Detecting and Evaluating Semantic Influences of Aspect Weaving in Aspect Oriented Models

Chunhua Yang
School of Information, Shandong Institute of Light Industry, Jinan, China
Email: jnych@126.com

***Abstract*—Weaving an aspect may introduce undesired impacts on behavior of the base model or other afore-woven aspects. An approach to detect and evaluate the semantic influences of an aspect weaving on a pattern is presented. The pattern specifies a desired behavior that should stay unaltered or occur in the woven model. The detection and evaluation are based on the semantic relationship between a pattern and its projection that represents its actual semantics in the woven model. Five types of aspect weaving influences are identified. The approach has the Process Algebras (PA) as the underlying formalisms and has been implemented by mapping to the Concurrency Workbench (CWB) tool. An example shows the effectiveness.**

***Index Terms*—aspect weaving, influences, aspect interaction, aspect interference, aspect-oriented software development**

I. INTRODUCTION

With the aim of reducing complexity and enhancing maintainability, aspects have been extended to software modeling stage to modularize crosscutting concerns.

An aspect-oriented model generally consists of a base model and aspect models. The base model encapsulates the main functionality of the software system, while the aspect models encapsulate concerns that crosscut the main functionality. Aspect models combine with the base model through a weaving process. Many approaches have been proposed to specify aspect oriented models using different notations, i.e. UML statecharts[1, 20], UML activity diagrams[2, 3], Visual Contract Language (VCL) [4], process algebra[18] etc.

However, the separate development of aspects may introduce semantic problems in the process of weaving [5]. The weaving of a new aspect may introduce undesired impacts on behavior of the base model or an afore-woven aspect, or result in emerging behaviors that conflict with some intended behaviors, which threaten the reliability of the software.

Many attempts have been made to detect whether there exist aspect influences by checking aspect-oriented models against desired properties [6, 7, 8, 9, 10, 19]. However, it has been neglected that how an aspect affects the base model or other aspects.

This work was supported by the Natural Science Foundation of Shandong Province China under Grant No.ZR2011FQ017 and Grant No.ZR2012FM032, and Shandong Engineering Laboratory of Key Technology for Flow Process Enterprise Information Integration.

Corresponding author: Chunhua Yang

We present an approach to evaluate whether and how an aspect weaving influences a pattern. The pattern specifies the behavior of the base model or an afore-woven aspect, or an expected behavior in the woven model. Its underlying formalisms are PA algebras.

The rest of the paper is organized as follows: In section 2, a motivation example is introduced. In section 3, the approach is presented. Thereafter, section 4 describes the implementation. Section 5 briefly describes the related work and section 6 concludes.

II. A MOTIVATION EXAMPLE

When weaving an aspect into a well-designed system, it is generally expected that certain behaviors occur or stay unaltered in the augmented system.

Consider a property listing subsystem (or PLS for short) in an online real estate system (see Fig.1). When a seller has a house for sale, he or she should list the property information for publicity through a broker. The work flow is as follows: A broker inputs the property information to the property listing system. Then system verifies the correctness of the provided information. If the verification result is ok, the information is saved to the DB for listing publicity and the broker receives an accepted response. Otherwise, the broker receives a refused response.

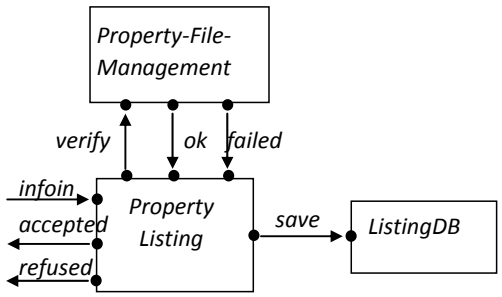

Figure 1. The property listing system.

Now, three aspects *Timing*, *Auth*, and *Log* are designed to augment the system with new requirements. The *Timing* aspect counts the average time of the process of successful property listing, while the *Auth* and *Log* aspects add the authentication and logging functions to the PLS. Moreover, it is desired that:

R1. The original system behavior should be preserved after the *Timing* aspect is woven; and

© 2013 ACADEMY PUBLISHER
doi:10.4304/jsw.8.11.2675-2681

R2. Before the broker inputs the property information to the property listing system, he or she should be authenticated; and

R3. Each time the broker is authenticated, it should be logged.

