Specifying and Verifying Web Transactions *

Jing Li, Huibiao Zhu, and Jifeng He

Shanghai Key Laboratory of Trustworthy Computing
East China Normal University
Shanghai, China, 200062
E-mail: {jli, hbzhu, jifeng}@ sei.ecnu.edu.cn

Abstract. New evolving internet technologies are extending the role of the World Wide Web from a platform of information exhibition to a new environment for service interactions. While new business opportunities are brought in under this new era of internet, novel challenges are coming out at the same time. Current technologies have been found lacking efficient support for web transactions. Because transactions in the context of web services have distinct features, such as autonomous and interactive, the traditional automatic mechanisms of resource locking and rollback are proved to be inappropriate. For this reason, we suggest that web transactions are constructed through a series of compensable transactions, using the concept of compensation to ensure a relatively relaxed atomicity. This paper formally expresses the composition structures and behavioral dependencies of compensable transactions. Based on the formal description for a transaction model, we are able to further verify its transactional behavior according to the specified requirement of relaxed atomicity and more precise behavioral properties with temporal constraints.

1 Introduction

Web Services are becoming the current most promising paradigm for enabling business interactions through the Internet. They have greatly influenced the way for application development. Web services can be regarded as computational entities, generally independent and autonomous. They are driven by XML related technologies which make services able to be described, discovered and invoked across the internet. Based on more and more accessible services over the Web, there is an opportunity to provide new value-added services to customers by combining individual services. Therefore, extensions of web service technologies have been considered. Several proposals for describing web service composition (XLANG, WSFL, BPEL and WSCDL) have been already put forward.

When a composite web service combines several existing services to complete a given task, a web transaction is required to orchestrate the loosely coupled services into a unit of work so as to guarantee a reliable execution. However, setting

* Supported by National Basic Research Program of China (No.2005CB321904), National High Technology Research and Development Program of China (No.2007AA010302), National Natural Science Foundation of China (No.90718004).

up an efficient web transaction is not a trivial task. First of all, web transactions usually require a long time to complete, due mainly to lengthy computation and pause for input from users, which may cause severe performance problems. Such long lived transactions greatly increase the contention for resources and finally lead to more deadlocks. Secondly, for a web transaction, its participating services may belong to different and even competing companies such that services are generally independent and autonomous. In such systems, there is no chance to intentionally block the resources residing in other services. Lastly, web transactions usually involve communications between companies and human beings, where the outcome of interactions cannot be physically undone. Thus, the pure rollback mechanism is not applicable at all. Therefore, the key features of traditional transactions become impracticable for web transactions. To solve this problem, the degree of atomicity needs to be relaxed. A weaker notion of atomicity based on compensation has been proposed to recover from failure.

The notion of compensation has its root to the seminal work of Sagas [8] which is one of the first proposals for extended transactional models. A saga is a long lived transaction which is partitioned into a set of sub-transactions. When a sub-transaction completes, it commits prior to the completion of the whole saga. This enables resources to be released earlier and thus reduces the possibilities of deadlock. However, partial effects have been exposed to the outside. In this case, each sub-transaction is associated with a compensation whose responsibility is to semantically undo the effect of its sub-transaction. Thus, whenever a sub-transaction aborts in the middle, the partial results made by the committed sub-transactions will be removed by executing their compensations.

This paper suggests to construct a web transaction in terms of compensable transactions. A compensable transaction is a new type of transaction whose effect can be semantically undone even after it has committed. Basically, a compensable transaction consists of two parts: forward flow and compensation flow. The forward flow describes the required normal logic according to system requirements, while the compensation flow is properly defined so as to undo the effect of its forward flow semantically. Unlike the traditional ACID transactions, a compensable transaction has distinct features. Its features allow web transactions to incorporate different transactional semantics as well as different behavioral dependency patterns. This provides a manner to enhance the flexibility and reliability of web transactions. In order to provide a precise description, we adopt the novel transactional language t-calculus [14] to model the composition structure of compensable transactions. Moreover, the behavioral semantics is precisely defined for different transactional constructs.

Compensation is a kind of backward recovery in the presence of failure. In order to promote the possibility of successful completion, our transaction model also supports forward recovery so as to survive the failure caused by a certain sub-transaction. In this case, there is no need to obey the strict requirement of all-or-compensated. The notion of *acceptable termination states* (ATS) is used to express the relaxed atomicity requirement. Given a web transaction, we are able to verify its behavior according to the specified ATS. Further, we can also

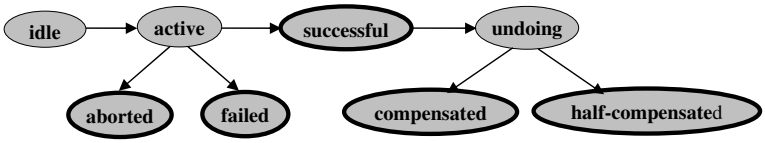

Fig. 1. The state transition diagram of compensable transactions

help to find inconsistencies if its behavior is proved to be invalid. In addition, depending on particular applications for which web transactions are designed, additional properties capable of expressing temporal constraints would be required. Here, we propose a specification language to specify this kind of transactional properties. The corresponding process for verifying whether these properties are satisfied is formally presented subsequently.

This paper is organized as follows. Section 2 gives a detailed description about compensable transactions whose execution structure is clearly depicted by a state transition diagram. The transactional model with the corresponding language is presented in Section 3, where related behavioral dependencies are formally specified. Section 4 presents strict solutions to the verification of web transactions according to the specified requirements. We discuss some related work in Section 5 and conclude this paper at last.

2 Compensable Transaction

Compensable transactions are the basic constituents for building web transactions in our approach. A compensable transaction aims to do a specific task as part of a business goal. Different from traditional transactions, a compensable transaction can withdraw its result after its commitment in case of error. On the transaction level, we do not model its internal operations but only deal with its external visible aspects. We adopt the *state transition diagram* to describe its behavior structure by providing a set of execution states and a set of transitions between these states. The Figure 1 shows the state transition diagram for an arbitrary compensable transaction.

A compensable transaction has eight states and five of them are terminal states shown by bold ellipses in Figure 1. Initially, its state is marked as *idle* and it becomes *active* when it is arranged for execution. Once it is active, it can be either *aborted* or *failed* due to the presence of failure, or it finally achieves its objective and leads to the *successful* state. Whenever some wrong has happened during its execution or within other concurrent transactions, it tries to erase its partial outcome as soon as possible. If all the partial effects have been erased, it becomes aborted as if nothing has ever been done. Otherwise, it turns into failed since some partial effect still exists. This case possibly results in data inconsistency. Especially, the effect after completion for compensable transactions does not remain for ever. When it has succeeded, its effect can be semantically undone by properly defined compensation. Once the system decides to undo a successful

compensable transaction, this transaction enters into the *undoing* state. During this period, its associated compensation is executing. The final state is totally dependent on the execution result of its compensation. When the compensation fails in the middle, it is marked as *half-compensated*, otherwise as *compensated*.

