Implementing a Flexible Compensation Mechanism for Business Processes in Web Service Environment

Zaihan Yang and Chengfei Liu
Faculty of Information and Communication Technologies
Swinburne University of Technology
Melbourne, VIC 3122, Australia
{zyang, cliu}@ict.swin.edu.au

Abstract

Web services have been emerging as a promising technology for business process integrations. Due to their long-duration and loosely coupled properties, Web service based applications require transactional support beyond traditional transactions. Some service standards have been proposed to deal with the transaction aspect of Web service applications. Compensation is a commonly used mechanism in these standards for backward recovery. However, the compensation mechanism usually adopted is too fixed and cannot satisfy the various requirements of different applications. In this paper, we first analyze the compensation protocol of current standards. Then we enrich the protocol by allowing flexible compensation and extend our proposed multiple-compensation mechanism in web service environment. The implementation of the extended compensation mechanism is discussed and the incorporation of the mechanism into current standards is also addressed.

1. Introduction

A web service is a self-contained modular program that can be discovered and invoked across the Internet. It is an autonomous, standards-based component whose public interface is defined and described using XML. Other systems may interact with the web service in a manner prescribed by its definition, using XML based messages conveyed by internet protocols. Web services are the key component of the emerging, loosely coupled, web-based computing architecture [1, 2].

The architecture of the layered standards has been defined that allows technical interoperability of web services. The current web service architecture is typically built upon XML, SOAP, WSDL and UDDI specifications [3, 4]. SOAP defines the basic formatting of a message and the basic delivery options independent of programming language, operating system, or platform. WSDL [5] describes the static interface of a web service. It defines the protocol and the message characteristics of end points. UDDI allows publishing the availability of a web service and its discovery from service requesters using sophisticated searching mechanisms.

Recently the role of web services has been extended from a support of information interaction to a middleware of business process integrations. Nowadays, enterprises are willing to outsource their internal business process as services and make them accessible via the web. In addition, they can dynamically combine individual services to provide new value-added services. To achieve this, transaction support has become a more and more important aspect for web service technology, since it is critical to ensure a correct integration and a reliable execution of the integrated business process.

Several web service standards relevant to transaction support have been proposed. Among them are BPEL4WS [6], WS-Coordination [7], WS-Transaction [8], WSCI [9], and WS-CDL [10]. WS-Transaction can be further divided into two protocols of WS-AtomicTransaction (WS-AT) [11] and WS-BusinessActivity (WS-BA) [12]. Compensation is the basic mechanism adopted in all of these standards for backward recovery.

The mechanism of compensation is originally proposed by Gray in [13], and then widely used in both advanced transaction models (ATMs) [14] and transactional workflows [15] to maintain atomicity when the isolation property has to be relaxed. In ATMs, for a transaction T, its compensating transaction C is a transaction that can semantically eliminate the effects of the transaction T after T has been successfully committed. For a transactional workflow, it is assumed that users can define for each task in a business process one compensating task [16]. When some committed tasks need to be undone, their corresponding compensating tasks will be invoked. Confirmation, a new mechanism proposed in [17], can modify some non-compensatable tasks to make them compensatable. For the business process integrated in a web service environment, tasks of the business process can be compensated as a set of operations that span a single or multiple web services. Web service operations can be compensated if the corresponding business process task needs to be semantically undone.
However, the compensation mechanism used in the current web service standards has some deficiencies. Firstly, the compensation operation is defined at the level of scopes (in BPEL4WS) or context (in WSCI) or choreography (in WS-CDL) which may lead to duplicated definition and extra work when scopes or contexts change. Secondly, for each scope, there exists only one corresponding compensation transaction, which is too fixed and not flexible enough to adjust to different application requirements. For example, when penalty has to be considered for carrying out compensation, different penalty policies will result in different compensation strategies. As a result, a more flexible compensation mechanism is necessary.

