A Comprehensive Failure Recovery Algorithm for LRTs Based on Paired Net

Xiaoyong Mei1,2, Yiyan Fan1, Changqin Huang3, Xiaolin Zheng4

1School of Computer Science and Technology, Hunan University of Arts and Science, Changde 415000, China
2School of Information Science and Technology, Sun Yat-sen University, Guangzhou 510275, China
3College of Educational Information Technology, South China Normal University, Guangzhou 510631
4College of Computer Science and Technology, Zhejiang University, Hangzhou 310027

Email: Email: cdmxy@126.com, yyfan@gmail.com, cqhuang@zju.edu.cn, xlzhen@zju.edu.cn

Abstract—Failure recovery optimization is an important way for enhancing efficiency of Long Running Transaction (LRT) processing. In this paper, to solve the efficiency problem of LRT failure recovery, a Comprehensive Recovery Model for LRTs (LCRM) is constructed, which divides LRTs into a series of sub-transactions in different levels, and supports versatile transaction properties of LRT. Based on LCRM, a Comprehensive Failure Recovery algorithm for LRTs (LCFR) is proposed. This algorithm uses methods of forward recovery, backward recovery and alternative recovery. It supports auto-recovery of failures during the execution of LRTs. LCFR guarantees LRT’s semantic atomicity property and durability property. By restricting the recovery scope in lower level of complex LRTs, LCFR limits the quantity of sub-transactions to be recovered. Thus, it reduces unnecessary loss of time and enhances the efficiency of failure recovery. Experiment results show that LCFR can reduce the time required for failure recovery and decrease the failure rate of LRT processing.

Index Terms—long running transaction modeling, scope-based recovery, hierarchical recovery strategy, failure recovery algorithm

I. INTRODUCTION

Since the long-lived nature of composition Web transaction (it may last for several hours, several months or longer), it brings difficulty to transaction handling. Traditional transaction mechanism is no longer suitable for LRTs to deal with the coordination between several loose coupled Web services and long holding of resources, pure roll back mechanism is not suitable to all situations to ensure atomic semantics of LRTs, it is difficult or even impossible to eliminate the result of execution. Therefore, in a loosely coupled LRTs environment, it is inevitable to take more relax transaction mechanism, which is called relaxed-ACID.

Nowadays the transaction handling strategy of LRTs usually uses simple compensation mechanism, which has the following shortages: (i) the transaction handling strategies are provided by most protocols or specifications (WS4BPEL, WSCI, WS-CDL etc.) which execute compensation tasks to eliminate the effect of failure while ignore transactional properties of composites. However, for uncompensable and nonretriable tasks, this method is infeasible. (ii) compensation operations are usually defined at the level of scope (WS4BPEL), context (WSCI) or choreography (WS-CDL) which may lead to duplicated definition and extra work when scopes or contexts change. (iii) for each scope, there exists only one corresponding compensation transaction, which is too fixed and not flexible enough to adjust for different application requirements. Actually, users want to select appropriate recovery strategies according to different requirements of certain failed task. Our research focuses on the dynamic construction of several kinds of failure recovery strategies, in order to specify failure constraint rules in recovery handler and calculate recovery scope according to Terminate Dependency Point (TDP) to reduce unnecessary failure handling. Therefore, we separate failure recovery strategy from business process to implement modeling and dynamically choreograph failure recovery process.

In this paper, we introduce Paired Net to formally describe the failure recovery mechanism of LRTs, and discuss the execution semantics of aggregation control structure. Paired Net is chosen to model failure recovery for following reasons: (i) it has a formal semantics representation, analyzing techniques and verifying tools; (ii) it has well graphical representation and supports modeling and analyzing in the way of graph; (iii) it is suitable to represent typical control flow construction and support prototype design and simulation; (iv) it provides a much broader foundation for computer aided verification than abstract state machines and process algebras, which lacks in exception handing, compensation and recovery strategy.

To implement relaxed-ACID transaction, we propose a comprehensive failure recovery algorithm based on extended Paired Net, which introduces state token, input/output data token, QoS token and control token respectively, and constructs failure transition and recovery transition. The failure type of each task has a corresponding recovery transition, that is, recovery token...
fires recovery transition to start corresponding recovery strategies.

II. FORMAL DESCRIPTION OF WEB SERVICES BASED ON PAIRED-NET

Firstly, we propose the formal definition of Web services based on Petri-Net.

Definition 1 Web Service (WS) is a tuple \( I = (P,T,F) \) where:

(i) \( P = P^s \cup P^o \cup P^w \cup P^f \) where \( P^s \) denotes the finite set of state places of task \( I \), \( P^o \) denotes a set of reverse actions \( \{\text{Compensate}, \text{Half} \} \) of \( I \), \( P^w \) denotes a set of output parameters of \( I \), it usually refers to functional metrics of WS. \( P^w \) denotes finite set of place of control token. To simplify the formal definition, we use \( I.p^o, I.p^w \) to describe the states, functional metrics and non-functional metrics of \( I \) respectively.

