On the Development of a Multiple-Compensation Mechanism for Business Transactions

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. Compensation is a widely used concept for maintaining atomicity in both the advanced transaction models and transactional workflow systems. Some Web service protocols also adopt the compensation mechanism for failure recovery when providing transaction management. However, the compensation mechanisms used in these models or protocols are too fixed and cannot satisfy the various requirements of different applications. In this paper, a multiple-compensation mechanism is proposed and defined explicitly in a business process model. An algorithm on how to implement this multiple-compensation mechanism for backward recovery is designed and its computation complexity is analysed.

1 Introduction

These years have seen the widespread use of transaction management in non-traditional applications. The transactions in these applications are different from traditional transactions [1,2] for their long-time running and for that they may access data held in heterogeneous, autonomous and distributed systems. The ACID properties will be too strict for them to follow. To overcome the limitations of traditional transactions, some advanced transaction models (ATMs) [3] have been proposed, such as Sagas [4], closed/open nested transactions [5, 6], multi-level transactions [6], flexible transactions [7] and Contracts [8].

The mechanism of compensation is originally proposed by Gray in [9], and then widely used in ATMs to maintain atomicity when the isolation property has been relaxed [9, 10]. 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 example, for a DEPOSIT transaction, its compensating transaction can be a WITHDRAW. We take the Sagas model as an example to clarify how the compensation mechanism is used. In Sagas, the long transaction is divided into several short subtransactions each of which strictly follows the ACID properties. The isolation for the global transaction is relaxed, since subtransactions can release the resources they hold and publicise their effect to other subtransactions before the global transaction commits. For each subtransaction (except for the very last one),

there exists a corresponding compensating transaction. When a subtransaction fails, it will firstly be rollbacked by a transaction manager and all its preceding subtransactions will be compensated by executing their corresponding compensating transactions in a reverse order.

A compensating transaction has some special characteristics besides the fundamental properties of a transaction. First of all, a compensating transaction eliminates a transaction's effect in a semantic manner, rather than by physically restoring to a prior state. Secondly, a compensating transaction is retriable, namely, once the compensating transaction is invoked to execute, it will ultimately commit successfully. Thirdly, a compensating transaction is always regarded as being associated with a compensated-for transaction. In most situations, it is the programmer's responsibility to pre-define a compensating transaction.

Compensation mechanism is not only widely used in ATMs but also adopted by transactional workflows to maintain reliability and consistency of business processes. It is assumed that users can define for each task in a business process one compensating task [11, 12]. When some committed tasks which are called compensated-for tasks need to be undone, their corresponding compensating tasks will be invoked.

The loosely coupled property of Web services offers a good environment for business process collaborations. Some existing Web service protocols, such as WSCI [13], BPEL4WS [14] and WS-CDL [15] also provide some transaction management by supporting the open nested transaction model and compensation mechanism.

Currently, each task can only have one compensating task. This compensation mechanism 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 polices will result in different compensation strategies. As a result, a multiple-compensation mechanism is necessary. This paper proposes a concept of multiple-compensation and describes how to incorporate it in workflow systems. The rest of the article is organised as follows. Section 2 gives a motivating example to clarify the importance of multiple-compensation. Section 3 defines a business process model with the multiple-compensation feature. Section 4 introduces an algorithm on how to implement the multiple-compensation mechanism and analyse its complexity. Section 5 discusses the related work on compensation. Section 6 concludes the paper and indicates the future work.

2 Motivating Example for Multiple-Compensation

Consider a travel reservation process shown in Figure 1 as an example. The whole business process has ten tasks. Travellers will send their trip requests to a travel agent (SR). After receiving the request and sending back acknowledgment (SA), the travel agent will invoke two concurrent activities at the same time: to reserve proper tickets for the traveller via the airline company (BAT) and to book a hotel for the traveller to reside in the destination place (BH). Whether to rent a car in the visiting place is an optional task determined upon the traveller's requirements (RC). During the booking process, travellers should provide their credit card information for identity validation. After all the necessary reservations have been completed, the travel agent will send an

itinerary describing the reservation information and an invoice to the traveller (SBS). The traveller can send acknowledgment to confirm his or her bookings (ACK). Before the airplane departure, the traveller can still choose to cancel the booking (TC) or confirm the booking by paying the money (TP). After the traveller finishes purchasing, the travel agent will send airplane ticket and confirmation letter for hotel booking and for car rental to the traveller. If the traveller cancels the booking or does not complete purchasing after departure, a penalty will apply.