In this paper, we extend the multiple-compensation mechanism we proposed [16] into the web service environment. First we define compensation independently from business processes and target at the web service operation level. As such, compensation can be defined just once. Besides, we may define multiple compensation operations for each web service operation. This makes compensation more flexible and can tailor the requirements from different business processes. We take the BPEL4WS and WS-BA as the basic models and explicitly describe an algorithm on how to implement this extended multiple-compensation mechanism in the web service environment. The revision brought to some current web service standards will also be addressed.

The rest of the article is organised as follows. Section 2 describes the existing business processes based on BPEL4WS and transaction support based on WS-BA in web service environments without the multiple-compensation extension. Section 3 extends the WS-BA compensation protocol with multiple-compensation and extends the multiple-compensation mechanism we proposed into the web service environment. Section 4 introduces an algorithm on how to implement the extended multiple-compensation mechanism in the web service environment. Section 5 discusses the extensions and revisions we will make on the current web service standard when adopting the flexible-compensation mechanism. Section 6 concludes the paper.

2. Business Processes and Transaction Support in Web Service Environment

Among all the current web service standards, BPEL4WS has emerged as the most important standard to define and manage business process tasks and business interaction protocols comprising collaborating web services. BPEL4WS is a workflow-like language which defines a model and grammar for describing the behaviour of a business process based on interactions between the process and its partners. At the core of the BPEL4WS process model is the notion of Peer-to-Peer interactions between services described in the form of WSDL. Both the whole business process workflow and each of its composing counterpart tasks are specified in WSDL services. Figure 1 illustrates the relationship between business process tasks and the corresponding web service they invoke.

![Figure 1. Relationship between business processes and web services](image)

As we can see from Figure 1, the whole business process can be modelled as a WSDL service and each task can be implemented by invoking a set of operations which span over a single or multiple web services. For simplicity, we assume that one task is corresponding with one web service in the form of WSDL. WSDL defines an XML grammar for describing network services as collections of communication endpoints capable of exchanging messages. In one WSDL document, several port types are defined, each of which acts as the static interface of this web service. Port type is composed of multiple operations, which are described in the form of input or output of messages. The execution of business process tasks can be turned into invoking of the WSDL operations.

The message flow among business process tasks are modelled by BPEL4WS as some synchronous or asynchronous manipulation on web service operations such as receive/reply, invoke/receive and etc. The control flow of business process can also be specified by BPEL4WS as sequence (for ordered sequence of tasks), switch (for branching), while (for loop), pick (for alternative paths), flow (for concurrent task execution) and link (for expressing synchronisation dependency). The interactions between a service and each of the parties with which it interacts are indicated by the Partner Link element which defines the messages and port types used in the interactions in both directions, along with role names.
The WS-Coordination and WS-Transaction specification complement BPEL4WS in that they provide a web services-based approach to improve the dependability of automated long-running business transactions in an extensible and interoperable way. WS-Coordination provides a framework for coordinating the actions of distributed applications via coordination-context sharing. It consists of three components, namely, the activation service which creates a coordination context with an identifier, coordination type, the registration service which enables an application to register for coordination protocols as participants and the coordination type specific set of coordination protocols. WS-Transaction leverages WS-Coordination by describing the coordination types that are used with the extensible coordination framework outlined in WS-Coordination specifications. There exist two coordination types: atomic transaction (AT) and business activity (BA). WS-AT provides protocol for short-lived atomic units of work, while WS-BA is the standard suggested for processes that are of long-duration. Since we discuss about business processes, we will focus on WS-BA.

WS-BA is established upon the open nested transaction model and adopts the compensation mechanism for backward recovery. Figure 2 shows the state transition diagram that specifies the behaviour of the protocol between the web service coordinator and participant. The coordinator is responsible for the execution or compensation of a business process instance (or parent scope) while the participant is responsible for the execution or compensation of an individual task (or nested scope).

As shown in Figure 2, the coordinator will generate Complete, Fault, Compensate, Closed, Cancelled and Exit signals. The participant will generate Close, Cancel, Compensated, Faulted and Exited signals. The working details are described as follows.