(ii) \( T = T^s \cup T^w \cup T^f \cup \tau \), where \( T^s \) denotes a set of normal actions \( \{\text{Activate}(), \text{Run}(), \text{Fail}(), \text{Abort}(), \text{Cancel}(), \text{Commit}()\} \) of \( I \), \( T^o \) denotes a set of directed actions \( \{\text{Compensate}(), \text{Hcompensate}(), \text{Retry}()\} \) of \( I \), \( \tau \) denotes a silent transition which executes no operation and takes no time, so it does not change the execution semantics. To simplify notation, \( r^w \) denotes that task will be executed when \( \text{Activate}() \) is activated; \( r^o \) denotes that task will be handled when \( \text{Run}() \) is executed. \( \text{Fail}() \) will be triggered if failure occurs during task execution, or \( \text{Retry}() \) will be executed several times until task is successfully finished, denoted as \( r^w \) and \( r^o \) respectively. Sending \( \text{Abort}() \) to task in \( \text{Activate}() \) can abort it, denoted as \( r^o \). When finished, task will reach \( \text{Committed} \) after \( \text{Commit}() \) is executed, denoted as \( r^w \). \( \text{Cancel}() \) can cancel task and \( \text{Compensate}() \) can eliminate the effects of the committed task, denoted as \( r^o \) and \( r^w \) respectively. For convenience, \( I.t \) is used to acquire actions of \( I \).

(iii) \( F = (P \times T) \cup (T \times P) \) denotes a set of directed arcs from \( P \) to \( T \) or from \( T \) to \( P \), which is called control flow. The constraint relations on \( (t_i, p_i) \) and \( (t_i, p_i) \) from different actions and states are: \( (t_i, p_i) < (t_j, p_j) \) if and only if \( (t_i, p_i) < (t_j, p_j) \).

The Web tasks can transfer among different states by executing different action transitions. WS transaction behavior can happen if both preconditions and constraints are satisfied, the postposition of the behavior shows the result of action executing. To other tasks, a Web task can be considered as a black box, denoted as \( I \). Only the interactive interfaces and external actions through which \( I \) interact with the environment are visible.

III. COMPOSITION TRANSACTION MODEL

A. Transaction Type of Atomic WS

Each atomic WS has its own transaction behaviors. According to different functional semantics and behaviors of Web transaction, WS can be classified into four types: Pivot WS (\( WS^p \)), Compensable WS (\( WS^c \)), Retriable WS (\( WS^r \)) and Vital WS (\( WS^v \)), which is denoted as \( \text{TBP}(WS) \), \( \text{TBP}(WS) \in \{\text{Pivot}, \text{Compensable}, \text{Vital}, \text{Retriable}\} \).

To analyze compensation transaction and how specific compensable transaction can affect its behavior exactly, it is necessary to discuss the behavior dependency between each atomic WS and the environment. The behavior of atomic transaction is formally described as follows.

\( WS^c \) can neither be retried nor be compensated. Once successfully executed, the effects of task in WS can not be eliminated. The Petri Net of \( I^c \) (which in \( WS^c \)) is shown in Fig. 1(a), there exist following transaction behaviors:

(i) If \( I^c \) is executed along path \( t_{12} \), then \( I^c \) is successfully committed and task reach \( \text{Committed} \), thus \( \text{Run}(), \text{Running}() \prec \text{Commit}(), \text{Committed} \) is satisfied.

(ii) If \( I^c \) is executed along path \( t_{13} \), then \( I^c \) failed and transfers to \( \text{Failed} \); thus \( \text{Run}(), \text{Running}() \prec \text{Fail}(), \text{Failed} \) is satisfied. Because \( I^c \) is not committed and no compensation needed, there exist no constraint \( \text{Fail}(), \text{Failed} \prec \text{Compensate}(), \text{Half} \).

(iii) If \( I^c \) is executed along path \( t_{14} \), then the executing of \( I^c \) is aborted and transfers to \( \text{Abort} \), thus \( \text{Run}(), \text{Running}() \prec (\text{Abort}(), \text{Aborted} \) is satisfied.

(iv) If \( I^c \) is executed along path \( t_{15} \), then the commit of \( I^c \) is cancelled and transfers to \( \text{Cancelled} \), thus \( \text{Commit}(), \text{Committed} \prec (\text{Cancel}(), \text{Cancelled} \) is satisfied.

For \( I^c \) in \( WS^c \), if its effects can not be eliminated after \( I^c \) committed, it is considered to be unrecoverable.

\( WS^c \) are the successfully committed tasks whose effect can be semantically eliminated by invoking their corresponding compensation tasks. The difference between transaction behaviors of \( WS^c \) and those of \( WS^c \) is that the former is compensable. As shown in Fig. 1(b), there exist following transaction behaviors:

(i) If \( I^c \) is executed along path \( t_{24} \), \( I^c \) is executed after \( I^c \) is committed successfully to eliminate its effects. Then \( I^c \) transfers to \( \text{Compensated} \) Thus \( \text{Commit}(), \text{Committed} \prec (\text{Compensate}(), \text{Compensated} \) is satisfied.

(ii) If \( I^c \) is executed along path \( t_{25} \), \( I^c \) is cancelled and may have partial effects on business transaction. Therefore, the task transfer to \( \text{Hcompensate} \). Thus \( \text{Run}(), \text{Running}() \prec (\text{Abort}(), \text{Aborted} \prec (\text{Hcompensate}(), \text{Half} \prec \text{Compensated} \) is satisfied.

For \( WS^c \), there are \( I \) and \( I^c \), where \( I^c \) can eliminate the effects of \( I \).
WSr can be executed for several times to ensure that they will be committed successfully. The difference between the transaction behaviors of WSr and those of WSps is that the former is retriable. As Fig. 1(c) shows, if
\[ jvWS ≤ jI \]
is satisfied, the temporal constraint is that the former is retriable. As Fig. 1(c) shows, (i) \( jvWS ≤ jI \) and we call is .