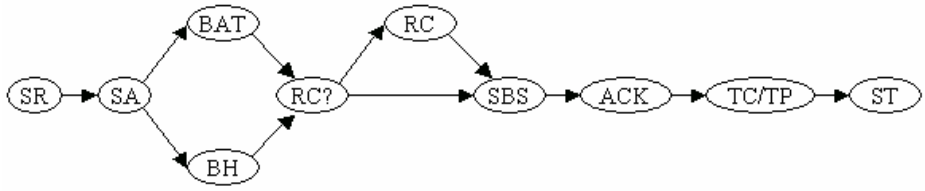

Fig.1. A travel reservation business process example

There exists compensation dependency among tasks. For example, since BAT and BH are concurrent tasks, and only when both of them successfully complete can the succeeding task be executed, consequently, either BAT or BH fails, the other committed task should be compensated.

Consider the situation for the traveller to cancel the booking after having sent out the acknowledgment information. Some corresponding compensation tasks should be carried out due to the cancelling behaviour. Associated with these cancellations, the companies such as the airline company will normally take actions based on some penalty policies for the sake of their own interests. The following table illustrated the penalty policies taken by an airline company.

Table 1. An example of the penalty policy of an airline company

Penalty / Time / Users	2 weeks	5 days	2 days	0 days
VIP Users	None	None	None	None
Members	None	10%	20%	50%
Non-Members	None	20%	50%	100%

Time column indicates when the traveller cancels his booking, 2 weeks before departure, 5 days, 2 days or right before departure (0 days); User column indicates the different status of the users and correspondingly they have different privileges. Penalty column indicates the different charges the company will ask for due to the time to cancel and the user status.

A penalty policy is associated with a compensation task and can be regarded as part of the compensation task. Different penalty policies will be adopted in different cases, leading to different compensation tasks. Our multi-compensation mechanism is motivated to deal with this situation.

3 Business Processes with Multiple-Compensation Mechanism

From the motivating example described in the previous section, we can see that the multiple compensations are common phenomena in real applications. Consequently, a corresponding multiple-compensation mechanism should be considered and reflected in the business process models. In this section, we introduce the multiple-compensation mechanism in a business process environment which associates for each task several compensating tasks. We give formal definitions of a business process model with a multiple-compensation feature in the following.

Definition 1. A business process can be modelled as an acyclic directed graph in the form of $G(N, E, t, n_S)$, where

(1) N is a set of nodes. Each node corresponds to a task in the business process. Namely, $N = \{n_1, n_2, ..., n_m\}$, $n_i \in N (1 \le i \le m)$ represents a task.

(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={normal, And-Join, And-Split, Or-join, Or-Split}*. It is easy to see that:
If *t(n) = "normal"* then *ind(n) = outd(n) = 1*.
If *t(n) = "And-Split"* or *"Or-Split"* then *ind(n) = 1, outd(n) > 1*.
If *t(n) = "And-Join"* or *"Or-Join"* then *ind(n) > 1, outd(n) = 1*.

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

Tasks are the main components of a business process. A task can be modelled as a combination of a normal part (of operations), which is used for forward execution and a compensation part (of operations), which is used for backward recovery. In order to introduce the mechanism of multiple-compensation, we define for the compensation part of each task not only one, but a set of compensating tasks. We can model a task as follows.

Definition 2. A task n is defined as (tf, tb, C) where,

(1) *tf* defines the forward execution part (normal part) of *n*. The set of input and output parameters of *tf* is denoted as *Par*. When *tf* is invoked, *Par* will be recorded in a system log.

(2) *tb* defines the backward execution part (compensation part) of *n*. When *tb* 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 tasks defined for the task *n*. When a task needs to be compensated, its backward execution part *tb* will be invoked. Then the *tb* will select from the set *C* one appropriate compensating task for execution according to some decision criteria.

