An Approach to Weaving Stateful Aspects using LTS

CHUN-HUA YANG AND HAI-YANG WANG

School of Information Science and Technology
Shandong Institute of Light Industry
Shandong, 250353 P.R. China

School of Computer Science and Technology
Shandong University
Shandong, 250100 P.R. China

Stateful aspects is a recent topic in AOSD. As the trigger of a stateful aspect depends on the system execution trace, it needs complicate modeling mechanisms for the pointcuts and weaving operations. We present an approach to weaving stateful aspects using Labelled Transition System (LTS). In the approach, the base system is specified as a LTS. The pointcut is modeled as an extended LTS that observes the trace of the base system and decides the activation points for the advices. The composition of the pointcut and the base system can achieve a LTS that is semantically consistent with the base system. The advice weaving is modeled as transformation operations on the composition of the base system and the pointcut. These weaving operators can preserve the semantic equivalence between systems. The approach has been implemented on a FSP based prototype tool.

Keywords: stateful aspects, aspects, weaving, labeled transition system, aspect-oriented software development

1. INTRODUCTION

Stateful aspects is a recent research topic in AOSD, which was first introduced by Douence et al., in their work on Event-Based AOP and trace-based aspects [1-3]. This technique allows aspects to trigger on a sequence of join points instead of a single join point. The activation condition of an advice depends on the current and past state of the program execution. Consider the example where some additional behavior must be invoked after a particular sequence of events has been executed. In order to match this sequence, the history of the program state must be traced. This is an extra concern in its own and without language support, will be tangled with the base module and the concern that depends on this history tracking.

Essentially, the need for stateful aspects indicates that the base system exhibits reactive behavior. However, as most of aspect-oriented programming languages are based on procedural, functional, object-oriented or component-based decomposition languages and frameworks, they do not provide adequate abstractions for specifying and implementing reactive behavior [4]. This is an issue that is frequently encountered not only in programs but also in the software design.

In contrast to general aspects, stateful aspects needs more complicate modeling mechanisms for the pointcuts and weaving operators. Currently, many aspect-oriented

Received October 26, 2009; revised December 20, 2010 & January 13, 2011; accepted February 8, 2011.
Communicated by Chih-Ping Chu.
* This work was also partially supported by the National Natural Science Foundation of China under grant No. 60673130.
programming languages have provided support for stateful aspects, such as Declarative Event Patterns (DEP) [5] or JAsCo [6]. Generally, the proposed pointcut descriptors implicitly or explicitly maintain a finite state machine to determine the trigger condition of an advice. Due to the difference between the state machine and the program, extra instrumentation should be coded to hook the two mechanisms.

The paper focuses on modeling and weaving stateful aspects at the architecture level and presents a Labelled Transition System (LTS) [7] based approach. The main contributions of the approach include:

- modeling aspects using the LTS. The LTS is a powerful tool for specifying software architecture behavior, which makes the approach applicable for general software system;
- adopting the same mechanism for the pointcuts as that for the base system, which avoids the extra overheads caused by harmonizing the pointcut mechanism with the base system;
- presenting operators to weave observer aspects and assistant aspects [8], which can preserve the semantic equivalence between systems.

The approach has been implemented on a prototype weaving tool.1

The rest of the paper is structured as follows: Section 2 describes the related LTS concepts and introduces a motivation example; section 3 formalizes the approach; section 4 presents the implementation; section 5 explores related works; section 6 concludes.

2. MOTIVATION

Labelled Transition System (LTS) [7] is a powerful tool for modeling and analyzing software system behavior at the architecture level. We first describe concepts related to LTS, then define the base system and introduce a motivation example.

2.1 A Brief Introduction to LTS

**Definition 1** Labelled Transition System: Let States be a universal set of states, Act be a universal set of observable action labels, and τ represent unobservable action label. Let Actτ = Act ∪ {τ}. A labeled transition system (LTS) is a tuple P = (S, L, T, sinit), where S ⊆ States is a finite set of states, L ⊆ Act, is a set of labels, T ⊆ (S × L × S) is a transition relation between states, and sinit ∈ S is the initial state.

Given a LTS P = (S, L, T, sinit) we say P transits on l to P′, denoted P l→ P′, if P′ = (S, L, T, sinit′) and (sinit, l, sinit′) ∈ T. Let w = w1, …, wk be a word over Actτ. Then P w→ P′ means that there exist P0, …, Pk such that P = P0, P′ = Pk, and Pi→ P_{i+1} for 0 ≤ i < k. We write P w→ to mean ∃P′ · P w→ P′ and P w→ to mean ∃l, P′ · P l→ P′.

**Definition 2** Trace: A word π = l1, …, ln for l_i ∈ Act (0 ≤ i < k) is a trace on LTS P if P π→ P′.

1 http://code.google.com/p/ltsaspectweaving/downloads/list.
Definition 3  Strong Bisimulation Equivalence: Let $\wp$ be the universe of all LTSs, and $P, Q \in \wp$. $P$ and $Q$ are strong equivalent, written $P \sim Q$, if $(P, Q)$ is contained in some bisimulation relation $R \subseteq \wp \times \wp$ for which the following holds for all $\ell \in \text{Act}_\tau$:

1. $(P \xrightarrow{\ell} P') \Rightarrow (\exists Q' \cdot Q \xrightarrow{\ell} Q' \land (P', Q') \in R)$,
2. $(Q \xrightarrow{\ell} Q') \Rightarrow (\exists P' \cdot P \xrightarrow{\ell} P' \land (P', Q') \in R)$.