The expectation **R1** implies that behavior of the base model should stay unaltered after weaving the *Timing* aspect, while **R2** or **R3** represents a desired behavior that should occur in the augment system after weaving the *Auth* aspect and the *Log* aspect.

To check whether the augment system satisfies these expectations, influences of weaving the three aspects on the behavior of the base model or the desired behaviors (**R2** and **R3**) should be evaluated.

III. THE APPROACH

Process Algebra [11] is a popular tool for modeling software system [12]. Moreover, its notion of behavior equivalence makes is feasible for comparing the semantic relationship between two models. Therefore, our approach has Process Algebra as the underlying formalism.

A. Brief Introduction to PA

This section briefly introduces concepts related to Process Algebra (PA in short) [11].

Assuming an infinite collection A of *names*, the set $\bar{A}=\{\bar{a} \mid a\in A\}$ is the set of *complementary names* (or co-names for short). Let $Act = A\cup\bar{A}\cup\{\tau\}$ be the set of actions, where action τ is a distinguished unobservable action or inner action.

Definition 1 The collection of process terms of the Process Algebra is generated by the following grammar:

$P ::= 0 \mid a.P \mid P+P \mid P|P \mid P\backslash L \mid P[f] \mid K$

where a is an action in Act, f: $Act\rightarrow Act$ is a relabelling function, $L\subseteq Act-\{\tau\}$ is a set of labels, and K is a constant possessing a defining equation of the form $K \triangleq P$.

In the syntax above, the null term "0" is the term that cannot execute any action. The action prefix operator "$a.P$" denotes the sequential composition of an action and a term. The hiding operator "$\backslash L$" makes the executed actions belonging to L unobservable. The alternative composition operator "+" expresses a nondeterministic choice between two terms. The parallel composition operator "|" expresses the concurrent execution of two terms according to the following synchronization discipline: two (observable) actions can synchronize iff they are a pair of complementary actions.

The application of the semantics for PA is a Labeled Transition System, where states are in correspondence with process terms and transitions are labeled with actions.

Definition 2 A labeled transition system (LTS) is a tuple (S, Act, T, s_{init}), where S is a set of states which include an initiate state s_{init}, Act is a set of actions, $T\subseteq S\times Act\times S$ is a transition relation.

We write $s \xrightarrow{a} s'$ for a transition $(s, a, s')\in T$.

A *trace* of a process P is a sequence $a_1 \cdots a_k \in Act^*(k \geq 0)$ such that there exists a sequence of transitions $P = P_0 \xrightarrow{a_1} P_1 \xrightarrow{a_2} \cdots \xrightarrow{a_k} P_k$ for some $P_1, \ldots, P_k$. We write $Traces(P)$ for the collection of traces of P.

Definition 3 [Trace equivalence] Two processes and Q are trace equivalent if, and only if, they afford the same traces, i.e. $Traces(P) = Traces(Q)$.

Let P and Q be processes. We write $P \overset{\varepsilon}{\Rightarrow} Q$ iff there is a (possibly empty) sequence of τ-labeled transitions that leads from P to Q. (If the sequence is empty, then $P = Q$.)

For each action a, we write $P \overset{a}{\Rightarrow} Q$ iff there are processes P' and Q' such that

$P \overset{\varepsilon}{\Rightarrow} P' \xrightarrow{a} Q' \overset{\varepsilon}{\Rightarrow} Q$.

For each action a, we use $\hat{a}$ to stand for ε if $a = \tau$, and for a otherwise.

Definition 4 [Weak Bisimulation] A binary relation R over the set of states of an LTS is a *weak bisimulation* iff whenever $s_1 \, R \, s_2$ and a is an action:

- if $s_1 \xrightarrow{a} s_1'$, then there is a transition $s_2 \overset{\hat{a}}{\Rightarrow} s_2'$ such that $s_1' \, R \, s_2'$;
- if $s_2 \xrightarrow{a} s_2'$, then there is a transition $s_1 \overset{\hat{a}}{\Rightarrow} s_1'$ such that $s_2' \, R \, s_1'$.

Two states s and s' are observationally equivalent (or weakly bisimilar), written $s \approx s'$, iff there is a weak bisimulation that relates them.

B Definition of Models

As PA is a powerful tool for modeling software system behavior[12], we define the base model and advice models as labeled transition systems.