After registration, the state of the participant will become “Active”. If it then completes successfully, it will send a “Completed” signal to the coordinator, which will then set the current state of the participant to “Completed”. When all the participants finish successfully, the coordinator will send a broadcasting “Close” signal to the participants. Upon receiving this signal, the participant will turn its state into “Closing”. Then it can send a “Closed” signal to the coordinator, which will finally set its state to “Ended”. The coordinator may send the “Compensate” signal to a participant which has already completed in case of the failure of some other participants. Upon receipt of the “Compensate” signal, the participant will change its state to “Compensating” and executes the corresponding compensation operations. If the compensation operations completes successfully, the participant will send a “compensated” signal to the coordinator, which will finally set its state to “Ended”. If the compensation operations fail, a “Fault” signal will be sent by the participant to the coordinator and its state will be set into “Faulting”. After receiving the “Fault” signal, the coordinator will change the state of the participant to “Ended”. A participant may choose to exit from the coordination by sending the “Exit” signal. Its current state will firstly become “Exiting” and then ultimately set to “Ended” after receiving the signal of “Exited” sent by the coordinator. For the failure of some other participants, the coordinator can send the signal “Cancel” to those participants with the current state of “Active”. Upon receiving the “Cancel” signal, the participant will change its state to “Cancelling” and send signal “Cancelled” to the coordinator. The coordinator will then turn the state of the participant to “Ended”.

The State Transition Diagram of WS-BA shown in Figure 2 only emphasises that when a participant receives the compensate signal from the coordinator, it will carry out compensation. However, it does
not indicate the specific compensation order when several participants need to be compensated. Furthermore, the compensation mechanism adopted is not flexible enough to satisfy different applications requirements. To overcome these limitations, we extend WS-C/WS-BA by incorporating the multiple-compensation we proposed into web service environment.

3. A Flexible Compensation Mechanism for Web Service Transactions

We extend the compensation model proposed by WS-C/WS-BA by allowing multiple compensations. The extended State Transition Diagram is shown in Figure 3.

The most prominent extension lies in two aspects. Firstly, each web service operation is associated with multiple compensation operations. Secondly, we add a new state of “Selecting” which allows a participant to select from multiple compensation operations based on some system logged information. The original state for a participating task is “Initial”. After it registers with the coordinator via the Registration Service defined in WS-C framework, its current state will be turned into “Active”. Upon receiving the Compensate signal generated by the coordinator, the participant will change its state to “Selecting”. Then it will select from its multiple compensation operations one appropriate operation for execution and change its current state to “Compensating”.

We can model a business process in the web service environment using the BPEL4WS standards, where the whole business process can be regarded as a composite web service with each task corresponding to an operation of a component web service. Different from WS-Transaction and WS-Coordination, we define compensation at the operation level and allow more than one compensation operations for a web service operation.

Definition 1. A web service operation $o$ is defined as $(o_f, o_b, C)$ where,
(1) $o_f$ defines the forward execution part (normal part) of $o$. The set of input and output parameters of $o_f$ is denoted as $Par$. When $o_f$ is invoked, $Par$ will be recorded in a system log.
(2) $o_b$ defines the backward execution part (compensation part) of $o$. When $o_b$ is invoked, the $Par$, which is stored in a system log, will be adopted.
(3) $C$ is a set which consists of a set of compensating operations defined for the operation $o$. When an operation needs to be compensated, its backward execution part $o_b$ will be invoked. Then the $o_b$ will select from the set $C$ one appropriate compensating operation for execution according to some decision criteria.

The business process in a web service environment with the multiple compensation nature is modelled as follows.