Definition 4  Parallel Composition: Let $P = (S_P, L_P, T_P, s_{initP})$ and $Q = (S_Q, L_Q, T_Q, s_{initQ})$ be LTS. Then parallel composition ($||$) is a symmetric operator and $P || Q$ is the LTS $(S_P \times S_Q, L_P \cup L_Q, T, (s_{initP}, s_{initQ}))$, where $T$ is the relation achieved according to the rules shown in Fig. 1.

\[
\begin{align*}
P &\xrightarrow{\ell} P' \quad \ell \in L_P \land P || Q &\xrightarrow{\ell} P'||Q' \quad \ell \in L_Q \\
Q &\xrightarrow{\ell} Q' \quad \ell \notin L_P \land P || Q &\xrightarrow{\ell} P'|Q' \quad \ell \in L_Q
\end{align*}
\]

Fig. 1. parallel composition ($||$) rules.

2.2 Defining the Base System

The base system encapsulates the business concern, which is composed of components. A LTS describes how a component behaves and communicates with the environment.

Definition 5  Component: A component is a LTS $e = (S, L, T, s_{init})$, where $L \subseteq \text{Act}_\tau$ is the action set, while $L - \{\tau\}$ is the communicating action set.

A component interconnects with the environment through the communicating actions. Components interact on each other through their shared common actions according to the parallel composition ($||$) rules.

Definition 6  The base system: A base system $P_B$ is a parallel composition of components $e_1, \ldots, e_n$: $P_B = (e_1 || \ldots || e_n)$ for $n \in \mathbb{N}$.

2.3 A Motivation Example

Consider an international travel agent example that provides services for obtaining information for traveling destinations and registering bookings [9]. Fig. 2 illustrates its base system. It has two states and state $0$ is the initial state. To use the services of the agent, customer should first log in. Then, they can query certain information (queryInfo) or book a travel package before they log out the system.

Now, the agent wants to inspire the booking service. When a user logs out the system, if he (or she) has booked a travel package, they will get a bonus that is to be settled later. Bonus is not part of the main functionality of the example, and should be separated in an aspect. Once the transition $1 \to 0$ occurs, the bonus aspect should apply. Moreover, the trigger condition is that action book has been executed in the system history. For ex-
ample, if the system trace is like:

\[ login, queryInfo, queryInfo, queryInfo, logout, \]

the bonus aspect cannot be triggered. Otherwise, if the system trace is as follows,

\[ login, queryInfo, book, queryInfo, logout, \]

then the aspect should be activated. Therefore, bonus is a stateful aspect.

Nowadays, the majority of proposed pointcut mechanisms for stateful aspects are based on finite state machines \([3, 5, 6]\). The finite state machine based pointcut can accept a legal join point sequence to decide the trigger conditions of advices. For example, a finite state machine that accepts the language:

\[ login(queryInfo) \ast book(queryInfo \mid book) \ast logout \]

can be used to model the pointcut of the bonus aspect. However, we do not want to use finite state machine to model the pointcut as it is inconsistent with the LTS mechanism for the base system.

We extend the LTS to achieve the same effect as the finite state machine pointcuts. The extended LTS can observe the trace of the base system and label transitions where advices would apply special join point tags. For example, Fig. 3 illustrates the extended LTS for the bonus aspect. It observes the trace of the base system:

\[ (queryInfo) \ast book(queryInfo \mid book) \ast logout. \]

Moreover, the transition \( 0 \rightarrow 1 \) with a special tag “/ JP” indicates the position to trigger advices. When the extended LTS combines with the base system, an extended LTS that is semantically consistent with the base system can be achieved. Advices can be woven on it afterwards.

3. THE APPROACH

In our approach, it is assumed that the aspect be woven on a well-defined system. A well-defined system should be deterministic and progressive.
Definition 7  Determinism: A LTS $P = (S, L, T, s_{init})$ is deterministic iff $\forall \ell \in L - \{\tau\}$: $(\exists (s, \ell, s_1) \in T \land (s, \ell, s_2) \in T \Rightarrow (s_1 = s_2))$.

Given a deterministic LTS $P$, if trace $\pi_1 = \pi_2$ and $P \rightarrow P'$, then $P \rightarrow P'$. In other words, two identical traces arrive at the same state.

Definition 8  Progress: A LTS $P = (S, L, T, s_{init})$ is progressive iff any state $s \in S$ is progressive: $\exists (s) \in S \cdot (s \rightarrow (s \rightarrow s))$.

Definition 9  Well-defined: A LTS $P$ is well-defined iff $P$ is progressive and deterministic.

An aspect may have various influences on the base system. For example, an assistant aims at modifying the behavior of the base system [8]. But, the way aspects are allowed to interfere with the system behavior should be clearly defined and limited. Indeed, one would intuitively expect that whenever two systems $P_1$ and $P_2$ are semantically equivalent, applying such an aspect to $P_1$ gets the same result (semantically speaking) as applying it to $P_2$ [10].

Definition 10  Semantic equivalence preservation: An aspect weaving preserves the semantic equivalence iff given two base systems $P_1$ and $P_2$ and an aspect $A$, the following holds:

$P_1 \sim P_2 \Rightarrow P_1' \sim P_2'$

where $P_1'$ and $P_2'$ is the result when applying aspect $A$ to $P_1$ and $P_2$ respectively.

