A Compensation Paired Net-based Refinement Method for Web Services Composition

Xiaoyong Mei, Aijun Jiang, Shixian Li, Changqin Huang, Xiaolin Zheng, Yiyan Fan

Abstract

With the development of advanced Internet technology, business applications across multiple enterprises based on Web Services Composition (WSC) paradigm are widely used. When system failure occurs, some tasks of the flow may be committed while others unscheduled, in this situation, it is important to accurately analyze execution log of composition business process. In this paper, we formally defines Web services composition compensation based on paired net, and analyze execution logs, dataflow dependency and behavior dependency between tasks in composition process, and discuss the refinement for Web services composition compensation process and the abstract replacement of complex compensation process. The classical composition transaction application Trip Reservation Process (TRP) shows the feasibility of ensuring consistent execution of reliable composition.

Keywords: Web Services Composition, Compensation Paired Net, Execution Log,
Process Refinement

1. Introduction

Refinement of Petri nets is usually a static concept which is used to represent a model at different abstract levels, it is well suited for the hierarchical design of system models. The requirement of dynamic refinement of WSC at runtime is necessary, transition refinement is implemented by a model transformation which allows for dynamic WSC constructing. B. Acu and W. Reisig introduce a refinement method, namely to thin the abstract transition in cursory main business process into more specific sub business process, to obtain more accurate business process [1]. M. Felder and A. Gargantini et al. proposed a theory of implementation and refinement in timed Petri nets and presented the proof that properties are remained before and after refinement [2]. C. Ferreira and M. Butler discussed the process refinement of compensation business process and explicitly embedded the behavior and compensation information of StAC into B to realize the refinement of business process [3]. Focus on different abstract level, M. Kohler and H. Rolke proposed a dynamic transition refinement method and extended it to transition refinement of nets-within-nets [4].

Similar to nested workflow, WSC are often bound to scope, enclosed tasks will be compensated one by one from the innermost level until it terminates successfully. N. B. Lakhal and T. Kobayashi et al. put forward a failure endurable nested-transaction based execution of WSC [5]. U. Greiner and E. Rahm propose a new approach to dynamically handle exception in WSC, exception handling component automatically deals with exceptions [6]. S. Choi and H. Kim et al. put forward a mechanism that ensures the consistent execution of WSC with relaxed isolation [7].

We choose extended Petri nets to model composition transaction for the 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; (iii) It is suitable to represent the 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. H.Y. Sun and J. Yang put forward a Chorographical Business Transaction Net model (CoBTx-Net) based on Hierarchical Colored Petri nets (HCPN) [8]. H.Y. Sun and J. Yang put forward a novel transactional consistency model BTx-Net based on HCPN [9]. M. Kovacs and D. Varro et al. propose a formal modeling technique for BPEL workflows including fault and compensation handling which provides exact semantics [10]. R. Hamadi and B. Benatallah et al. propose a Self-Adaptation Recovery Net (SARN) based on Petri nets, which extends Petri nets model to specify fault or exceptional behavior in business process [11-12].

Based on the previous works, firstly, we formally define Web services composition compensation based on paired net, and analyze execution logs, dataflow dependency and behavior dependency between tasks in WSC. Then we introduce the refinement for WSC compensation process and discuss the abstract replacement of complex compensation process to ensure the reliability and consistency of WSC after composition compensation refinement.

2. Formal definition of WSC based on Petri-net

Execution of WSC has common behaviors: either successfully committed or compensated. Each execution of WSC can be one of the following states: ready, running, suspend or committed. If the WSC is ready, the token is at place $\text{Ready}$. And the committed state implies that the token will be at place $\text{Committed}$ after the execution is committed.

WSC is formal defined as a tuple $WSC = (\text{desc}, \text{URL}, \text{component}, IIC)$, where:

(i) $\text{desc} = \{\text{Name}, \text{ActID}, \text{Desc}, \text{QoS}, \text{TBP}, \text{Behavior}, \text{State}, \text{In}, \text{Out}\}$ represents the service functions and related information description of WSC, where $\text{Name}$ represents the name of the composition service, uniquely identifies a WSC; $\text{ActID}$ denotes the identifier of task; $\text{Desc}$ denotes the functional description of task; $\text{QoS}$ denotes the performance metric of task; $\text{TBP}$ denotes the type of process; $\text{Behavior}$ denotes the process behavior of task; $\text{State}$ denotes the state of task, $\text{State} \in \{\text{activate}, \text{fail}, \text{abort}, \text{cancel}, \text{commit}, \text{compensated}\}$; $\text{In}$ denotes a list of input parameters; $\text{Out}$ denotes a list of output parameters.

