

Web Service Composition Transaction Management

Benchaphon Limthanmaphon

Department of Computer and Information Science
Faculty of Applied Science
King Mongkut's Institute of Technology
North Bangkok
Bangkok, Thailand
blt@kmitnb.ac.th

Yanchun Zhang

School of Computer Science and Mathematics
Victoria University of Technology
Melbourne, Australia
yzhang@matilda.vu.edu.au

Abstract

The development of new web services by composition of existing services is becoming an extensive approach. This has resulted in transactions that span in multiple web services. These business transactions may be unpredictable and long in duration. Thus they may not be acceptable to lock resources exclusively for such long period. Two-phase commit is also not suitable for transactions with some long sub-transactions. Compensation is a way to ensure transaction reliability. However, rolling back a previously completed transaction is potentially expensive. Thus, tentative holding is another option. This paper presents a transaction management model for web service composition. We apply the approach of tentative hold and compensation for the composite transaction. We also present a multi-dimension negotiation model for the service composition.

Keywords: Web services composition, Web Service Transaction Management, Compensation, Tentative hold, Negotiation

1 Introduction

A web service can be described broadly as a service available via the Internet that conducts transactions. Service composition refers to the process of creating customised services from existing services by a process of dynamic discovery, integration and execution of those services in a deliberate order to satisfy user requirements (Chakraborty et al. 2002). Integrating or composing services from different and heterogeneous business entities is necessary to the discussion of transaction management issues. Web services that are capable of intelligent interaction would be able to discover and negotiate with each other, mediate on behalf of their users and compose themselves into more complex services.¹

A transaction involving multiple web services is composed of many autonomous sub-transactions that abort or commit independently. In other words, web service transactions are loosely coupled. They are possibly involving and spanning many enterprises. They

may be unpredictable and long in duration. Thus they may not be acceptable to lock resources exclusively for such long period. Also, two-phase commit protocol involves one or other form of resource locking. Longer periods of resource locking will result in serious scalability issues. If each web service invocation is executed as an independent transaction, the way to guarantee the desired all-or-nothing property of transactions is through the notion of compensation. However, not all web services will support compensation. Moreover, rolling back a previously completed transaction is potentially expensive. Thus tentative holding is another option. The effect of a tentative hold is to allow tentative, non-blocking holds or reservations to be requested for a business resource. The resource owners grant non-blocking reservations on their services, preserving control of their resources while allowing many potential clients to place their requests. This facility will provide clients with up-to-date data and minimise the need for cancellations.

In this paper we apply two concepts of tentative hold and compensation to manage service composition. We also present the negotiation model for service composition.

The rest of this paper is organised as follows: Section 2 presents an overview of service composition and related works. Transaction models are presented in section 3. Our tentative hold and compensation transaction model for service composition is presented in section 4. Section 5 presents the negotiation concept during composition. Finally, the conclusion and future research direction are presented in section 6.

2 Overview and Related Work

There are several ways of classifying service composition. Chakraborty and Joshi (2001) classified service composition in terms of offline and on-line processing which refer to pro-active and reactive composition respectively. Pro-active service composition means offline or pre-compiled composition of available services to form new services. Services that compose in a pro-active manner are usually stable and used at a very high rate over the Internet. Reactive service composition refers to the type of composition that is executed only upon request or created on the fly (Chakraborty et al. 2002). It requires a component manager to take the responsibility of collaborating with the different sub-services to provide the composite service to the client. The interaction cannot be predefined and varied according to the dynamic situation.

Copyright ©2004, Australian Computer Society, Inc. This paper appeared at the Fifteenth *Australasian Database Conference (ADC2004)*, Dunedin, New Zealand, Conferences in Research and Practice in Information Technology, Vol. 27. Klaus-Dieter Schewe and Hugh Williams, Ed. Reproduction for academic, not-for profit purposes permitted provided this text is included.

There are some works focus on transaction management (Strandenaes and Karlsen 2002; Mikalsen, et, al. 2002; Pires, et, al. 2002; Park and Choi 2003). The WSTx framework was proposed based on reliability in web service composition (Mikalsen et, al. 2002). The model describes three transactional attitudes: pending-commit, group-pending-commit, and commit-compensate. The WebTransact framework was proposed to treat the problem of building composition in an integrated way by providing mechanisms to describe the dissimilar transaction behaviour of web services (Pires, et, al. 2002). Strandenaes and Karlsen (2002) describe a technique for implementing compensating transactions based on the concept of triggers, while Park and Choi (2003) applied the concept of tentative hold to allow placement of business resources by considering the performance optimisation of tentative hold through two parameters: over hold size and hold duration.

Our previous work (Limthanmaphon and Zhang 2003) presented the model that composition process executes in two modes: proactive, and reactive which consists of three main components: request analyst, outsource agents and services composer. Service relationships are defined and used during the proactive phase. We used the Case-Based Reasoning (CBR) techniques to evaluate the customer's queries and plan the composition service.

3 Transaction Models

Basic transactions or traditional transactions refer to transactions endowed with the properties of atomicity, consistency, isolation and durability (ACID), while complex transactions refer to extended and relaxed transactions. Extended transactions permit grouping of their operations into hierarchical structures while relaxed transactions indicate that a given transaction model relaxes some of the ACID requirements.

In this section we briefly review the existing transaction models and then present the existing web service and business transactions.