In the rest of the paper, we shall define aspects as LTS transformations and show that aspect weaving may be considered as a new operator. We shall show that this new operator preserves determinism, progress and the semantic equivalence.

3.1 Modeling Stateful Aspects

3.1.1 Modeling the pointcut

We first introduce an extended LTS to label the LTS with join point tags.

Definition 11  $f$-Labelled Transition System: A $f$-Labelled Transition System (or $f$LTS for short) is a tuple $P = (S, L, T, f, s_{init})$, in which $(S, L, T, s_{init})$ is a LTS and $f$ is a function: $T \rightarrow \{0, 1\}$.

The function $f$ in $f$LTS aims at labeling the join points where advices would trigger. Given a transition $t$, if $f(t) = 1$, then it is labeled a join point tag. For convenient discussion, in the following section, we write $s \rightarrow^{\ell} s'$ to mean $s \rightarrow^{\ell} s' \land f(s \rightarrow^{\ell} s') = 0$ and $s \rightarrow^{\ell} s'$ to mean $s \rightarrow^{\ell} s' \land f(s \rightarrow^{\ell} s') = 1$.

Given a $f$LTS $P = (S, L, T, f, s_{init})$, we define an auxiliary operator $\partial$ to get its LTS:
\( \partial(P) = (S, L, T, s_{init}) \). In addition, we define an extended parallel composition operator \((\times)\) to compose a LTS and a fLTS.

**Definition 12** Extended parallel composition: Let \( P = (S_P, L_P, T_P, s_{init}) \) be a LTS and \( Q = (S_Q, L_Q, T_Q, f_Q, s_{init}) \) be a fLTS. The extended parallel composition \((\times)\) is a symmetric operator and \( P \times Q \) returns the fLTS \((S_{PQ}, L_{PQ}, T_{PQ}, f_{PQ}, s_{init})\), where \( T_{PQ} \subseteq (S_P \times S_Q) \times (L_P \cup L_Q) \times (S_P \times S_Q) \) and \( f_{PQ} : T_{PQ} \rightarrow \{0, 1\} \) is the relation and function achieved according to the rules shown in Fig. 4.

![Fig. 4. Extended parallel composition \((\times)\) rules.](image)

Fig. 4. Extended parallel composition \((\times)\) rules.

A pointcut is a fLTS that its action set is a subset of the base system’s action set and the composition of the pointcut and the base system is semantically consistent with the base system.

**Definition 13** Pointcut: Let \( P_B = (S_B, L_B, T_B, s_{init}) \) be a base system. A fLTS \( P_{pc} = (S_{pc}, L_{pc}, T_{pc}, f_{pc}, s_{init}) \) is a pointcut on \( P_B \) iff the following conditions hold:

\( \partial(P_{pc}) \) is well-defined \( \land L_{pc} \subseteq L_B \land \partial(P_B \times P_{pc}) \sim P_B \).

Fig. 5 illustrates the fLTS derived from the composition \((\times)\) of the pointcut of the bonus aspect and the base system.

![Fig. 5. The fLTS derived from the composition \((\times)\) of the pointcut of the bonus aspect and the base system.](image)

In the composition of a pointcut and the base system, transitions labelled with the special tag are joins points where advices would apply. We call them transition join points.

**Definition 14** Transition join point: Let \( P_B \) be a base system, \( P_{pc} \) be the pointcut on \( P_B \), and \( P_M = (S_M, L_M, T_M, s_{init}) \) be the composition of \( P_B \) and \( P_{pc} \). A transition \( t \) is a transition join point iff \( t \in T_M \land f(t) = 1 \).

In addition to the transition join points determined by the pointcut, some aspects need a target state for recovery. For example, in [10], a Tolinit aspect can recover the system execution to the initial state. Such a target state is another type of join points. We call it a state join point. In a LTS, states are unobservable, whereas traces are. Moreover, a trace in a well-defined system can identify a unique target state. Therefore, we use a trace to identify a state join point.
Definition 15  State join point: Let $P_B$ be a base system, $P_{pc}$ be the pointcut on $P_B$, and $P_M = (S_M, L_M, T_M, f_M, s_{initM}) = P_B \bowtie P_{pc}$ be the composition of $P_B$ and $P_{pc}$. A state $s$ is a state join point iff $s \in S_M \land P_M \rightarrow s$, where $\pi$ is a trace of $P_B$.

Note that in Definition 15 $\pi$ is a trace on $P_B$ instead of $P_B \bowtie P_{pc}$. Although the advice weaving operates on $P_B \bowtie P_{pc}$, the trace of $P_B \bowtie P_{pc}$ is unknown before weaving. Let $s_{\mu B}$ be the target state in $P_B$ identified by $\pi: P_B \rightarrow s_{\mu B}$. According to definition of the pointcut, the target state $s$ in $P_B \bowtie P_{pc}$ identified by $\pi$ is equivalent with $s_{\mu B}: s \sim s_{\mu B}$.

3.1.2 Modeling advices and aspects

An advice can be an observer that only inserts extra actions when certain event occurs but does not change the trace of the base system, or an assistant that makes the base system recover to another state [8]. Tracing, debugging, and logging [11] are the typical observer examples, whereas the access control [12] and recovery aspects [10] are the assistant examples.

Let $WT = \{\text{bef-seq, aft-seq, choice, bef-branch, aft-branch}\}$ be the set of predefined advice types, in which the bef-seq and aft-seq type of advices insert extra actions before or after a transition join point, the choice type of advices make the base system go into another state join point, whereas the bef-branch and aft-branch type of advices execute advice actions before or after a transition join point and then make the base system go into a state join point conditionally.

