A Comprehensive Failure Recovery Algorithm for LRTs Based on Paired Net

Xiaoyong Mei[1,2], Yiyan Fan[1], Changqin Huang[3], Xiaolin Zheng[4]

[1]School of Computer Science and Technology, Hunan University of Arts and Science, Changde 415000, China
[2]School of Information Science and Technology, Sun Yat-sen University, Guangzhou 510275, China
[3]College of Educational Information Technology, South China Normal University, Guangzhou 510631
[4]College 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 composited tasks. 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

Footnotes: 8-point Times New Roman font;
Manuscript received January 1, 2009; revised June 1, 2009; accepted July 1, 2009.
Copyright credit, project number, corresponding author, etc.

doi:10.4304/jsw.7.2.420-430

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) $P=P^s\cup P^{io}\cup P^{qos}\cup P^c$, P^s denotes the finite set of state places of task I, $P^s\in\{Ready, Activated, Running, Failed, Aborted, Cancelled, Committed, Compensated, Half\_Compensated\}$. P^{io} denotes the input or output parameters of I, it usually refers to functional parameters of WS. P^{qos} denotes the QoS parameters of I, it usually refers to non-functional metrics of WS. P^c denotes finite set of place of control token. To simplify the formal description, we use $I.p^s, I.p^{io}$ and $I.p^{qos}$ to describe the states, functional metrics and non-functional metrics of I respectively.

(ii) $T=T^n\bigcup T^b\bigcup\tau$, where T^n denotes a set of normal actions $\{Activate(), Run(), Fail(), Abort(), Cancel(), Commit()\}$ of I, T^b denotes a set of reverse actions $\{Compensate(), Hcompensate(), Retry()\}$ of I, $\tau\in T$ denotes a silent transition which executes no operation and takes no time, so it does not change the execution semantics. To simplify notation, t^{act} denotes that task will be executed when *Activate*() is activated; t^{run} denotes that task will be handled when *Run*() is executed. *Fail*() will be triggered if failure occurs during task execution, or *Retry*() will be executed several times until task is successfully finished, denoted as t^{fal} and t^{rty} respectively. Sending *Abort*() to task in *Activated* can abort it, denoted as t^{abt}. When finished, task will reach *Committed* after *Commit*() is executed, denoted as t^{cmt}. *Cancel*() can cancel task and *Compensate*() can eliminate the effects of the committed task, denoted as t^{cnl} and t^{cmp} respectively. For convenience, $I.t$ is used to acquire actions of I.

(iii) $F=(P\times T)\bigcup(T\times P)$ 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_1,p_1) and (t_2,p_2) from different actions and states are: $(t_1,p_1)\prec(t_2,p_2)$, $(t_1,p_1)\lhd\rhd(t_2,p_2)$ and $(t_1,p_1)\approx(t_2,p_2)$.

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 $TBP(WS)$, $TBP(WS)\in\{Pivot, Compensable, Vital, Retriable\}$.

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^p 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^p (which in WS^p) is shown in Fig. 1(a), there exist following transaction behaviors:

(i) If I^p is executed along path t_1t_2, then I^p is successfully committed and task reach *Committed*, thus $(Run(), Running)\prec(Commit(), Committed)$ is satisfied.

(ii) If I^p is executed along path t_1t_6, then I^p failed and transfers to *Failed*; thus $(Run(), Running)\prec(Fail(), Failed)$ is satisfied. Because I^p is not committed and no compensation needed, there exist no constraint $(Fail(), Failed)\not\prec(Hcompensate(), Half\_Compensated)$.

(iii) If I^p is executed along path t_1t_3, then the executing of I^p is aborted and transfers to *Aborted*, thus $(Run(), Running)\prec(Abort(), Aborted)$ is satisfied.

(iv) If I^p is executed along path t_1t_5, then the commit of I^p is cancelled and transfers to *Cancelled*, thus $(Commit(), Committed)\prec(Cancel(), Cancelled)$ is satisfied.

For I^p in WS^p, if its effects can not be eliminated after I^p 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^p 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 $t_1t_2t_8$, $I^{c'}$ is executed after I^c is committed successfully to eliminate its effects. Then I^c transfers to *Compensated*. Thus $(Commit(), Committed)\prec((Compensate(), Compensated)$ is satisfied.

(ii) If I^c is executed along path $t_1t_3t_9$, I^c is cancelled and may has partial effects on business transaction. Therefore, the task transfer to *Hcompensated*. Thus $(Run(), Running)\prec(Abort(), Aborted)\prec(Hcompensate(), Half\_Compensated)$ is satisfied.

For WS^c, there are I and I', where I' can eliminate the effects of I.

WS^r can be executed for several times to ensure that they will be committed successfully. The difference between the transaction behaviors of WS^r and those of WS^p is that the former is retriable. As Fig. 1(c) shows, there exist following transaction behavior:

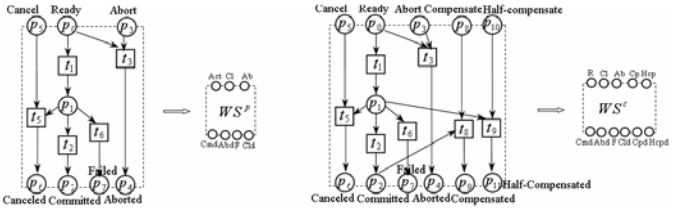
If I^r is executed along path $t_1(t_6t_7)^n t_2$ (where $n \geq 1$, n is the times of retry), the failed task is executed repeatedly, finally it is committed successfully and reaches *Committed*. Thus $(Run(), Running) \prec (Fail(), Failed) \prec (Run(), Running) \prec (Commit(), Committed)$ is satisfied.