Let $\mathcal{T}$ be a finite set of compensable transactions, and Σ be a finite set of transactional states. Here, Σ is defined as:

$$\Sigma = \{idl, act, suc, abt, fal, und, cmp, hap\}$$

where they stand for the fore-mentioned eight states (idle, active, successful, aborted, failed, undoing, compensated and half-compensated). Moreover, we let Δ be the set of terminal states, that is:

$$\Delta = \{suc, abt, fal, cmp, hap\}$$

An *action* a on $\mathcal{T}$ and Σ is a pair $(T, \sigma) \in \mathcal{T} \times \Sigma$, saying the transaction T enters into the state σ. Especially, a *terminal action* is a special kind of action, which is a pair satisfying $(T, \sigma) \in \mathcal{T} \times \Delta$. Actions may follow a specified order or conform to some other policies. Here, we propose five relations to enforce different constraints on the occurrence of actions:

1. $a < b$: only a can fire b.
2. $a \prec b$: b can be fired by a.
3. $a \ll b$: a is the precondition of b.
4. $a \leftrightarrow b$: a and b both occur or both not.
5. $a \nleftrightarrow b$: the occurrence of one action inhibits the other.

The first three relations specify a kind of order, whereas the last two do not. $a < b$ indicates a must precede b when two actions both occur. In addition, the two actions must both appear or not, same as $a \leftrightarrow b$ in this sense. However, $a \leftrightarrow b$ does not enforce any temporal constraint on actions. $a \prec b$ tells that b can also be fired by other event except for a. In other words, b is able to occur without the previous occurrence of a. $a \ll b$ tells that a must occur earlier whenever b occurs. However, the occurrence of a does not guarantee a following occurrence of b. Finally, $a \nleftrightarrow b$ denotes that the two actions must be mutually exclusive. These relations have some useful and interesting properties. $<$, $\prec$ and $\ll$ are anti-symmetric and transitive. $\leftrightarrow$ is reflexive, symmetric and transitive, while $\nleftrightarrow$ is irreflexive, symmetric and intransitive. Besides, there are more useful properties when these relations are combined.

Law 1. If $a < b$ and $a \leftrightarrow c$ then $b \leftrightarrow c$

Law 2. If $a < b$ and $b \leftrightarrow c$ then $a \leftrightarrow c$

Law 3. If $a < b$ and $b \circ c$ $(\circ \in \{\prec, \ll\})$ then $a \circ c$

Law 4. If $a \circ b$ $(\circ \in \{\prec, \ll\})$ and $b < c$ then $a \circ c$

Law 5. If $a \circ b$ $(\circ \in \{<, \prec, \leftrightarrow\})$ and $b \nleftrightarrow c$ then $a \nleftrightarrow c$

Table 1. Intra-constraints of a compensable transaction

$(T, idl) \ll (T, act)$	$(T, act) \ll (T, suc)$	$(T, act) \ll (T, abt)$	$(T, act) \ll (T, fal)$
$(T, suc) \nleftrightarrow (T, abt)$	$(T, suc) \nleftrightarrow (T, fal)$	$(T, abt) \nleftrightarrow (T, fal)$	$(T, cmp) \nleftrightarrow (T, hap)$
$(T, suc) \ll (T, und)$	$(T, und) \ll (T, cmp)$	$(T, und) \ll (T, hap)$	

Law 6. If $a \circ b$ ($\circ \in \{<, \ll, \leftrightarrow\}$) and $a \nleftrightarrow c$ then $b \nleftrightarrow c$

As for an arbitrary compensable transaction T, all the actions occurring during its execution must satisfy some constraints which are clearly shown in Table 1. For instance, the compensation can only be activated when the transaction T has succeeded, expressed as $(T, suc) \ll (T, und)$. Besides, terminal states must be exclusive, e.g., $(T, suc) \nleftrightarrow (T, abt)$, $(T, cmp) \nleftrightarrow (T, hap)$.

3 The transactional model

Though the structure of a compensable transaction is relatively complex, its distinct properties provide the opportunity for engineers to resolve the performance problem caused by web transactions. In this section, we explain how to build reliable web transactions on the basis of compensable transactions. A novel transactional language t-calculus is adopted to describe the composition structure of compensable transactions. Its syntax is shown below:

$$\begin{aligned} S, T ::= {} & BT \mid S; T \mid S \parallel T \mid S \sqcap T \mid S \otimes T \mid \\ & S \rightsquigarrow T \mid S \trianglerighteq T \mid S \triangleright T \mid S \divideontimes T \\ P ::= {} & \{T\} \end{aligned}$$

The basic transaction BT is the primary block to form a compensable transaction. Generally, BT is composed of two activities, one denotes the forward flow while the other one represents its compensation. More details of basic transactions can be found in [14] but not relevant in this paper. The key point is that compensable transactions support composition, that is, it can be constructed out of simpler ones and still preserves the features of compensable transactions. To deal with specific requirements of web transactions, we incorporate some distinct composition constructs in this transactional model. These new constructs help to build a web transaction with a higher quality which mainly comes from three directions:

- *Flexibility* is enforced by allowing users to provide functionally equivalent sub-transactions for a given objective. These equivalent transactions may have different or even priorities. Transactions with even priorities can be arranged to run in parallel, expressed as $S \otimes T$. Otherwise, the transaction with higher priority must be executed first, expressed as $S \rightsquigarrow T$. Note that if one of these transactions completes successfully, the objective is achieved.
- *Reliability* is enhanced by properly dealing with partial compensations. We have shown in Figure 1 that partial compensation will make a compensable

transaction into the *half-compensated* state in which data consistency does not hold at all. In order to keep the whole system consistent, partial compensation needs further treatment which is referred as *exception handling*. $S \trianglerighteq T$ and $S \triangleright T$ are two kinds of proposed mechanisms for exception handling.

– *Specialization* is added by offering specialized compensations for specific applications. Basically, the compensation of a composite transaction is constructed by the accumulation of those of its sub-transactions. Concerning the requirement of a concrete application, it is more satisfactory for developers to directly define an appropriate compensation, expressed as $S \circledast T$.

Finally, we use $\{T\}$ to stand for a complete web transaction. Once a web transaction completes, its effect holds for ever. An aborted web transaction looks like doing nothing from the view of an external observer. A failed web transaction causes some loss due to its inconsistent status. Our transactional model can not avoid inconsistency, but it helps to reduce the possibilities of such failure.

3.1 Behavioral Dependencies

In t-calculus, each composite construct stipulates distinct behavioral dependencies between compensable transactions. It specifies how simpler compensable transactions are coupled and how the behavior of a certain compensable transaction influences the behavior of the other. Further, it specifies how the behaviors of sub-transactions determine the behavior of their composite transaction. In the following, we formally describe the behavioral dependencies for all the transactional constructs.