More details on the process of selecting will be explained in Section 4.

Definition 3. An instance of a business process graph $G(N, E, t, n_s)$ is defined as an acyclic graph $\overline{G}(\overline{N}, \overline{E}, t, st, et, s, \overline{n}_s)$, where

(1) $\overline{N} \subseteq N$ Each $\overline{n}_i \in \overline{N}$ corresponds to a task instance in the business process instance.

(2) $\overline{E} \subseteq E$. Each edge $\overline{e} = (\overline{n}_1, \overline{n}_2) \in \overline{E}$ corresponds to the control dependency between task instances $\overline{n}_1$ and $\overline{n}_2$, where $\overline{n}_1, \overline{n}_2 \in \overline{N}$.

(3) $t : \overline{N} \to Type$ is the same mapping function as that defined in the business process model G.

(4) $st, et : \overline{N} \to Time$ are functions which map a $\overline{n}_i \in \overline{N}$ to a specific system time, where $st(\overline{n}_i)$ indicates the starting time of $\overline{n}_i$ and $et(\overline{n}_i)$ indicates the terminating time of $\overline{n}_i$.

(5) $s : \overline{N} \to States$ is a function which maps each task instance in set $\overline{N}$ to a certain kind of states in set $States$, where $States = \{initial, active, complete, ended, selecting, compensating, faulting\}$.

(6) $\overline{n}_s$ indicates the starting task instance.

(7) $prec, succ : \overline{N} \to 2^{\overline{N}}$ are functions which define for each task instance $\overline{n}_i \in \overline{N}$ its preceding task instances and succeeding task instances respectively. $\overline{n}_j$ is said to be the preceding task instance of $\overline{n}_i$ when it exists that $(\overline{n}_j, \overline{n}_i) \in \overline{E}$. $\overline{n}_j$ is said to be the succeeding task instance of $\overline{n}_i$ when it exists that $(\overline{n}_i, \overline{n}_j) \in \overline{E}$.

Definition 4. The executed part of $\overline{G}(\overline{N}, \overline{E}, t, st, et, s, \overline{n}_s)$ is denoted as $\overline{G}_E(\overline{N}_E, \overline{E}_E, t, st, et, s, \overline{n}_s)$, where $\overline{N}_E$, $\overline{E}_E$ are subsets of $\overline{N}$ and $\overline{E}$ respectively and for each $\overline{n}_i \in \overline{N}_E$, $s(\overline{n}_i) \neq "initial"$.

4 Implementing Multiple-Compensation

Upon the definitions given in Section 3, we present an algorithm on how to implement the multiple-compensation mechanism in this section. Before the presentation of the algorithm, the main ideas of it will be firstly introduced. The analysis for the computational complexity of the algorithm will be given in the end.

4.1 Algorithm Introduction

The algorithm describes what should be done with the multiple-compensation mechanism to maintain atomicity and consistency of the whole business process in

the presence of tasks' failures. The algorithm is invoked by the input of the executed part of a business process instance $\overline{G}_E$ with one or more failed task instances. A system log will play an important role in the algorithm. For each executed task instance $\overline{n} \in \overline{G}_E$, the input/output parameters *Par* of $\overline{n}.tf$, the starting time $st(\overline{n})$, the terminating time $et(\overline{n})$ and the current state $s(\overline{n})$ will all be kept in a system log.

Due to the compensation dependencies among tasks, the abortion or compensation of some tasks will lead to the abortion or compensation of other tasks. For example, when a "*normal*" task is aborted or compensated, its only one preceding task should be compensated. When an "*And-Join*" task is aborted or compensated, all of its multiple preceding tasks should be compensated. When a task that is one of the succeeding tasks of an "*And-Split*" task is aborted or compensated, not only the "*And-Split*" task itself but all the tasks on its succeeding branches should also been compensated for. The abortion or compensation of tasks should be executed in a reverse order with the business process control flow.