The above-mentioned compensability and retriability of WS are orthogonal transactional properties. That is, compensable WS are not retriable while retrievable WS are not compensable.

WSr are both retrievable and compensable. The Petri Net description of I' (which in WSr ) is shown in Fig. 1(d). Obviously, it has the transaction behaviors of both WSps and WSr.

Fig. 1 shows the execution states and transition of the tasks. During the execution, tasks can be in one of the following states: Ready \( ⇔ p_0 \), Running \( ⇔ p_0 \), Committed \( ⇔ p_0 \), Aborted \( ⇔ p_0 \), Cancelled \( ⇔ p_0 \), Failed \( ⇔ p_0 \), Compensated \( ⇔ p_0 \), and may trigger internal or external transitions: Activate( ) \( ⇔ t_0 \), Run( ) \( ⇔ t_0 \), Abort( ) \( ⇔ t_0 \), Cancel( ) \( ⇔ t_0 \), Fail( ) \( ⇔ t_0 \), Retry( ) \( ⇔ t_0 \), Compensate( ) \( ⇔ t_0 \), Hcompensate( ) \( ⇔ t_0 \).

### B. Composition Transaction Model

Each WS has its own transaction properties, therefore, when a certain service fails, how to coordinate other WS to ensure the reliability of the whole system is a critical problem to be solved. Composition services can be considered as a business process where each task is related to a certain kind of WS in service layer and is implemented through the execution of a specific WS.

#### Definition 2

**Transactional Composition Services (TCS)** is defined as a tuple \( TCS = (I,O,α,β,γ) \), where:

1. \( I = \{ l_1, l_2, ..., l_n \} \), where \( l_i \in \{ I', I', I', I' \} \) \( (1 ≤ i ≤ m) \)
denotes task in the business process.
2. \( O \) is a binary operator \( I_1 ⊙ I_2 \) on LRTs, where

   \[ α ∈ \{<, ≤, =\} \]

   \( I_i ⊙ I_j \) denotes that \( I_i \) is executed before \( I_j \), where \( < \) is call strong order . \( I_i ≤ I_j \) means neither \( I_i < I_j \) nor \( I_j < I_i \) and we call \( ≤ \) weak order.
3. \( β : I → AggregationType \) is a function, where \( AggregationType \in \{ null, sequence, And-join, Or-join, And-split, Or-split, loop \} \). Atomic tasks can be constructed to complex composition through aggregation operators such as \( ⊕, ||, Θ, \cdots \) and \( \sqsubseteq \). The usual aggregation pattern is

\[ CS = (I_1 ⊕ I_2) \cdots (I_{n-1} ⊕ I_n) \]

where \( I_i ⊕ I_j (1 ≤ i, j ≤ n) \) satisfies the temporal constraint before(\( t_i, t_j \)) , \( i < j \). Only if \( I_i \) is activated after \( I_j \), is successfully committed, the failure of \( I_i \) may depend on \( I_j \), as shown in Fig. 2(a);

Parallel aggregation pattern \( CS = I_1 || I_2 \cdots || I_n \), where \( I_i || I_j (1 ≤ i, j ≤ n) \) meets the temporal constraint equals(\( I_i, I_j \)) or finishes(\( I_i, I_j \)) , only if concurrent execution of \( I_i \) and \( I_j \) are both successfully committed, \( I_i || I_j \) is successfully committed. If \( I_i \) or \( I_j \) failed, \( I_i || I_j \) will be aborted, as shown in Fig. 2(b);

Selection aggregation pattern \( CS = I_1 ∨ I_2 \cdots ∨ I_n \), where \( I_i || I_j (1 ≤ i, j ≤ n) \) denotes \( I_i \) or \( I_j \) is executed according to selection condition case(cond) . \( I_i ∨ I_j \) is successfully committed if and only if one of the branches is successfully committed, as shown in Fig. 2(c);

Discriminator aggregation pattern \( CS = I_1 Θ I_2 \cdots Θ I_n \), where \( I_i Θ I_j (1 ≤ i, j ≤ n) \) is similar to operator ||. A guard function \( guard(status) \) is added to operator \( Θ \) to capture the first committed branch, as shown in Fig. 2(d);

Iteration aggregation pattern \( CS = \{ I \} \), where \( \{ I \} \) is executed repeatedly according to the times of iteration \( λ = |I| \), as shown in Fig. 2(e).

#### Figure 1. State/transition diagram of atomic Web services.

#### Figure 2. Aggregation patterns.

#### (iv) \( β : I → State \) is a function, where \( State \in \{ initial, active, failed, completed, aborted, cancelled \} \). The state of LRTs can be determined by the execution progress of the composition task, denoted as \( CS.state = \bigcup_{i=1}^{n} (I_i.sparse) \).

#### (v) \( γ : I → TransactionType \) is a function, where \( TransactionType ∈ \{ Vital, Retriable, Pivot, Compensable \} \). The
transaction behaviors of each atomic task are composed as transaction behavior of composition services, denoted as $CS.TransactionProperty = \bigcup_{i=1}^{n}(I,TransactionProperty)$.

IV. THE WEB SERVICES COMPOSITION BASED ON PAIRED NET

The execution of LRTs can be successfully committed, failed or cancelled. The service compensation model is so important that it will capture the occurrence of failure as well as cancellation, then take measures to activate the related compensation service according to the compensation strategies.