3.1 Extended and Relaxed Transaction models

An important part of the evolution of a traditional transaction model is the extension of the flat or single level transaction structure to multi-level structures (Zhang and Jia 1999, Zhang et, al. 1999).

- **Nested Transactions** permits transactions to be nested within transactions to form a transaction tree (Moss 1981). A child transaction may start after its parent has started and a parent transaction may terminate only after all its children terminate. If a parent transaction is aborted, all its children are aborted. These commit/abort and resource inheritance strategies are applied recursively throughout the tree. Nested transactions have benefits in three areas. First, they provide full isolation on the global level, while permitting increased modularity – each transaction can be decomposed into a hierarchy of cooperating sub-transactions. The next benefit is to provide finer granularity of failure handling. Recovery action can be taken at

the level of failed sub-transactions. Last, non-conflicting sub-transactions can be executed concurrently (so called intra-transaction parallelism).

- **Open Nested Transactions** relax the isolation requirements by making the results of committed sub-transactions visible to other top-level transactions (Wiekum and Schek 1992). In open nested transactions, the abort of a top-level transaction requires compensation for committed sub-transactions. In other words, a sub-transaction can commit and release the resources before the global transaction successfully completes and commits. If the global transaction later aborts, its failure atomicity may require that the effects of already committed sub-transactions be undone by executing compensating sub-transactions.
- **The Saga Transaction Model** permits a long-lived transaction to be divided into a sequence of sub-transactions (Garcia-Molina and Salem 1987). Each transaction has an associated compensating sub-transaction that can be triggered to semantically undo the effects of its committed associate. This means that a saga consists of a set of ACID sub-transactions T_1, \dots, T_n with a predefined order of execution, and a set of compensating sub-transactions CT_1, \dots, CT_{n-1} , corresponding to T_1, \dots, T_{n-1} . If a saga sub-transaction T_k fails and cannot recover, its partial effects are undone by executing compensating sub-transactions CT_{k-1}, \dots, CT_1 .
- **The Split-Join Transaction Model** was designed as its name implies, to split itself into two independent or dependent transactions and later join together to form a single transaction [Pu 1988, Pu et, al. 1988]. It was designed for open-ended activities characterised by uncertain, but normally very long-duration, unpredictable development, and interaction with other activities.
- **ConTracts** is a mechanism for grouping transactions into a multi-transaction activity (Reuter 1989). It consists of a set of predefined actions called steps, and an explicitly specified execution plan called a script. An execution of a ConTract must be forward-recoverable. In the case of a failure the state of the ConTract must be restored and its execution may continue.
- **Long-Running Activity** is modelled as a set of execution units that consist recursively of other activities or transactions (Dayal et, al. 1991). Control flow and data flow of an activity may be specified statically in the activity's script or dynamically by Event-Condition-Action (ECA) rules.

3.2 Web Service and Business Transactions

Web service transacting would be exactly the same as any other distributed transaction management system. However, the following characteristics of web service mean that its requirements are different:

- Web service transactions will usually be conducted across organisational boundaries. This implies that transaction participants will be autonomous and distributed across the Internet (Mani and Nagarajan 2002). Due to the limited utilisation of transaction management protocol standards in general, participants are likely to be using incompatible transaction management implementations.
- Web services transactions can be long running, however organisations cannot afford to allow their resources to be consumed unpredictably in an open environment such as the Internet. This implies that extreme care should be taken to make sure that resources are not blocked for long periods of time (Mani and Nagarajan 2002). However, this conflicts with the application of strict ACID properties, because ACID implies resources must be locked until the transaction terminated, so that isolation and consistency are preserved. Moreover, the commit protocol is particularly vulnerable to the effects of network instability and malicious attacks, which can lead to resources being locked for long or indefinite periods. The natural long time frame of web service transactions and the blocking potential of commit protocols may leave web service resources vulnerable to extended locking.

To solve the above problems, the earlier attempts to define an Internet-based transaction protocol that simplifies the distributed application like web service is the proposed Transaction Internet Protocol (TIP) as well as the motivation of creating a business transaction protocol (BTP) to be used in business transaction that require transactional support beyond ACID and extended transactions. Next, tentative hold protocol (THP) is another concept that allows multiple clients to place holds on the same resource to eliminate blocking problems. This protocol minimises the need for cancellations. Lastly, web services transaction (WS-Transaction) defines two models for transactions over web services: atomic transactions and business activity transactions. An important property of activity transactions is that they provide a compensation mechanism, which is needed to support business processes.

3.2.1 Transaction Internet Protocol

Transaction Internet Protocol (TIP) 3.0 as defined in RFC2371 is a transport protocol enabling distributed transaction coordinators to communicate over the Internet (Lyon et, al. 1998, Papazoglou 2003). TIP allows transaction coordination protocols for the recovery of collapsed connections between transaction participants. It

does not attempt to ameliorate the issue of blocking that can occur at the participant endpoints due to 2PC protocol. In the case of failure during the PREPARE and COMMIT/ABORT, a TIP coordinator is expected to wait until communication with the transaction manager restored. Hence TIP has the same flaw of blocking issues as affects other distributed transaction processing systems (DTPs). Moreover, TIP is not a secure protocol and does not require or support authentication. It requires the participants to open an additional bi-directional TCP port (3372), which will allow remote parties to block their local resources by delaying their vote on transaction outcomes.