Sequential composition: $S;T$ arranges the first transaction S to be executed first. Only when S finishes its task, the following transaction T will be activated to run. However, whenever T is aborted or compensated, the completed transaction S would be compensated so as to remove all the partial effects. The above description reflects the following behavioral dependencies:

$$(S, suc) \prec (T, act) \qquad (;1)$$
$$(T, abt) \prec (S, und) \qquad (;2)$$
$$(T, cmp) \prec (S, und) \qquad (;3)$$

Recall that any compensable transaction satisfies some internal constraints (Table 1), e.g., $(S, suc) \nleftrightarrow (S, abt)$ and $(T, act) \ll (T, suc)$. By using $(;1)$ and Law 6, we have that $(T, act) \nleftrightarrow (S, abt)$. Then using Law 6 and the symmetry property of $\nleftrightarrow$, we get that $(S, abt) \nleftrightarrow (T, suc)$ which says the abortion of S would never lead to the success of T. Similarly, we can deduce more dependencies as follows:

$$(S, suc) \ll (T, \alpha) \quad \alpha \in \Delta \qquad (;4)$$
$$(S, \alpha) \nleftrightarrow (T, \beta) \quad \alpha \in \{abt, fal\},\ \beta \neq idl \qquad (;5)$$
$$(S, \alpha) \nleftrightarrow (T, \beta) \quad \alpha \in \{cmp, hap\},\ \beta \in \{fal, hap\} \qquad (;6)$$

The last item $(;6)$ tells that if T is failed or half-compensated, the compensation of the former transaction S will never be enabled. Now we are going to investigate

the relationship between the composite transaction and its sub-transactions. A composite transaction is also a compensable transaction and its terminal state is totally dependent on the terminal states of its sub-transactions. Here, we focus our attention on terminal actions which are associated with terminal states. The behavior of a composite transaction is recorded by sequences of terminal actions which are referred as *transactional traces*. For any composite transaction T, the symbol $[(T, \sigma)]$ denotes all these transactional traces which finally cause T to end in the terminal state σ ($\sigma \in \Delta$). Especially, if T is a basic transaction, then it just has one single transactional trace, that is:

$$[(T, \sigma)] = \{\langle (T, \sigma) \rangle\}$$

Next we define the behavior of T as the union of transactional traces for all terminal states, denoted as $[\![T]\!]$:

$$[\![T]\!] = \bigcup_{\sigma \in \Delta} [(T, \sigma)]$$

In the following, we will define the behavior of sequential composition in terms of transactional traces. We write st for the concatenation of two traces s and t. The successful completion of $S; T$ requires both S and T to succeed.

$$[(S; T, suc)] = \{st \mid s \in [(S, suc)] \wedge t \in [(T, suc)]\}$$

Abortion of $S; T$ is caused by either an abortion or a full compensation of S.

$$[(S; T, abt)] = [(S, abt)] \cup \{sts' \mid s \in [(S, suc)] \wedge t \in [(T, abt)] \wedge s' \in [(S, cmp)]\}$$

The failure of S or T directly leads to the failure of $S; T$. Besides, the half-compensation of S results in the same outcome too.

$$\begin{aligned}[(S; T, fal)] = {} & [(S, fal)] \cup \{st \mid s \in [(S, suc)] \wedge t \in [(T, fal)]\} \cup \\ & \{sts' \mid s \in [(S, suc)] \wedge t \in [(T, abt)] \wedge s' \in [(S, hap)]\}\end{aligned}$$

A full compensation of $S; T$ means both sub-transactions have been compensated utterly. Note that sub-transactions are compensated in a reverse order to their original sequence.

$$[(S; T, cmp)] = \{ts \mid t \in [(T, cmp)] \wedge s \in [(S, cmp)]\}$$

A partial compensation of $S; T$ is rendered by a failed compensating execution of either sub-transaction.

$$[(S; T, hap)] = [(T, hap)] \cup \{ts \mid t \in [(T, cmp)] \wedge s \in [(S, hap)]\}$$

Note that the occurrences of $(S; T, cmp)$ and $(S; T, hap)$ require a precondition implicitly, that is, the whole transaction $S; T$ has already completed successfully.

Parallel composition: For $S \parallel T$, both branches S and T are executed in parallel. Likewise, their compensations are also activated concurrently when semantic rollback is needed. If one branch aborts or fails, the other branch is willing

to disrupt its flow and yield to this failure. In other words, two branches must both succeed or both not. The related behavioral dependencies are formalized as follows:

$$(S, act) \leftrightarrow (T, act) \qquad (\| \ 1)$$
$$(S, und) \leftrightarrow (T, und) \qquad (\| \ 2)$$
$$(S, suc) \leftrightarrow (T, suc) \qquad (\| \ 3)$$

More dependencies can be deduced:

$$(S, \alpha) \nleftrightarrow (T, \beta) \ \alpha \in \{abt, fal\}, \beta \in \{suc, cmp, hap\} \qquad (\| \ 4)$$
$$(T, \alpha) \nleftrightarrow (S, \beta) \ \alpha \in \{abt, fal\}, \beta \in \{suc, cmp, hap\} \qquad (\| \ 5)$$

The behavior of parallel composition needs to take interleaving into consideration. In the following, the symbol $s \ ||| \ t$ denotes the set of transactional traces which are combinations of s and t.
The successful completion of $S \| T$ achieves only when both branches succeed.

$$[(S \parallel T, suc)] = \{r \mid r \in s \ ||| \ t \wedge s \in [(S, suc)] \wedge t \in [(T, suc)]\}$$

The abortion of $S \parallel T$ is caused by the abortion of both branches.

$$[(S \parallel T, abt)] = \{r \mid r \in s \ ||| \ t \wedge s \in [(S, abt)] \wedge t \in [(T, abt)]\}$$

The failure of either branch causes the failure of $S \parallel T$.

$$\begin{aligned}[(S \parallel T, fal)] = &\{r \mid r \in s \ ||| \ t \wedge s \in [(S, fal)] \wedge t \in [(T, abt)]\} \\ &\cup \{r \mid r \in s \ ||| \ t \wedge s \in [(S, abt)] \wedge t \in [(T, fal)]\} \\ &\cup \{r \mid r \in s \ ||| \ t \wedge s \in [(S, fal)] \wedge t \in [(T, fal)]\}\end{aligned}$$

A full compensation of $S \parallel T$ means that both branches have been compensated successfully. Note that both branches are compensated in parallel too.

$$[(S \parallel T, cmp)] = \{r \mid r \in s \ ||| \ t \ \wedge s \in [(S, cmp)] \wedge t \in [(T, cmp)]\}$$

A partial compensation of $S \parallel T$ arises from a failed compensation of either branch.

$$\begin{aligned}[(S \parallel T, hap)] = &\{r \mid r \in s \ ||| \ t \wedge s \in [(S, hap)] \wedge t \in [(T, cmp)]\} \\ &\cup \{r \mid r \in s \ ||| \ t \wedge s \in [(S, cmp)] \wedge t \in [(T, hap)]\} \\ &\cup \{r \mid r \in s \ ||| \ t \wedge s \in [(S, hap)] \wedge t \in [(T, hap)]\}\end{aligned}$$

Internal choice: In some cases, only one branch is selected in accordance with internal decisions. In the construct of $S \sqcap T$, only S or T will be activated:

$$(S, act) \nleftrightarrow (T, act) \qquad (\sqcap \ 1)$$

Further, we get that only one branch will terminate:

$$(S, \alpha) \nleftrightarrow (T, \beta) \quad \{\alpha, \beta\} \subseteq \Delta \qquad (\sqcap \ 2)$$

The behavior of $S \sqcap T$ is directly defined as follows:

$$[\![S \sqcap T]\!] = [\![S]\!] \cup [\![T]\!]$$

Speculative choice: This construct provides a way for developers to design two or more threads to finish one task. If one thread is aborted, the other thread is still active trying to achieve the same target. In this construct of $S \otimes T$, S and T are two sub-transactions with equivalent functions. The two sub-transactions have even priorities and they are arranged to be executed concurrently. The choice is delayed when one sub-transaction has succeeded. That is, only one sub-transaction will be finally selected to achieve its business goal. When one sub-transaction terminates successfully, the other one cannot succeed but aborts either internally or forcibly. Especially, if one sub-transaction fails halfway, the other one should yield to this failure. The related behavioral dependencies are formalized below:

$$(S, act) \leftrightarrow (T, act) \qquad (\otimes\ 1)$$
$$(S, suc) \nleftrightarrow (T, suc) \qquad (\otimes\ 2)$$
$$(S, suc) \nleftrightarrow (T, fal) \qquad (\otimes\ 3)$$
$$(T, suc) \nleftrightarrow (S, fal) \qquad (\otimes\ 4)$$

The above dependencies imply that only one sub-transaction can be compensated if compensation is needed.

$$(S, \alpha) \nleftrightarrow (T, \beta) \quad \{\alpha, \beta\} \subseteq \{cmp, hap\} \qquad (\otimes\ 5)$$

Speculative choice is a construct which behaves partially like parallel composition by allowing concurrent executions and partially like internal choice by choosing one sub-transaction to fulfil the objective.