Now we construct service compensation model in two steps:

(i) Constructing failure place and compensation transition: Consider that the failure of transition $t_i$ occurs, compensation transition $t'_i$ is introduced for $t_i$, and $t'_i$ is executed when $t_i$ needs to be compensated. Then two failure places $p'_i$ and $p_i'$ are introduced for $p_i$ and $p_i$ to represent prepositive state place and postpositive state place of $t'_i$ respectively, as shown in Fig. 3(a).

(ii) Constructing mapping. Adding an activating place $p'_i$ with uncertain state between $t_i$ and $t'_i$. Adding failure transition $t'_i$ and $t'_i$ between $p_i$ and $p_i$ and between $p_i$ and $p_i$ respectively, as shown in Fig. 3(b).

Compensation tasks are colored in grey to distinguish from the normal tasks. The constructed model is similar to the normal task model in term of structure except that the flow relation is reversed. This implies that the execution of compensation service is always in a reverse order of the execution of normal service. According to the compensation model above, the formal representation of the compensation mechanism of WS process is described as follows:

![Figure 3. Service compensation modeling.](image)

**Definition 3** Given a transactional composition services $TCS = (I,O,\alpha,\beta,\gamma)$, where $I = (P,T,F,1)$, suppose that $T'$ and $P'$ are the sets of failed transitions and uncertain state activating places respectively such that for each $p_i \in P'$ there is a unique $t'_i \in T'$ and for each $l_i,l \in T'$ there is a unique $p_i' \in P'$, then the Service Compensation Paired Net of $I$ is described as $SCP\bar{N} = (P,\bar{T},\bar{F},\bar{R})$, as shown in Fig. 4, where: (i) $\bar{T} = P \cup \{p_i' \mid p_i' \in P' \land p_i \in P\} \cup \{t'_i \mid t'_i \in T' \land t_i \in T\}$; (ii) $\bar{F} = F' \cup F'' \cup F'''$, where $F' = \{(p_i \times T) \mid P \subset F \cup \{(T \times P') \mid (P \times T) \in F\}$, $F'' = \{(t_i,p_i') \mid p_i \in P \land t_i' \in T' \cup \{(t_i',p_i') \mid t_i' \in T' \land p_i' \in P' \}$, and $F''' = \{(p_t,p_f') \mid p_f' \in P' \} \cup \{(t_i',p_i') \mid t_i' \in T' \}$.

![Figure 4. Composition compensation paired net](image)

According to the structure of $SCP\bar{N}$, $P$ and $P'$, $T$ and $T'$ are respectively matched so that it is easy to construct the mapping from the normal service process to the compensation process, which implies to extend the service behavior $I = (P,T,F,1)$ in LRTs to $I' = (P,T',F,1')$.

V. FAILURE RECOVERY STRATEGIES OF COMPOSITION TRANSACTIONS

To meet the requirements of composition transaction recovery, we propose a flexible method to dynamically calculate recovery scope according to the dependency between task and execution environment in LRTs.

A. Recovery Scope

For given execution trace $\sigma$ in LRTs, if $\exists \sigma I,l_i \in \sigma \land I,TBP,l_i,TBP \in \{Vital,Compensable\}$, one of the transaction dependency $I_j ||_{tod} I_i$, $I_j ||_{tod} I_i$, $I_j ||_{to} I_i$, $I_j ||_{to} I_i$ exists between $I_i$ and $I_j$, and $I_i,Compensated \land I_j,Compensated$, then $\sigma$ is called a compensation sequence, denoted as $\sigma,Compensated$.

It is critical to determine TDP when a task in the process fails. Recovery Handler (RH) takes charge of backward executing of all the tasks in recovery scope until it reaches TDP. When failure occurs, RH executes compensation tasks or fixes the failure or chooses an alternative execution path to avoid it. For a given LRTs, if $\exists \sigma I,l_i \in \sigma \land I,TBP,l_i,TBP \in \{Vital,Compensable\}$, and one of the data flow dependency $I_j ||_{tod} I_i$, $I_j ||_{to} I_i$, $I_j ||_{tod} I_i$, $I_j ||_{to} I_i$ exists between $I_i$ and $I_j$, then $I_i$ is called the TDP of $I_i$, as shown in Fig. 5. If $I_i$ fails, $\sigma,Compensated$ will be executed from $I_i$. $\sigma,Compensated$ is called the dependency path of $I_i$. Each task in LRTs has a specification of TDP.

After TDP is determined, RH executes $\sigma,Compensated$ in reverse from failed task to try to eliminate the effects brought by committed tasks in recovery scope. For a given LRTs, if $I_i(l_i \in \sigma)$ fails (the set of dependency path of $I_i$ is $\{\sigma_{i_1} \land \sigma_{i_2} \land \ldots \land \sigma_{i_k}\}$), RH rolls back to $I_i$ and injects corrected parameters, executes the compensation tasks of $\sigma_{i_1} \land \sigma_{i_2} \land \ldots \land \sigma_{i_k}$, after that, $I_i$ is committed successfully. The set of recovery scopes of $\{\sigma_{i_1} \land \sigma_{i_2} \land \ldots \land \sigma_{i_k}\}$ is $\{E_i \cup E_i \cup E_k\}$.
The cost of reverse compensation is so high that it is necessary to minimize the compensation tasks in recovery scope, which is called minimal recovery scope, denoted as $\Xi$. The control flow structure. As shown in Fig. 6, $S[p]$, is the sub scope of $S$. The number in subscript indicates the nesting level. In general, $S[p] = \Xi$ represents that scope $\Xi$ has $n$ nesting levels which includes zero or more tasks.