Definition 16  Advice: An advice is a tuple $Ad = (\text{type}, S, L, T, s_{init}, S_{final})$, where type $\in WT$ is the advice type, $(S, L, T, s_{init}, S_{final})$ is an extended LTS to specify the advice behavior. $S_{final} = \{s \mid s \in S \land \exists s' \in S \cdot (s, a, s') \in T\}$ is a terminate state set that satisfies $|S_{final}| = 1$ for type $\in \{\text{bef-seq, aft-seq, choice}\}$, and $|S_{final}| = 2$ for type $\in \{\text{bef-branch, aft-branch}\}$. Moreover, the extended LTS $(S, L, T, s_{init}, S_{final})$ is deterministic and all states except the final states are progressive.

Definition 17  Aspectlet: An aspectlet is a tuple $(P_{pc}, \pi, P_{ad})$ in which $P_{pc}$ is a pointcut, $P_{ad}$ is an advice, $\pi$ is a trace of the base system to identify a state join point which is null if $P_{ad}.\text{type} = \{\text{bef-seq, aft-seq, choice}\}$ and not null otherwise.

Definition 18  Aspect: An aspect $A$ is a set of aspectlets: $A = \{e \mid e$ is an aspectlet$\}$.

3.2 Modeling Aspect Weaving

Let $P = (S, L, T, f, s_{init})$ be a $\lambda$LTS. We define an auxiliary operator $\text{select}(P)$ to get all transition join points: $\text{select}(P) = \{t \mid t \in T \land f(t) = 1\}$.

In addition, given two state set $S_1$ and $S_2$ with distinct states, define a function:

$$\text{Union}(S_1, S_2, \{s_{11} \rightarrow s_{21}, \ldots, s_{1n} \rightarrow s_{2n}\}) = \{s \mid \{s_{11}, \ldots, s_{1n}\} \subseteq S_1 \land \{s_{21}, \ldots, s_{2n}\} \subseteq S_2\} \land (s = s_{11}$ for $s_{11} \in S_1 - \{s_{11}, \ldots, s_{1n}\} \land (s = s_{21}$ for $s_{21} \in S_2) \land (s = s_{22}$ for $s_{22} \in S_2$ for $s_{11} \in \{s_{11}, \ldots, s_{1n}\}) \land x \in \{1, \ldots, n\})$$
to return a “union” state set of $S_1$ and $S_2$. In the “union” state set, the original state $s_{11}$, ..., $s_{1n}$ in $S_1$ unite with the state $s_{21}$, ..., $s_{2m}$ in $S_2$ as “$s_{11}$$^{′′}$”, “$s_{2m}$$^{′′}$” respectively. Other states in $S_1$ or $S_2$ add to the union state independently.

Firstly, define a micro weaving operator, $\downarrow$, to insert an advice to a join point in a $\pi$LTS.

**Definition 19** Let $P_M = (S_M, L_M, T_M, f_M, s_{initM})$ be a $\pi$LTS, $t_{jp} = (s_{jp}, \ell_{jp}/JP, s_{jp}^{′}) \in T_M$ be a transition join point, $P_{ad} = (type_{ad}, s_{ad}, L_{ad}, T_{ad}, s_{initad}, S_{finalad})$ be an advice, $\pi$ be a trace on $P_M$. In addition, let $s_{jp}^{′}$ be the state join point identified by $\pi, P_M \rightarrow s_{jp}^{′′}$. Then, the micro weaving of an aspectlet is implemented through inserting its cloned ad-

$\downarrow$. The weaving of an aspect is a process of weaving all its aspectlets.

**Definition 20** Let $AL = (P_{pc}, \pi, P_{ad})$ be an aspectlet, $P_B = (S_B, L_B, T_B, s_{initB})$ be the base
system. The operator, \( \angle_{AL} \), inserts \( AL \) to \( P_B \) and returns a LTS:

\[
P_B \angle_{AL} AL = \hat{\alpha}(\ldots((P_B \times P_{pc}) \angle (t_{p1}, \pi, \text{clone}(P_{ad}))) \angle (t_{p2}, \pi, \text{clone}(P_{ad}))) \angle \ldots \angle (t_{pn}, \pi, \text{clone}(P_{ad}))),
\]

where \( \text{select}(P_B \times P_{pc}) = \{t_{p1}, \ldots, t_{pn}\} \) \( (n \geq 1) \).

**Definition 21** Let \( P_B \) be the base system and \( A = \{AL_1, \ldots, AL_n\} \) \( (n \geq 1) \) be an aspect. The aspect weaving operator, \( \angle \), weaves \( A \) into \( P_B \) and returns a LTS:

\[
P_B \angle A = (\ldots(P_B \angle_{AL1} \ldots \angle_{ALn})).
\]

Consider the weaving of the bonus aspect. We define the bonus aspect as follows,

\[
\text{bonus} = \{<P_{PC}, \pi, P_{Ad}>\},
\]

where \( P_{PC} \) is the pointcut illustrated in Fig. 3, and \( P_{Ad} = \{\text{bef-seq}, \{s_{init}\}, \{\text{bonus}\}, \{s_{init}, \text{bonus}, s_{final}\}\} \) is a bef-seq advice. According to Definition 21, the LTS resulted from weaving the bonus aspect is as shown in Fig. 6.