For I in WS^r, if I has failed k $(k \leq n)$ times, n is the maximum times of retry times, then I is sure to be committed successfully within m times ($(k+1) \leq m \leq n$).

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

WS^v are both retriable and compensable. The Petri Net description of I^v (which in WS^v) is shown in Fig. 1(d). Obviously, it has the transaction behaviors of both WS^c and WS^r.

Fig. 1 shows the execution states and transition of the tasks. During the execution, tasks can be in one of the following states: $Ready \leftrightarrow p_0$, $Running \leftrightarrow p_1$, $Committed \leftrightarrow p_2$, $Aborted \leftrightarrow p_4$, $Cancelled \leftrightarrow p_6$, $Failed \leftrightarrow p_7$, $Compensated \leftrightarrow p_9$, and may trigger internal or external transitions: $Activate() \leftrightarrow t_1$, $Run() \leftrightarrow t_2$, $Abort() \leftrightarrow t_3$, $Cancel() \leftrightarrow t_5$, $Fail() \leftrightarrow t_6$, $Retry() \leftrightarrow t_7$, $Compensate() \leftrightarrow t_8$, $Hcompensate() \leftrightarrow t_9$.



(a) Pivot Web services (b) Compensable Web services

(c) Retriable Web services (d) Vital Web services

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

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 $TCS = (I, O, \alpha, \beta, \gamma)$, where:

(i) $I = \{I_1, I_2, ..., I_m\}$, where $I_i \in \{I^p, I^c, I^r, I^v\}$ $(1 \leq i \leq m)$ denotes task in the business process.

(ii) O is a binary order $I_i \circ I_j$ on LRTS, where $\circ \in \{\prec, \lessdot\}$. $I_i \prec I_j$ denotes that I_i is executed before I_j where $\prec$ is call *strong order*. $I_i \lessdot I_j$ means neither $I_i \prec I_j$ nor $I_j \prec I_i$ and we call $\lessdot$ is *weak order*.

(iii) $\alpha : I \rightarrow AggregationType$ is a function, where $AggregationType \in \{null, sequence, And\text{-}join, Or\text{-}join, And\text{-}split, Or\text{-}split, loop\}$. Atomic tasks can be constructed to complex composition through aggregation operators such as $\oplus, \|, \otimes, \Theta$ and $\lfloor \_ \rfloor$. The usual aggregation pattern is $CS = (I_i \oplus I_{i+1}) | (I_i \| I_{i+1}) | (I_i \otimes I_{i+1}) | (I_i \Theta I_{i+1}) | \lfloor I_i \rfloor$ where:

Sequence aggregation pattern $CS = I_1 \oplus I_2 \oplus \cdots \oplus I_n$, where $I_i \oplus I_j (1 \leq i, j \leq n)$ satisfies the temporal constraint $before(t_i, t_j)$, where $i < j$. Only if I_j is activated after I_i is successfully committed, the failure of I_j may depend on I_i, as shown in Fig. 2(a);

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

Discriminator aggregation pattern $CS = I_1 \Theta I_2 \Theta \cdots \Theta I_n$, where $I_i \Theta I_j (1 \leq i, j \leq n)$ 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 $CS = \lfloor I_1 \rfloor$, where $\lfloor I_1$ is executed repeatedly according to the times of iteration $\lambda = ||I_1|$, as shown in Fig. 2(e).

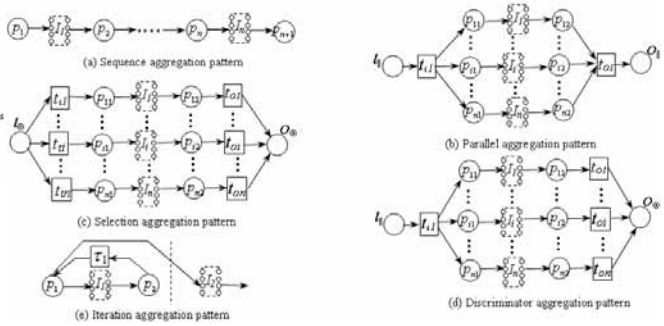


Figure 2. Aggregation patterns.

(iv) $\beta : I \rightarrow State$ 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 $CS.state = \bigcup_{i=1}^{n} (I_i.state)$.

(v) $\gamma : I \rightarrow TransactionType$ is a function, where $TransactionType \in \{Vital, Retriable, Pivot, Compensable\}$. The

transaction behaviors of each atomic task are composed as transaction behavior of composition services, denoted as $CS.TransactionProperty = \bigcup_{i=1}^{n}(I_i.TransactionProperty)$.

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_1 occurs, compensation transition t_1' is introduced for t_1, and t_1' is executed when t_1 needs to be compensated. Then two failure places p_2' and p_1' are introduced for p_2 and p_1 to represent prepositive state place and postpositive state place of t_1' respectively, as shown in Fig. 3(a).

(ii) Constructing mapping. Adding an activating place p_1^f with uncertain state between t_1 and t_1'. Adding failure transition t_1^f and t_2^f between p_1 and p_1' and between p_2 and p_2' respectively, as shown in Fig. 3(b). If t_1 fails, p_1 fires t_1^f and arrives at p_1' to forward compensate t_1' before t_1. If the failure happens after t_1, p_2 fires t_2^f and arrives at p_2' to compensate t_1' before t_1 (including t_1).