(ii) $\text{URL}$ provides URLs of service invocation;

(iii) $\text{component}$ represents a set of service components;

(iv) $\text{IIC} = \{I_1, I_2, \cdots, I_n\}$, $I_i = (P, T, F, \ell)$ represents the formal description of activities of WSC based on Petri-net, as shown in Figure 1, where:

(a) $P = P^0 \cup P^\text{on} \cup P^\text{off} \cup P^\text{input}, P^0$ denotes the finite set of state places of task $I_i, P' \in \{\text{Ready, Activated, Running, Failed, Aborted, Cancelled, Committed, Compensated, Half Compensated}\}$. $P^\text{on}$ denotes the input or output parameters of $I_i$, which refers to functional parameters of Web services. $P^\text{off}$ denotes the QoS parameters of $I_i$ which refers to non-functional metrics of Web services; $P^\text{input}$ denotes finite set of place of control token.

(b) $T = T^n \cup T^b \cup \tau$, where $T^n \cup T^b$ denotes a set of normal handling actions $\{\text{Activate()}, \text{Run()}, \text{Fail()}, \text{Abort()}, \text{Cancel()}, \text{Commit()}\}$ of $I_i$, $T^b$ denotes a set of reverse handling actions $\{\text{Compensate()}, \text{Hcompensate()}, \text{Retry()}\}$ of $I_i$, $\tau \in T$ denotes a silent transition which executes no operation and takes no time, and does not change the execution semantics.

(c) $F = (P \times T) \cup (T \times P)$ denotes a set of directed arcs from $P$ to $T$ or from $T$ to $P$, which is called control flow. The constraint relations on $(t_i, p)$ and $(t_2, p)$ from different actions and states are: $(t_i, p) \prec (t_2, p)$, $(t_i, p) \succ (t_2, p)$ and $(t_i, p) = (t_2, p)$. 

A Compensation Paired Net-based Refinement Method for Web Services Composition
Xiaoyong Mei, Aijun Jiang, Shixian Li, Changqin Huang, Xiaolin Zheng, Yiyan Fan
Advances in Information Sciences and Service Sciences. Volume 3, Number 4, May 2011

- 170 -
A Compensation Paired Net-based Refinement Method for Web Services Composition
Xiaoyong Mei, Aijun Jiang, Shixian Li, Changquin Huang, Xiaolin Zheng, Yiyan Fan
Advances in Information Sciences and Service Sciences. Volume 3, Number 4, May 2011

Figure 1. Formal description of $I_1$ based on Petri-net

For each WSC, the place $t$ is an initial token of $WSC$ based on Petri-net. When the token is at place $t$, $WSC$ begins to execute; When it arrives at place $o$, $WSC$ is completely executed.

Take the classical TRP for example, which has the following possible business behaviors, as shown in Figure 2. Firstly, traveler makes a Customer Requirements Specification (CRS: $I_1$) according to his traveling plan and submits it to travel agent. Then, according to CRS, traveler negotiates about Travel Route (TR: $I_2$), travel agent carries out Flight Booking (FB: $I_3$) which is refined into reserving airline ticket ($I_{31}$) and purchasing insurance ($I_{32}$) or Train Reserving (TR), Hotel Booking (HB: $I_4$) which is refined into reserving hotel room ($I_{41}$) and depositing valuables ($I_{42}$), Transport reservation (TR: $I_5$) which is refined into Car Renting (CR: $I_{51}$) or Bus Renting (BR: $I_{52}$) sequentially. Then, after the reservations are confirmed, traveler finishes Online Payment (OP: $I_6$). Finally, Express Delivery (ED: $I_7$) carries out Ticket Delivery with EMS (TDE: $I_{71}$) or UPS (TDU: $I_{72}$), and sends “reservation successful” message to traveler to make Traveler Confirmation (TC: $I_8$).

Figure 2. Composition Traveling Reservation Process.