B. Nesting of Scope

The nesting description of $\Xi$ is similar to that of control flow structure. As shown in Fig. 6, $S[p]$, denotes $S_{n+3}$ is the sub scope of $S$. The number in subscript indicates the nesting level. In general, $S[p] = \Xi$ represents that scope $\Xi$ has $n$ nesting levels which includes zero or more tasks.

V. RECOVERY STRATEGIES OF LRTS

Most existing specifications consider about backward recovery, which didn’t support forward recovery. The recovery strategies for LRTs proposed in this paper includes backward recovery, forward recovery and alternative recovery.

These three recovery strategies are always associated with $\Xi$, Fig. 7 depicts the mapping from LRTs to $\Xi$. The recovery strategy is enclosed in dashed lines. In this strategy, Compensation Handler (CH), RH and Abortion Handler (AH) will be activated by Compensate($Cp$), Recovery($Rv$) and Abort($A$) respectively to execute the compensation tasks in $\Xi$. If the handlers executed successfully, then failure transition is transferred to Compensated($Cpd$) and Recovered($Rvd$) respectively, which means they have finished the compensation or recovery of failed process successfully.

A. Backward Recovery Strategy

When the engine failed in the executing process, the failed task $I$, throw fail() to terminate execution of forward flow. RH is triggered to calculate $\Xi$ and to get the log ($ActID$, $Desc$, $QoS$, $TBP$, $State$, $Behavior$) of the successfully executed tasks. It constructs input interface $I_{\text{in}}$ of compensation task according to the input/output interface ($I_{\text{in}}$, $I_{\text{out}}$) of the committed tasks, sets the external state place $Cp$ of $I$, and fires Compensate($Cp$), activates and executes $I'$. For convenience, we assume that LRTs consists of forward execution flow and backward compensation flow when failure occurs, which are associated with normal transaction behavior and compensation behavior of a task, as shown in Fig. 7.

Since tasks in $\Xi$ hold $I_{\text{TP}} \in \{\text{Vital, Compensable}\}$, RH gets the corresponding $I'_{\text{TP}}$ and constructs the mapping between $I'$ and $I$. It is called compensation pair $I, I'$.  

![Image](https://example.com/image.png) 

Fig. 1 The recovery model of composition Web transaction.

Fig. 7. Recovery model of LRTs.

As Fig. 8 shows, $I$ and $I'$ both represent atomic transactions with states. Business process and its compensation process are enclosed by dash dotted lines. Transition models the internal behaviors of tasks while place carries out the internal and external interaction between sub transactions. If state place failed, it triggers RH to make control flow turn to compensation flow. Backward Compensation Paired Strategy (BCPS) is introduced to formally describe it:

The Petri Net of $I$ and their compensation tasks $I'$ in LRTs are denoted as $I = (P, T, F)$ and $I' = (P', T', F')$ respectively. Since $l_{\text{out}} \in \Xi_{\text{pair}}(I, I') = (T', T)$, where $I'$ is the transition from $I$ to $l_{\text{out}}$, $I'$ is the failure transition from $I$ to $I'$, and $I'$ is the compensation transition from $I'$ to $l_{\text{in}}$. The triple of BCPS is denoted as $(\mathcal{P}, \mathcal{T}, \mathcal{F})$, where: (i) $\mathcal{P} = P \cup P' \cup F'$; (ii) $\mathcal{T} = T' \cup T' \cup T' \cup T' \cup T' \cup T';$ (iii) $\mathcal{F} = F' \cup F' \cup F' \cup F' \cup F' \cup F' \cup F' \cup F' \cup F' \cup F'$.
Algorithm 1 Construct recovery process.

Input: The set of tasks in business process $BP$, failed task $I$, and compensation dependency set $CP$

Output: Compensation Business Process $BP$, failed task $I$, and compensation dependency set $CP$

1. $t' = \cup (t_i', I_i', p_i')$, $F_{t'} = (I_i', p_i', I_i', p_i') \cup (t_i', I_i', p_i')$, as shown in Fig. 8.

BCPS is constructed by RH in two steps. Firstly, it captures the failure of $I_i$, analyzes failure type of $I_i$, sends execution request to $I_i'$, gets $I_i'$ in accordance to $I_i$, $in$ and $I_o$, and constructs recovery flow. Secondly, it starts up the compensation flow to execute backward recovery. Fig. 8 illustrates that the failed $I_i$ aborts the forward control flow and turns to recovery flow with the help of $p'_i$ or $t'_i$.

The complexity of this algorithm depends on the number of tasks in LRTs and the number of compensation tasks of $\Xi_i, \Xi_2, \Xi_3$. If $\max(|BP|,|\Xi|) = n$, then the time complexity of this algorithm is $O(n)$.

Algorithm 2 The execution of RH.