Definition 2. A business process (or a composite web service) can be modelled as an acyclic directed graph in the form of $G(N, E, t, n_s)$, where
(1) $N = \{n_1, n_2, \ldots, n_m\}$, $n_i \in N$ represents a task in the business process. A task is implemented by a web service operation defined in Definition 1.
(2) $E$ is a set of directed edges. Each edge $e = (n_1, n_2) \in E$ corresponds to the control dependency between $n_1$ and $n_2$, where $n_1, n_2 \in N$.
(3) For each $n \in N$, $Ind(n)$ and $Outd(n)$ define the number of edges which take $n$ as the terminating node and starting node, respectively.
(4) \( t: N \rightarrow Type \) is a mapping function, where \( Type = \{ \text{normal}, \text{And-Join}, \text{And-Split}, \text{Or-Join}, \text{Or-Split} \} \). It is easy to see that:
- If \( t(n) = \text{"normal"} \) then \( ind(n) = outd(n) = 1 \).
- If \( t(n) = \text{"And-Split"} \) or \( \text{"Or-Split"} \) then \( ind(n) = 1, outd(n) > 1 \).
- If \( t(n) = \text{"And-Join"} \) or \( \text{"Or-Join"} \) then \( ind(n) > 1, outd(n) = 1 \).

(5) \( n_e \) is the starting task of the business process, which satisfies that \( n_e \in N \) and \( Ind(n_e) = 0 \).

In BPEL4WS, the primitive activities within a sequence structured activity are of the normal type. The And-Split and And-Join correspond to the start and end tags of \(<\text{flow}>\) element, respectively. Similarly, the Or-Split and Or-Join correspond to the start and end tags of either \(<\text{switch}>\) or \(<\text{pick}>\) elements.

**Definition 3.** An instance of a business process graph \( G(N,E,t,n_i) \) is defined as an acyclic graph \( \overline{G}(N,E,t,st,et,\pi) \), where

1. \( N \subseteq N \) Each \( \pi \in N \) corresponds to a task instance in the business process instance.
2. \( E \subseteq E \). Each edge \( \overline{e} = (\pi, \pi) \in E \) corresponds to the control dependency between task instances \( \pi \) and \( \pi \), where \( \pi, \pi \in N \).
3. \( t: N \rightarrow Type \) is the same mapping function as that defined in the business process model \( G \).
4. \( st,et: \overline{N} \rightarrow Time \) are functions which map a \( \pi \in N \) to a specific system time, where \( st(\pi) \) indicates the starting time of \( \pi \) and \( et(\pi) \) indicates the terminating time of \( \pi \).
5. \( s: \overline{N} \rightarrow States \) is a function which maps each task instance in set \( \overline{N} \) to a certain kind of states in set \( States \), where \( States = \{ \text{Initial}, \text{Active}, \text{Complete}, \text{Ended}, \text{Selecting}, \text{Compensating}, \text{Faulting}, \text{Cancelling}, \text{Existing}, \text{Closing} \} \).
6. \( \pi \) indicates the starting task instance.
7. \( prec,succe: \overline{N} \rightarrow 2^N \) are functions which define for each task instance \( \pi \in N \) its preceding task instances and succeeding task instances respectively. \( \pi \) is said to be the preceding task instance of \( \pi \) when it exists that \( (\pi, \pi) \in E \). \( \pi \) is said to be the succeeding task instance of \( \pi \) when it exists that \( (\pi, \pi) \in E \).

The set \( States \) in (5) corresponds to the set of states defined in the extended State Transition Diagram shown in Figure 3.

**Definition 4.** The executed part of \( G(\overline{N}, \overline{E}, t, st, et, s, \pi) \) is denoted as \( \overline{G}_e(\overline{N}_e, \overline{E}_e, t, st, et, s, \pi) \), where \( \overline{N}_e, \overline{E}_e \) are subsets of \( \overline{N} \) and \( E \) respectively and for each \( \pi \in \overline{N}_e, s(\pi) \neq \text{"Initial"} \).

4. Implementation of the Mechanism

Upon the extended state transition diagram and definitions given in previous section, we present the algorithm on how to implement the extended multiple-compensation mechanism in the web service environment. The algorithm distinguishes two roles: coordinator and participant. In addition, the algorithm also considers the ordering of the compensation process.