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:

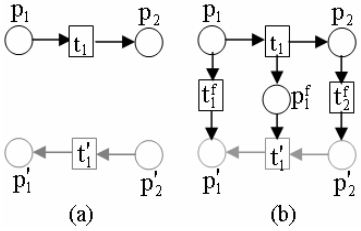


Figure 3. Service compensation modeling.

Definition 3 Given a transactional composition services $TCS = (I, O, \alpha, \beta, \gamma)$, where $I = (P, T, F, \ell)$, suppose that T^f and P^f are the sets of failed transitions and uncertain state activating places respectively such that for each $p_i \in P$ there is a unique $t_i^f \in T^f$ and for each $I_j.t \in T$ there is a unique $p_j^f \in P^f$, then the Service Composition Paired Net of I is described as $SCPN = (\overline{P}, \overline{T}, \overline{F}, \overline{\ell})$, as shown in Fig. 4, where: (i) $\overline{P} = P \cup \{P' \mid p' \in P' \wedge p \in P\}$; (ii) $\overline{T} = T \cup \{T' \mid t' \in T' \wedge I.t \in T\} \cup T^f$; (iii) $\overline{F} = F \cup F' \cup F^f$, where $F' = \{(P' \times T' \mid (T \times P) \subset F\} \cup \{(T' \times P') \mid (P \times T) \subset F\}$, $F^f = \{(p_i, t_i^f) \mid p_i \in P \wedge t_i^f \in T^f\} \cup \{(t_i^f, p_i') \mid t_i^f \in T^f \wedge p_i' \in P'\} \cup \{(t_j, p_j^f) \mid t_j \in T \wedge p_j^f \in P^f\} \cup \{(p_j^f, t_j') \mid p_j^f \in P^f \wedge t_j' \in T'\}$; (iv) $\overline{\ell} = \ell \cup \ell' \cup \{t_j^f \mid t_j^f \in T^f\}$.

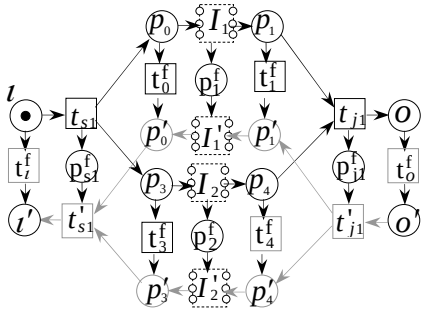


Figure 4. composition compensation paired net

According to the structure of $SCPN$, 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 $I = (P, T, F, \ell)$ in LRTs to $\overline{I} = (\overline{P}, \overline{T}, \overline{F}, \overline{\ell})$.

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 σ in LRTs, if $\exists\sigma \forall I_i, I_j \in \sigma \wedge I_i.TBP, I_j.TBP \in \{Vital, Compensable\}$, one of the transaction dependency $I_j \rangle\rangle^{exd} I_i$, $I_j \rangle\rangle^{inexd}_{I_k} I_i$, $I_j \rangle\rangle^{inexd}_{\sigma'} I_i$, $I_j \rangle\rangle^{imd} I_i$ and $I_j \rangle\rangle^{inimd} I_i$ exists between I_i and I_j, and $I_i.Compensated \wedge I_j.Compensated$, then σ 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 $\exists\sigma \forall I_i, I_j \in \sigma \wedge I_i.TBP, I_j.TBP \in \{Vital, Compensable\}$, and one of the data flow dependency $I_j \rangle\rangle^{exd} I_i$, $I_j \rangle\rangle^{inexd}_{I_k} I_i$, $I_j \rangle\rangle^{inexd}_{\sigma'} I_i$, $I_j \rangle\rangle^{imd} I_i$ and $I_j \rangle\rangle^{inimd} I_i$ exists between I_i and I_j, then I_i is called the TDP of I_j, as shown in Fig. 5. If I_j fails, $\sigma_m (\sigma_m \subseteq \sigma)$ will be executed from I_i. σ_m is called the dependency path of I_j. Each task in LRTs has a specification of TDP.

After TDP is determined, RH executes σ_m in reverse from failed task to try to eliminate the effects brought by committed tasks in recovery scope. For a given LRTs, if I_j ($I_j \in \sigma$) fails (the set of dependency path of I_j is $\{\sigma_{m_1}, \sigma_{m_2}, ..., \sigma_{m_k}\}$), RH rolls back to I_i and injects corrected parameters, executes the compensation tasks of σ_{m_i} ($1 \le i \le k$), after that, I_j is committed successfully. The set of recovery scopes of $\{\sigma_{m_1}, \sigma_{m_2}, ..., \sigma_{m_k}\}$ is $\{\Xi_1, \Xi_2,$

$\cdots,\Xi_k\}$. 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 Ξ_{min}.

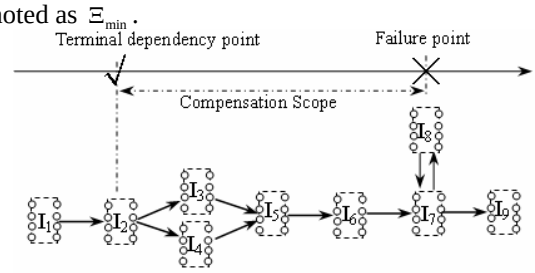


Figure 5. Recovery scope of the failed business process.