3. The composition compensation paired net for WSC

To deal with failure compensation, during the normal execution of WSC, logs are created to maintain the dynamic information of the current active process and its sub-process. The active process here means the process that has already begun to execute but its parent process hasn’t be committed or aborted. Log record maintains an item for each active task or committed task, and all sub-process log of execution progress.

Given atomic tasks $I_1 = (P_1, T_1, F_1, \ell_1), I_2 = (P_2, T_2, F_2, \ell_2), \ldots, I_n = (P_n, T_n, F_n, \ell_n)$, suppose that $T'$ and $P'$ are the sets of failed transitions and uncertain stated activating places respectively such that for each $p_i \in P$ there is a unique $t'_i \in T'$ and for each $t_j \in T$ there is a unique $p'_j \in P'$, then the Compensation Paired Net of $I$ is described as $I'_1 = (P'_1, T'_1, F'_1, \ell'_1), I'_2 = (P'_2, T'_2, F'_2, \ell'_2), \ldots, I'_n = (P'_n, T'_n, F'_n, \ell'_n)$ as shown in Figure 3, where: (i) $P = P \cup \{ P' | p' \in P' \land p \in P \} \cup P'$; (ii) $T = T \cup \{ T' | t' \in T' \land t \in T \} \cup T'$; (iii) $F = F \cup (F')^{-1} \cup F'$, where $(F')^{-1} = \{(P' \times T') \circ F_1 \cup (T' \times P') \circ F_2 \}$, $F = \{(p, t') \mid p \in P \land t'_i \in T' \cup \{ t' \in T' \}) \cup \{(t, p') \mid t \in T \land p'_j \in P' \cup \{(p', t') \mid p' \in P' \land t' \in T' \}}$. 

- 171 -
The above definition shows that $I'$ is composed of $I$, compensation transition $T'$, failure places $P'$, failure transition $fT$ and activating place $fP$. According to the structure of $I'$, $P$ 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.

Compensation tasks are colored in grey to distinguish from the normal tasks. The constructed model is similar to the normal process model in term of structure except that the flow relation is reserved. This implies that the execution of compensation service is always in a reserve order of the execution of normal service.

From the formal definition of compensation paired net, the constructing of reverse compensation process depends on the context of normal execution process. To get the compensation behavior context, an effective log file needs to be created for executed service and record the start and end time, the resource required, the input and output data and other information of the tasks during execution.

Process Execution Context Log file (PECLog) records the execution context of the tasks, where the context of each executed task can be represented as a tuple $<PID, ActID, TBP, State, In, Out>$, where $PID$ represents Web service process identifier, $ActID$ represents identifier of task, $TBP$ represents the type of transaction, $State$ denotes the state of task, $In$ and $Out$ represents input and output parameters of task, respectively.

### 4. Execution log description of WSC

Let an execution log trace, which belongs to a WSC, denotes as $\sigma = e_1 e_2 \cdots e_n \in \mathcal{L}^*$, where $e_i$ corresponds to the execution log item. Such as $ActID, Desc, QoS, TBP, State, In, Out$ of sub tasks, $dom(\sigma) = \{1, 2, \cdots, n\}$ is the domain of $\sigma$, where $e_i$ and $e_j$ satisfy the following five properties:

(i) If $\sigma$ is a empty trace, then it has zero element, denoted as $|\sigma| = 0$;

(ii) $e_i$ is the $i$ th log item of $\sigma$, $i \in dom(\sigma)$;

(iii) $\zeta(t_1, \cdots, t_j)$ is the action sequence of task execution in $e_i$ if and only if $1 \leq i \leq j \leq n \land t_j = \text{Activate}() \land t_i = \text{Commit}() \land \forall \psi(t_1, \cdots, t_{j-1}) (\zeta \neq \psi)$;

(iv) $\xi \in \psi$ represents $\psi$ is the direct successor of $\zeta$ in the $Log$, and only if there exist a $\sigma=(e_1 e_2 \cdots e_n)$ and two tasks in $\sigma$ have action sequence $\zeta(t_1, \cdots, t_j)$ and $\psi(t_1, \cdots, t_k)$ respectively such that $j < k$ and there does not exist task $\lambda(t_1, \cdots, t_m)$ that satisfies $j < m < n < k$;

(v) $\xi \times \psi$ represents the intersection of $\psi$ and $\zeta$ in the $Log$, if and only if there exist $t_1, t_2, \cdots, t_n$ and two tasks in $\sigma$ have action sequence $\zeta(t_1, \cdots, t_i)$ and $\psi(t_1, \cdots, t_k)$ respectively, where $i < k < j$ or $k < i < l$.