Input: the set of tasks in $BP$, $R_{flow}$

Output: the information of compensation service that is executed successfully or not

1. For each LogItem = seek(RecordLog,$I$) of task $I$ in the committed process if $I_i$.state = Completed then
   for each binding($I_i$, ws)
      $I_i'$ ← ActID, Desc, QoS, TBP, Behavior, State, $I_i$, $in$, Log informatin of ws;
   if $I_i$.Failed = true then
      trigger Compensating Handler CH;
   if $I_i$.TBP ∈ (Compensable,Vital) and $I_i$.state = Completed and $I_i$ is atomic task then
      for each compensation task $I_i'$;
      CH execute $I_i'$ and record execution information of $I_i'$;
      else if $I_i$ is composition task then
      $\{\text{divide task } I_i' \text{ into subtask } I_i', I_i', \cdots I_i'\}$;
      for each subtask $I_i' (I_i \text{ from } i_i \text{ to } i_e)$
      execute($I_i'$);
      else if $\text{compensate}(I_i) = \text{false}$ and $I_i$ is atomic task then
      Forward $I_i'$, compensation continue;
      if notExit(CBP) then
      $\text{Compensation flow } CPB \text{ is not exist;}$
      for each task $I_i$
      if $I_i$.in_Exit and $I_i$.state = running then
      $\{\text{CH send abort() to ws;}$
      set task $I_i$'s state as "Aborted";
      $\}$ for each task $I_i$
      if $I_i$.in_Exit and $I_i$.state = active then
      $\{\text{CH send cancel() to ws;}$
      set task $I_i$'s state as "canceled";
      $\}$ for each task $I_i$
      if $I_i$.enter then
      Compensate task $I_i'$, set $I_i$'s state as "canceled";
      if allcompensation(CBP) = true then
      return success
      else
      return failure;
   } For $|R_{flow}| = n$, $\text{Edge}(I) = m$, $I_i \in R_{flow}$, where $|R_{flow}|$ and $\text{Edge}(I)$ is the number of tasks and edges in CBP respectively, then the time complexity is $O(nm)$.

B. Forward Recovery Strategy

Forward recovery strategy is similar to BCPS except that when rolling back to TDP, which RH injects proper input parameters and restart the expected execution of
forward flow. The Paired Net of each task in LRTs can be denoted as \( I = (P, T, F) \). For failed task \( I \), TDP of \( I \) is calculated and \( \Xi \) is determined. Similar to BCPS, a triple \( (\tilde{P}, \tilde{T}, \tilde{F}) \) is used to represent Forward Compensation Paired Strategy (FCPS), where:

1. \( \tilde{P} = P \cup P' \cup \{p'\} \);
2. \( \tilde{T} = T \cup T' \cup \{t'\} \);
3. \( \tilde{F} = F \cup F' \cup \{t'_c, p', (t'_c, p'), \ldots\} \),
   \( j \in \Xi \), as shown in Fig. 9.

RH constructs FCPS in 3 steps: Firstly, it terminates the execution of LRTs and determines the TDP and \( \Xi \) of \( I \); secondly, it executes recovery; finally, it injects proper parameters at \( I \) and restarts the LRTs from \( I \). Note that if TDP is \( I \), there is no difference between BCPS and FCPS during recovery.

**Algorithm 3** Forward recovery algorithm

Input: failure task \( I \), log BP

Output: the information of success or failure of the recovery

1. \( \text{taskstate} = \text{getTaskState}(I, \text{RecordLog}); \)
2. if \( \text{taskstate} = \text{Failed} \) then
3. \( \Xi = \text{calculateCompensationScope}(I); \)
4. if \( \text{taskstate} = \text{Aborted} \) then
5. for each task \( I \), \( \text{state} = \text{Running} \)
6. \( \text{if}(I) \}\textit{execute} \( I \)
7. \( \text{interrupt each task} I \) in running flow,
8. for each \( I \), \( \in \Xi \),
9. if \( \text{getTaskState}(I, \text{RecordLog}) = \text{committed} \) then
10. \( \text{LogItem} = \text{seek}(\text{RecordLog}, I) \)
11. \( \text{input} \) the value of the LogItem to task \( I \) and compensate \( I \);
12. change the value of the LogItem to task \( I \) and re-execute \( I \);
13. if \( I \).state = Failed or \( I \).state = Aborted then
14. \( \text{delete log record in FailedLog, AbortedLog, CancelledLog}; \)

The time complexity of reverse execution of recovery process and restart forward recovery from TDP are both \( O(n) \), thus the complexity of this algorithm is \( O(n) \).

**C. Alternative Recovery Strategy (ARS)**

When executing \( I \) fails, TDP of \( I \) is calculated and \( \Xi \) is determined. If there exists \( \{I\} = \{I\} = \Xi \), occurrence of failure is caused by executing task itself rather than output of the committed tasks. In this situation, it is not necessary to perform backward compensation recovery. A simple and direct way to solve the problem is to trigger alternative recovery tasks \( I' \) of \( I \), then the execution of the subsequent process proceeds. The formal description of ARS is given as follows.

The Petri Net of each task in LRTs is denoted as \( I = (P, T, F) \). For failed task \( I \), TDP of \( I \) is calculated and \( \Xi \) is determined. If \( \exists I \in \Xi | I = I' \), then alternative task \( I' = (P', T', F') \) and alternative state place \( p' \) is added. ARS can be denoted as a triple \( (\tilde{P}, \tilde{T}, \tilde{F}) \), where:

1. \( \tilde{P} = P \cup P' \);
2. \( \tilde{T} = T \cup T' \cup T' \cup T' \cup \{t'\} \); \( \tilde{F} = F \cup F' \cup F' \cup \{t'_c, p', (t'_c, p'), \ldots\} \),
   as shown in Fig. 10.