The successful completion of $S \otimes T$ realizes when one branch succeeds and another one aborts.

$$\begin{aligned}[(S \otimes T, suc)] = &\{r \mid r \in s \,|||\, t \wedge s \in [(S, suc)] \wedge t \in [(T, abt)]\} \\ &\cup \{r \mid r \in s \,|||\, t \wedge s \in [(S, abt)] \wedge t \in [(T, suc)]\}\end{aligned}$$

The abortion of $S \otimes T$ is due to the abortion of both branches.

$$[(S \otimes T, abt)] = \{r \mid r \in s \,|||\, t \wedge s \in [(S, abt)] \wedge t \in [(T, abt)]\}$$

Likewise, the failure of either branch will finally lead to the failure of the whole composition $S \otimes T$.

$$\begin{aligned}[(S \otimes T, fal)] = &\{r \mid r \in s \,|||\, t \wedge s \in [(S, fal)] \wedge t \in [(T, abt)]\} \\ &\cup \{r \mid r \in s \,|||\, t \wedge s \in [(S, abt)] \wedge t \in [(T, fal)]\} \\ &\cup \{r \mid r \in s \,|||\, t \wedge s \in [(S, fal)] \wedge t \in [(T, fal)]\}\end{aligned}$$

Since only one branch can succeed, then just the successful one will be compensated. A full compensation or a partial compensation is determined by the successful branch.

$$[(S \otimes T, \alpha)] = \begin{cases} [(S, \alpha)] & \text{if } S \text{ has succeeded} \\ [(T, \alpha)] & \text{if } T \text{ has succeeded} \end{cases} \quad \alpha \in \{cmp, hap\}$$

Alternative forwarding: Similar to speculative choice, this construct also provides two functionally equivalent sub-transactions to achieve one business goal. The difference is that these two sub-transactions have distinct priorities. In addition, the one with the higher priority is executed first and the other is activated only when the first one has been aborted. In $S \rightsquigarrow T$, S is planed to run first and T is the backup of S. The related dependency is given below:

$$(S, abt) < (T, act) \qquad (\rightsquigarrow 1)$$

Then we get the following dependencies:

$$(S, abt) \ll (T, \alpha) \quad \alpha \in \Delta \qquad (\rightsquigarrow 2)$$
$$(S, \alpha) \nleftrightarrow (T, \beta) \quad \alpha \in \Delta - \{abt\}, \beta \in \Delta \qquad (\rightsquigarrow 3)$$

The behavior of alternative forwarding corresponds to a sequential execution. Different from sequential composition, the right branch is enabled on abortion instead of on success of the left branch.

The successful completion of $S \rightsquigarrow T$ achieves when there is one branch which succeeds eventually.

$$[(S \rightsquigarrow T, suc)] = [(S, suc)] \cup \{st \mid s \in [(S, abt)] \wedge t \in [(T, suc)]\}$$

The abortion of $S \rightsquigarrow T$ is caused by the abortion of its alternative T.

$$[(S \rightsquigarrow T, abt)] = \{st \mid s \in [(S, abt)] \wedge t \in [(T, abt)]\}$$

The failure of $S \rightsquigarrow T$ is due to the failure of either branch.

$$[(S \rightsquigarrow T, fal)] = [(S, fal)] \cup \{st \mid s \in [(S, abt)] \wedge t \in [(T, fal)]\}$$

In this construct, only one branch can succeed in the end. Thus, the definitions of $[(S \rightsquigarrow T, cmp)]$ and $[(S \rightsquigarrow T, hap)]$ are as same as those for speculative choice.

Backward handling: Partial compensation leads to inconsistency which is unwelcome by the users. The failure of a composite transaction is ultimately caused by the half-compensated state of one of its sub-transactions provided that basic transactions cannot fail. Backward handling $S \trianglerighteq T$ is such a construct that provides a backward handler T to remedy the failure thrown by S. This handler tries to undo all the remaining effects which are not covered by partial compensation. The handler T is triggered on the failure of S, that is:

$$(S, fal) < (T, act) \qquad (\trianglerighteq 1)$$

Consequently, we get that:

$$(S, fal) \ll (T, \alpha) \quad \alpha \in \Delta \qquad (\trianglerighteq 2)$$
$$(S, \alpha) \nleftrightarrow (T, \beta) \quad \alpha \in \Delta - \{fal\}, \beta \in \Delta \qquad (\trianglerighteq 3)$$

Note that this construct offers a kind of backward recovery mechanism, for the backward handler is devised to undo the remaining partial effects.
The successful completion of $S \trianglerighteq T$ realizes when the left branch S succeeds.

$$[(S \trianglerighteq T, suc)] = [(S, suc)]$$

The abortion of $S \trianglerighteq T$ is caused by the abortion of the left branch S or the successful completion of the handler T.

$$[(S \trianglerighteq T, abt)] = [(S, abt)] \cup \{st \mid s \in [(S, fal)] \wedge t \in [(T, suc)]\}$$

The failure of S fires the activation of T and the thrown failure can be thoroughly cleared only when the handler T succeeds. Hence, either failure or abortion of T will cause the final failure of the whole composition $S \trianglerighteq T$.

$$[(S \trianglerighteq T, fal)] = \{st \mid s \in [(S, fal)] \wedge t \in [(T, abt)] \cup [(T, fal)]\}$$

Since only the success of the left branch S leads to the successful composition of $S \trianglerighteq T$, the compensation result is totally dependent on the compensation of S.

$$[(S \trianglerighteq T, cmp)] = [(S, cmp)] \qquad [(S \trianglerighteq T, hap)] = [(S, hap)]$$

Forward handling: This is another manner to deal with partial compensation apart from backward handling. In $S \triangleright T$, T is the forward handler to fix the failure thrown by S. Different from backward handling, this construct adopts the forward recovery mechanism trying to fulfill the business goal in the presence of failure. In other words, if the forward handler completes, the whole composition is regarded as success though some error has occurred previously. Likewise, this handler T can only be activated by the failure of S:

$$(S, fal) < (T, act) \tag{$\triangleright$ 1}$$

Consequently, we get that:

$$(S, fal) \ll (T, \alpha) \quad \alpha \in \Delta \tag{$\triangleright$ 2}$$
$$(S, \alpha) \nleftrightarrow (T, \beta) \quad \alpha \in \Delta - \{fal\}, \beta \in \Delta \tag{$\triangleright$ 3}$$

The successful completion of $S \triangleright T$ realizes when either branch succeeds.

$$[(S \triangleright T, suc)] = [(S, suc)] \cup \{st \mid s \in [(S, fal)] \wedge t \in [(T, suc)]\}$$

The abortion of $S \triangleright T$ is caused by the abortion of S.

$$[(S \triangleright T, abt)] = [(S, abt)]$$

The abortion of T means the handler does nothing essential and thus the previous failure has not been resolved at all. Then either failure or abortion of T will make the whole composition $S \triangleright T$ into the failed state.

$$[(S \triangleright T, fal)] = \{st \mid s \in [(S, fal)] \wedge t \in [(T, abt)] \cup [(T, fal)]\}$$

Since both branches can lead to the success of $S \triangleright T$, we should first judge which branch has succeeded and then be sure whose compensation is activated while undoing. Hence, the definitions of $[(S \triangleright T, cmp)]$ and $[(S \triangleright T, hap)]$ are as same as those for speculative choice.