Definition 5 A base component $e=(S, Act, T, s_{init})$ is a LTS.

Definition 6 A base model $m=(S, Act, T, s_{init})$ is a LTS that is a parallel composition of the base components $e_1, \ldots, e_n$, i.e. $m = e_1\|\ldots\|e_n (n\geq 1)$.

Definition 7 [Join Point] Given a base component $Bc=(S, Act, T, s_{init})$, a join point jp is a state $s\in S$ or a transition $(s, a, s')\in T$ where a is an observable action.

If a join point is a transition, we call it *transition join point*. Otherwise, we call it *state join point*.

We identify three advice types:

- *sequential*: A sequential advice applies to a transition join point. The advice begins to run when the join point is active. Until the advice arrives at a final state, the join point executes (or continues) immediately.
- *branched*: A branched advice applies to both a transition join point and a state join point. The advice executions before (or after) the transition join point. If the conditions set by the advice satisfy, the advice runs to a *true* final state and then the transition join point executes immediately. Otherwise, the advice runs to a *false* final state and the state join point becomes active.
- *synchronized*: A synchronized advice is an advice applies to transition join points. Execution of a synchronized advice and the base model are concurrent according to the synchronization rules: a

© 2013 ACADEMY PUBLISHER

transition join point synchronizes with a predefined observable transition of the advice.

Definition 8 [Advice] An advice ad=($Type$, $Beha$) consists of a type $Type$ and a behavior $Beha$, in which $Type$=($sequential$, $branched$, $synchronized$) and $Beha$=(S, Act, T, s_{init}, S_{final}) is an extended LTS. An extended LTS introduces a final state set S_{final} to the base LTS. A final state is a state from which no transitions direct. Moreover, $Beha.S_{final}=\emptyset$ for $Type$= $synchronized$, and $Beha.S_{final}\neq\emptyset$ for $Type$=($sequential$, $branched$).

Definition 9[Aspect] An aspect $Ap=\{(jp_1,ad_1),\ldots,(jp_n, ad_n)\}(n\geq 1)$ is a set of pairs of an advice and a join point set where the advice would apply.

C. Definition of Aspect Weaving

According to the advice types, we define five operators ($\angle^{bef}_{seq}$, $\angle^{aft}_{seq}$),($\angle^{bef}_{brach}$, $\angle^{aft}_{brach}$),($\angle_{syn}$) to weave *sequential*, *branched*, and *synchronized* advices, respectively. Moreover, we use $M=\angle_{op}(Bc, jp\_x, ad)$ to denote the advice weaving, where:

- M is a LTS that represents the composition model resulted from the weaving, $\angle_{op}$ is the weaving operator, ad is an advice, Bc is the base component that jp_x belongs to; and
- $jp\_x = s \xrightarrow{a} s'$ stands for a transition join point for a sequential advice; and

  $jp\_x = \{s \xrightarrow{a} s', s_{false}\}$ stands for a pair of a transition and a state join point for a branched advice; and

  $jp\_x= \{<s_1 \xrightarrow{a_1} s_1', t_1 \xrightarrow{b_1} t_1'>,\ldots,<s_n \xrightarrow{a_n} s_n', t_n \xrightarrow{b_n} t_n'>\}$ $(n\geq 1)$ stands for a set of pairs of a join point and the corresponding synchronized transition in ad for an interactive synchronized advice.

Fig.2 illustrates the two types of sequential advice weaving. Formally, let $M=\angle^{bef}_{seq}(Bc, s \xrightarrow{a} s', ad)$, then:

- $M.S=Bc.S \cup\{s_{new}\}\cup ad.Beha.S$, $M.s_{init}= Bc.s_{init}$, $M.Act = Bc.Act \cup ad.Act$, and
- $M.T=(Bc.T-\{s \xrightarrow{a} s'\})\cup\{s_{new} \xrightarrow{a} s', s \xrightarrow{\varepsilon} t, t_{final} \xrightarrow{\varepsilon} s_{new}\} \cup ad.Beha.T$.

The process of the after-sequential weaving is similar to the before-sequential weaving, which is as shown in Fig.2(3).