4.1 Coordinator and Participant

A coordinator is created for each instance of a business process and be responsible for the execution of the instance. In this paper, we concentrate our discussion on the compensation process. A coordinator takes care of the execution of all of its participating tasks and monitors their state transitions. \( \overline{G}_e \) defined in Definition 4 keeps the execution status of the business process instance and is kept by the coordinator. A participant takes care of the execution of one particular participating task and does registration when the task starts from “Initial” state to “Active” state. Figure 3 shows the relationship between the coordinator and one of its participants. There are two situations the coordinator needs to deal with. (1) positive: when all participating tasks reach the “Complete” state, the coordinator may signal “Close” to confirm the execution of all the tasks (hence the business process). (2) negative: when one participant signal “Exit” or “Fault”, the coordinator needs to signal “Compensate” to all executed tasks reaching “Complete” state and signal “Cancel” to all executing tasks in “Active” state. When a participant receives a “Compensate” signal, it changes the state to “Selecting”, triggering the multiple-compensation mechanism for selecting one appropriate compensation operation for execution based on the input and output parameters of the executed task and other information such as starting and terminating times.

4.2 Coordination Logic

The compensation process is triggered when the coordinator receives the signal “Exit” or “Fault” from one of its participant. To guarantee the correct compensation semantics and that all participating tasks
rolling back to the “Ended” state, the coordinator needs to compensate in an order that is reverse to the execution of all its executed or executing tasks.

The main principle of the algorithm is to traverse the graph $\overline{G}_k$ twice in opposite directions. One is backward traversing (recovery), which keeps processing and removing nodes from set $NP$ (Nodes-to-be-Processed) as well as repetitively adds new traced preceding tasks into set $NP$ for processing. The other is forward traversing (tracing), which keeps tracing succeeding tasks until some certain tasks are reached.

The algorithm starts from a failed task in graph $\overline{G}_k$ and invokes the backward traversing first. During the process of backward traversing, the preceding tasks except those And-Split tasks of the currently processed task will be put into set $NP$ in order for processing. The order of adding tasks into set $NP$ indicates the corresponding compensation order. The tasks in $NP$, which have not been completed successfully, will be aborted by system. Other tasks in $NP$, which have already successfully committed will be compensated for. When a task is going to be compensated, the backward part of its invoked operation $o_b$ will be invoked. The backward part $o_b$ will then select from the set of compensating tasks one appropriate compensating operation to execute according to that system-logged information of the task.

When the preceding task of the currently processed task is an And-Split task, a forward traversing process will be needed. The forward traversing process will traverse all the succeeding branches of the And-Split task until a certain task of each branch which has no further succeeding task or which has already been in set $NP$ is reached. The whole algorithm will be terminated when the starting task instance in graph $\overline{G}_k$ is reached.

In $\overline{G}_k$, only one preceding and succeeding of Or-Join and Or-Split will be selected. So we can treat them as normal tasks.

### 4.3 Algorithm

We now present the algorithm in terms of two roles: coordinator and participant. The dialogue between the coordinator and a participant is realized in an event-driven manner. The coordinator always keeps the execution graph $\overline{G}_k$ of the business process instance it is responsible and keeps updating $\overline{G}_k$.