Programmable compensation: Primarily, the compensation is attached through the transactional pair which is a central construct to compose activities [14]. As for a composite transaction, the developers sometimes need to program a new compensation so as to satisfy a specific application requirement. Thus, the construct of $S \circledast T$ is added to meet this demand. Formerly, the compensation of S is constructed by the accumulation of those of its sub-transactions. Here, T is the newly programmed compensation for S, while the original accumulated one is simply discarded. When S has completed, its new compensation T is waiting to be enabled so that the effect of S can be semantically removed in case some error occurs later:

$$(S, suc) \ll (T, act) \qquad (\circledast\ 1)$$

By using the transitive property of $\ll$ and dependencies listed in Table 1, we have that:

$$(S, suc) \ll (T, \alpha) \quad \alpha \in \Delta \qquad (\circledast\ 2)$$

In our model, compensation is also a compensable transaction. Thus we allow this kind of composition, such as $S \circledast (T_1 \rightsquigarrow T_2)$ in which S has two alternative compensations. Moreover, the actions (T, cmp) and (T, hap) have no chance to take place since compensation for compensation will never be used.

The behavior of $S \circledast T$ is quite simple, S decides whether the whole composition is successful, aborted or failed. Whereas the new compensation T determines whether the whole composition is compensated completely or partially.

$$\begin{array}{ll} [(S \circledast T, \alpha)] = [(S, \alpha)] & \alpha \in \{suc, abt, fal\} \\ [(S \circledast T, cmp)] = [(T, suc)] & \\ [(S \circledast T, hap)] = [(T, \alpha)] & \alpha \in \{abt, fal\} \end{array}$$

3.2 A Case Study

Now let us presents a real web transaction dedicated to the processing of customer orders. This description is carried out by compensable transactions illustrated in Figure 2. Firstly, an order request from a customer is accepted and this step is compensated by notifying the customer this request is canceled. Then money will be deducted from the credit card providing that the credit checking has passed. Afterwards, all the ordered items are packed for shipment and this step is compensated by unpacking. Simultaneously with packing items, the seller books shippers for delivery. In this example, we make an assumption that this seller has only two shippers (shipper A and shipper B) to contact with. Shipper A is cheaper but hard to book whereas shipper B is more expensive but always available. For the sake of saving money, shipper A is preferred and shipper B is booked only when shipper A is unavailable. At last, the selected shipper is responsible for delivering these items. Note that if the customer cancels the order during processing, the compensation for completed parts will be activated. When the compensation cannot properly undo the partial effects, the

$$
\begin{aligned}
OrderTrans &= \{(ProcessRequest; OrderProcess) \trianglerighteq GetIndemnity\} \\
OrderProcess &= PayByCard; (PrepareOrder \,\|\, ContactShipper); DeliverOrder \\
ProcessRequest &= AcceptOrder \circledast CancelOrder \\
PayByCard &= (CheckCredit; DeductMoney) \circledast RefundMoney \\
PrepareOrder &= PackItems \circledast UnpackItems \\
ContactShipper &= BookShipperA \rightsquigarrow BookShipperB
\end{aligned}
$$

Fig. 2. Transaction for order fulfillment

seller would ask for extra indemnities from the customer which is transacted by backward handling.

The transactional flow described above is not always satisfactory. Sometimes, the customer needs the goods urgently because of his timing requirement. However, the process of credit checking may cost so much time that transportation will be delayed. In this case, the seller would like to deal with payment in parallel with packing items. On the other hand, in order to be more reliable, sometimes transporting items by more than one shipper would be a better solution. Overall, the transactional flow designed in hand should satisfy specific application requirements. In the following section, we will introduce two kinds of representations to express application requirements. Further we will propose strict solutions to the verification of web transactions.

4 Verification

In this section, we mention two notions for specifying application requirements. One is referred as acceptable termination states (ATS) as a correctness criteria to specify relaxed atomicity requirements. Another one is a specification language used for expressing temporal constraints with regard to transactional properties.

4.1 Acceptable Termination States

Let $T.\sigma$ ($\sigma \in \Delta \cup \{idl\}$) represent the final state σ of the compensable transaction T. Especially, $T.idl$ denotes that T terminates without activation. As for a web transaction $\{T\}$, a termination state of $\{T\}$ is described by a set of final states of its sub-transactions $\{T_1.\sigma_1, T_2.\sigma_2, \ldots, T_n.\sigma_n\}$, where T_i is a sub-transaction of T. For example, given a transaction $T = \{(T_1 \trianglerighteq T_2); T_3\}$, one of its termination states is $\{T_1.fal, T_2.suc, T_3.idl\}$. It says that T terminates when T_1 fails and T_2 succeeds without activating T_3.

Let Ω be the set of transactions in which the designer is interested when investigating the termination states. Each transaction in Ω can be a composite transaction. Further, we require that for two arbitrary transactions $S, T \in \Omega$, T cannot be the sub-transaction of S and vice versa. An *acceptable termination*

state of a web transaction $\{T\}$ is a termination state limited to Ω in which the designer expect to see $\{T\}$ ends without raising any inconsistency. In other words, an acceptable termination state is a termination state in which this transaction finally succeeds or aborts but not fails. The set of all acceptable termination states specified by the designer is denoted as ATS.

Let $\lfloor T.\sigma \rfloor$ be the set of termination states in which T ends in σ. A web transaction $\{T\}$ is invalid if there is a termination state belonging to ATS which cannot make $\{T\}$ successful or aborted.

Definition 4.1. *A web transaction $\{T\}$ is said to be valid according to its acceptable termination states ATS if and only if $ATS \subseteq \lfloor T.suc \rfloor \cup \lfloor T.abt \rfloor$*

It is worth noting that it is much better if there are more termination states in which $\{T\}$ cannot fail except for those in ATS. In order to verify a valid web transaction, we should know the values of $\lfloor T.suc \rfloor$ and $\lfloor T.abt \rfloor$. As mentioned before, the transactional behavior is recorded by transactional traces. Now we transform transactional traces to termination states such that $\lfloor T.\sigma \rfloor$ can be derived from $[(T, \sigma)]$. Five steps need to be done in sequence:

1. While computing transactional traces, every interested transaction does not expand. That is, $[(T, \sigma)](T \in \Omega)$ is defined as $\{\langle (T, \sigma) \rangle\}$ without considering its sub-transactions.
2. If there are two actions related to one transaction in a transactional trace, the first one is removed while the latter one is preserved.
3. For any transactional trace t, its element (S, σ) is removed if S is not in Ω. New element (S, idl) is inserted if S is in Ω but not mentioned in t.
4. Elements of a transactional trace are actions with the form of (T, σ), while elements of a terminate state are states with the form of $T.\sigma$. Thus the form of elements should be changed from (T, σ) into $T.\sigma$.
5. Elements in a transactional trace are ordered but elements in a termination state are not. Finally, we extract the order information from traces by turning sequences into sets.