**Algorithm 4** Alternative recovery algorithm

Input: failure task \( I \), log BP

Output: the information of success or failure of the recovery

1. \( \text{taskstate} = \text{getTaskState}(I, \text{RecordLog}); \)
2. if \( \text{taskstate} = \text{Failed} \) then
3. \( \Xi = \text{calculateCompensationScope}(I); \)
4. if \( \text{taskstate} = \text{Aborted} \) then
5. for each task \( I \), \( \text{state} = \text{Running} \)
6. \( \text{if}(I) \}\textit{execute} \( I \)
7. \( \text{interrupt each task} I \) in running flow,
8. for each \( I \), \( \in \Xi \),
9. if \( \text{getTaskState}(I, \text{RecordLog}) = \text{committed} \) then
10. \( \text{LogItem} = \text{seek}(\text{RecordLog}, I) \)
11. \( \text{input} \) the value of the LogItem to task \( I \) and compensate \( I \);
12. change the value of the LogItem to task \( I \) and re-execute \( I \);
13. if \( I \).state = Failed or \( I \).state = Aborted then
14. \( \text{delete log record in FailedLog, AbortedLog, CancelledLog}; \)

The time complexity of reverse execution of recovery process and restart forward recovery from TDP are both \( O(n) \), thus the complexity of this algorithm is \( O(n) \).

If failure occurs during execution, proper recovery strategy will be selected according to failure types. In general, the cost of BCPS is the biggest while ARS is the smallest and FCPS between them. This is the main reason why FCPS is introduced in this paper. When a task fails, retry will be selected firstly considering the cost, if it still fails, ARS will be selected. Finally, if RH has ability of
compensation or abortion, BCPS or FCPS will be selected to ensure the consistency of the whole composition service.

VI. EXPERIMENTAL RESULTS AND ANALYSIS FOR TRAVELING RESERVATION PROCESS (TRP)

We take TRP for example to analyze and verify LRT’s algorithm, as shown in Fig. 11. The classical TRP which is composed of more than 40 tasks and 100 services (recovery services included). We use LCFR algorithm and non-LCFR algorithm respectively to reduces execution time and cost, and decrease the failure rate. The experiment is taken on PCs with Intel Pentium Dual Core E5200 2.5 GHz, 2GB RAM, Windows XP SP2, Apache Tomcat server, jUDDI registry, Active Bpel designer, Axis, eclipse 3.2.2.

The TRP has the following possible transaction behaviors. Firstly, traveler makes a Customer Requirements Specification (CRS) according to his traveling plan and submits it to travel agent. Then, according to CRS, travel agent carries out Flight Booking (FB) or Train Reserving (TR), Hotel Booking (HB), Car Renting (CR) or Bus Renting (BR) sequentially. Then, after the reservations are confirmed, traveler carries out Online Payment (OP). Finally, express company carries out Ticket Delivery with EMS (TDE) or UPS (TDU), and sends “reservation successful” message to traveler to do Traveler Confirmation (TC). Using hierarchical nesting enhances the efficiency of aggregation nesting TRP.

This work implements as part of the TRPFlow projects which objectively compares the providers’ and requests’ service profiles. TRPFlow implements with the Activekit platform which permits the development of distributed applications using composition transaction principles. Fig. 12 shows a screenshot of TRPFlow, which partially shown in Fig. 12 is implemented and users can visualize its corresponding PNO-tree and a partial view of TRP. Interface of RH transforms the eXtended Markup Language (XML) output from other RHs to the current platform with XML Stylesheet Language (XSL).

As shown in Fig. 13, the LCFR curve is the accumulative execution time and cost curve, while non-LCFR curve does not takes execution time and cost into account. The LCFR curve means once the executing engine executes a task in LRTs, execution time and cost is evaluated. This experiment aims to compare the difference of execution time and cost between LCFR and non-LCFR and to discuss performance of LRTs. Observe that during execution of the LRT process, the number of composed tasks of the orchestration increases, so does the dependency between tasks. Compared with LCFR, in non-LCFR the average growth rate of execution time and cost of TRP is 0.69 and 0.16 respectively, while in LCFR they are effectively cut down. Also, the average negative growth rate of reliability, reputation and availability is 1.93, 1.12 and 1.24 respectively. As Fig. 13 shows, the more tasks are executed in the LRT, and the more complicate nesting levels are, the greater benefit will be gained by LCFR.
As shown in Fig. 14, LCFR1, LCFR2, LCFR3, LCFR4 and LCFR5 are LCFR curves that lead to different failure rate. We select five sets of WS to compose TRP. Only a set of WS is successfully executed and the other four groups failed in the 3rd, 4th, 7th and 8th task respectively. Observe that the inflection point of failure rate is in accordant with that of response time in the 7th and 8th task and the execution time and cost almost unchanged compared with that before starting recovery strategies. However, for composition process which requires high real-time performance, response time and execution time are prior to the other metrics.

The LCFR1, LCFR2, LCFR3, LCFR4 and LCFR5 in Fig. 15 are curves after recovery respectively. Forward recovery is executed respectively at the 3rd, 4th, 7th, 8th and 9th task and number of corresponding compensated tasks is 7, 11, 15, 21 and 28 respectively. Observe that when failure occurs, the RH will be triggered and compensation context be constructed so that the response time will increase, but it has little effect to execution time and cost.