#### Algorithm 1: coordinator

1. **Upon receiving** “Exit” or “Fault” from $\overline{\pi}$, \{ 
2. \> $s(\overline{\pi}) =$ “Exiting” or “Faulting”; /* update state */
3. \> send “Exited” or “Failed” to $\overline{\pi}$;
4. \}
5. $NP = \{\overline{\pi}\}$;
6. **for each** $\overline{\pi} \in NP$ \{
7. \> /* Processing Part*/
8. \> if $s(\overline{\pi}) =$ “Active” then send “Cancel” to $\overline{\pi}$;
9. \> if $s(\overline{\pi}) =$ “Complete” then send “Compensate” to $\overline{\pi}$;
10. \> $NP = NP - \{\overline{\pi}\}$;
11. \> if $\overline{\pi} = \pi_n$ then stop; /* Generating Part*/
12. \> if $t(\overline{\pi}) =$ “Normal” or $t(\overline{\pi}) =$ “And-Split” then \{
13. \> $\overline{\pi}_p = \text{getone}(\text{prec}(\overline{\pi}))$; /* get a prec task */
14. \> if $t(\overline{\pi}_p)$ ∉ “And-Split” then $NP = NP \cup \{\overline{\pi}_p\}$;
15. \> else if $\overline{\pi}_p \notin \text{ASMarked}$ then { /* the And-Split node has not been marked*/
16. \> $\text{AJMarked} = \phi$; /* forward tracing */
17. \> $\text{Asucc} = \text{succ}(\overline{\pi}_p) - \{\overline{\pi}_p\}$;
18. \> for each $\overline{\pi} \in \text{Asucc}$ \{
19. \> $\text{Asucc} = \text{Asucc} - \{\overline{\pi}_k\}$;
20. \> if $\overline{\pi}_k \notin NP$ and $\text{suc}(\overline{\pi}_k) = \phi$ then
21. \> $NP = NP \cup \{\overline{\pi}_k\}$
22. \> else if $\text{suc}(\overline{\pi}_k) \neq \phi$ then \{
23. \> $\text{Asucc} = \text{Asucc} \cup (\text{succ}(\overline{\pi}_k) - \text{AJMarked})$;
24. \> if $t(\overline{\pi}_k) =$ “And-Split” then $\text{ASMarked} = \text{ASMarked} \cup \{\overline{\pi}_k\}$
25. \> else if $t(\overline{\pi}_k) =$ “And-Join” then $\text{AJMarked} = \text{AJMarked} \cup \{\overline{\pi}_k\}$
26. \} \}
27. \} \}
28. else { /* the And-Split node has been marked*/
29. \> $\text{Asucc} = \text{suc}(\overline{\pi}_p)$;
30. \> for each $\overline{\pi} \in \text{Asucc}$ if $s(\overline{\pi}) =$ “Ended” then $\text{Asucc} = \text{Asucc} - \{\overline{\pi}\}$;
31. \> if $\text{Asucc} = \phi$ then \{ $NP = NP \cup \{\overline{\pi}_p\}$;
32. \> $\text{ASMarked} = \text{ASMarked} - \{\overline{\pi}_p\}$ \} \}
33. \}
34. else if $t(\overline{\pi}) =$ “And-Join” then $NP = NP \cup \text{prec}(\overline{\pi})$;
35. \}
36. **upon receiving** “Compensated” or “Closed” or “Cancelled” from $\overline{\pi}$, $s(\overline{\pi}) =$ “Ended”;
37. **upon receiving** “Completed” from $\overline{\pi}$, $s(\overline{\pi}) =$ “Complete”;
38. **upon receiving** “Completed” from all \( \pi \) **broadcast**
   “Close” to all \( \pi \);

   The main part of Algorithm lines 1-35 is for compensation. In order to avoid reduplicate traversing, a set \( \text{ASMarked} \) is constructed. The And-Split tasks, which have once been processed, will be added into set \( \text{ASMarked} \). They will not be forward traced again even though they will be reached later during the traversing.

   Lines 15-25 are used for a forward tracing process when an And-Split task is first reached. For those And-Split tasks, all of its succeeding branches except those that have been processed will be traversed until the task of each branch that has already been in set \( \text{NP} \) or has no succeeding task is reached. In the latter situation, the task that has no succeeding tasks will be put into set \( \text{NP} \). To avoid reduplicate traversing, two sets \( \text{ASMarked} \) and \( \text{AJMarked} \) are used to contain those And-Split tasks and And-Join tasks that have once been traversed.