The main principle of the algorithm is to traverse the graph $\overline{G}_E$ 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 $\overline{G}_E$ 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, its backward part *tb* will be invoked. The backward part *tb* will then select from the set of compensating tasks one appropriate compensating task to execute according to those 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 $\overline{G}_E$ is reached.

Please note that we only consider the execution part of the business process instance. So for those *Or-Join* and *Or-Split* tasks, their proceding tasks and succeeding tasks will be specific. We can treat them as normal tasks.

4.2 Algorithm Description

We now describe the algorithm for implementing the multiple-compensation mechanism in a more formal way as follows.

Algorithm 1: backward-recovery

Input

The executed part of a business process instance $\bar{G}_E(\bar{N},\bar{E},t,st,et,s,\bar{n}_s)$, where $\exists \bar{n}_i(\bar{n}_i \in \bar{N} \wedge s(\bar{n}_i) =$" *faulting*"$)$.

Output

The updated executed part of a business process instance $\bar{G}_E(\bar{N},\bar{E},t,st,et,s,\bar{n}_s)$, where $\forall \bar{n}_i(\bar{n}_i \in \bar{N} \rightarrow s(\bar{n}_i) =$"*ended*"$)$.

Steps:

1. **for *each*** $\bar{n}_i \in \bar{N}$, **if** $s(\bar{n}_i) =$" *faulting*" **then** {*NP*={ $\bar{n}_i$ }; Skip} /* put one faulting task in *NP* */
2. *ASMarded* = ϕ /* used for marking tasks of the type "*And-Split*" */
3. **for *each*** $\bar{n}_i \in NP$ **{**

/* Processing Part*/

4. **if** $s(\bar{n}_i) =$"*active*" **then** $s(\bar{n}_i) =$"*ended*";
5. **if** $s(\bar{n}_i) =$" *faulting*" **then** $s(\bar{n}_i) =$"*ended*";
6. **if** $s(\bar{n}_i) =$"*complete*" **then** { $s(\bar{n}_i) =$"*selecting*"; *multiple-compensate*($\bar{n}_i$);} /* invoke algorithm 3 of *multiple-compensate**/
7. $NP = NP - \{\bar{n}_i\}$;
8. **if** $\bar{n}_i = \bar{n}_s$ **then return** updated $\bar{G}_E$.

/* Generating Part */

9. **if** $t(\bar{n}_i) =$"*normal*" or $t(\bar{n}_i) =$" $And - Split$" **then {**
10. $\bar{n}_p = getone(prec(\bar{n}_i))$; /* getone(s) take one element from set s */
11. **if** $t(\bar{n}_p) \neq$" $And - Split$" **then** $NP = NP \bigcup \{\bar{n}_p\}$;
12. **else if** $\bar{n}_p \notin ASMarked$ **then {** */* the And-Split node has not been marked*/*
13. *forward-tracing* ($\bar{G}_E, NP, ASMarked, succ(\bar{n}_p) - \{\bar{n}_i\}$); /* invoke algorithm 2 of *forwardtracing**/
14. $ASMarked = ASMarked \bigcup \{\bar{n}_p\}$;
15. **}**
16. **else {** */* the And-Split node has been marked*/*
17. $Asucc = succ(\bar{n}_p)$;
18. **for each** $\bar{n}_j \in Asucc$ **if** $s(\bar{n}_j) =$"*ended*" **then** $Asucc = Asucc - \{\bar{n}_j\}$;
19. **if** $Asucc = \phi$ **then** { $NP = NP \bigcup \{\bar{n}_p\}$; $ASMarked = ASMarked - \{\bar{n}_p\}$;}
20. **}**
21. **}**
22. **else if** $t(\bar{n}_i) =$" $And - Join$" **then** $NP = NP \bigcup prec(\bar{n}_i)$;
23. **}**