The log records maintains an item for each active task or committed task, and all sub transaction log of an execution progress, such as $ActID, Desc, QoS, TBP, State, Behavior, \Gamma, In, Out$, where $ActID$ denotes the identifier of a task, $Desc$ denotes the functional description of a task, $QoS$ denotes the performance metric of a task, $TBP$ denotes the type of transaction, $Behavior$ denotes the
transaction behavior of a task, $State$ denotes the state of a task, $\Gamma$ is the time stamp of a task in a specific state, so $\Gamma_{\text{state}}$ is a set of $\Gamma_{\text{activate}}$, $\Gamma_{\text{failed}}$, $\Gamma_{\text{aborted}}$, $\Gamma_{\text{cancelled}}$, $\Gamma_{\text{committed}}$ and $\Gamma_{\text{compensated}}$. $In$ denotes a list of input parameters, $Out$ denotes a list of output parameters.

When WSC begins to execute, a log item of an activating task will be created. If $CS$ executes several sub transactions at the same time, then the log items of its sub transactions will be created and be appended to the tail of this log list. When a sub transaction is committed, a $\text{Sub\_Commit}$ record is created. When a sub transaction is aborted, the abortion message is sent to a transaction manager to abort all its sub transactions, an $\text{Abort}$ record is created after receiving the abortion confirmation messages. When the root transaction is aborted, an $\text{Abort}$ record will be created for each sub transactions and a log list of a sub transaction will be deleted. The execution log generation algorithm is as follows:

```
Algorithm 1 Automatic log generation
Input: states and actions of task $I$
Output: log
{state = getstate(excutingTask);
event = getaction(excutingTask);
while (!$EndExecution(BP)) /* BP is the business process */
{(place.transition) = executeTask(state, event);
switch((place,transition)){
case("Activated","Activate")
RecordLog(ActivatedLogItem); /* LogItem = ActID, Desc, QoS, TBP, Behavior, State, $\Gamma$, In, Out */
case("Running","Run")
RecordLog(RunningLogItem);
case("Failed","Fail")
RecordLog(FailedLogItem);
case("Aborted","Abort")
RecordLog(AbortedLogItem);
case("Cancelled","Cancel")
RecordLog(CancelledLogItem);
case("Committed","Commit")
RecordLog(CommittedLogItem);
break};

nowState = GetTaskNowState(Task I);
prevState = GetPreviousStateLog(Task I);
For each nowState and prevState of Task $I$ in the executing business process
{If (nowState is Failed)
{writeLogRecord(FailedLogItem);
for each task $I$ in Logrecord
if $I$.state = running then
Interrupt executing task $I$;
Transfer prevState to Compensation Handler;}
If (nowState is Committed)/*Delete the previous state before Committed from the $I$'s state log*/
Delete State Log ActivatedLog, RunningLog, FailedLog, CommittedLog, RewriteLogRecord(CommittedLogItem);}
```

During the execution of WSC, logs should be recorded for each task process. Only when the root process of composite Web services is committed, the log is submitted to execution engine. When sub process is committed, log will only record in its private buffer.

Take travel reservation business process for example, the $\text{Committed}$ log items of the successful air tickets reservation are described as follows:
Sequence aggregation pattern 12::

Parallel aggregation pattern 12::

(i) $\bar{P} = I_1 \cup P \cup \cdots \cup I_n \cup P \cup I_1' \cup P \cup I_2' \cup \cdots \cup I_{n-1} \cup P \cup I_n' \cup P \cup I_1 \cup P \cup \cdots \cup I_n'$

(ii) $\bar{T} = I_1, T \cup I_2, T \cup \cdots \cup I_n, T \cup I_1', T \cup I_2', T \cup \cdots \cup I_{n-1}', T \cup I_n', T \cup I_1', \cdots \cup I_n'$