B. Nesting of Scope

The nesting description of Ξ is similar to that of control flow structure. As shown in Fig. 6, $S_n[\![S_{n-1}$ denotes S_{n-1} is the sub scope of S_n. The number in subscript indicates the nesting level. In general, $S_n[\![\cdots[\![S_2[\![S_1$ represents that scope Ξ has n nesting levels which includes zero or more tasks.

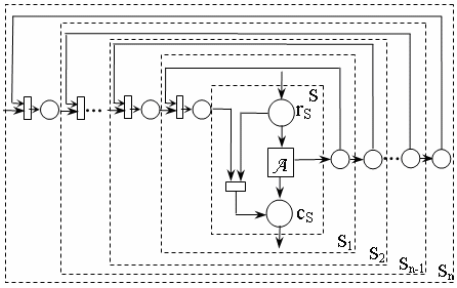


Figure 6. Nesting of scope based on Petri nets.

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 Ξ, fig. 7 depicts the mapping from LRTs to Ξ. 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 Ξ. If the handlers executed successfully, then failure transition is transferred to *Compensated*(*Cpd*) and *Recoveried(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_j throw *fail*() to terminate execution of forward flow. RH is triggered to calculate Ξ and to get the log $(ActID, Desc, QoS, TBP, State, Behavior, \Gamma, In, Out)$ of the successfully executed tasks. It constructs input interface $I_i'.in$ of compensation task according to the input/output interface $(I_i.in, I_i.out)$ of the committed tasks, sets the external state place *Cp* of I_i and fires *Compensate*(), activates and executes I_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 Ξ hold $I_i.TBP \in \{Vital, Compensable\}$, RH gets the corresponding I_i' and constructs the mapping between I_i' and I_i, which is called compensation pair $I_i \div I_i'$.

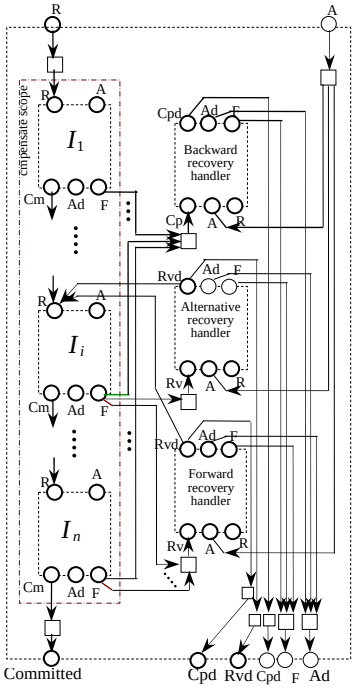


Fig. 1 The recovery model of composition Web transaction.

Figure 7. Recovery model of LRTs.

As Fig. 8 shows, I_i and I_i' both represent atomic transactions with states. Business process and its compensation process are enclosed by dashdotted 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. $\exists\Xi\forall I\in\Xi \mid cpair(I,I')\in(T,T')$, where t_i^n is the transition from I_i to I_{i+1}, t_i^f is the failure transition from I_i to I_i', and $t_i'^c$ is the compensation transition from I_i' to I_{i+1}'. The triple of BCPS is denoted as $(\bar{P},\bar{T},\bar{F})$, where: (i) $\bar{P}=P\cup P'\cup P^f$; (ii) $\bar{T}=T\cup T'\cup T^f\cup T^n\cup T'^c \cup\{\tau,t^r,t^a\}$; (iii) $\bar{F}=F\cup F'\cup F^n\cup F^f\cup F'^c\cup\{(t^a,I_1'.p),\cdots, (t^a,I_n'.p)\}$, where $F^n=(I_i.p,\ t_i^n)\cup(t_i^n,I_{i+1}.p)$, $F^f=(I_i.p,$

$t_i^f) \cup (t_i^f, I_i'.p)$, $F'^c = \{(I_{i+1}'.p, t_i'^c) \cup (t_i'^c, I_i'.p)$, 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 $I_n.Failed$, calculates Ξ of I_i, sends execution request to I_i', get $I_i'.In$ according to $I_i.In$ and $I_i.Out$, 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^f or t_i^f .

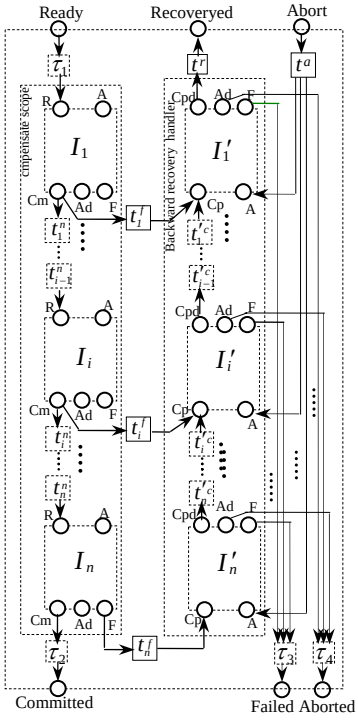


Figure 8. Backward recovery model.

According to all committed tasks and compensation dependency between them, RH calculates Ξ and I' to construct compensation business process. The algorithm of BCPS is given as follows:

Algorithm 1 Construct recovery process.