![Fig. 6. The LTS derived from weaving the bonus aspect.](image)

### 3.3 About the Aspect Weaving Operator

Theorem 1 below expresses that weaving over a well defined system preserves both determinism and progress, and Theorem 2 shows that the semantic equivalence is preserved when weaving an aspect.

**Theorem 1** Preservation of determinism and progress: Let \( P_B \) be a well-defined base system and \( A \) be an aspect. Then, \( P_B \angle A \) is also well-defined.

**Theorem 2** Preservation of equivalence: Let \( P_1 \) and \( P_2 \) be two base systems and \( A \) be an aspect for \( P_1 \) and \( P_2 \). Then \( P_1 \sim P_2 \Rightarrow P_1 \angle A \sim P_2 \angle A \).

Proofs for the two theorems are given in the appendix.

The determinism, progress and equivalence being preserved, the system resulted from weaving an aspect can combine (\( || \)) with other systems or weave (\( \angle \)) other aspects. Therefore, aspect weaving can be considered as a new operator on LTS. In contrast to
4. IMPLEMENTATION AND DISCUSSION

4.1 Implementation

A prototype weaving tool has been developed. The tool is based on the FSP (Finite State Processes) language. FSP is a process algebra style language introduced in Darwin ADL [13], which has an open tool-Labeled Transition System Analyzer (LTSA) [14] for the LTS evaluation and analysis.

The prototype weaving tool inputs a FSP LTS and an aspect specification, and then outputs the weaving result. As the output result is a legal FSP LTS, it can be pictured and analyzed in LTSA.

In FSP, a process \( P \) corresponds to a LTS in which the process \( P \) and its local processes compose the state set and process \( P \) corresponds to the initial state. A FSP process consists of one or more local processes which is separated by comma and ends with a period. A local process is defined using action prefix ("->"), choice ("|") and recursion. Fig. 7 is the FSP specification of the base system of the international travel agent example.

The aspect specification is an extension to FSP syntax. Fig. 8 describes the specification of the bonus aspect. In the figure, Aspect in the first line is a reserved word which is followed by the aspect id-Bonus. In the aspect body, an aspectlet \( A_1 \) is defined whose weaving type, pointcut and advice is Bef-Seq, PC and AD respectively. Then, following the declaration of aspectlet \( A_1 \), the LTSs of \( PC \) and \( AD \) are defined.

\[
\begin{align*}
B &= (\text{login} \to B_1), \\
B_1 &= (\text{book} \to B_1| \text{queryInfo} \to B_1| \text{logout} \to B)
\end{align*}
\]

Fig. 7. The base system specification of the travel agent example.

\[
\begin{align*}
\text{Aspect Bonus} \\
&\text{DEF } A_1 = <\text{BeforeSeq, PC, AD}> \\
&PC = (\text{book} \to PC_1 | \text{queryInfo} \to PC | \text{logout} \to PC), \\
&PC_1 = (\text{book} \to PC_1 | \text{logout/JP} \to PC | \text{queryInfo} \to PC_1), \\
&AD = (\text{bonus} \to \text{EndAD}). \\
\end{align*}
\]

Fig. 8. Specification of the bonus aspect.

\[
\begin{align*}
B_{PC} &= (\text{login} \to B_{PC}), \\
B_{1_{PC}} &= (\text{logout} \to B_{PC_1} | \text{queryInfo} \to B_{1_{PC_1}}), \\
B_{1_{PC_1}} &= (\text{book} \to B_{1_{PC_1}} | \text{queryInfo} \to B_{1_{PC_1_1}} | \text{bonus} \to \text{Ad_1_New1}), \\
\text{Ad_1_New1} &= (\text{logout} \to B_{PC}).
\end{align*}
\]

Fig. 9. The result LTS generated by the weaving tool.

\[
\begin{align*}
\text{Aspect Bonus} \\
&\text{DEF } A_1 = <\text{BeforeSeq, PC, AD}> \\
&PC = (\text{book} \to PC_1 | \text{queryInfo} \to PC | \text{logout} \to PC), \\
&PC_1 = (\text{book} \to PC_1 | \text{logout/JP} \to PC | \text{queryInfo} \to PC_1), \\
&AD = (\text{bonus} \to \text{EndAD}). \\
\end{align*}
\]

Fig. 10. The result LTS graph pictured in LTSA.
Fig. 9 shows the result LTS specification generated by the prototype tool after weaving the bonus aspect, which is pictured in LTSA as shown in Fig. 10.

Now, we are developing more examples on the prototype tool. The current definitions offer a good support for implementing the aspects we need.

4.2 Discussion

4.2.1 Supports for general aspects

The approach applies to not only stateful aspects but also general aspects.

Let $P_B = (S_B, L_B, T_B, s_{initB})$ be a base system and $P$ be a LTS that has only one transition:

$$\frac{\ell}{JP} P \rightarrow P$$

for any action $\ell \in L_B$. It can be seen easily that $P$ is a pointcut on $P_B$. In the composition $P \approx P_B$, all the $\ell$-labelled transitions become transition join points, where advices can apply. Therefore, the approach can describe aspects that insert extra actions when action $\ell$ occurs.

4.2.2 Limitations

Based on the proposed pointcut model, the approach can identify the join points where a stateful aspect would apply. The weaving is a transformation on the composition of the base system and the pointcut. Therefore, the approach allows stateful aspects to be woven statically. The weaving result is formal which thus makes it easy for future analysis and reasoning.