(iii) $\bar{F} = I_1, F \cup I_2, F \cup \cdots \cup I_n, F \cup I_1', F \cup I_2', F \cup \cdots \cup I_{n-1}', F \cup I_n', F \cup I_1', \cdots \cup I_n'$$$
\cup (I_1, t_1'), (I_1', t_1'), (I_2, t_2'), (I_2', t_2'), \cdots (I_n, t_n'), (I_n', t_n') \cup (I_1, t_1'), (I_2, t_2'), \cdots (I_n, t_n')$$$
Parallel aggregation pattern CS ::= I_i \parallel I_j$, where $I_i \parallel I_j (1 \leq i, j \leq n)$ meets the temporal constraint $\text{finishes}(I_i, I_j)$ or $\text{equals}(I_i, I_j)$, only if concurrent execution of $I_i$ and $I_j$ are both successfully committed, $I_i \parallel I_j$ is successfully committed. If $I_i$ or $I_j$ is failed, $I_i \parallel I_j$ will be aborted, as shown in Figure 4(b).

(i) $\bar{P} = I_1 \cup P \cup \cdots \cup I_n \cup P \cup I_1' \cup P \cup I_2' \cup \cdots \cup I_{n-1} \cup P \cup I_n' \cup P \cup I_1 \cup P \cup \cdots \cup I_n'$

(ii) $\bar{T} = I_1, T \cup I_2, T \cup \cdots \cup I_n, T \cup I_1', T \cup I_2', T \cup \cdots \cup I_{n-1}', T \cup I_n', T \cup I_1', \cdots \cup I_n'$

(iii) $\bar{F} = I_1, F \cup I_2, F \cup \cdots \cup I_n, F \cup I_1', F \cup I_2', F \cup \cdots \cup I_{n-1}', F \cup I_n', F \cup I_1', \cdots \cup I_n'$$$
\cup (I_1, t_1'), (I_1', t_1'), (I_2, t_2'), (I_2', t_2'), \cdots (I_n, t_n'), (I_n', t_n') \cup (I_1, t_1'), (I_2, t_2'), \cdots (I_n, t_n')$$$
Selection aggregation pattern CS ::= I_i \circledast I_2 \circledast \cdots \circledast I_n$, where $I_i \circledast I_j (1 \leq i, j \leq n)$ denotes $I_i$ or $I_j$ is executed according to condition $\text{case}(cond)$. $I_i \circledast I_j$ is successfully committed if and only if one of the branches is successfully committed, as shown in Figure 4(c).

(i) $\bar{P} = I_1 \cup P \cup \cdots \cup I_n \cup P \cup I_1' \cup P \cup I_2' \cup \cdots \cup I_{n-1} \cup P \cup I_n' \cup P \cup I_1 \cup P \cup \cdots \cup I_n'$

(ii) $\bar{T} = I_1, T \cup I_2, T \cup \cdots \cup I_n, T \cup I_1', T \cup I_2', T \cup \cdots \cup I_{n-1}', T \cup I_n', T \cup I_1', \cdots \cup I_n'$$$
\cup (I_1, t_1'), (I_1', t_1'), (I_2, t_2'), (I_2', t_2'), \cdots (I_n, t_n'), (I_n', t_n') \cup (I_1, t_1'), (I_2, t_2'), \cdots (I_n, t_n')$
(iii) $\bar{F} = I_1,F \cup I_2,F \cup \cdots \cup I_n,F \cup I'_1,F \cup \cdots \cup I'_n,F \cup \{(t_o,t'_o),(t'_1,o'_1),(t'_2,o'_2),\ldots,(t'_n,o'_n)\} \cup \{(t_o,t_o),(t'_1,t'_1),(t'_2,t'_2),\ldots,(t'_n,t'_n)\}$

Discriminator aggregation pattern $12::$ $\text{CS} \equiv I_1 \Theta I_2 \Theta \cdots \Theta I_n \equiv I_1$, where $(1,i)$ is similar to operator $\|$. A guard function $\text{guard(status)}$ is added to operator $\Theta$ to capture the first committed branch, and ignore the other branches’ actions as shown in Figure 4(d).