When a transaction T is compensated, some transactional traces contain two actions about T. The first action is (T, suc) and the following action is either (T, cmp) or (T, hap). The second step above is intended to remove the first action which is actually an interim state.

So far, we are able to compute $\lfloor T.\sigma \rfloor$ $(\sigma \in \Delta)$ for any T. Further, we need one more definition to help compute $\lfloor T.idl \rfloor$:

$$\begin{array}{ll} \lfloor T.idl \rfloor = \{\{T.idl\}\} & T \in \Omega \\ \lfloor T.idl \rfloor = \{s \cup t \mid s \in \lfloor T_1.idl \rfloor \wedge t \in \lfloor T_2.idl \rfloor\} & T \notin \Omega \wedge T = T_1 \odot T_2 \end{array}$$

The operator $\odot$ here and below denotes an arbitrary operator in t-calculus. Restricted by behavioral dependencies, not all pairs of final states can exist simultaneously. For instance, $T_1 \rightsquigarrow T_2$ does not have such a termination state $\{T_1.suc, T_2.abt\}$, since (T_1, suc) and (T_2, abt) are exclusive derived from $(\rightsquigarrow 3)$.

Definition 4.2. *Suppose* $T = T_1 \odot T_2$ *and* T_1, T_2 *are not expanded* [1]*,* $T_1.\sigma_1$ *and* $T_2.\sigma_2$ *are said compatible with regard to* $\odot$ *if and only if* $\exists \sigma \bullet \{T_1.\sigma_1, T_2.\sigma_2\} \in \lfloor T.\sigma \rfloor$

When a web transaction is proved to be invalid, we further provide a solution to help designers to find the possible reason for this problem. Let $\mathcal{A}$ represent the set of termination states in ATS but not in $\lfloor T.suc \rfloor \cup \lfloor T.abt \rfloor$. Apparently, $\mathcal{A}$ is not empty when the web transaction $\{T\}$ is invalid. We try the two steps below to locate the error.

1. Firstly, construct a syntax tree according to the structure of $\{T\}$. This is a binary tree and the contents of its nodes are either operators in t-calculus or transactions in Ω. All these transactions in Ω lie in the leaves of this tree.[2] Every node n relates to a compensable transaction denoted by $\mathbb{T}(n)$. For a leaf node n, $\mathbb{T}(n)$ is equal to its content. For a non-leaf node m with an operator $\odot$, $\mathbb{T}(m)$ is equal to $\mathbb{T}(m_1) \odot \mathbb{T}(m_2)$, in which m_1, m_2 are its children nodes.
2. Secondly, try to traverse this tree in post-order. Given a termination state θ, for any tree node n, its related transaction $\mathbb{T}(n)$ has a final state denoted by $\ell(n, \theta)$. While visiting a leaf node n, for each θ in $\mathcal{A}$, $\ell(n, \theta)$ is computed and it is equal to $\mathbb{T}(n).\sigma$ which is an element of θ. While visiting a non-leaf node m with an operator $\odot$ and two children nodes m_1, m_2, for each θ in $\mathcal{A}$, we need to check whether the two states $\ell(m_1, \theta)$ and $\ell(m_2, \theta)$ are compatible with regard to $\odot$. If something incompatible is found, it means this operator is wrongly written and the process of traverse is stopped. Otherwise, let $\ell(m, \theta)$ equal to $\mathbb{T}(m).\sigma$ where $\{\ell(m_1, \theta), \ell(m_2, \theta)\} \in \lfloor \mathbb{T}(m).\sigma \rfloor$ for each θ in $\mathcal{A}$. After that the next node in post-order is visited until there is a node associated with an improper operator.

Next, we take the example in Figure 2 to explain our method for locating the problem. Suppose the designer is interested in these sub-transactions below:

$$\Omega = \{ProcessRequest, PayByCard, PrepareOrder\} \\ \cup \{ContactShipper, DeliverOrder, GetIndemnity\}$$

The syntax tree for $OrderTrans$ is given in Figure 3. It has eleven nodes marked by $n_1, n_2, \ldots, n_{11}$. All the leaves denote the interested sub-transactions. While traversing the tree in post-order, the nodes are visited in the following order:

$$n_4, n_8, n_{10}, n_{11}, n_9, n_6, n_7, n_5, n_2, n_3, n_1$$

For simplicity, we assume $\mathcal{A}$ just includes one termination state:

$$\theta = \{ProcessRequest.cmp, PayByCard.abt, PrepareOrder.abt\} \\ \cup \{ContactShipper.abt, DeliverOrder.idl, GetIndemnity.idl\}$$

[1] While computing $\lfloor T.\sigma \rfloor$ for a composite transaction T with its sub-transactions T_1, T_2 not expanded, we temporarily set Ω to be $\{T_1, T_2\}$. Thus, $\lfloor T.\sigma \rfloor$ just includes the sets with two elements denoting the final states of T_1, T_2 respectively.

[2] If the transactions in Ω cannot cover all the leaves, the designer should provide more information by enlarging the elements of Ω until all the leaves can be covered.

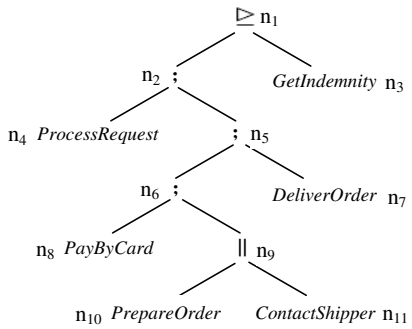

Fig. 3. The syntax tree of order transaction

When visiting the first four nodes in post-order, we get their related transactions and states as follow:

$$\mathbb{T}(n_4) = ProcessRequest \qquad \ell(n_4, \theta) = ProcessRequest.cmp$$
$$\mathbb{T}(n_8) = PayByCard \qquad \ell(n_8, \theta) = PayByCard.abt$$
$$\mathbb{T}(n_{10}) = PrepareOrder \qquad \ell(n_{10}, \theta) = PrepareOrder.abt$$
$$\mathbb{T}(n_{11}) = ContactShipper \qquad \ell(n_{11}, \theta) = ContactShipper.abt$$

Node n_9 is equipped with a parallel operator. The transaction related to n_9 is:

$$\mathbb{T}(n_9) = \mathbb{T}(n_{10}) \parallel \mathbb{T}(n_{11}) = PrepareOrder \parallel ContactShipper$$

It is not difficult to prove that $\ell(n_{10}, \theta)$ and $\ell(n_{11}, \theta)$ are compatible with regard to this parallel operator by figuring out $\lfloor \mathbb{T}(n_9).abt \rfloor$:

$$\lfloor \mathbb{T}(n_9).abt \rfloor = \{\{PrepareOrder.abt, ContactShipper.abt\}\}$$

Thus, we get that: $\ell(n_9, \theta) = \mathbb{T}(n_9).abt$. The next node n_6 is associated with a sequential operator and the transaction related to n_6 is: $\mathbb{T}(n_6) = \mathbb{T}(n_8); \mathbb{T}(n_9)$.