However, in cases that join points cannot be identified statically or multiple pointcuts should be applied to a base system, the approach shows limitations in efficiency and scale. For example, in order to weave a bonus aspect that applies after $N$ bookings having been performed, the approach would need to add $N$ states to the base system. Moreover, the application of multiple pointcuts to a base system may cause the number of states to increase exponentially with the number of pointcuts.

One way to overcome the limitation is through introducing state variables to the pointcut model to monitor the occurrence of join points, as do in other programming languages supporting stateful aspects [1-3]. To do so, the pointcut model should be extended to add predicates that can express the side effects of the occurrence of join points on the state variables to transitions. Meanwhile, the model of the base system should be extended to add predicates on the pointcut state to the candidate join points to express the activation conditions of aspects. Moreover, to harmonize and schedule the execution of the base system and the aspects, a monitor should be introduced. The monitor would receive actions from the base system, send them to all the monitoring aspects, and schedule the running of the base system and the advices when a pointcut is identified.

By adding such supports, the proposed approach can weave stateful aspects whose join points need to be identified dynamically and would scale with complex stateful aspects. Furthermore, the monitor can deal with the logic relationship and even the interferences among aspects. However, such a weaving needs complex modeling mechanisms.
Meanwhile, the whole behavior after aspect weaving can not be understood easily, which makes it difficult for the analysis and reasoning on the weaving.

Therefore, for stateful aspects whose pointcuts can be identified statically, the proposed approach is suitable and economic. To cover other cases where weaving cannot be efficiently performed at compile time, the approach should be extended to add the support stated above. Furthermore, to make it feasible for analysis and reasoning, a weaving semantics should be built. The weaving semantics would interpret the logic relationship between the behavior that before and after aspect weaving, and the scheduling relationship between the base system, the monitor and the stateful aspect during the weaving. To date we have developed the weaving semantics for the proposed approach in the paper. Based on the former work, we believe the semantics for the extended approach can be built.

In addition, the prototype tool can only support aspect weaving on the primitive process consists of local processes defined by action prefix (“->”) and choice (“|”) to date. To weave a composite process or other complicated FSP processes, an available method is through first getting the transition specification of the composite process (or other complicated processes) from the LTSA and then weaving it and an aspect.

### 4.2.3 The aspect interference problem

A key point when dealing with aspects is the notion of interference. In our approach, as aspectlet is the unit of aspect weaving, interference may occur between aspectlets.

Aspect interference can be classified as syntactic interference and semantic interference. Given the base system $P$ and two aspectlets $AL_1$ and $AL_2$, if $P \preceq_{AL} AL_1$ and $P \preceq_{AL} AL_2$ are feasible but ($P \preceq_{AL} AL_1 \preceq_{AL} AL_2$ or ($P \preceq_{AL} AL_2 \preceq_{AL} AL_1$)) are not, then $AL_2$ syntactically interferes with $AL_1$ (or $AL_1$ syntactically interferes with $AL_2$). Such interference often occurs when the pointcut of aspectlet $AL_2$ is disabled by aspectlet $AL_1$ (or otherwise).

If ($P \preceq_{AL} AL_1 \preceq_{AL} AL_2$ or ($P \preceq_{AL} AL_2 \preceq_{AL} AL_1$)) is feasible but the weaving of $AL_2$ changes the weaving effect of aspectlet $AL_1$ (or otherwise), then $AL_2$ semantically interferes with $AL_1$ (or otherwise). For example, some properties $\phi$ hold for ($P \preceq_{AL} AL_1$), but $\phi$ do not hold for ($P \preceq_{AL} AL_2$).

Syntactic interference can be detected by the weaving operation $\preceq_{AL}$, while semantic interference can be checked through analyzing ($P \preceq_{AL} AL_1 \preceq_{AL} AL_2$ (or ($P \preceq_{AL} AL_2 \preceq_{AL} AL_1$))) on tools such as LTSA.

### 5. RELATED WORK

Trace-based aspects [3] use finite state automata to recognize the set of all authorized sequences of events. To integrate the automata, the weaver first abstracts the base program into a graph and then instruments the graph with additional structures to implement the state transition of the underlying automaton. Finally, the graph is translated back into a program. Declarative Event Patterns (DEPs) [5] allow a developer to describe a sequence of events in the system execution as an event pattern. The weaver first generates an event parser, a pushdown automaton, to recognize each declarative event pattern. Then, the instrumentation that will announce the occurrence of particular events at run-time to the event parsers are coded, and the specification of the join points where the instrumentation must be in-
jected are specified.

As the differences between the automata and the program, the two approaches have to abstract events used in the automata from the program and inserts extra instrumentation to hook the two mechanisms. We avoid this by adopts the same mechanisms for the pointcut as that for the base system. In addition, trace-based aspects and DEPs can only model observer aspects, whereas our approach can model observer and assistant aspects.

The approach proposed in [10] is similar to ours. As a matter of fact, we get inspiration from the approach. The approach uses Mealy Automata to model the base system and the pointcut. After the parallel composition of the pointcut and the base system, all join points are labeled. An advice adds a transition from a join point to a target state. The approach can implement the ToInit aspects and Recovery aspects. The approach in [10] applies to reactive systems. Mealy Automata is an ideal tool for modeling reactive systems.

By contrast, our approach applies to general systems and LTS is an ideal tool for modeling the software system behavior. In addition to the two kinds of aspects in [10], our approach can model branched aspects.

In [9], the author introduce a pointcut language to express aspect activation depending on past and current state of the workflow execution and propose a weaving strategy based on finite state automata in order to transparently weave history tracking code in the workflow.