(i) $\bar{F} = I_1,P \cup I_2,P \cup \cdots \cup I_n,P \cup I'_1,P \cup \cdots \cup I'_n,P \cup \{(t_o,\theta_1),o'_o,\theta'_o,p'_o,p'_1,p'_2,\ldots,p'_n\} \cup \{p'_1,p'_2,\ldots,p'_n\}$

(ii) $\bar{F} = I_1,T \cup I_2,T \cup \cdots \cup I_n,T \cup I'_1,T \cup \cdots \cup I'_n,T \cup \{(t_o,\theta_1),t'_o,t'_1,t'_2,\ldots,t'_n\}$

(iii) $\bar{F} = I_1,F \cup I_2,F \cup \cdots \cup I_n,F \cup I'_1,F \cup \cdots \cup I'_n,F \cup \{(t_o,t'_o),(t'_1,o'_1),(t'_2,o'_2),\ldots,(t'_n,o'_n)\} \cup \{(t_o,t_o),(t'_1,t'_1),(t'_2,t'_2),\ldots,(t'_n,t'_n)\}$

Iteration aggregation pattern $\text{CS} \equiv I_1||L_1$, where $L_1$ is executed repeatedly according to the times of iteration $\lambda = |L_1|$, as shown in Figure 4(e).

(i) $\bar{F} = I_1,P \cup L_1,P \cup \{p_1,p'_1,p_2,p'_2\}$

(ii) $\bar{F} = I_1,T \cup L_1,T \cup \{t_1,t_2\}$

(iii) $\bar{F} = I_1,F \cup L_1,F \cup \{(p_1,t_1),(t_1,p_2)\} \cup \{(p_1,t'_1),(t'_1,p'_2)\}$

- 175 -
6. Process refinement of WSC based on compensation paired net

The complex business process discussed in this paper is not only the simple connection of the above five aggregations patterns, but also has complex aggregation nesting and involves compensation scope analysis. Therefore, refinement is essential to properly dealing with complex composition compensation process. The basic idea of refinement is to replace the abstract transition in complex business process with logically independent process till the business process is detailed enough.

Given $WSC = (P, T, F)$ and $WSC_c = (P_c, T_c, F_c)$, $WSC_c$ is called the sub net of $WSC$ if the following hold: (i) $P_c \subseteq P$ and $P_c \neq \emptyset$; (ii) $T_c \subseteq T$ and $T_c \neq \emptyset$; (iii) $F_c = F \cap \{(p \times T_c) \cup (T_c \times p)\}$.

Given $WSC = (P, T, F)$ and $WSC_c = (P_c, T_c, F_c)$, $WSC_c$ is the sub net of $WSC$, $WSC$ is called PP-Type subnet if the following hold: (i) $T_c \cup T'_c \subseteq P_c$, where $T_c$ represents the direct predecessor of $T_c$, $T'_c$ represents the direct successor of $T_c$; (ii) $P_c = P$ and $P_c \cup P'_c \subseteq P_c$.

Given a composition compensation Petri net $WSC = (P, T, F)$, if there is an abstract task $I_x$, then replace it with a pp-type sub net $WSC_c = (P_c, T_c, F_c)$, refinement compensation Petri net can be formalized as a tuple $WSC_{ref} = (P_{ref}, T_{ref}, F_{ref})$, where: (i) $P_{ref} = P \cup P_c$; (ii) $T_{ref} = T \cup T_c - \{t_i, t_i'\}$; (iii) $F_{ref} = F \cup F_c \cup \{(p, t) | p \in I_x, t \in \{t_i, t_i', t_e\}, t \in \{t_i, t_i', t_e\}\} \cup \{(t, p) | t \in \{t_o, t_i, t_o\}, p \in t_m, t_m \in \{t_i, t_i', t_e\}\}$.

Each complex structured compensation paired net process can be constructed by basic patterns.
Further more, composition constructs can be nested in other service process to describe more complex business logic. In the composition compensation paired net modeling, the places and transitions of adjacent composition patterns should be merged. Figure 5 shows the refinement of traveling reservation process.

**Algorithm 2** Refinement of composition compensation process

**Input:** Abstract composition compensation process \(Cbp\)

**Output:** Refinement process

1. Derive parameter list of task \(act \in \{a_1, a_2, \cdots, a_n\} \in Cbp\_task\) for each task \(act \in Cbp\_task\)
2. if \(act\_type\) is COMPOSITION then
   \(Act\_Refine = \text{refinementProcess}(act)\);
3. else if \(act\_type\) is AggregationPattern then
   \{ case(SEQUENCE): \(Act\_Refine = \text{SequenceRefinement}(act)\);
   case(PARALLEL): \(Act\_Refine = \text{ParallelRefinement}(act)\);
   case(SELECT): \(Act\_Refine = \text{SwitchRefinement}(act)\);
   case(Loop): \(Act\_Refine = \text{LoopRefinement}(act)\); \}
4. Return refinement process.