TIP is a simple 2PC protocol that removes the restrictions of conventional 2PC protocols by providing ubiquitous distributed transaction support in a heterogeneous and cross-domain environment. This is made possible by employing a two-pipe model separating the transaction protocol from the application communications protocol. TIP supports both “push” and “pull” models for starting transactions with multiple servers. In the push model, a client will first asks its transaction manager (TM) to export the transaction to a remote node’s transaction manager to instantiate a transaction, make it as a subordinate to an existing transaction on the client’s TM, and then return the remote TM’s context for the transaction. Then the client will send the work request to the remote node with the remote’s transaction context, and tell it to execute it as part of that transaction. In the pull model, the client requests a remote node to do some work and make it as a part of an existing transaction. The remote node’s TM will pull the transaction over from the client. As a result of this pull, the client’s TM knows to involve the remote node’s TM in a 2PC process.

In summary, the TIP offers flexibility for 2PC protocol-based short-lived transactions, but it falls short in the case of long-lived business transactions. Business transactions consist of a large number of component transactions with largely different response times, thus blocking resources controlled by short-lived transactions for unacceptably long periods of time, making them unable to process new service requests. This is an undesirable model from an autonomous service provider’s point of view.

3.2.2 Business Transaction Protocol (BTP)

BTP is an XML based standard interoperation protocol that defines the role of transaction participants and the messages being passed between them over the Internet (BTP, Dalal et, al. 2001, Papazoglou 2003). The objective of BTP is to orchestrate loosely coupled web services into a single business transaction. It aims to be an underlying protocol that offers transactional support in terms of coordinating distributed multiple autonomous business functionality, in the form of services. The goals of the BTP specification can be summarised as follows (BTP, Potts et, al. 2002):

- Provide a model for transactions over the Internet.
- Integrate reliable outcomes over unreliable communication channels.

- Manage the transaction life-cycle and support the ACID properties.
- Support asynchronous communication between loosely-coupled systems.
- Provide support for long-lived transactions.
- Coordinate multiple autonomous related transactions and sub-transactions.
- Provide a foundation for workflow.

A traditional transaction is normally viewed as atomic, which means that it is a consistency preserving state update. ACID transactions include this consistency along with guarantees on isolation and durability. With ACID transactions, any failure that occurs within the transaction will be rolled back and its effects reversed or erased. For long-lived business transactions, individual constituent work may be ACID in nature, but the overall business transaction employs a compensatory approach to reverse or erase partial work. The concept of isolation in an ACID transaction is relaxed in this model.

BTP is based on two-phase commit for short duration interactions known as atoms, which can be combined into larger non-ACID transactions known as cohesions. Atomic business transactions are small scale interactions made up of services that all agree to enforce a common outcome: either commit or abort of the entire transaction. The cohesive business transactions or cohesions are aggregations of several atomic transactions, which allow the selective confirm (commit) or cancel (rollback) of participants. A cohesive business transaction relaxes the isolation property by allowing the effects of a cohesive interaction to be externally visible before the interaction is committed. Moreover, a cohesion may deliver different termination results to its participants such that some will confirm while the remainder will cancel. Finally, consistency is determined by agreement and interaction between the initiator (or the client) and the coordinator. The initiator is the only participant that is allowed to terminate the transaction. In order to terminate the transaction, the initiator sends a terminate request to the main coordinator. The main coordinator then together with all the subordinate coordinators jointly executes the termination protocol. A transaction can be terminated with success or with error. Transaction termination with error triggers the appropriate compensating transaction.

3.2.3 Tentative Hold Protocol (THP)

Tentative Hold Protocol is an open, loosely coupled, messaging-based framework for information exchange between business partners prior to the actual transaction itself (Roberts et, al. 2001, Papazoglou 2003). The objective of THP is an effort to facilitate automated coordination of multi-business transactions. It defines an architecture that allows tentative, non-blocking holds or reservations to be requested for a business resource. In this paradigm of online ordering, these lightweight reservations are placed prior to the sale, allowing multiple clients to place holds on the same item (thus non-blocking). Whenever one client finishes the purchase of that item, the other clients receive notifications that their

holds are no longer valid. The resource owners grant non-blocking reservations on their products, preserving control of their resources, while allowing many potential clients greater flexibility in coordinating their purchase. The following four states (as shown in figure 1) are associated with the use of THP:

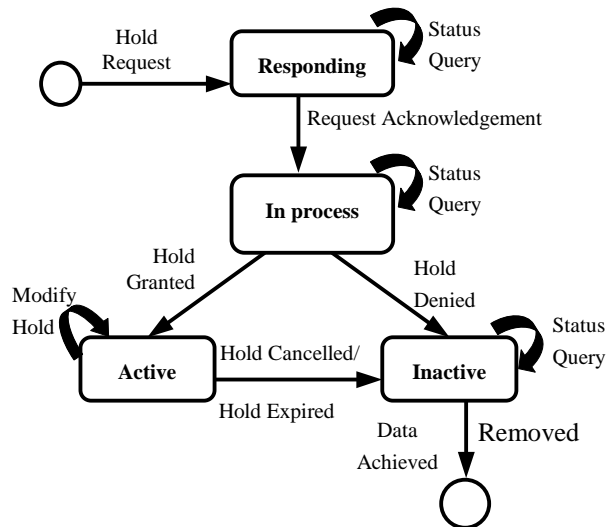


Figure 1. Tentative Hold Protocol State Diagram

- *Responding* is an initial state when an application sends a Hold Request message.
- *In Process* is an intermediate state indicating that the hold request has been received and an acknowledgement of the request has been returned to the hold requestor.
- *Active* is reached when the requested hold has been granted. A Hold Granted message has been sent by the resource owner. In this state the hold requestor may attempt to modify the specifics of this hold by sending a Modify Hold message.
- *Inactive* is reached from the Active state when the tentative hold is no longer valid. A 'Hold Cancellation' message has been sent by the client or the resource owner, or the hold has expired.