In [15], the author presents an algorithm to weaving a Hierarchical Message Sequence Chart (HMSC) and a behavior aspect expressed with scenarios. An aspect consists of a pointcut that specifies certain behavior occurred in the HMSC and an advice that will replace the behavior identified by the pointcut. As the pointcut is interpreted as a predicate over the base behaviors, the aspect proposed in [15] can be seen as a stateful aspect. The weaving algorithm takes into account the compositional semantics of HMSCs, which is similar to our weaving that based on the compositional semantics of LTSs. However, the aspect is a kind of replacement aspects, which is different from our sequential, choice and branched aspects that insert extra actions to the join points.

In [4], the author propose that aspect-oriented modeling languages can capture stateful aspects in terms of module specifications as opposed to implementation-dependent sequence of events in order to achieve better decoupling between aspects and base modules. However, the approach is not formalized.

Other work on design-level aspect weaving include: a signature-based composition approach proposed in [16] and relationship-based composition approach proposed in [17]. These approaches base on UML, which is informal. Moreover, they do not provide supports for stateful aspects.

6. CONCLUSION AND FUTURE WORK

We have proposed to model stateful aspects using LTS. We introduce an extended LTS pointcut mechanism to trace the base system and identify the activation condition of advices. The advice weaving is a transformation over the composition of the pointcut and the base system. The weaving operators preserve the semantic equivalence of systems. Moreover, we have developed a FSP based prototype weaving tool. The output woven results can be analyzed on LTSA.
The approach requires that the composition of pointcut and the base system should be semantically consistent with the base system. Sometimes, it may be not easy to specify such pointcuts. In the future, we shall further research how to derive a pointcut from a trace. Besides, the future work include: extending the approach to support dynamic weaving, developing more examples on the prototype tool and exploring the aspect interference issues.

REFERENCES

Theorem 1  Preservation of determinism and progress: Let $P_B$ be a well-defined base system and $A$ be an aspect. Then, $P_B \triangleleft A$ is also well-defined.

**Proof:** Let $AL = \langle P_{PC}, \pi, P_{ad} \rangle \in A$ be an aspectlet of aspect $A$. Let’s first consider the determinism and progress of $P_B \triangleleft AL$.

As it is well-defined, $P_B$ satisfies determinism and progress. Meanwhile, according to their definitions, the pointcut $P_{PC}$ satisfies determinism and progress and the advice $P_{ad}$ satisfies determinism and all states except the terminate states satisfy progress. Let $P_M = P_B \triangledown P_{PC}$ be the composition of $P_B$ and $P_{PC}$. As $P_M$ is equivalent with $P_B \triangledown (P_{PC} \triangledown P_{ad})$, $P_M$ also satisfies determinism and progress.

The weaving $P_B \triangleleft AL$ is implemented by inserting the clone of its advice $P_{ad}$ into $P_M$ according to the rules set by the corresponding advice type. Now, we prove that the weaving of a bef-seq type of advice $P_{ad}$ can preserve determinism and progress. Other types of advices can be proved by analogy.

According to the definition of operator, $\triangledown$, the weaving of a bef-seq type advice $P_{ad}$ is through:

- first introducing a clone $(P_{ad})$ and unite its initiate state $s_{initial}$ with $s_{jp}$ that is the source state of transition join point $t_{jp} = (s_{jp}, \ell_{jp} / JP, s_{jp}^{'})$ through the operation: $\text{Union}(S_{ad}, S_M, \{s_{initial} \rightarrow s_{jp}\})$, 
- and then deleting the transition $t_{jp} = (s_{jp}, \ell_{jp} / JP, s_{jp}^{'})$, and adding a new $\ell_{jp}$-labelled transition from the terminate state $s_{adfinal}$ of clone($P_{ad}$) to the target state of $t_{jp}$: $s_{adfinal}, \ell_{jp}, s_{jp}^{'}$.

As $P_{ad}$ is deterministic, clone $(P_{ad})$ is also deterministic. Although state $s_{initial}$ is united with $s_{jp}$, it does not cause the non-determinism of state $s_{jp}$ because the action names in clone $(P_{ad})$ differ from those in the base system $P_B$. Similarly, as transition $t_{jp} = (s_{jp}, \ell_{jp} / JP, s_{jp}^{'})$ has been deleted, the new introduced transition $(s_{adfinal}, \ell_{jp}, s_{jp}^{'})$ can not cause the non-determinism of state $s_{jp}^{'}$. In addition to states $s_{jp}$ and $s_{jp}^{'}$, other states in $P_B \triangleleft AL$ come from the base system $P_B$ and clone($P_{ad}$). As the action set of $P_B$ and clone ($P_{ad}$) are different, the determinism of these states stay unaltered.

As $P_M$ is progressive, all its states are progressive. In clone ($P_{ad}$), all states except the terminate state $s_{adfinal}$ are progressive. The operation $\text{Union}(S_{ad}, S_M, \{s_{initial} \rightarrow s_{jp}\})$ makes
all states except the terminate state $s_{\text{sadfinal}}$ of clone $(P_\text{Ad})$ preserve progress in $P_B \preceq_A L. AL$, while the new introduced transition $(s_{\text{sadfinal}}, l|_m, s'_m)$ makes the terminate state $s_{\text{sadfinal}}$ become progressive in $P_B \preceq_A L. AL$. Besides these states, other states in $P_B \preceq_A L. AL$ come from $P_B$ and preserve the progress thereby.