**VII RELATED WORK**

When using LRTs to composite WS, it is necessary to make sure the atomicity of a set of interactive WS. LRTs undo the effects of failed transactions by executing compensation tasks. N. B. Lakhal and T. Kobayashi et al. put forward a failure endurable execution framework of nested-composition transaction and composition transaction architecture, which discuss the effects to the current layer that brought by failed tasks, support a on-off failure dependency that orients the whole composition transaction, define arbitrary nesting, state, vitality degree and compensation, and analyze the execution semantics of recursive nested transactions [1, 2, 3]. R. Yi and Q.Y. Wu et al. put forward a failure recovery algorithm ensuring LRTs which avoid unnecessary compensation and enhance the efficiency of failure recovery, they focus on handling the complexity and long-running and reducing the failure rate of transaction execution[4]. K. Wiesner and R. Vacul et al. present a new dynamic recovery mechanism based on semantics, which dynamically discover failure and execute recovery with equivalent alternative service, in the context of recovery and process adaptation [5]. It is hard for LRTs to select partner and predict execution time, thus it is forbidden to suspend LRTs, T. Minh et al. proposed a web transaction based on semantics [6]. S. Bhiri and O. Perrin et al. put forward a reliable and flexible Web composition transaction method, which according to control flow and data flow dependency of aggregation patterns, ensure the highly cohesive of control flow and transaction patterns and the relaxed atomicity of composition transactions using the properties of Acceptable Termination State(ATS) [7, 8].

Compensation transaction is a new transaction which includes two flow types: normal flow and compensation flow, where the first describes normal business logic and later semantically undo the effects of normal flow. S. Rinderle and M. Reichert et al. present a method to realize forward recovery according to dynamic workflow changes. It allows semi-adaptive workflow in case of
failure, which introduces operations such as deleteAct, jumpTo and insertAct to implement transaction recovery [9, 10]. However, there are two main disadvantages: firstly, it can only realize semi-automatic adaptation. Secondly, this kind of workflow change needs rigid definition. Z. Yang and L. Lin et al. put forward a backward recovery compensation method that supports business process, which mentions forward recovery but doesn’t focus on it. This method considers compensation logic as part of coordination logic, which leads poor flexibility of compensation [11, 12].

Existing transaction mechanism only provides limited compensation ability. In most cases, backward recovery is used to maintain consistency after aborting the executing tasks. M. Schafer and P. Dolog et al. extend the current WS coordination architecture and infrastructure, propose an advanced compensation environment based on forward recovery strategy. This method separates compensation logic from coordination logic and realizes the dynamic plug in and plug out of compensation strategies [13]. M. Schafer and P. Dolog et al. put forward an engineering compensation method based on WS environment that discuses engine dependency rules and constructs compensation method based on rules [14]. Y. Kim and J. Kim propose a WS composition framework allowing user-specified failure handling, which defines user requirement based on general business logic and failure acceptable specification [15]. R. Bruni and G. Ferrari et al. put forward an framework called Java Transactional WS (JTWS), which is a Java API providing suitable primitives for wrapping and invoking WS as activities in LRTs [16]. Therefore, with the backward recovery approach, the failure of any single participating WS can trigger the abort of many transactions and thus lead to cascading compensations (called the domino effect), which can result in a huge loss of time and cost. To sum up, it is necessary to further research flexible recovery strategy based on LRTs.

Most of the existing formal composition business-process-modeling methods based on Petri nets do not support transaction mechanism, and exception behavior is beyond normal behavior logic, we need to extend Petri net. G. Dobson and M. Kovacs et al. propose a formal modeling technique for BPEL business process including fault and compensation handling based on Petri net, which give the mapping of various fault patterns to WS-BPEL [17, 18, 19]. W.L. Dong and X.G. Deng et al. use Colored Petri Nnets(CPNs) to model WS choreography and orchestration, and they analyzes and tests BPEL composition business process using HPNs [20, 21, 22, 23]. L. Garcia-Banuelos proposes an executable transaction model based on ASML that allows seamless add/modify behavior, and extends failure handler in WS-BPEL [24]. R. Hamadi and B. Benatallah propose a Self-Adaptation Recovery Net (SARN) based on Petri net, which extends Petri net model to specify fault or exceptional behavior in business process. SARN incorporates with recovery region [25, 26, 27, 28], but it is not suitable for handling business transaction in P2P environment.

VIII CONCLUSION AND FUTURE WORK

Composition transactions incorporate different transactional semantics and behavioral patterns, composed tasks usually have following properties: (i) different tasks may have different execution behaviors or transactional properties; (ii) service provider can define different transaction coordination mechanism; (iii) different failure handling and recovery strategies should be defined. Therefore, ATS is introduced in this paper to implement relaxed atomicity, and recovery strategies ensure consistency. To ensure reliable LRTs execution, state token, functional token and nonfunctional token are introduced, transaction isolation and atomicity are relaxed. Failure recovery starts from log analysis, which uses a set of dependency rules to construct the recovery flow of transaction. Our future work includes several issues such as global transaction structure, subtransaction properties, inter-subtransaction dependencies, mechanisms of handing-over, success and failure criteria should be considered.

ACKNOWLEDGMENT

This work is one of the projects supported by the National Key Technologies R&D Program of China (2008BAH24B03), the National Natural Science Foundation of China (60673122, 60940033), the Postdoctoral Science Foundation of China (20080440121), the Natural Science Foundation of Province (06017089, 60940033), the Science and Technology Planning Project of Hunan Province (2010GK3020).

REFERENCES