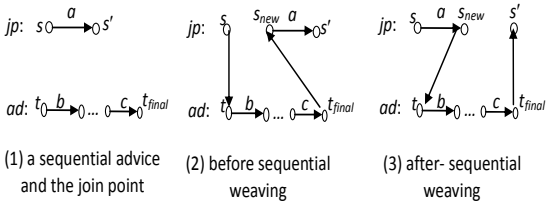

Figure 2. The sequential weaving.

Given a branched advice ad, which has a *true* final state t_{final}' and a *false* final state t_{final}'', the before-branched and after-branched advice weaving are depicted in Fig.3(2) and Fig.3(3) respectively. Take the before-branched weaving for instance(see Fig.3(2)). Let $M=\angle^{bef}_{brach}(Bc, \{s \xrightarrow{a} s', s_{false}\}, ad)$, then:

- $M.S = Bc.S \cup \{s_{new}\} \cup ad.Beha.S$, $M.s_{init} = Bc.s_{init}$, $M.Act = Bc.Act\cup ad.Act$, and
- $M.T=(Bc.T-\{s \xrightarrow{a} s'\})\cup\{s_{new} \xrightarrow{a} s', s \xrightarrow{\varepsilon} t, t_{final}' \xrightarrow{\varepsilon} s_{new}, t_{final}'' \xrightarrow{\varepsilon} s_{false}\}\cup ad.Beha.T$.

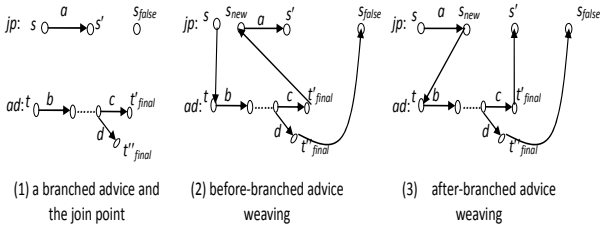

Figure 3. The branched weaving.

As for a synchronized advice, its weaving is implemented by parallel composition of PA. Formally, let ad is an advice and $M=\angle_{syn}(Bc, \{<s_1 \xrightarrow{a_1} s_1', t_1 \xrightarrow{b_1} t_1'>,\ldots,< s_n \xrightarrow{a_n} s_n', t_n \xrightarrow{b_n} t_n'>\}, ad)$, then:

- $Bc.Act=Bc.Act\cup\{b_1,\ldots, b_n\}$, $Bc.T=(Bc.T-\{ <s_1 \xrightarrow{a_1} s_1'\} -\ldots- \{s_n \xrightarrow{a} s_n'\})\cup\{ <s_1 \xrightarrow{a_1.b_1} s_1', s_n \xrightarrow{a_n.b_n} s_n'\}$, and
- $M = Bc| ad$.

Fig.4 illustrates the process of weaving synchronized advices.

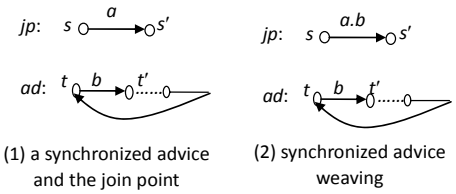

Figure 4. The synchronized weaving.

The weaving of an aspect is achieved by weaving its advices in turn. Given an aspect A and a base model B, we use $B\angle A$ to denote the process of weaving A to B in the following sections. As the woven model is a LTS, it can act as a base model which allows weaving another aspect.

D. Detection and Evaluation of Aspect Weaving Influences

A pattern specifies a certain behavior that is desired to be checked in the woven model.

Definition 10[Pattern] Given a woven model M'=(S', Act', T', s'_{init}), a pattern is a model M=(S, Act, T, s_{init}) which satisfies $Act\subseteq(Act'\cup\{\tau\})$.

As the action set of a base model is a subset of the woven model according to the definition of weaving, a base model is a pattern.

Definition 11[Projection] Given a pattern M=(S, Act, T, s_{init}) and a woven model M'=(S', Act', T', s'_{init}) which satisfies $Act\subseteq(Act'\cup\{\tau\})$, then the *projection* of M' on M, written as $\nabla^{M'}_{M}$, is a LTS (S_P, Act_P, T_P, $s_{init\text{-}P}$), where

© 2013 ACADEMY PUBLISHER

- $S_P = S'$, $Act_P = Act \cup \{\tau\}$, $s_{init\text{-}P} = s'_{init}$, and
- $T_P = \{(s_1', a, s_2') | (s_1', a', s_2') \in T' \wedge (\text{if } a' \in Act \cup \{\tau\} \text{ then } a = a', \text{otherwise } a = \tau)\}$.