In the case, the states $\mathbb{T}(n_8).abt$ (i.e., $\ell(n_8, \theta)$) and $\mathbb{T}(n_9).abt$ (i.e., $\ell(n_9, \theta)$) are proved to be incompatible with regard to the sequential operator. It is easy to predict this result because in the sequential composition $S;T$, T does not have the chance to be activated without mentioning the possibility of abortion when S is aborted. Thus, we find that this sequential operator attached to n_6 is wrongly written. In fact, the designer here expects the credit check to be performed in parallel not in sequence with preparing order because such check normally succeeds. Moreover, the seller in this case has more time to process order so as not to delay the transportation unnecessarily.

4.2 Verifying Temporal Constraints

In order to express more specific transactional properties with temporal constraints, we devise a specification language whose syntax is given below:

$$\psi ::= \diamond\, a \mid a \prec b \mid a \ll b \mid a < b \mid a \leftrightarrow b \mid a \nleftrightarrow b$$
$$\neg\psi \mid \psi \wedge \psi \mid \psi \vee \psi$$

$$
\begin{array}{ll}
(\!|\diamond\, a|\!) & = \{s : M^* \mid \exists i \bullet s[i] = a\} \\
(\!|a \prec b|\!) & = \{s : M^* \mid \forall i \bullet (s[i] = a \Rightarrow \exists j \bullet (j > i \wedge s[j] = b))\} \\
(\!|a \ll b|\!) & = \{s : M^* \mid \forall i \bullet (s[i] = b \Rightarrow \exists j \bullet (j < i \wedge s[j] = a))\} \\
(\!|a < b|\!) & = \{s : M^* \mid \exists i, j \bullet (i < j \wedge s[i] = a \wedge s[j] = b) \vee \forall i \bullet (s[i] \neq a \wedge s[i] \neq b)\} \\
(\!|a \leftrightarrow b|\!) & = \{s : M^* \mid \exists i, j \bullet (s[i] = a \wedge s[j] = b) \vee \forall i \bullet (s[i] \neq a \wedge s[i] \neq b)\} \\
(\!|a \nleftrightarrow b|\!) & = \{s : M^* \mid \exists i \bullet s[i] = a \Rightarrow \forall j \bullet s[j] \neq b\} \\
(\!|\neg\psi|\!) & = M^* - (\!|\psi|\!) \\
(\!|\psi_1 \wedge \psi_2|\!) & = (\!|\psi_1|\!) \cap (\!|\psi_2|\!) \\
(\!|\psi_1 \vee \psi_2|\!) & = (\!|\psi_1|\!) \cup (\!|\psi_2|\!)
\end{array}
$$

Fig. 4. Interpretations over traces for formulae

where a, b are terminal actions. Except for these five relations $(<, \prec, \ll, \leftrightarrow, \nleftrightarrow)$, one more unitary relation $\diamond$ is introduced. $\diamond\, a$ says that a will definitely occur in the future.

Let M be the set of all terminal actions, and M^* be the set of all finite sequences (including $\langle\rangle$) which are formed from elements of M. For a trace s, $s[i]$ denotes the i^{th} element. From another view, a formula ψ can be regarded as a set of transactional traces defined by the function $(\!|\cdot|\!)$, whose definition is inductively given in Figure 4. Given a transaction T and a property ψ, the designer expects to verify whether ψ is satisfied by a specific behavior $[(T, \sigma)]$ for a concrete state σ $(\sigma \in \Delta)$.

Definition 4.3. *ψ holds for $[(T, \sigma)]$ (written as $[(T, \sigma)] \models \psi$) if and only if $[(T, \sigma)] \subseteq (\!|\psi|\!)$.*

The problem is that the number of traces in $(\!|\psi|\!)$ is too large due to a large amount of elements in M. To improve this, we reset M for each formula ψ and make it be the set of actions only mentioned by ψ. In addition, we need to restrict the traces of $[(T, \sigma)]$ to the actions in M. For a trace s, we use the expression $s \restriction M$ to denote a new trace formed from s simply by omitting all actions outside M. Thus, the traces from $[(T, \sigma)]$ restricted to M is defined as follows:

$$[(T, \sigma)]\downarrow_M = \{t \mid \exists s \bullet (t = s \restriction M \wedge s \in [(T, \sigma)])\}$$

Then the verification process can be improved by getting rid of actions independent of properties.

Theorem 4.1. *Let M be the set of actions occurring in ψ. $[(T, \sigma)] \models \psi$ holds if and only if $[(T, \sigma)]\downarrow_M \subseteq (\!|\psi|\!)$.*

Example: Given a simple transaction $T = (T_1; T_2) \trianglerighteq T_3$ and a specified property $\psi = (T_1, hap) \prec (T_3, suc)$, we require to check whether $[(T, abt)] \models \psi$ holds.

First, we compute the behavior of T provided it is aborted.

$$
\begin{array}{rl}
[(T, abt)] = & \{\langle (T_1, abt)\rangle, \langle (T_1, suc), (T_2, abt), (T_1, cmp)\rangle\} \\
& \cup \{\langle (T_1, suc), (T_2, abt), (T_1, hap), (T_3, suc)\rangle\} \\
& \cup \{\langle (T_1, fal), (T_3, suc)\rangle, \langle (T_1, suc), (T_2, fal), (T_3, suc)\rangle\}
\end{array}
$$

This property only refers to two actions and we get that:

$$M = \{(T_1, hap), (T_3, suc)\}$$

Then we limit the traces of $[(T, abt)]$ to the set of M:

$$\begin{array}{ll}
\langle(T_1,abt)\rangle \upharpoonright M & = \langle\rangle \\
\langle(T_1,fal),(T_3,suc)\rangle \upharpoonright M & = \langle(T_3, suc)\rangle \\
\langle(T_1,suc),(T_2,abt),(T_1,cmp)\rangle \upharpoonright M & = \langle\rangle \\
\langle(T_1,suc),(T_2,fal),(T_3,suc)\rangle \upharpoonright M & = \langle(T_3, suc)\rangle \\
\langle(T_1,suc),(T_2,abt),(T_1,hap),(T_3,suc)\rangle \upharpoonright M & = \langle(T_1,hap),(T_3,suc)\rangle
\end{array}$$

Consequently, we have that

$$[(T, abt)]\downarrow_M = \{\langle\rangle, \langle(T_3, suc)\rangle, \langle(T_1, hap), (T_3, suc)\rangle\}$$

At last, we convert the property to a set of traces:

$$\begin{aligned}
(\!|\psi|\!) &= \{s : M^* \mid \forall i \bullet (s[i] = (T_1, hap) \Rightarrow \exists j \bullet (j > i \wedge s[j] = (T_3, suc)))\} \\
&= \{\langle\rangle, \langle(T_3, suc)\rangle, \langle(T_1, hap), (T_3, suc)\rangle\}
\end{aligned}$$

Obviously, $[(T, abt)]\downarrow_M \subseteq (\!|\psi|\!)$ holds for this example.