There is a THP coordinator on both client and resource owner sides, responsible for communicating various messages such as hold requests, cancellations, and so on. The THP client and resource coordinators should be able to communicate through firewalls. The functionality requirements for the coordinators can be summarised as follows:

- **Client Coordinator Requirement**

First, at start up, the client coordinator shall determine the status of any previously granted holds. Second, the client coordinator shall be a lightweight implementation, capable of running on a wide range of platforms. Third, the client coordinator shall provide an interface that permits a client application to (1) request a hold from a specific resource owner for a specific resource, (2) query existing holds

owned by this client, (3) cancel an existing hold owned by this client, (4) query for the logged THP activities, and (5) request a modification of an existing hold.

• Resource Coordinator Requirements

First, at start up, the resource coordinator shall determine the status of any previously granted holds, then verify the expiration times. Second, the resource coordinator shall respond to client coordinator hold requests either synchronously (hold request respond is sent immediately) or asynchronously (resource coordinator shall send an acknowledgement of the request, followed at some later time by a hold request response). Third, the resource coordinator shall use the resource owner developed *Rules Integration Module* to satisfy new hold requests. Fourth, the resource coordinator shall asynchronously notify affected client coordinators should a resource become unavailable, such as the resource is allocated to another party. Last, the resource coordinator shall provide an interface that permits a resource owner's application to (1) query existing holds granted by this resource owner, (2) cancel an existing hold granted by this resource owner, and (3) query for the logged THP activities.

The benefits of adding a THP phase for business services are: firstly, minimising the need for cancellations. Both the requesting clients and resource owners would benefit from THP. For example, the requesting clients would be less likely to cancel a purchase after they have placed it. Consequently, the resource owner is less likely to have a need to process a cancellation. Secondly, providing clients with up-to-date data to base their decision upon. A client would place a tentative hold on an item and be aware of its current availability. If that item becomes unavailable, the client would be notified by a tentative hold protocol message. Without THP, the client would have no knowledge of the change of state, and might make significant decisions based on obsolete data or data that is no longer valid.

3.2.4 Web Services Transaction

The current set of web service specifications (such as WSDL, SOAP) defines protocols for web service interoperability. Web services increasingly tie together a large number of participants forming large distributed applications. WS-Transaction (Cabrera et, al. 2001) defines how web services coordinate their activities in order to ensure the integrity of underlying database operations. WS-Transaction defines two models for transactions over web services: atomic transactions and business activity transactions.

An atomic transaction (AT) is used to coordinate activities having a short duration executed within limited high level of protection domains. Atomic transactions are small scale interactions made up of services that all agree to enforce a common outcome: either commit or abort the entire transaction (an "all-or-nothing" property). The atomic transaction follows the ACID properties and guarantees that all participants will see the same outcome

(atomic). Each participant typically locks any database records involved in the transaction to prevent any changes from being made to the data while the transaction is being processed. Only when all participants have indicated a readiness to commit does the coordinator instruct them to make the changes. If any participant either rejects the transaction or fails to respond, the coordinator instructs all participants to abort the transaction and discard any changes. This process is typically known as two-phase commit. The flaw of this approach is that each database involved in the transaction must hold some kind of lock on records for the duration of the transaction thereby making those records unavailable to other clients. While the duration of a transaction on an internal network is likely to be relatively small, the wide spanning of web service transaction will cause the problem of the endpoint's resource manager (such as a database) to be locked and blocked for potentially very long periods of time due to network latency.

Business Activity (BA) is defined to support transactions without requiring locks on all database records. It handles long-life transactions. Typically, a business activity is designed as an activity that consists of a sequence of tasks, where each task satisfies the constraints of an atomic transaction. The key behind business activities is compensation. Rather than requiring each participant in the transaction to lock data and hold off on committing changes until all participants approve, compensation assumes that all updates will succeed and commits the changes immediately, but prepares a way to undo the changes and therefore compensate for the failure of any component. Figure 2 shows the business activity state diagram.

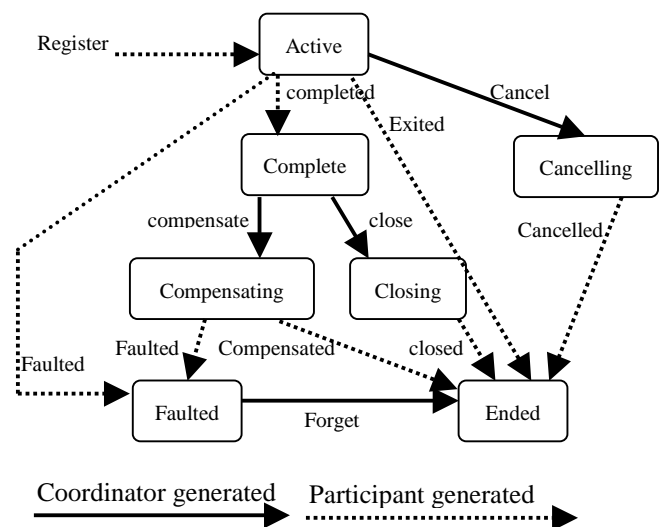


Figure 2. Business Activity State Diagram

It is a fact that going back to the past is not a realistic option in an online system. The only possibility is to install a compensating transaction for what is about to be committed at that moment and keep the compensating transaction around in case the pre-committed results turn out to be invalid. The state diagram is figure 2 specifies the behaviour of the protocol between a coordinator and a participant. The state reflects what both sides know of their relationship.