In the projection, actions of the pattern are observable while others are invisible. Therefore, projection $\nabla_M^{M'}$ represents the actual behavior of the pattern M in the woven model M'.

We introduce a function *Ifl*: $LTS \times LTS \rightarrow$(*influence-free*, *may-influence-free*, *extension*, *narrowing*, *alteration*) to evaluate the semantic relationship between two LTSs.

Definition 12 Given two LTSs M and N which satisfies $M.Act \cup \{\tau\} = N.Act \cup \{\tau\}$, then the *Ifl*($M$, N) is:

- *influence-free* iff $M \approx N$; or
- *may-influence-free* iff *Traces*(M) = *Traces*(N); or
- *extension* iff *Traces*(M) $\subset$ *Traces*(N); or
- *narrowing* iff *Traces*(M) $\supset$ *Traces*(N); or
- *alteration* otherwise.

The influence of an aspect weaving on a pattern is evaluated according to the semantic relationship between the pattern and its projection in the woven model. As a pattern and its projection are LTSs, their semantic relationship is evaluated by the function *Ifl*.

Definition 13 [Aspect-Weaving Influence] Given a pattern M, an aspect weaving $M' = B \angle A$ of base model B and aspect A, and the projection $\nabla_M^{M'}$, then the influence of the aspect weaving on the pattern M is *Ifl*(M, $\nabla_M^{M'}$).

According to the definition, if only a desired behavior can be specified as a pattern, the influence of an aspect weaving on it can be detected. For example, if let the pattern $M=B$, the weaving influence on the base model B can be evaluated by *Ifl*(B, $\nabla_B^{B \angle A}$). Meanwhile, let the pattern $M = \nabla_{A_1}^{B \angle A_1}$, we can evaluate the influence of weaving another aspect A_2 on the afore-woven aspect A_1according to *Ifl*(M, $\nabla_M^{(B \angle A_1) \angle A_2}$).

IV. IMPLEMENTATION

In this section, the approach is implemented by mapping to a popular verification tool- the Concurrency Workbench (CWB) [13], and the example introduced in section 2 is implemented.

A. Mapping the Approach to the CWB

The CWB is an automated tool that helps in the manipulation and analysis of concurrent system specifications [13], in which a variety of equivalence relationships are supported. As the main modeling language used by the CWB is the process calculus CCS [11], it is convenient to mapping our approach to the tool.

We map our approach to the CWB according to the following rules:

- The base components, the base model, the patterns, and the synchronized advices are LTSs. For a Labeled Transition System (S, Act, T, s_{init}), a state $s \in S$ is represented as an agent in the CWB, while a transition $s \xrightarrow{a} s_1 \in T$ is represented as an expression like "agent $S = a.S1$;" in the CWB.
- The mapping of other types of advices is similar to that of LTSs except that they have special undefined agents to indicate the final state of the advice. The special undefined agent's name begins with 'End'.
- The join point is a state or a transition in the base model, which is mapped as an agent or an expression in the CWB. Advice weaving is mapped according to the advice type and steps defined by the corresponding weaving operator, while the projection is mapped according to its definition.
- Finally, three commands *eq*, *mayeq*, and *maypre* are used to check semantic relationship between two LTSs. The three commands return true iff two agents are weak bisimilar, trace equivalent, or languages contained, respectively. In other words, given two LTSs M and M', *Ifl*(M, M') is:
  - *influence-free* iff *eq*(M, M');or
  - *may-influence-free* iff *mayeq* (M, M'); or
  - *extension* iff *maypre*(M, M'); or
  - *narrowing* iff *maypre*(M', M); or
  - *alteration* otherwise.

B. Implementing the Example

Fig.5 is the definition of the base model and the aspects of the property listing system introduced in section 2.

The base model *PLS* is a parallel composition of three base components *LDB*, *PFM* and *PL*, which correspond to the three components of the system: *ListingDB*, *Property File Management*, and *Property Listing* respectively (see Fig.1).In the CWB, ''*a*' represents coname $\bar{a}$ and 'tau' represents the 'τ'.