---

### Algorithm 2 Participant

1. **Upon the task cannot be finished** send “Exit” or “Fault” to coordinator;
2. **upon receiving** “compensate” {
3. \( s(\pi) = "\text{Selecting}"; \)
4. \( \text{invoke } \pi.o_{\pi}; \)
5. \( o_{\pi}(\text{par}, s(\pi), e(\pi)) \rightarrow c_{j}; c_{j} \in C; /* \text{select from} \) set \( C \) one compensating operation */
6. \( s(\pi) = "\text{Compensating}"; \)
7. \( \text{execute } c_{j}; \)
8. \( \text{send } "\text{Compensated}" \text{ to coordinator;} \)
9. }
10. **upon receiving** “Cancel” {
11. \( s(\pi) = "\text{Cancelling}"; \)
12. \( \text{rollback}; /* \text{abort the executing task */} \)
13. \( \text{send } "\text{Cancelled}" \text{ to coordinator;} \)
14. }
15. **upon receiving** “Exited” or “Faulted”
   \( s(\pi) = "\text{Ended}"; \)
16. **upon receiving** “Close” { \( s(\pi) = "\text{Closing}"; \)
   \( \text{send } "\text{Closed}" \text{ to coordinator;} \)

---

5. **Impact on Current Web Service Standards**

   One of the baseline specifications in the current web service architecture is WSDL, which is an XML format for describing network services as collections of communication endpoints capable of exchanging messages. BPEL4WS, WSCI, WSCDL, WS-C/WS-T are all established on WSDL and use it to describe the static interface of individual web service.

   Instead of defining compensation that is dependent on business processes in BPEL4WS, we choose the way of defining it independent from business processes. We simply add a compensation part in defining a web service operation.

   We extend the original WSDL as follows.

   ```xml
   <wsdl: portType name="">
     <wsdl: documentation>…….</wsdl:documentation >
     ........
     <wsdl: operation name="">
       <wsdl: normalPart >
         <wsdl: input message=""/>
         <wsdl: output message=""/>
       </wsdl: normalPart >
       ........
       <wsdl: compensationPart >
         <wsdl: input message=""/>
         <wsdl: output message=""/>
       </wsdl: compensationPart >
       ........
     </wsdl: operation >
     ........
   </wsdl: portType>
   ```

   Three new elements have been introduced in WSDL. <normalPart>, <compensationPart> and <compensatingOperation>. The normalPart element is used to specify the original operation defined in WSDL. The compensationPart is introduced to specify the multiple-compensation mechanism defined on the operation. The multiple compensatingOperation subelements of the compensationPart are used to specify more than one compensating operations. All these elements are further defined in the form of input and output of messages.

   Once compensation is defined in WSDL, we do not need to define it at business process level in BPEL4WS. In case something negative happens, the algorithm introduced in the previous section will be used to automatically compensate all invoked operations of those completed tasks.

6. **Conclusion**

   The recent evolution of Internet, driven by the Web Service technology, has extended the role of Web Service from a support of information interaction to a
middleware of business process integrations. In order to provide a reliable and consistent integration environment, transaction support of web service has become one of the focuses of many web service researchers. Some web service standards have been proposed to support compensation based transaction management. However, the current compensation mechanism adopted is far from satisfactory. Firstly, the compensation are normally defined upon scope level and are dependent on the business process workflow, thus can easily lead to repetitive definition work. Secondly, the current standards have not considered the multiple compensation requirements from real applications. Only one compensation task is defined for a scope. Thirdly, an explicit description on the compensation order has not been given in current web service standards.

To overcome such limitations mentioned above, we extended a multiple compensation mechanism proposed in our previous work for business process specification into the web service environment. We targeted at the operation level and defined for each web service operation multiple compensation operations. We took the BPEL4WS and WS-BA as the basic model and made extensions on WS-BA by allowing the participant to select one appropriate compensation operation for execution. We introduced an algorithm on how to implement this multiple compensation mechanism in the web service environment via the event-driven communication between web service coordinator and participants. The compensation order (coordination logic) has also been explicitly addressed in the algorithm. Finally, we analysed the impact on WSDL after introducing the multiple compensation mechanism.

7. Acknowledgment

This work is supported by the Australian Research Council Discovery Project under the grant number DP0557572.

8. References