Figure 5. Composition Process Refinement.

Time complexity of the algorithm is related to the tasks number \(|BP| = n\) and nesting level of aggregation patterns of business processes, thus the time complexity is \(O(n^2)\).

7. Experiment Results and Analysis

We choose open source tool ActiveBPEL [13] to implement composition compensation mechanism based on paired net, the experiment is carried out on a testbed consisting of 2 IBM x3650 servers and 12 PCs connected by a 100Mbps Ethernet. Each server is equipped with 4 Intel 2930MHz processors and 8GB memory running SuSE 10.2 Linux. Each PCs is equipped with Intel Pentium Dual Core E5200 2.5 GHz, 2GB RAM, Windows XP SP2.
Take TRP for example to discuss the refinement of composition process, as can be seen from Figure 6. To show the effectiveness of the proposed approach, we performed an experimental evaluation. 50 services and composition service templates that including four atomic services are considered, since the approach employed a generation algorithm, it is feasible to execute it in a compensation environment. The result of compensation execution of TRP is shown in Figure 7. The compensation approach locates any conforming compensation service with the dependency reasoning rules.

Travel agent first sets up a cursory process of traveling reservation (main process) and then performs refinement according to business logic (replace the tasks with the sub process indicated by the dashed) till finally forms a detail business process. The refinement and execution of TRP is addressed by algorithm 2 and algorithm 1 as follows:

(i) If all tasks in TRP are successfully executed, namely no failure occurs, then after the execution of $I_1, I_2, \ldots, I_i$, ActiveBPEL engine gets execution log, constructs and loads compensation context of each task and installs Compensation Handler (CH) for the whole process. When users apply for cancellation, CH starts compensation.

(ii) During TRP executing, the failure of any task in composition compensation pattern will terminate the process and activate CH to implement compensation. CH is described as $(I, C, \hat{c}, E)$ where $I$ represents the failure task, $C$ represents the corresponding compensation task of $I$, $\hat{c}$ represents the compensation task context constructed according to execution log, and $E$ represents the event activated by compensation. During process executing there are three possible situations:

   (a) Sequence task $I_1 \oplus I_2$ fails. According to the compensation execution semantics of $I_1 \oplus I_2$, $\text{CSCP}_{\text{sequence}}$’s compensation context $\hat{c}$ is loaded. CH will construct the mapping from $I_1, I_2$ to their corresponding compensation task $I'_1, I'_2$ respectively. CH is triggered and $I'_2$ and $I'_1$ will be executed.

   (b) Parallel task $I_1 \parallel I_2$ fails, namely either $I_1, I_2$ or all fail, then the compensation tasks $I'_1, I'_2, I'_3$ are loaded. If $I_1 \parallel I_2 \parallel I_3$ is executing, then its executing task is immediately aborted and $CH(I_1 \parallel I_2 \parallel I_3 \parallel I'_1 \parallel I'_2 \parallel I'_3, \hat{c}_{\text{sequence}}, \text{compensate}())$ is triggered according to $\text{SCP}N_{\text{paralle}}$ composition compensation strategy. CH constructs compensation context $\hat{c}_{\text{hotelbook}}$ or $\hat{c}_{\text{hotelorder}}$ or both and executes $\text{SCP}N_{\text{paralle}}$ composition compensation. As $I_1 \parallel I_2 \parallel I_3$ is executed after $I_1 \oplus I_2$, if $I_1 \parallel I_2 \parallel I_3$ is compensated, then $I'_1 \parallel I'_2 \parallel I'_3$ should be performed.
A Compensation Paired Net-based Refinement Method for Web Services Composition
Xiaoyong Mei, Aijun Jiang, Shixian Li, Changqin Huang, Xiaolin Zheng, Yiyan Fan
Advances in Information Sciences and Service Sciences. Volume 3, Number 4, May 2011

Figure 7. Execution Result of TRP

(c) The execution semantics of $I_a || I_b$ can be analyzed combining that of parallel and selection aggregation patterns, either $I_a || I_b$ or both fail, then the compensation tasks $I_a' || I_b'$ are loaded.