4 Web Service Tentative Hold and Compensation Composition Transaction Model

From the above extended and relaxed transaction models, each of those approaches has its own strengths and weaknesses. Their suitability depends on the individual business's needs. Because no single technology can provide a solution that successfully supports all businesses and overcomes all the challenges facing the automation of multi-service interactions, mixed technologies are needed.

4.1 State Transition

Consider the example of business trip where a client acquires the hotel and airline services independently. If he/she can reserve the hotel and issue payment but cannot reserve the air ticket on that day, the client should be able to cancel or change the hotel reservation or request partial refund of the payment. Hence, the hotel service may not be happy. With tentative holds, the client would place tentative holds on both services thus ensuring their availability, and then invoke the payment.

We present figure 3, a state machine diagram that captures the behaviour of our model. The multiple paths correspond to the executions of different types of action commands. The diagram consists of nodes, arrows and labels. The nodes describe the different states for an action. The arrows describe the transitions between states, and the labels on the arrows correspond to the conditions required by the respective transitions. The 'Active state' is the initial state. The final states are 'Abort' and 'Ended'.

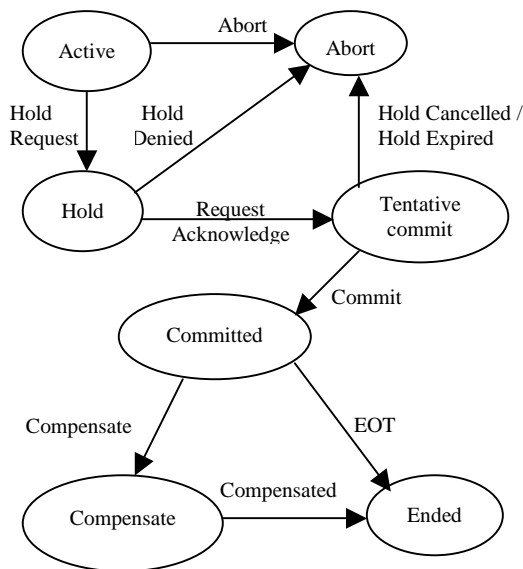


Figure 3. Tentative Hold Protocol and Compensation Transaction State Diagram

The transaction starts in the *active* state. Then it will enter into the *hold* state when an application sends a hold request message. The *tentative commit* is an intermediate state indicating that the hold request has been returned to

the hold requestor. The *abort* state is reached from the hold state when the tentative hold is no longer valid. A hold cancellation or expire will be sent by the service owner. On the other hand, the *committed* state corresponds to the state in which the effects of the action have already been committed to the other services, and therefore, the action can only be compensated for a part of transactions abort. All actions that are either committed or compensated will move into the *ended* state. The EOT indicates end of transaction.

4.2 Reactive relational service composition

Our previous work (Limthanmaphon and Zhang 2003) designed the reactive service composition according to the service relationships. The service composer will orchestrate the most preferred participating services to form a composition service by considering the relationships and constraints of each service. As a result, the service composition will be formed differently. We give some examples below:

Ex1: the business trip service consists of airline and hotel services. They are independent and there is no other requirement as shown in figure 4 (a). The airline and hotel services start independently on different transaction flows T_{11} and T_{21} from the initial state. The result of transaction flow T_{12} is either failure or success in flight reservation and the result of hotel reservation on transaction flow T_{22} will be concluded and evaluated at the intermediate state. For example, if the airline and hotel reservation services are "success", the "start" command will be issued and sent to the payment service (on transaction flow T_3). Vice versa, if one of the results from airline or hotel reservation services is "failure", the "wait" command will be sent. If a failure result is returned from the whole service, the "cancel" command will be sent to cancel the payment service. The payment service will apply and execute the command that issue from transaction flow T_3 . The payment service may have some other constraints such as authentication and / or other security mechanism to be executed. It then generates the result on transaction flow T_4 to complete the whole service composition process.

Ex2: The business trip consists of two services. Each service starts at the same time but the hotel has to wait until the air ticket is available or confirmed by the airline service. Figure 4 (b) describes the situation where the airline and hotel services can start at the same time but the hotel service has to wait the trigger on transaction flow T_{31} . The trigger here is the condition according to the airline services' result (from transaction flow T_{12}). This means that if the air ticket is available or implies that the airline reservation is successful and confirmed, then the hotel service can continue (on transaction flow T_4). The transaction flow T_5 will be issued after the intermediate state evaluates the result of transaction flows T_{32} and T_4 . The payment service will apply the command from transaction flow T_5 and generate the result to T_6 (as described in Ex1 (a)) to finish the service composition process.