Algorithm 1 describes the backward traversing process. It takes the executed part of a business process instance graph $\bar{G}_E$ as an input and starts from an arbitrary faulting task in the graph. After the execution of the algorithm, all the current states of tasks in $\bar{G}_E$ will be set into "ended". The main body of the algorithm consists of two

parts, processing part and generating part. During the processing part, tasks in set *NP* will be processed differently. For those tasks with current states of "active" or faulting", they will be undone by the transaction manager, while if their states are "complete", they will be compensated for. Algorithm 3 will be invoked to compensate these compensated-for tasks. During the generating part, the preceding tasks of the currently processed task will be traced. For a *normal* task or *And-Split* task, its preceding task that is not an *And-Split* task will be added into set *NP*. For an *And-Join* task, all its preceding tasks will be added into set *NP*. The process happens repetitively until at last the starting task is reached. When an *And-Split* task is first reached, a forward tracing process is associated, which will be described explicitly in algorithm 2. In order to avoid reduplicate traversing, a set *ASMarked* is constructed. The *And-Split* tasks, which have once been processed, will be added into set *AS-Marked.* They will not be forward traced again even though they will be reached later during the traversing.

Algorithm 2 forward-tracing
Input: $\overline{G}_E, NP, ASMarked, Asucc$
Output: NP
Steps:
1. $AJMarked = \phi$
2. **for each** $\overline{n}_i \in Asucc$ **{**
3. $Asucc = Asucc - \{\overline{n}_i\}$;
4. **if** $\overline{n}_i \notin NP$ and $succ(\overline{n}_i) = \phi$ **then** $NP = NP \bigcup \{\overline{n}_i\}$
5. **else if** $succ(\overline{n}_i) \neq \phi$ **then {**
6. $Asucc = Asucc \bigcup (succ(\overline{n}_i) - AJMarked)$;
7. **if** $t(\overline{n}_i) = "And - Split"$ **then** $ASMarked = ASMarked \bigcup \{\overline{n}_i\}$
8. **else if** $t(\overline{n}_i) = "And - Join"$ **then** $AJMarked = AJMarked \bigcup \{\overline{n}_i\}$
9. **}**
10. **}**
11. **return** NP.

Algorithm 2 describes a forward tracing process invoked 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 *NP* or has no succeeding task is reached. In the latter situation, the task that has no succeeding tasks will be put into set *NP*. To avoid reduplicate traversing, two sets *ASMarked* and *AJMarked* are used to contain those *And-Split* tasks and *And-Join* tasks that have once been traversed.

Algorithm 3 multiple-compensate
Input: $\overline{n}_i$
Steps:
1. **invoke** $\overline{n}_i.tb$;

2. $tb(par, st(\bar{n}_i), et(\bar{n}_i)) \rightarrow c_j : c_j \in C$; /* select from set *C* one appreciate compensating task based on some system-logged information*/
3. $s(\bar{n}_i) = "compensating"$;
4. **execute** c_j ;
5. $s(\bar{n}_i) = "ended"$;
6. **return.**

Algorithm 3 describes the multiple-compensation process. When a task in set *NP* is going to be compensated, its *tb* part will be invoked. Then it will choose from the set of its compensating tasks one appropriate task for executing.

4.3 Computational Complexity Analysis

Algorithms 1, 2 and 3 describe the whole process of backward recovery using the multiple-compensation mechanism. The main principle is to traverse the graph $\bar{G}_E$ for two times, one for backward traversing, and the other for forward traversing.

For algorithm 1, we can see that it traverses backward through edges in the graph $\bar{G}_E$ from a faulting node to the starting node and repetitively adds preceding nodes into set *NP*. Set *NP* grows dynamically during the process of traversing. Consequently, the complexity of algorithm 1 should be equal to *O (|E|).*

For algorithm 2, it describes a forward traversing process from any *And-Split* node in the graph to the node of each of its branch paths that is in set *NP* or has no succeeding nodes. New found succeeding nodes during traversing are added into set *Asucc* thus makes it grow gradually. Its complexity should also be equal to *O (|E|).* However, extra cost comes from step 6, which contains two set computation between $succ(n_i)$ (through traced edges) and *Asucc* and *AJMarked*, respectively. We consider the worst situation when *Asucc* and *AJMarked* are proportional to |*N*|, so the complexity for $Asucc = Asucc \bigcup (succ(\bar{n}_i) - AJMarked)$ will be equal to $O(|E| \log |N|)$ (we may use indices for both *Asucc* and *AJMarked*). As a result, the complexity of algorithm 2 should be $O(|E| \log |N|)$.