The *Timing* aspect has a synchronized *Timing* advice (agent Timing in Fig.5), which would execute the action *starttimer* at the join point '*agent PL = infoin.PL1*' and execute the action *endtimer* at the join point *'agent PL2 = ok.'accepted.'save.PL*'. The *Log* aspect has a sequential *Log* advice (agent Log in Fig.5), which would insert after the join point '*agent PL = infoin.PL1*'. The *Auth* aspect has a sequential *Auth* advice (agent *Auth* in Fig.5), which would weave before the join point '*agent PL = infoin.PL1*'. The sequential agent *Log* and *Auth* have a final state *EndLog* and *EndAuth* respectively.

```
**PLS**
agent LDB = save.LDB;
agent PFM = verify.PFM1;
agent PFM1 = 'ok.PFM + 'failed.PFM;
agent PL = infoin.PL1;
agent PL1 = 'verify.PL2;
agent PL2 = ok.'accepted.'save.PL + failed.'refused.PL;
agent PLS = (PL|PFM|LDB)\{failed, ok, save, verify};
**Aspects**
agent Timing = starttimer.endtimer.tau.'totaltime.Timing;
agent Log=log.EndLog;
agent Auth = auin.Auth1;
agent Auth1 = 'unauthorized.0 + 'authorized.EndAuth;
agent TimeLog1 = 'timelog.EndTimeLog1;
agent TimeLog2 = 'timelog.EndTimeLog2;
```

Figure 5. The base model and the aspects of the example.

The advice weaving is implemented by two stages:

- Firstly, agents in the base model and the aspect are

© 2013 ACADEMY PUBLISHER

included in the woven model. Moreover, to distinguish them from their original definitions, every agent is renamed as '*n*_W' where *n* is their original names.

- Secondly, the weaving is implemented according to the type of the advice and the definition of advice weaving.

For example, Fig.6 despicts the woven model *PLS_W* resulted from weaving the *Log* advice into the *PLS*.

```
**Woven PLS**
agent LDB_W = save.LDB_W;
agent PFM_W = verify.PFM1_W;
agent PFM1_W = 'ok.PFM_W + 'failed.PFM_W;
agent PL_W = infoin.PL1_W;
agent PL1_W = 'verify.PL2_W;
agent PL2_W = ok.'accepted.'save.PL_W + failed.'refused.PL_W;
Agent PLS_W = (PL_W|PFM_W|LDB_W)\{failed,ok,save,verify};
**weaving aspect individually****
**Log used after infoin**
agent Log_W=log.EndLog_W;
agent PL_W = infoin.PL1_W_new;
agent PL1_W_new = Log_W;
agent EndLog_W=PL1_W;
```

Figure 6. The woven model

The projection is implemented by applying its definition. For example, in Fig.7, agent *PLS_W_P* is the projection of the woven model *PLS_W* in Fig.6 on the base model *PLS*, in which action *log* has been replaced by 'tau'.

```
**Projection of Woven PLS on PLS**
agent LDB_W_P = save.LDB_W_P;
agent PFM_W_P = verify.PFM1_W_P;
agent PFM1_W_P = 'ok.PFM_W_P + 'failed.PFM_W_P;
agent PL_W_P = infoin.PL1_W_P;
agent PL1_W_P = 'verify.PL2_W_P;
agent PL2_W_P = ok.'accepted.'save.PL_W_P + failed.'refused. PL_W_P;
agent Log_W_P=tau.EndLog_W_P;
agent PL_W_P = infoin.PL1_W_new_P;
agent PL1_W_new_P = Log_W_P;
agent EndLog_W_P=PL1_W_P;
Agent PLS_W_P = (PL_W_P | PFM_W_P | LDB_W_P)
                \{failed,ok,save,verify};
```

Figure 7. The projection of the woven model on PLS.

Now, the influences of aspect weaving on certain patterns can be checked. Firstly, the influences on the base model when weaving the three advices into it individually is checked. Table 1 lists the results. In Table 1, agent *PLS_W* is the woven model resulted from weaving every advice into the *PLS* individually, while agent *PLS_W_P* is the projection of *PLS_W* on *PLS*.