Input: the set of tasks in business process *BP* , failed task I_ℓ and compensation dependency set *CP*

Output: Compensation Business Process *CBP*

taskstate = *getTaskState*(I_ℓ, *RecordLog*);
if *taskstate* = *Failed* then
  Ξ_1 = *calculateCompensationScope*(I_i); /*calculate the failure recovery scope Ξ_1 of I_ℓ */
else if *taskstate* = *Aborted* then
  Ξ_2 = *calculateCompensationScope*(I_i); /*calculate the abortion recovery scope Ξ_2 of I_ℓ */
else if *taskstate* = *Cancelled* then
  Ξ_3 = *calculateCompensationScope*(I_i); /*calculate the cancellation recovery scope Ξ_3 of I_ℓ */
for each $I_i \in \Xi_1$ or $I_i \in \Xi_2$ or $I_i \in \Xi_3$
  if *binding* (I_i, *ws*) and *getTaskState*(I_ℓ, *RecordLog*) = *Committed* then for committed task I_i in *BP* , $I_i \to I_i'$, $(P, T, F) \to (P', T', F')$;
for each *cpair*(I_i, I_i') and ($I_i \in \Xi_1$ or $I_i \in \Xi_2$ or $I_i \in \Xi_3$)
  if *dep*(I_i, I_i') = *true* then
    Constructing I_i, I_i' and the side $(p_i, t_i^c), (p_i, t_i^f), (t_i^f, p_i'), (p_{i+1}', t_i'^c), (t_i'^c, p_i')$ between adjacent I_i', I_{i+1}' , construct compensation business process *Rflow* ;
return *Rflow*

The complexity of this algorithm depends on the number of tasks in LRTs and the number of compensation tasks of Ξ_1, Ξ_2, Ξ_3 . If $\max(|BP|, |\Xi|) = n$, then the time complexity of this algorithm is $O(n)$.

RH monitors the execution of business process, once failure occurs during execution, RH is triggered to deal with failure recovery. After commit of each task in LRTs, the initial execution condition of its compensation task will be created. If *fail*() is sent by executing I_i , which then transited to *Failed* , RH is triggered to execute compensation. If I_i' has corresponding compensation service, then Compensation Business Process (CBP) is invoked and executed. Meanwhile, abortion or cancellation is executed according to the compensation dependency of I_i' . The compensation handling algorithm is described as follows:

Algorithm 2 The execution of RH.

Input: the set of tasks in *BP* , *Rflow*

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

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*)
  $I_i' \leftarrow$ *ActID, Desc, QoS, TBP, Behavior, State,* Γ*, In, Out log informatin of ws*;
if I_i.*Failed* = *true* then
  trigger *Compensating Handler CH*;
if I_i.*TBP* $\in$ {*Compensable, Vital*} *and* I_i.*state* = *Completed and* I_i *is atomic task* then
  for *each compensation task* I_i'
    CH excute I_i' and record excution information of I_i'
else if I_i is composition task then
  {*divide task* I_i' *into subtask* $I_{i1}', I_{i2}', \cdots, I_{ik}'$;
    for each subtask $I_i'(\ell$ *from* i_1 *to* $i_k)$
      excute(I_ℓ');}
else if *compensate*(I_i) = *false* and I_i is atomic task then
  Forward I_i' , compensation continue;
if *notExit*(*CBP*) then
  Compensation flow CBP is not exist;
for each task I_i
  if $I_j \rangle\rangle^{Abt} I_i$ and I_i.*state* = *running* then
    {*CH send abort*() *to* ws_i ;
      set task $I_i's$ state as "*Aborted*";}
for each task I_i
  if $I_j \rangle\rangle^{Cnl} I_i$ and I_i.*state* = *active* then
    {*CH send cancel*() *to* ws_i ;
      set task $I_i's$ state as "*cancelled*";}
for each task I_i
  if $I_j \rangle\rangle^{Cpt} I_i$ then
  Compensate task I_i', set I_i''s state as "*cancelled*";
if allcompensation(*CBP*)=true then
  return success
else
  return failure;
}

For $|Rflow| = n, Edge(I_\ell) = m, I_\ell \in Rflow$, where $|Rflow|$ and $Edge(I_\ell)$ is the number of tasks and edges in CBP respectively, then the time complexity is $O(mn)$.

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_j, TDP of I_i is calculated and Ξ is determined. Similar to BCPS, a triple $(\vec{P},\vec{T},\vec{F})$ is used to represent Forward Compensation Paired Strategy (FCPS), where: (i) $\vec{P}=P\cup\bar{P}\cup P^f\cup\{p^r\}$; (ii) $\vec{T}=T\cup\bar{T}\cup\{t^r\}$; (iii) $\vec{F}=F\cup\bar{F}\cup\{(t_n^{rc},p^r),(t_{n-1}^{rc},p^r),\cdots,(t_1^{rc},p^r)\}\cup\{(p^r,\tau_2)\}$, as shown in Fig. 9.

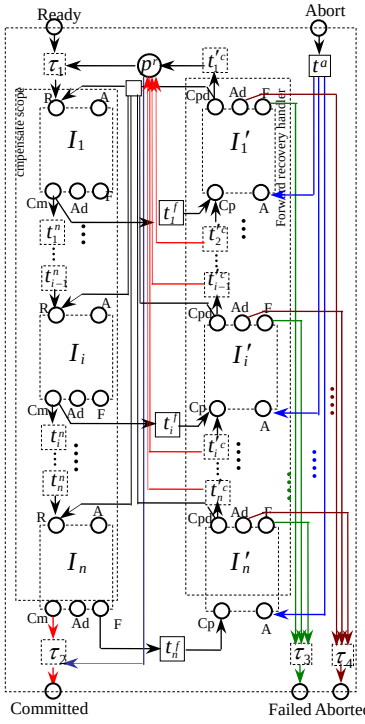


Figure 9. Forward Recovery Model.