For algorithm 3, it will be invoked for all nodes that have been completed successfully. The complexity for selecting one appropriate compensating task among several compensating tasks would be a constant. So, the complexity for algorithm 3 would be *O (|N|),* which is less than *O (|E|).*

We can conclude that the total complexity for algorithms 1, 2 and 3 is $O(|E| \log |N|)$.

5 Related Work

Compensation mechanism is firstly proposed in ATMs. It is then widely adopted by transactional workflows and Web service transaction protocols to maintain atomicity when isolation property is relaxed.

For transactional workflow systems, the notion of compensation is of great importance, since most workflow instances tend to be long running and the processing entities of some tasks do not support transaction management (such as file systems or legacy systems). The backward recovery based on compensation is well supported in some workflow systems, the most typical of which are the FlowMark workflow systems and the Virtual Transaction Model.

In FlowMark [16] workflow systems, the notion of sphere of joint compensation, which is proposed by Frank Leymann [17] for providing partial backward recovery, is well supported. A sphere is a collection of tasks in a workflow. It should be satisfied that either all the tasks in the sphere successfully complete or all of them should be compensated. Each sphere and each task enclosed in the sphere is defined to be associated with a compensating task. The sphere can be aborted by compensating its composed tasks individually or by invoking the compensation task for the sphere as a whole. Spheres can overlap and be nested. If a task fails, the sphere that immediately encloses it is compensated. Optionally, other spheres that enclose this sphere can be compensated and this can go on recursively.

The Virtual transaction model [18] specifies Virtual Transaction (VT) regions on top of a workflow graph. Upon a failure during the execution of a task enclosed in a VT region, all tasks in the region are compensated in the reverse order of their forward execution, until a compensation end point is reached.

Confirmation is a new mechanism proposed in [19]. It is able to modify some non-compensatable tasks to make them compensatable. While compensation is to semantically eliminate the effects of some completed tasks, confirmation is to semantically commit them. With confirmation mechanism, a task in a business process will not only be associated with a compensating task but also a confirmation task. Once a workflow process instance is executed successfully, the confirmation tasks of all the executed tasks will be executed automatically.

The technology of Web service is developing rapidly. It offers a good environment for business process execution since the Web service components are loosely coupled with each other. Some Web service protocols include transactional support mechanism. For example, the WSCI, WSBPEL and WS-CDL all support open nested transaction model and compensation mechanism. The Web service business activity transaction protocol (WS-BA) [20] is also compensation-based.

Compared with our multiple-compensation mechanism, those compensation mechanisms proposed in ATMs, transactional workflows are not flexible enough. They associate for each task only one compensating task. The compensation mechanism adopted in some Web service protocols is targeted at a scope (or context) level. Scopes and contexts can be nested, which will lead to redundant definition of compensation tasks and cannot be executed automatically. Our multiple-compensation mechanism defines for each task several compensating tasks, thus can satisfy various

application demands. The compensating task can be invoked and executed automatically once its corresponding task needs to be compensated for.

6 Conclusion

Compensation is an important mechanism for backward recovery in long running business processes. Its main principle is to semantically eliminate the effects of some successfully committed tasks in the business process. System developers or users can define for each task in the business process a corresponding compensating task. When a certain task needs to be compensated, its compensating task will be invoked.

In the previous studies, only one compensating task is defined to be associated with a task, which cannot satisfy the different requirements in real applications when some other conditions should be considered, such as penalty, time limits, different user privilege, etc. In this paper, we took into account this problem and proposed a new mechanism of multiple-compensation, which associates for each task several compensating tasks. When a task should be undone, one appropriate compensating task will be selected to invoke under some pre-fixed conditions.

We incorporated the multiple compensation mechanism into a business process model by giving some formal definitions. We then introduced and described in detail an algorithm on how to decide which tasks should be compensated, in which order they should be compensated and which one specific compensating task should be selected to compensate them. The algorithm is efficient, which basically traverses the executed part of a business process graph twice. In most cases, the complexity of the algorithm is $O(|E|)$, with the worst case to be $O(|E| \log |N|)$.