(d) Selection task $I_{ab} \otimes I_{bc}$ fails. The execution engine ActiveBPEL gets $I_{ab} \otimes I_{bc}$ compensation conditions to determine compensation context of branch $c_1$ or $c_2$ and performs corresponding compensation. Since the composition compensation is automatic reverse compensation, compensation should be performed on $I_a || I_b$, $I_a || I_b || I_c$, and $I_a \otimes I_c$, sequentially.

In real world, TRP is composed by normal business process and failure recovery process. Combining the compensation pair of partner tasks, extended TRP can be described as $(CRS+CRS') \oplus (TA+TA') \otimes ((FB+FB') \oplus (HB+HB')) \otimes (CR+CR') \otimes (BR+BR') \oplus (OP+OP') \otimes (TD+TD') \otimes (TDE+TDE') \otimes (TDU+TDU')) \oplus TC$. Corresponding compensation action pair of $I + I'$ is denoted as $I.action + I'.action$, where $I.action$ and $I'.action$ represent the actions of $I$ and $I'$ respectively, in particular, $\varnothing$ is dummy action. Hence, execution semantic of compensation pairs related to partner travel agent can be described as follows:

$CRS.SendRequest() + CRS.UndoTravelRequest()$, where $CRS.SendRequest()$ represents that CRS submits a travel request and $CRS.UndoTravelRequest()$ represents that CRS cancels the travel request;

$TA.NegotiateRouting() + TA.UndoNegotiateRouting()$, where $TA.NegotiateRouting()$ represents that TA negotiates a travel routing, and $TA.UndoNegotiateRouting()$ represents that TA cancels travel routing;

$TR.ReserveTrainTickets() + TR.UndoReserveTrainTickets$, where $TR.ReserveTrainTickets()$ represents that TR reserves train tickets, and $TR.UndoReserveTrainTickets$ represents that TR cancels train tickets reservation;

$FB.ReserveAirlineTickets() + FB.UndoReserveAirlineTickets()$, where $FB.ReserveAirlineTickets()$ represents that FB reserves airline tickets, and $FB.UndoReserveAirlineTickets()$ represents that FB cancels airline tickets reservation;

$HB.RoomsReservation() + HB.UndoRoomsReservation()$, where $HB.RoomsReservation()$ represents that HB makes room reservation, and $HB.UndoRoomsReservation()$ represents that HB cancels rooms reservation.
reservation;

\( CR.CarRenting() + CR'.UndoCarRenting() \), where \( CR.CarRenting() \) represents that \( CR \) rents a car, and \( CR'.UndoCarRenting() \) represents that \( CR' \) cancels car renting;

\( BR.BusRenting() + BR'.UndoBusRenting() \), where \( BR.BusRenting() \) represents that \( BR \) rents bus, and \( BR'.UndoBusRenting() \) represents that \( BR' \) cancels bus renting;

\( OP.OnlinePayment() + OP'.UndoOnlinePayment() \), where \( OP.OnlinePayment() \) represents that \( OP \) makes online payment, and \( OP'.UndoOnlinePayment() \) represents that \( OP' \) cancels online payment;

\( TDE.EMSDelivery() + TDE'.UndoEMSDelivery() \), where \( TDE.EMSDelivery() \) represents that \( TDE \) makes EMS delivery request, and \( TDE'.UndoEMSDelivery() \) represents that \( TDE' \) cancels EMS delivery;

\( TDU.UPSDelivery() + TDU'.UndoUPSDelivery() \), where \( TDU.UPSDelivery() \) represents that \( TDU \) sends UPS delivery request, and \( TDU'.UndoUPSDelivery() \) represents that \( TDU' \) cancels UPS delivery.

8. Conclusion and future work

WSC compensation is a hot issue in the field of reliability and integrity of Service Oriented Computing (SOC), and refinement is an effective method to implement the hierarchical modeling of complex composition process. This paper proposes a novel approach of WSC compensation and its refinement based on paired net, which discusses how to replace the abstract transitions in the cursory process with sub net and finally constructs the detailed composition compensation process.

Our future work includes three aspects: (i) give more details about composition business based on previous works; (ii) study semantic recovery matching technique based on multi-agent system; (iii) keep a reasonable size of state space, and improve decision capability of failure recovery to reduce human intervention.

9. Acknowledgements

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 (2010GBK 3020).

10. References