From the result, we can conclude that the influence of advice *Log* on the base model *PLS* is *influence-free* and that of advice *Auth* is *may-influence-free*. However, the influence of advice *Timing* on *PLS* is *narrowing*, i.e. the base model has been altered, which violates the expectation **R1.**

Through analysis, there are deadlocks in the woven model after weaving the *Timing* advice. To overcome the problem, we design an alternative aspect which consists of two advices *TimeLog1* and *TimeLog2* (see Fig.5). The two advices will log the time at the join points '*agent PL = infoin.PL1*' and '*agent PL2 = ok.'accepted.'save.PL*'. Then, certain computation for the average time can be conducted on the log files afterwards. The two timing advices are designed as depicted in Fig.5. Moreover, as shown in table 1, their influences on the base model are *influence-free*, which satisfy **R1.**

Then we detect the aspect weaving influences when weaving these advices into the base model incrementally in a sequence *Auth->TimeLog1->TimeLog2->Log*. Table 2 lists the checking results. To make it clearly understood, parameters *M* and *M'*are specified in the weaving notation '∠'and the projection notation'∇'. From the table, it can be seen that aspect *Auth* has a *may-influence-free* influence on the base model *PLS*, while the weaving of *TimeLog1* and *TimeLog2* have *influence-free* influences on the base model *PLS∠Auth* and *influence-free* impacts on the afore-woven aspect *Auth*. Finally, the weaving of *Log* does not influence the base model and the afore-woven aspects *Auth*, *TimeLog1*, and *TimeLog2*.

Finally, we check the influences of aspect weaving on the desired behavior. We design two patterns as follows to describe the desired behavior in **R2** and **R3**:

agent Pattern1 = 'authorized.infoin.Pattern1 + 'unauthorized.0; and
agent Pattern2 = 'authorized.log.Pattern2 + 'unauthorized.0;,

Through evaluation (see Table 3), weaving all aspects into the PLS has *influence-free* on *Pattern1* and *Pattern2*, i.e. it satisfies these expectations.

TABLE I.
INFLUENCES OF ASPECTS ON THE BASE MODEL WHEN WOVEN INDIVIDUALLY

The base model	The advice	*M*	*M'*	eq (*M*, *M'*)	mayeq (*M*, *M'*)	maypre (*M*, *M'*)	maypre (*M'*, *M*)
PLS	Log	PLS_W	PLS_W_P	T	T	T	T
PLS	Timing	PLS_W	PLS_W_P	F	F	F	T
PLS	Auth	PLS_W	PLS_W_P	F	T	T	T
PLS	TimeLog1	PLS_W	PLS_W_P	T	T	T	T
PLS	TimeLog2	PLS_W	PLS_W_P	T	T	T	T

© 2013 ACADEMY PUBLISHER

TABLE II.
INFLUENCES OF ASPECTS WHEN WEAVING THEM INCREMENTALLY.

M	M'	eq (M, M')	mayeq (M, M')	maypre (M, M')	maypre (M', M)
PLS	$\nabla^{PLS\angle Auth}_{PLS}$	F	T	T	T
$PLS\angle AUTH$	$\nabla^{PLS\angle Auth\angle TimeLog1\angle TimeLog2}_{PLS\angle Auth}$	T	T	T	T
$\nabla^{PLS\angle Auth}_{Auth}$	$\nabla^{PLS\angle Auth\angle TimeLog1\angle TimeLog2}_{Auth}$	T	T	T	T
$PLS\angle AUTH\angle TIMELOG1\angle TIMELOG2$	$\nabla^{PLS\angle Auth\angle TimeLog1\angle TimeLog2\angle Log}_{PLS\angle Auth\angle TimeLog1\angle TimeLog2}$	T	T	T	T
$\nabla^{PLS\angle Auth\angle TimeLog1\angle TimeLog2}_{Auth}$	$\nabla^{PLS\angle Auth\angle TimeLog1\angle TimeLog2\angle Log}_{Auth}$	T	T	T	T
$\nabla^{PLS\angle Auth\angle TimeLog1\angle TimeLog2}_{TimeLog1}$	$\nabla^{PLS\angle Auth\angle TimeLog1\angle TimeLog2\angle Log}_{TimeLog1}$	T	T	T	T
$\nabla^{PLS\angle Auth\angle TimeLog1\angle TimeLog2}_{TimeLog2}$	$\nabla^{PLS\angle Auth\angle TimeLog1\angle TimeLog2\angle Log}_{TimeLog2}$	T	T	T	T