For future work, we would like to take into account the concept of sphere of the joint compensation to see how the multiple compensation mechanism can be applied to it. We also would like to incorporate the mechanism of multiple-compensation into a Web service environment to see what benefits it will bring to improve the existing Web service protocols on Web service transactions.

Acknowledgement

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

References

1. J. Gray and A.Reuter. Transaction Processing: Concepts and Techniques, Morgan Kaufmann,1993.
2. N.Lynch, M.Merritt, W.Weihl and A. Fekete. Atomic Transactions. Morgan Kaufmann, 1993.

3. A. Elmagarmid (Ed.). Database Transaction Models for Advanced Applications, Morgan Kaufmann, 1992.
4. H. Garcia-Molina, K. Salem. Sagas. In the Proceedings of the ACM Conference on Management of Data, 1987, pp.249-259.
5. J.Moss. Nested Transactions and Reliable Distributed Computing. In Proceeding of the 2nd Symposium on Reliability in Distributed Software and Database Systems, 1982, pp. 33-39, Pittsburgh, PA. IEEE CS Press.
6. G.Weikum and H. Schek. Concepts and applications of multiple transactions and open-nested transactions. A.Elmagarmid(Ed.), Morgan Kaufmann, chapter 13, 1992.
7. A. Zhang, M. Nodine, B. Bhargava and Bukhres,O. Ensuring Rlaxed Atomicity for Flexible Transactions in Multidatabase Systems. In Proceedings of 1994 SIGMOD International Conference on Management of Data, 1994, pp. 67-78.
8. H. Wachter and A. Reuter. "The Contract Model", Database Transaction Models for Advanced Applications, A.Elmagarmid (Ed.) Morgan Kaufmann, San Francisco, CA, 1992.
9. J. Gray. The transaction concept: Virtues and Limitations. In Proceeding of the International Conference on Very Large Data Bases, Cannes, France, 1981, pp. 144-154.
10. H.F. Korth, E. Levy and A. Silberschatz. A formal approach to recovery by compensating transactions. In the Proceedings of the 16th VLDB Conference, 1990, pp. 139-146.
11. B. Kiepuszewski, R. Muhlberger and M. Orlowska. Flowback: Providing backward recovery for workflow systems. In the Proceedings of the ACM SIGMOD International Conference on Management of Data, 1998, pp. 555-557.
12. D.Kuo, M. Lawley, C. Liu and M. Orlowska. A model for transactional workflows. R. Topor (Ed.). In the Seventh Australasian Databases Conference Proceedings, vol. 18, Melbourne, Australia, 1996, Australian Computer Science Communications, pp. 139-146.
13. A.Arkin, et al. Web Service Choreography Interface (WSCI) 1.0, August 2002, http://www.w3.org/TR/wsci/.
14. T.Andrews, et al. Business Process Execution Language for Web Services (BPEL4WS) 1.1, May 2003, http://www.ibm.com/developerworks/library/ws-bpel.
15. N.Kavantzas. et al. Web Services Choreography Description Language (WS-CDL) 1.0. 2004. http://www.w3.org/TR/2004/WD-ws-cdl-10-20040427.
16. F.Leymann and D.Roller. Business process management with FlowMark. In the Proceedings of IEEE CompCon (San Francisco, CA, 1994) (Los Alamitos), CA: IEEE Computer Society Press), pp 230-234.
17. F.Leymann. Supporting business transactions via partial backward recovery in workflow management systems. In the Proceedings of BTW'95, 1995, pp. 51-70.
18. V.Krishnamoorthy and M.Shan. Virtual Transaction Model to support Workflow Applications. SAC (2), 2000, pp. 876-881
19. C. Liu, X. Lin, M. E. Orlowska and X. Zhou. Confirmation: increasing resource availability for transactional workflows. Inf. Sci. 153, 2003, pp. 37-53.
20. L. F. Cabrera et al. Web Services Business Activity Framework (WS-BusinessActivity). 2005. http://ftpna2.bea.com/pub/downloads/webservices/WS-BusinessActivity.pdf.