OR operators are involved in the processes of composite services, CM needs to check the corresponding transition condition before executing a service. If the condition is true, the service starts; otherwise, CM uses CEP to recheck consistent termination condition. After adding these modifications, A4 can now be removed.

We now consider the assumption of A3. Actually, retrievable services do not affect compensatability since there is no permanent error during their execution. The termination of a pivot service means all its predecessors no longer need compensation. Consider G , an instance of process graph \underline{G} . If there is only one pivot service in G , we need first remove all retrievable services (i.e., vertices) and corresponding edges in G to obtain a new graph on which CEP works. If there are multiple pivot services, G is divided into several sub-graphs, each of which is handled like G with one pivot service. Hence, A3 can be removed.

Consider next the assumption of A2. If dynamic selection is used, we can only obtain the information of selected services. This is actually the combination of *IM1* and *IM2* problems. Hence, A2 can be removed. Finally, consider the assumption of A1. If a component service is a composite service, it will miss some information due to dynamic selection. This is still the combination of *IM1* and *IM2* problems. Summarizing all above discussions, our framework can always ensure consistent termination of composite services.

6. RELATED WORK

Much work has been done recently to address the requirements of transactional support for web services. Several standards, such as BTP, WS-C/WS-AT/WS-BA, and WS-TXM, are based on extended transaction models and have defined a standardized way for web services to interact with their coordinators. All of these provide support for compensation-based long running activities.

Compensation based transaction models have also been studied in earlier work in the area of multi-database [15] and process-based middleware [12]. These work highlighted how to achieve atomicity by using compensation when some local transactions and processes do not provide compensation methods. Based on these efforts, many transaction models have been designed for heterogeneous transactional web services. In [9], a framework called WSTx was presented to describe different transactional capabilities of service providers and requirements of client applications. [13] presented a multi-level composition model which allows specification of atomicity and guaranteed termination properties at different levels. The transaction model proposed in [4] adopted the original THP to avoid resource blocking, and extended the strict atomicity by defining a parallel composition operator with minimality and maximality constraints. In [3], an approach to ensure failure atomicity of composite web services was presented. It adapted the property of accepted termination states by relaxing the atomicity, and thus allowed designers to define different levels for transaction termination. In [11], designers could deduce the required transactional properties of every task based on a set of accepted termination states and use the automatically produced coordination rules to coordinate the service execution. A solution for fault-tolerant web services orchestration by using relaxed atomic execution and exception handling was presented in [8]. All of this work has ignored the constraints on compensation, especially temporal constraints which are common and important in the context of web services [1].

There is also some work focusing on temporal properties of web services. In [2], temporal abstractions were exploited for the compatibility and replaceability analysis of web service protocols. A formal approach for modeling and analyzing temporal properties of web service composition was presented in [6]. However, this work did not consider how these temporal properties would affect the web services transaction.

7. CONCLUSION AND FUTURE WORK

In this paper, we have presented a framework to ensure consistent termination of composite web services with temporal constraints. On the one hand, it can help designers construct a reliable composition, and if service selection is carried out during the execution of composite services, it also provides another criterion for selection. On the other hand, it can dynamically check the consistent termination condition of composite services. In case the condition gets changed during service execution, it will give another scheduling plan. If it can not find any scheduling plan, it will abort the composite service so that the service always terminates in a consistent state.

We only consider temporal constraints of compensation in this paper. In the web services world, however, cost is another important kind of constraints [1]. In particular, compensation may have different costs in different times. Next, we will consider this issue in our work.

As stated before, we are currently devising a solution to find an optimal delay, which determines the possibility of successful completion of composite web services in a dynamic environment. It will be more challenging when cost is considered.

Our approach is based on the notion of complete compensation where all the completed component services need to be compensated for. However, there may be some safe-points [5] in composite services and thus only partial compensation is needed. Support for partial compensation is one of the next steps of our work.

8. ACKNOWLEDGEMENTS

The research described here is supported by the National Basic Research Fund of China ("973" Program) under Grant No.2003CB317006, and has been benefited from various discussions among the group members of the Joint Research Lab between CityU and USTC in their advanced research institute in Suzhou (China), particularly Prof. Huang Liusheng, Dr. Xiao Mingjun, Mr. Liu Hai and Lin Baoping.

9. REFERENCES

- [1] Benatallah, B., et al. Web Service Conversation Modeling: A Cornerstone for E-Business Automation. *IEEE Internet Computing*, 8(1): 46-54, 2004.
- [2] Benatallah, B., et al. On Temporal Abstractions of Web Service Protocols. In *Proc. CAiSE'05*, 2005.
- [3] Bhiri, S., Perrin, O., and Godart, C. Ensuring Required Failure Atomicity of Composite Web Services. In *Proc. WWW'05*, 2005.
- [4] Fauvet, M., et al. Handling Transactional Properties in Web Service Composition. In *Proc WISE'05*, 2005.
- [5] Grefen, P., Vonk, J., Apers, P. Global Transaction Support for Workflow Management Systems: From Formal Specification to Practical Implementation. *VLDB J.* 10(4): 316-333, 2001.
- [6] Kazhamiakin, R., Pandya, P., and Pistore, M. Representation, Verification and Computation of Timed Properties in Web Services Compositions. In *Proc.ICWS'06*, 2006.
- [7] Little, M. Models for Web Services Transactions. In *Proc. SIGMOD'04*, 2004.
- [8] Liu, A., et al. Fault-tolerant Orchestration of Transactional Web Services. In *Proc. WISE'06*, 2006.
- [9] Mikalsen, T., Tai, T., and Rouvellou, I. Transactional Attitudes: Reliable Composition of Autonomous Web Services. In *Proc. DSN'02*, 2002.
- [10] Mehrotra, S., et al. A Transaction Model for Multidatabase Systems. In *Proc. ICDCS'92*, 1992.
- [11] Montagut, F. and Molva, R. Augmenting Web Services Composition with Transactional Requirements. In *Proc. ICWS'06*, 2006.
- [12] Schuldt, H., et al. Atomicity and Isolation for Transactional Processes. *ACM TODS*, 27(1): 63-116, 2002.
- [13] Vidyasankar, K. and Vossen, G. A Multi-Level Model for Web Services Composition. In *Proc. ICWS'04*, 2004.
- [14] Zeng, L., et al. Quality Driven Web Services Composition. In *Proc. WWW'03*, 2003.
- [15] Zhang, A., et al. Ensuring Relaxed Atomicity for Flexible Transactions in Multidatabase Systems. In *Proc. SIGMOD'94*, 1994.