RH constructs FCPS in 3 steps: Firstly, it terminates the execution of LRTs and determines the TDP and Ξ of I_i; secondly, it executes recovery; finally, it injects proper parameters at I_i and restarts the LRTs from I_i. Note that if TDP is I_1, there is no difference between BCPS and FCPS during recovery.

Algorithm 3 Forward recovery algorithm

Input: failure task I_ℓ, log, *BP*

Output: the information of success or failure of the recovery

{*taskstate* = *getTaskState*(I_ℓ, *RecordLog*);
if *taskstate* = *Failed* then
Ξ_1 = *caculateCompensationScope*(I_i); /*calculate the failure recovery scope Ξ_1 of I_ℓ */
if *taskstate* = *Aborted* then
for *each task* I_k.*state* = *Running*
if $I_\ell\rangle\rangle^{Abt} I_k$ and $I_k\in\Xi_1$ then
Interrupte each task I_k *in runningflow*;
for *each* $I_\ell\in\Xi_1$
{if *getTaskState*(I_ℓ, *RecordLog*) = *committed* then
{*LogItem* = *seek*(*RecordLog*, I_ℓ)
input the value of the LogItem to task I_l and *compensate* I_l;
change the value of the LogItem to task I_l and *re-excute* I_ℓ;
}}
if I_ℓ.*state* = *Failed* or I_ℓ.*state* = *Cancelled* or I_ℓ.*state* = *Aborted* then
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_j fails, TDP of I_j is calculated and Ξ is determined. If there exists $\{I_j\}=\{I_i\}=\Xi$, then 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_j'' of I_j, 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_j, TDP of I_j is calculated and Ξ is determined. If $\exists\Xi\forall I_j\in\Xi\,|\,I_i=I_j$, then alternative task $I''=(P'',T'',F'')$ and alternative state place p'' is added. ARS can be denoted as a triple $(\vec{P},\vec{T},\vec{F})$, where: (i) $\vec{P}=P\cup P''$; (ii) $\vec{T}=T\cup T''\cup T^f\cup T^r\cup T^a\cup\tau$; (iii) $\vec{F}=F\cup F''\cup\{(I_i.p,t_i^f),(t_i^f,I_i''.p),(t_i^r,I_i''.p),(I_i''.p,t_2^r)\}$, as shown in Fig. 10.

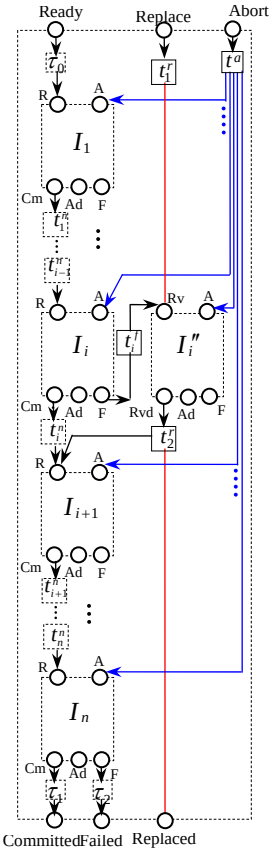


Figure 10. Alternative Recovery Model.

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).

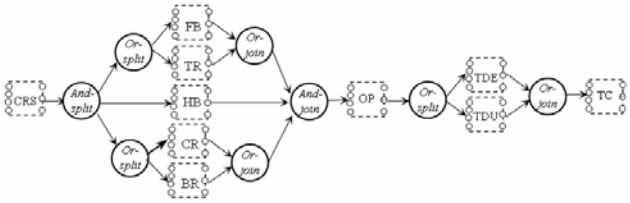


Figure 11. Traveling reservation process

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.