In conclusion, the weaving of any aspectlet $AL$ of aspect $A$ preserves the determinism and progress. Therefore, the aspect weaving $P_B \preceq A$ also preserves the determinism and progress. \hfill \square

**Theorem 2** Preservation of equivalence: Let $P_1$ and $P_2$ be two base systems and $A$ be an aspect for $P_1$ and $P_2$. Then

$$P_1 \sim P_2 \Rightarrow P_1 \preceq A \sim P_2 \preceq A.$$ 

**Proof:** If we prove that $P_1 \sim P_2 \Rightarrow P_1 \preceq_A L. AL \sim P_2 \preceq_A L. AL$ holds for any aspectlet $AL = \langle P_{pc}, \pi, P_{AL} \rangle \in A$, we can conclude that $P_1 \preceq A \sim P_2 \preceq_A A$ holds.

Let $P_{1} = (S_{1}, L_{1}, T_{1}, f_{1}, s_{\text{init1}})$ and $P_{2} = (S_{2}, L_{2}, T_{2}, f_{2}, s_{\text{init2}})$ be the two base systems, $P_{pc} = (S_{pc}, L_{pc}, T_{pc}, f_{pc}, s_{\text{adpc}})$ be the pointcut on $P_{1}$ and $P_{2}$, $P_{M_{1}} = P_{1} \propto P_{pc} = (S_{M_{1}}, L_{M_{1}}, T_{M_{1}}, f_{M_{1}}, s_{\text{initM_{1}}})$ be the composition on $P_{1}$ and $P_{pc}$ and $P_{M_{2}} = P_{2} \propto P_{pc} = (S_{M_{2}}, L_{M_{2}}, T_{M_{2}}, f_{M_{2}}, s_{\text{initM_{2}}})$ be the composition on $P_{2}$ and $P_{pc}$.

According to $P_{1} \sim P_{2}$ and $\partial(P_{M_{1}}) \sim P_{1}$ and $\partial(P_{M_{2}}) \sim P_{2}$, we get $\partial(P_{M_{1}}) \sim \partial(P_{M_{2}})$. This means that:

For any state $s_{M_{1}} \in S_{M_{1}}$ and transition $(s_{M_{1}}, \ell, s'_{M_{1}}) \in T_{M_{1}}$, there exist state $s_{M_{2}} \in S_{M_{2}}$ and transition $(s_{M_{2}}, \ell, s'_{M_{2}}) \in T_{M_{2}}$ that satisfy

$$s_{M_{2}} \sim s_{M_{1}} \wedge s'_{M_{2}} \sim s'_{M_{1}}. \quad (1)$$

Similarly, according to the rules of the composition operator, $\propto$, the function $f$ is also evaluated with the transition composition. Therefore, we can get that:

For any state $s_{M_{1}} \in S_{M_{1}}$ and transition $(s_{M_{1}}, \ell|_M, s'_{M_{1}}) \in T_{M_{1}}$, there exist state $s_{M_{2}} \in S_{M_{2}}$ and transition $(s_{M_{2}}, \ell|_M, s'_{M_{2}}) \in T_{M_{2}}$ that satisfy

$$s_{M_{2}} \sim s_{M_{1}} \wedge s'_{M_{2}} \sim s'_{M_{1}}. \quad (2)$$

According to Eqs. (1) and (2), we can get that $\partial(P_{M_{1}}) \sim \partial(P_{M_{2}})$ and $\text{select}(P_{M_{1}}) \sim \text{select}(P_{M_{2}})$, i.e. the states incident to the transition join points in $P_{M_{1}}$ and $P_{M_{2}}$ are equivalent.

As the weaving of aspectlet $AL$ is a transformation over $P_{M_{1}}$ and $P_{M_{2}}$, we will prove that weaving each type of advices preserves the equivalence of $P_{M_{1}}$ and $P_{M_{2}}$.

For a sequential or choice advice, as the weaving executes the same operations on the equivalent states incident to the transition join points in $P_{M_{1}}$ and $P_{M_{2}}$, it does not violate the equivalence of $P_{M_{1}}$ and $P_{M_{2}}$.

For a branched advice, let state $s_{M_{1}}'$ and $s_{M_{2}}'$ be the state join point identified by the trace $\pi$ in $P_{M_{1}}$ and $P_{M_{2}}$ respectively: $P_{M_{1}} \pi s_{M_{1}}', P_{M_{2}} \pi s_{M_{2}}'$. According to $\partial(P_{M_{1}}) \sim \partial(P_{M_{2}})$, we can get that $s_{M_{1}}' \sim s_{M_{2}}'$. The branched advice weaving executes the same operations on the equivalent states incident to the transition join points in $P_{M_{1}}$ and $P_{M_{2}}$ and equivalent state join point $s_{M_{1}}'$ and $s_{M_{2}}'$, it does not violate the equivalence of $P_{M_{1}}$ and $P_{M_{2}}$ thereby. \hfill \square
Chun-Hua Yang (楊春花) received her Ph.D., M.S. and B.S. degrees in Department of Computer Science and Technology from Shandong University, China, in 2010, 2002 and 1995, respectively. She is an Assistant Processor of School of Information Science and Technology of Shandong Light Industry, Shandong, China. Her research interests include aspect oriented technologies, business process, and web services.

Hai-Yang Wang (王海洋) has been a Professor and Ph.D. supervisor of the Department of Computer Science and Technology of Shandong University, Shandong, China. His research interests include software and data engineering, computer supported cooperative work (CSCW) and business process management (BPM).