Ex3: The business trip consists of two services. The hotel service will start when the airline service finished execution. There is no other constraint between the services as shown in figure 4 (c). It shows that the hotel service will start when the airline service finishes execution. There is no constraint between these services. The transition flows and trigger can be explained as the same as in Ex2 (b).

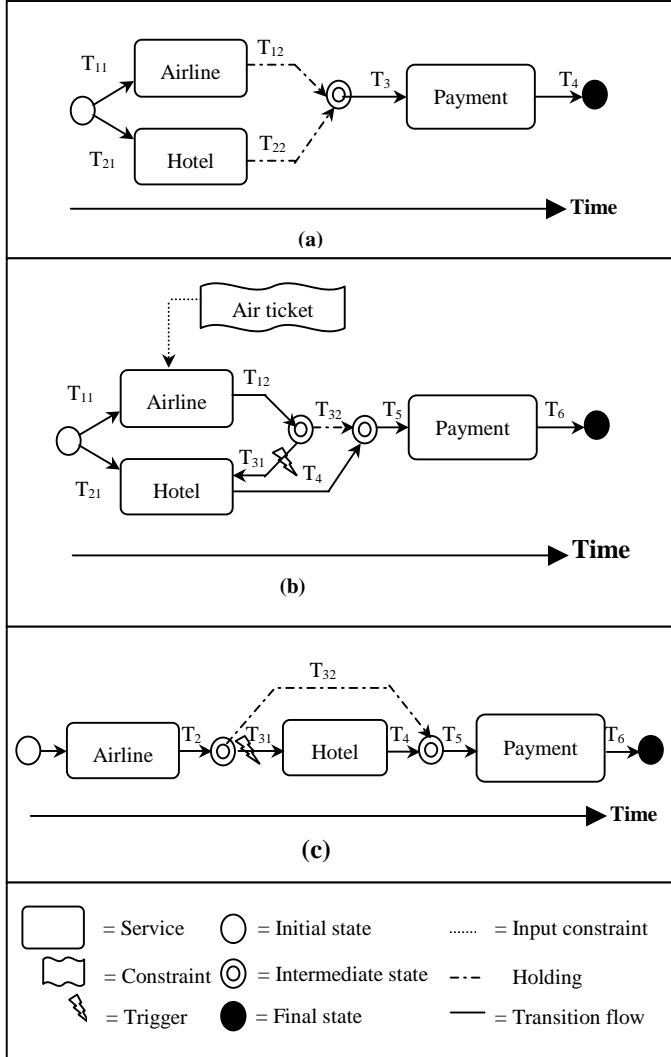


Figure 4. Relational service composition execution flows of business trip in different forms.

5 Negotiation During Composition

It is quite possible that budgetary constraints are the most important issue before placing a tentative hold on any service reservations. For example, a client has a budget for \$A2000. He/She intends to pay \$A1600 for an air ticket and \$400 for a hotel. Unfortunately, the hotel that suits his/her requirements (such as location) charges \$550. He/She may try to negotiate for discount, such as by changing to a lower class, no Internet access, no breakfast provided and so on. Meanwhile, he/she may try to negotiate with the airline service by changing flights that need to transit and stop many places, or changing to a flight that leaves in the very early morning in exchange for a lower price ticket. At the end of the day, the total cost to invoke both services needs to be under budget.

In our previous work (Limthanmaphon and Zhang 2003), we defined service relationships for a composite web service. One of the service relationships is *parallel-dependency relationship*. This means that the participated sub-services are able to execute freely but there are some constraints (for example, price) that share between these services. As a result, the negotiation process is required. Apparently, negotiation is modeled as a business process as it is a set of activities that are performed conforming to a set of activities in order to achieve a goal. We describe the negotiation process for web service composition by giving definitions and steps as follows:

Definition 1: Let S_A and S_B be sub-services of a composite service S . Denoted as $S = \{S_A, S_B\}$. C_t is a constraint to be negotiated between the service composer and sub-services S_A and S_B . $V_A(C_t)$ refers to a value of constraint C_t of sub-service S_A . For example, budget is a constraint to be negotiated. $V_{airline}(\text{budget}) = \1600 and $V_{hotel}(\text{budget}) = \400 .

Definition 2: Let $F(S_A)$ be a list of service features of service S_A while $V_A(C_t)$ is corresponding to $F(S_A)$. For instance, a list of hotel service features is {Type = Single Bed, Class = A, Breakfast = Oriental, Internet access = available, Kitchenettes = yes, Cable = yes, Duration = 3 nights} responding with the cost of \$600. Another example list of hotel service features is {Type = twin share, Class = B, Breakfast = no, Internet access = no, Kitchenettes = no, Duration = 3 nights} responding with the cost of \$400. Denote as $V_i(C_t) \propto F(S_i)$. Any feature of $F(S_i)$ effects the value of constraint C_t .

Note that we use the term 'budget' from the client's point of view and 'cost' from the service provider's point of view to refer to the same constraint.

Definition 3: Let $qV(C_t)$ be the value of a constraint query from a client. Then the summation of the value of constraints from each sub-service S_A and S_B must be less than or equal to the value of the constraint query at the end of the negotiation process. Denote as $qV(C_t) \geq V_A(C_t) + V_B(C_t)$. For example, the client's budget must cover the cost of hotel and airline services.