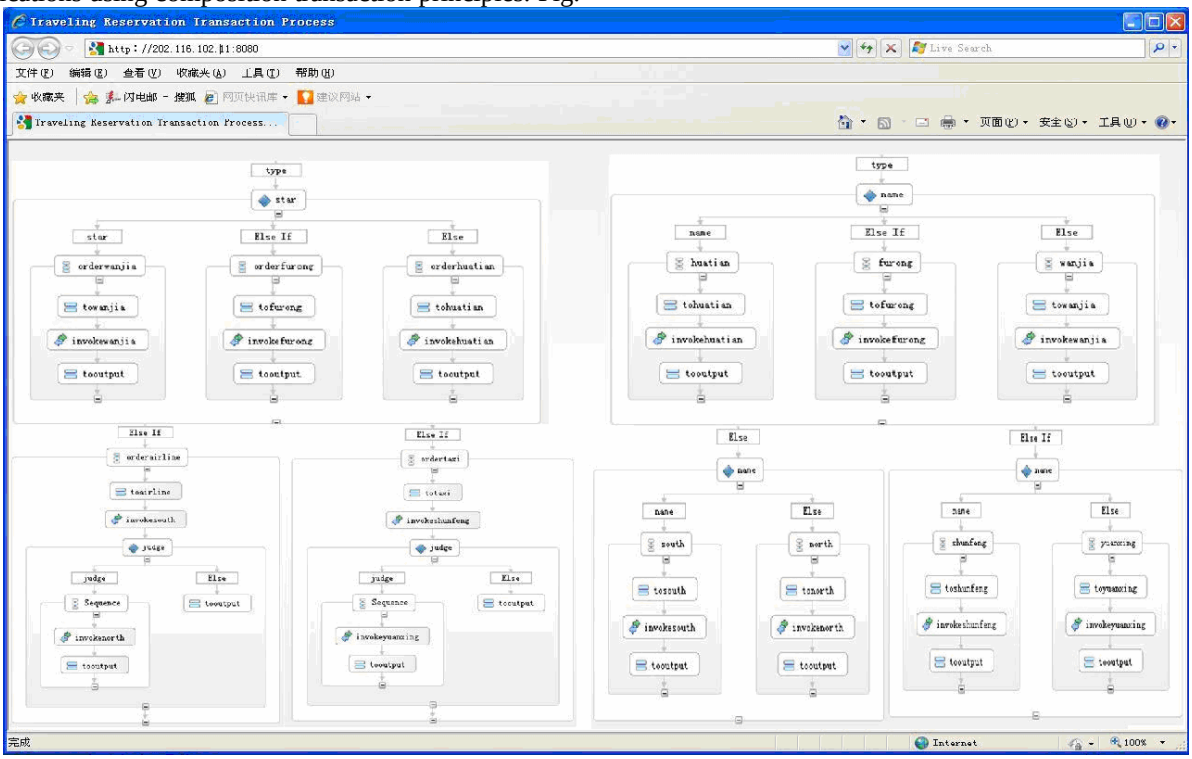


Figure 12. The Implementation of TRP with LCFR

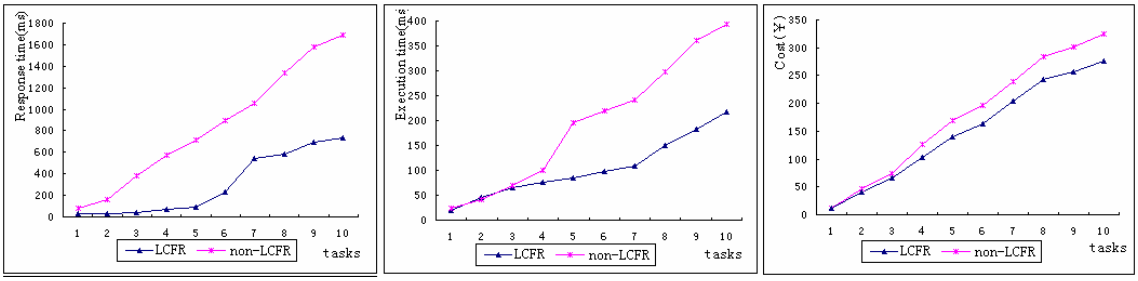


Figure 13. Comparison between *LCFR* and *non-LCFR*

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.

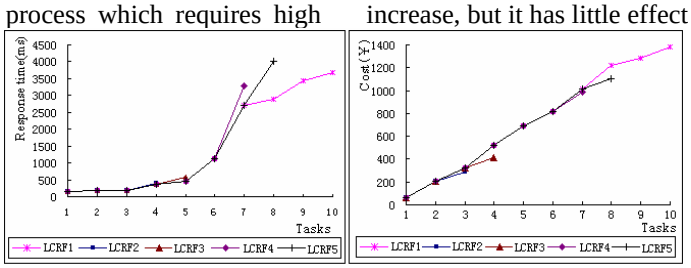


Figure 14. Comparison of *LCFR* with different failure rate

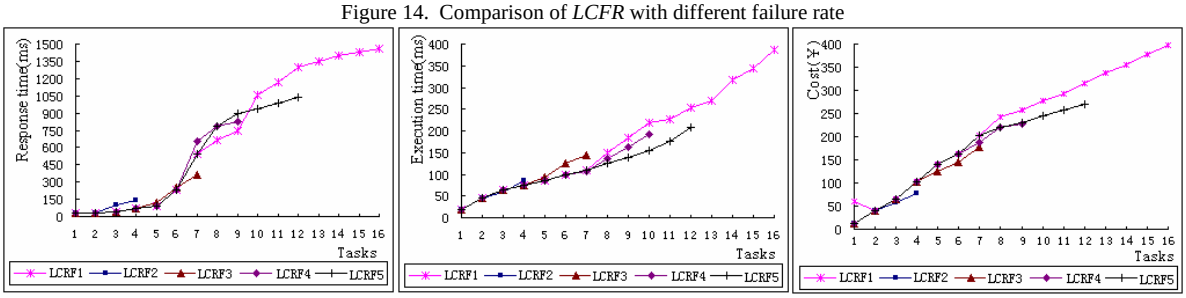


Figure 15. Comparison of response time, execution time and cost with *LCFR*

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 the 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 protocol 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. Garcıa-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. Benatallan 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 (200804401 21), the Natural Science Foundation of Province (06017089, 60940033), the Science and Technology Planning Project of Hunan Province (2010GK3020).

References

[1] N.B. Lakhal, T. Kobayashi, and H. Yokota, "FENECIA: failure endurable nested-transaction based execution of composite Web services with incorporated state analysis," *International Journal on Very Large Data Bases*, Vol.18, No.1, pp.1-56, 2009.

[2] N.B. Lakhal, T. Kobayashi, and H. Yokota, "THROWS: An architecture for highly available distributed execution of web services compositions," *In proceding of International Workshop on Research Issues on Data Engineering*, pp.103-110, 2004.