A complex formula ψ possibly makes $(\!|\psi|\!)$ much bigger, which increases the difficulty of verification. A practical solution is that we change the formula into the *disjunctive normal form* as follows:

$$\psi = \psi_1 \vee \psi_2 \vee \cdots \vee \psi_n$$

Apparently, $(\!|\psi_i|\!)$ $(1 \leq i \leq n)$ is smaller than $(\!|\psi|\!)$. If we can verify that one sub-formula is satisfied, the whole formula holds obviously.

Theorem 4.2. *Suppose* $\psi = \psi_1 \vee \psi_2 \vee \cdots \vee \psi_n$. $[(T, \sigma)] \models \psi$ *if* $[(T, \sigma)] \models \psi_1 \vee [(T, \sigma)] \models \psi_2 \vee \cdots \vee [(T, \sigma)] \models \psi_n$.

5 Related Work

Due to limitations of classical transactions, some models have adopted the concept of compensation to satisfy the autonomy requirement in distributed environment. Elmagarmid et al. [5] supported mixed transactions allowing compensable and noncompensable sub-transactions to coexist. Levy et al. [11] proposed a formal model to unify the two dual methods of compensation and retry. In their work, a whole transaction was modeled as a static partial order of steps. A new approach called XIP [17] was proposed to present an optimistic commit protocol to enable the Internet transaction semantics. In contrast with these works, we develop a formal language to explicitly model the logical precedence, causality and synchronization constraints among compensable transactions.

In recent literatures, process calculus is adopted extensively to formalize long-lived transactions. Several proposals [1, 10, 15] took the well-known π-calculus as

a starting point and extended it with some transactional features. Another language cCSP [3] as an extension of CSP supports automatic compensation on transactional failure. It is equipped with a simple operational semantics [4] and this semantics is made executable by encoding the rules in Prolog. Bruni et al. [2] have developed a sagas calculus with similar operators to cCSP. The difference is mainly that while considering parallel execution, cCSP encourages synchronized compensation whereas sagas calculus supports distributed compensation. None of these transactional languages treats compensations as compensable transactions like t-calculus [14] we proposed earlier does. The operational and algebraic semantics of t-calculus was studied and a kind of linking theory has been established [13, 12]. Unlike our previous works, this paper specifies web transactions by investigating compensable transactions in a higher level of granularity. In addition, the semantics of each transactional construct is explored in a different way by formally describing its behavioral dependencies. Moreover, this work proposes strict solutions to verify transactional behavior.

With regard to verification, there are already a series of work focusing on analyzing and verifying web service properties. Foster et al. [6] discussed a model-based approach to verify web service compositions, in which implementations were mechanically translated to FSP to perform an equivalence trace verification process. Temporal logics for compositional reasoning about web service interfaces have been proposed in [18]. Nakajima used model checking to analyze web service flow by translating BPEL descriptions into Promela [16]. The interactions of composite web services were analyzed by modeling them as conversations [7]. Service processes were first translated into guarded automata and then verified using SPIN. Pu et al. [9] adopted a similar approach to use the model checker UPPAAL to verify BPEL programs including timed properties. However, we find no relevant work on verifying web transactions with the feature of compensation. This paper is the first attempt in this area to the best of our knowledge.

6 Conclusion

Web transactions in this paper are built upon a set of compensable transactions which help to ensure a relatively relaxed atomicity. Compensable transactions support composition in different manners, aiming to enhance the reliability and flexibility of web transactions. Distinct transactional constructs correspond to different behavioral dependencies which have been explored from two aspects. On the one hand, the dependency between two sub-transactions on the same syntactic level is described by a series of relations between actions. On the other hand, the dependency among a composite transaction and its sub-transactions is defined in terms of transactional traces. Transactional traces are more precise than relations of actions, since they can assist designers to track the behavior of a whole web transaction.

The formal description of web transactions helps to clarify ambiguous concepts and it provides the basis for the following process of verification. At first, this paper has provided a method to verify web transactions according to the

relaxed atomicity requirement. Besides, the problem can be located if inconsistency exists. Afterwards, a specification language has been proposed for specifying temporal constraints about compensable transactions. The verification process is further optimized by restricting actions of interest and partitioning property formulae.

References

1. L. Bocchi, C. Laneve, and G. Zavattaro. A calculus for long-running transactions. In *Proc. of FMOODS'03*, LNCS 2884, pages 124–138. Springer, 2003.
2. R. Bruni, H. Melgratti, and U. Montanari. Theoretical foundations for compensations in flow composition languages. In *POPL'05*, pages 209–220. ACM Press, 2005.
3. M. Butler, T. Hoare, and C. Ferreira. A trace semantics for long-running transaction. In *25 Years of CSP*, LNCS 3525, pages 133–150. Springer, 2004.
4. M. Butler and S. Ripon. Executable semantics for compensating CSP. In *Proc. of WS-FM'05*, LNCS 3670, pages 243–256. Springer, 2005.
5. A. K. Elmagarmid, Y. Leu, W. Litwin, and M. Rusinkiewicz. A multidatabase transaction model for interbase. In *VLDB'90*, pages 507–518, 1990.
6. H. Foster, S. Uchitel, J. Magee, and J. Kramer. Model-based verification of Web service compositions. In *ASE'03*, pages 152–161. IEEE Computer Society, 2003.
7. X. Fu, T. Bultan, and J. Su. Analysis of interacting BPEL web services. In *Proc. of WWW'04*, pages 621–630. ACM Press, 2004.
8. H. Garcia-Molina and K. Salem. Sagas. In *Proc. of ACM SIGMOD'87*, pages 249–259. ACM Press, 1987.
9. P. Geguang, Z. Xiangpeng, W. Shuling, and Q. Zongyan. Towards the Semantics and Verification of BPEL4WS. In *In WLFM'05*, ENTCS 151, pages 33–52. Elsevier, 2006.
10. C. Laneve and G. Zavattaro. Foundations of web transactions. In *Proc. of FoSSaCS'05*, LNCS 3441, pages 282–298. Springer, 2005.
11. E. Levy, H. F. Korth, and A. Silberschatz. A theory of relaxed atomicity. In *PODC'91*, pages 95–110. ACM Press, 1991.
12. J. Li, H. Zhu, and J. He. Algebraic Semantics for Compensable Transactions. In *Proc. of ICTAC'07*, LNCS 4711, pages 306–321. Springer, 2007.
13. J. Li, H. Zhu, G. Pu, and J. He. A Formal Model for Compensable Transactions. In *Proc. of ICECCS'07*, pages 64–73. IEEE Computer Society, 2007.
14. J. Li, H. Zhu, G. Pu, and J. He. Looking into compensable transactions. In *Proc. of SEW-31*, pages 154–166. IEEE Computer Society, 2007.
15. M. Mazzara and R. Lucchi. A framework for generic error handling in business processes. In *WS-FM'04*, ENTCS 105, pages 133–145. Elsevier, 2004.
16. S. Nakajima. Model-checking of safety and security aspects in web service flows. In *Proc. of ICWE'04*, LNCS 3140, pages 488–501. Springer, 2004.
17. J. Ouyang, A. Sahai, and V. Machiraju. An approach to optimistic commit and transparent compensation for e-service transactions. *HP Laboratories Palo Alto*, February 2001.
18. M. Solanki, A. Cau, and H. Zedan. Augmenting semantic web service descriptions with compositional specification. In *WWW'04*, pages 544–552. ACM Press, 2004.