Definition 4: The list of service features is altered during the negotiation process. Let i be a service feature. $i \in I^+ = \{1, \dots, n\}$ represents the number of n features under negotiation, and each service feature i has m_i alternatives, where $m_i \in I^+$. For instance, feature 1 has m_1 alternatives, feature 2 has m_2 alternatives, and feature n has m_n alternatives. When all the worst or least expensive alternatives of each service feature have been chosen, no further reduction or negotiation over service features can occur.

Definition 5: Negotiation processes between service composer and each service are independent but respect the condition that the value of constraints needs to be compromised under the condition in definition 3 ($qV(C_t) \geq V_A(C_t) + V_B(C_t)$) as shown in figure 5. The service composer can spend time negotiating differently in each service.

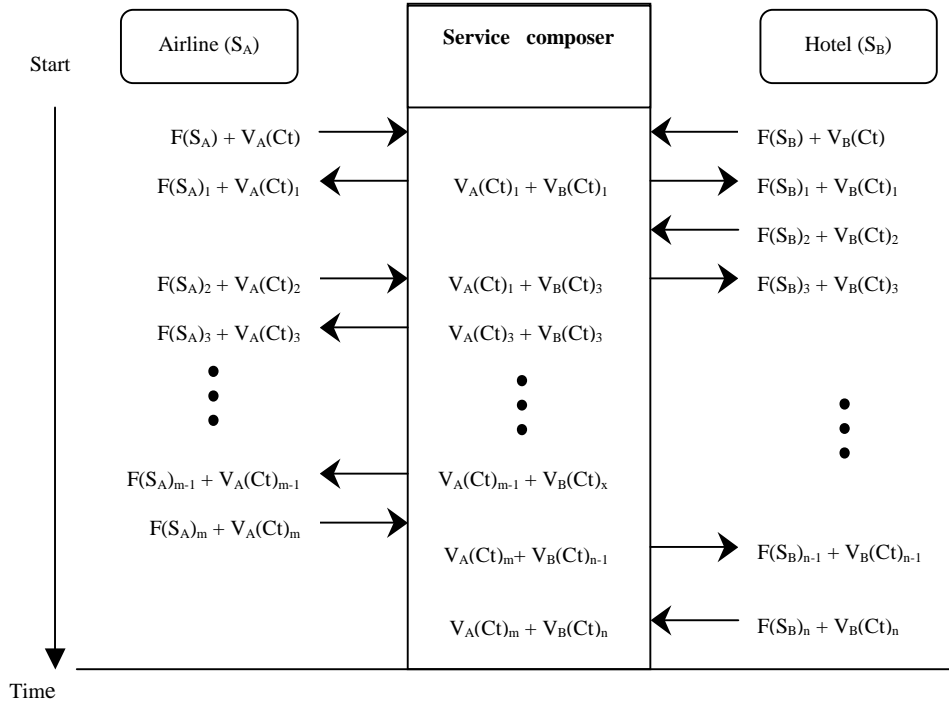


Figure 5. The changing of service features and the control of the value of constraints during the negotiation process.

For instance, service S_A may negotiate by placing the final proposal $F(S_A)_m + V_A(Ct)_m$ at round m . If the service composer satisfies this proposal, it will send a ‘tentative hold’ request to service S_A . However, three possible cases can occur:

- 5.1: Service S_B sent $F(S_B)_n + V_B(Ct)_n$ as a final proposal and $V_A(Ct)_m + V_B(Ct)_n$ meets the conditions in definition 3. In this case, all negotiation processes will terminate with a successful result. Figure 6 shows the transition state diagram for the successful negotiation.

- 5.2: Service S_B sent $F(S_B)_n + V_B(Ct)_n$ as a final proposal but $V_A(Ct)_m + V_B(Ct)_n$ does not meet the conditions in definition 3. In this case the service composer needs to find another candidate service S'_B to negotiate.

- 5.3: Service S_B could not complete negotiations within the limited time constraint. In this case the service composer may find another candidate service S'_B to negotiate, or cancel all services

6 Conclusion

This paper addresses the problems of web service composition transactions. When transactions are complex, can involve many parties, can span multiple organizations, and can potentially last for long periods of time, they are unable to lock any underlying resources exclusively or indefinitely. We present a transaction management model based on the tentative hold and compensation concepts. Our model also supports the negotiation process for service composition.

In future work, we plan to extend the proposed model to more complex situations in which the service composer is subject to failure. The participated composite services are able to complete the composite transaction.

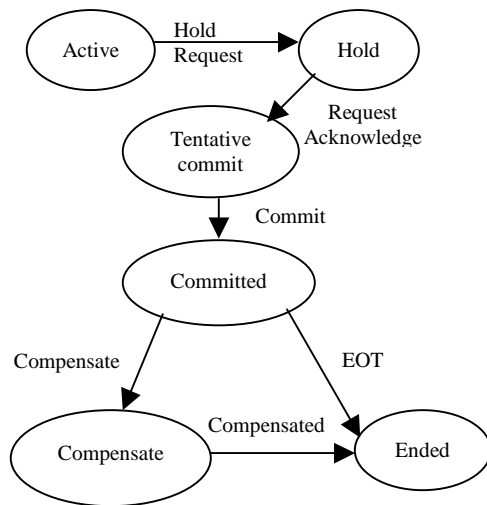


Figure 6. State diagram for the successful Negotiation.