[3] N.B. Lakhal, T. Kobayashi, and H. Yokota, "WS-SAGAS: Transaction model for reliable web services composition specification and execution," *DBSJ Letters*. Vol.12, pp.17-20, 2001.

[4] Y. Ren, Q.Y. Wu, Y. Jia, J.B. Guan, "An efficient hierarchical failure recovery algorithm ensuring semantic atomicity for workflow applications," *Advances in Web-Age Information Management*, LNCS 3129, pp.664-671, 2004.

[5] K. Wiesner, R. Vacul, M. Kollingbaum, and Katia Sycara, "Recovery mechanisms for semantic Web services," Distributed Applications and Interoperable Systems, *LNCS 5053*, pp.100-105, 2008.

[6] N.L. Minh, and J.L. Cao, "Flexible and semantics-based support for Web services transaction protocols," Advances

in grid and pervasive computing, *LNCS 5036*, pp.492-503, 2008.

[7] S. Bhiri, K. Gaaloul, O. Perrin, and Claude Godart, "Overview of transactional patterns: combining workflow flexibility and transactional reliability for composite Web services," Business process management, *LNCS 3649*, pp.440-445, 2005.

[8] S. Bhiri, O. Perrin, and C. Godart, "Ensuring required failure atomicity of composite Web services," *In proceding of International world wide Web conference committee*, Japan, pp.138-147, 2005.

[9] S. Rinderle, S. Bassil, and M. Reichert, "A framework for semantic recovery strategies in case of process activity failures," *In proceding of International conference on enterprise information systems*, pp.136-143, 2006.

[10] M. Reichert, S. Rinderle, U. Kreher, and P. Dadam, "Adaptive process management with ADEPT2," *In proceding of International conference on data engineering*, pp.1113-1114, 2005.

[11] Z.H. Yang, and C.F. Liu, "Implementing a flexible compensation mechanism for business processes in Web service environment," *In proceding of International Conference on Web Services*, Salt Lake City, pp.753-760, 2006.

[12] L. Lin, and F. Liu, "Compensation with dependency in Web services composition," *In proceding of International conference on next generation Web services practices*, Seoul, Korea, pp.183-188, 2005.

[13] M. Schafer, P. Dolog, and W. Nejdl, "An environment for flexible advanced compensations of Web service transactions," *ACM transactions on the Web*, Vol.2, pp.1-36, 2008.

[14] M. Schafer, P. Dolog, and W. Nejdl, "Engineering compensations in Web service environment," *In proceding of the International conference on Web Engineering*, LNCS4607, pp.32-46, 2008.

[15] Y. Kim, and J. Kim, "Allowing user-specified failure handling in Web services composition," *In proceding of the 2nd internationl conference on ubiquitous information management and communication*, pp.452-458, 2008.

[16] R. Bruni, G. Ferrari, H. Melgratti, and U. Montanari, "From theory to practice in transactional composition of Web services," *EPEW 2005 and WS-FM 2005*, LNCS 3670, pp.272-286, 2005.

[17] G. Dobson, "Using WS-BPEL to implement software fault tolerance for Web services," *In Proceedings of the EUROMICRO Conference on Software Engineering and Advanced Applications*, Dresden, Germany, pp.126-133, 2006.

[18] S. Hinz, K. Schmidt, and C. Stahl, "Transforming BPEL to Petri Nets," Business process mnagement, *LNCS 3649*, pp.220-235, 2005.

[19] M. Kovacs, D. Varro, and L. Gonczy, "Formal modeling of BPEL workflows including fault and compensation handling," *In Proceedings of the 2007 workshop on engineering fault tolerant systems*, 2007.

[20] W.L. Dong, H. Yu, and Y.B. Zhang, "Testing BPEL-based Web service composition using high-level Petri Nets," *In Proceedings of the Enterprise distributed object computing conference*, 2006.

[21] X.G. Deng, Z.Y. Lin, W.Q. Cheng, R.L. Xiao, L.N. Fang, and L. Li, "Modeling Web Service Choreography and Orchestration with Colored Petri Nets," *In Proceedings of International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/ Distributed Computing*, pp.838-843, 2007.

[22] H.Y. Sun, and J. Yang, "Exploiting CoBTx-Net to verify the reliability of collaborative business transactions," *In Proceedings of IEEE asia-pacific services computing conference*, pp.415-422, 2007.

[23] 23. H.Y. Sun, and J. Yang, "BTx-Net: a token based dynamic model for supporting consistent collaborative business transactions," *In Proceedings of IEEE International Conference on Services Computing*, 2007.

[24] L. Garcıa-Banuelos. "An asmL executable model for WS-BPEL with orthogonal transactional behavior," *Business process management*, LNCS 4102, pp.401-406, 2006.

[25] R. Hamadi, *Formal composition and recovery policies in service-based business processes*. PhD thesis, The University of New South Wales, Sydney, Australia, 2005.

[26] R. Hamadi, and B. Benatallah, "Recovery nets: towards self-adaptive workfow systems," *In Proceedings of International Conference on Web Information Systems Engineering*, LNCS 3306, pp.439-453, 2004.

[27] R. Hamadi, B. Benatallah, and B. Medjahed, "Self-adapting recovery nets for policy- driven exception handling in business processes," *Distribute parallel databases*, Vol.23, pp.1-44, 2008.

[28] R. Hamadi, and B. Benatallah, "Dynamic restructuring of recovery nets," *16th Australasian Database Conference*, pp.37-46, 2005,.