7 References

- BTP: OASIS Business Transaction Protocol (BTP).
<http://www.oasis-open.org/committees/business-transactions/>.
- Cabrera, F., Copeland, G., Cox, B., Freund, T., Klein, J., Storey, T. and Thatte, S. (2001): Web Services Transaction (WS-Transaction), BEA Systems, International Business Machines Corporation, Microsoft Corporation, Inc.,
<http://www.ibm.com/developerworks/library/ws-transpec>.
- Chakraborty, D. and Joshi, A. (2001): Dynamic Service Composition: State-of-the-Art and Research Directions, *Technical Report TR-CS-01-19, Department of Computer Science and Electrical Engineering, University of Maryland, Baltimore County, Baltimore, USA*.
- Chakraborty, D., Perich, F., Joshi, A., Finin, T. and Yesha, Y. (2002): A Reactive Service Composition Architecture for Pervasive Computing Environments, *In 7th Personal Wireless Communications Conference (PWC 2002)*, Singapore, October.
- Dalal, S. and Takacs-Nagy, P. (2001): Proposal for Business Transaction Protocol Version 1.0, BEA Systems, Inc.
- Dayal, U., Hsu, M. and Ladin, R. (1991): A Transaction Model for Long-Running Activities, *In Proceedings of the 17th VLDB Conference*, September.
- Garcia-Molina, H. and Salem, K. (1987): SAGAS, *In Proceedings of ACM SIGMOD Conference on Management of Data*.
- Limthanmaphon, B. and Zhang, Y. (2003): Web Service Composition with Case-Based Reasoning, Database Technologies 2003, *In Proceedings of the 14th Australasian Database Conference (ADC2003)*, K. Dieter-Schewe and X. Zhou Editors, Adelaide, Australia, February.
- Lyon, J., Evans, K. and Klein, J. (1998): Transaction Internet Protocol, version 3.0, RFC2371, <http://www.ietf.org/rfc/rfc2371.txt>, Network Working Group, July.
- Mani, A. and Nagarajan, A. (2002): Understanding quality of service for Web services, <http://www-106.ibm.com/developerworks/library/wsquality.html> [20 October 2002], January.
- Mikalsen, T., Tai, S., and Rouvellou, I. (2002): Transactional Attitudes: Reliable Composition of Autonomous Web Services, Workshop on Dependable Middleware-based Systems (WDMS 2002), *In the International Conference on Dependable Systems and Networks (DSN 2002)*, Washington D.C., June.
- Moss, J.E.B. (1981): Nested Transactions: An Approach to Reliable Distributed Computing, MIT/LCS/TR-260, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, U.S.A.
- Papazoglou, M.P. (2003): Web Services and Business Transaction, *World Wide Web Internet and Web Information Systems*, 6(1), M. Rusinkiewicz, Y. Kambayashi, Y. Zhang (Eds), Kluwer Academic Publishers, March.
- Park, J. and Choi, K-S. (2003), Design of an Efficient Tentative Hold Protocol for Automated Coordination of Multi-Business Transactions, *In Proceedings of the IEEE Conference on E-Commerce*, June.
- Pires, P.F., Benevides, R.F.M., and Mattoso, M. (2002): Building Reliable Web Services Compositions, Web, Web-Services, and Database Systems, NODe 2002, A. Chaudhri, M. Jeckle, E. Rahm, and R. Unland (Eds.), LNCS 2593, Springer.
- Potts, M., Cox, B., and Pope, B., (2002): Business Transaction Protocol Primer, An OASIS Committee Supporting Document, Version 1.0, June.
- Pu, C. (1988): Superdatabases for Composition of Heterogenous Databases *In Proceedings of the 4th International Conference on Data Engineering*.
- Pu, C., Kaiser, G., and Hutchinson, N. (1988): "Split-Transactions for Open-Ended Activities", *In Proceedings of the 14th International Conference on VLDB*.
- Reuter, A. (1989): ConTracts: A Means for Extending Control Beyond Transaction Boundaries, *In Proceedings of the 3rd International Workshop on High Performance Transaction Systems*.
- Roberts, J., and Srinivasan, K. (2001): Tentative Hold Protocol Part 1: White Paper, W3C Note 28 November 2001, <http://www.w3.org/TR/tenthold-1>.
- Roberts, J., Collier, T., Malu, P. and Srinivasan, K. (2001): Tentative Hold Protocol Part 2: Technical Specification, W3C Note 28 November 2001, <http://www.w3.org/TR/tenthold-2>.
- Strandenaes, T. and Karlsen, R. (2002): Transaction Compensation in Web Services, NIK 2002, The Norwegian Computer Science Conference, Norway, Buskerud College.
- Wiekum, G., and Schek, H-J. (1992): Concepts and applications of multilevel transactions and open nested transactions, *In A Elmagamid, editor, Database Transaction Models for Advanced Applications*, Morgan-Kaufmann.
- Zhang, Y., and Jia, X. (1999): Transaction Processing, *In Wiley's Encyclopedia of Electrical and Electronics Engineering*, Vol. 22, Ed. J. Webster, February, pp 298-311.
- Zhang, Y., Kambayashi, Y., Jia, X., Yang, Y., and Sun, C. (1999): On Interactions between Co-existing Traditional and Cooperative Transactions, *International Journal of Cooperative Information Systems*, Vol. 8, No.2.