TABLE III.
INFLUENCES OF ASPECTS WEAVING ON PATTERNS

M	M'	eq (M, M')	mayeq (M, M')	maypre (M, M')	maypre (M', M)
Pattern1	$\nabla^{PLS\angle Auth\angle TimeLog1\angle TimeLog2\angle Log}_{Pattern1}$	T	T	T	T
Pattern2	$\nabla^{PLS\angle Auth\angle TimeLog1\angle TimeLog2\angle Log}_{Pattern2}$	T	T	T	T

V. RELATED WORK

Majority of currently approaches to detecting aspect influences adopt model checking as the underlying technologies. They generally define the semantics as certain desired properties. Through checking the model derived from aspect weaving against desired properties, the influences of one aspect on the base model or other aspects can be detected. The approach proposed in [6] can verify whether the woven program contains unexpected behaviors such as deadlocks. MAVEN tool [10] can verify and analyze aspect interference modularly. It can not only verify the correctness of an aspect relative to its specification, but also allow establishing noninterference among aspects, or detecting potential interference. The approach proposed in [9] can detect aspect-base interactions modularly. As for the UML based aspect models, they should first be translated into some formal specifications. In [8], AO models written in Aspect-UML are translated into Alloy for verification. Similarly, in [7], aspect-oriented state models of a system are transformed into FSP processes which are checked by the LTSA model checker.

In contrast to the model-checking based approaches, the approach presented in the paper focuses on the influences of aspect weaving on the behavior of a pattern other than certain behavior properties.

In addition, other approaches employ technologies such as program slicing[14], graph transformation[15, 16] and semantics annotation[17] to detect the aspect interactions in AOSD. The approach in [14] uses programming slicing to detect the influences between aspects. The graph transformation based approach in [15] is to analyze potential inconsistencies caused by aspect composition. The graph transformation based approach in [16] can detect aspect interference at shared join points. The approach in [17] is presented to detect aspect interactions in aspect scenarios, which is based on lightweight semantic annotations of aspects. In contrast to these approaches, our approach operates on design artifacts. Nevertheless, the idea of program slicing is similar our notion of projection. In other words, the projection can be seen as a kind of the semantic slicing of the model.

VI. CONCLUSION

In this paper, we have presented and implemented an approach to detect semantic influences of aspect weaving on the base model, an afore-woven aspect, or an expected behavior in the woven model. The behavior to be detected is specified as a pattern. Through comparing the semantic relationship between a pattern and its projection, influences of the aspect weaving on the pattern are evaluated. The approach can detect and evaluate five types of influences, which can be used as the basis for estimating the correctness of aspects or be as clues to further improvements when undesirable influences occur.

Currently, we implement the weaving manually. As the CWB is a command line tool and the detection of aspect influences needs three models: a pattern, a woven model and the projection, there are much trivial work on

© 2013 ACADEMY PUBLISHER

saving copies of models. The problem can be resolved by building an automatic weaving tool in the future.

ACKNOWLEDGMENT

This work was supported by the Natural Science Foundation of Shandong Province China under Grant No.ZR2011FQ017 and Grant No.ZR2012FM032, and Shandong Engineering Laboratory of Key Technology for Flow Process Enterprise Information Integration.

REFERENCES

[1]B. Wang, C. Zhu, J. Sheng, "A formal description method for aspect-oriented statechart based on CSP", *In Proceedings of International Symposium on Computer Science and Computational Technology*, pp. 750-753, 2008

[2]D. Mouheb, D. Alhadidi, M. Nouh, M. Debbabi, L. Wang, M. Pourzandi, "Aspect weaving in UML activity diagrams: A semantic and algorithmic framework", *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol 6921 LNCS, pp. 182-199, 2012.

[3]M. Nouh, R. Ziarati, D. Mouheb, D. Alhadidi, M. Debbabi, L. Wang, M. Pourzandi, "Aspect weaver: A model transformation approach for UML models", *In Proceedings of the 2010 Conference of the Center for Advanced Studies on Collaborative Research, CASCON'10*, pp. 139-153, 2010.

[4]N. Amálio, P. Kelsen, Q. Ma, C. Glodt, "Using VCL as an aspect-oriented approach to requirements modeling", *Transactions on Aspect-Oriented Software Development VII - A Common Case Study for Aspect-Oriented Modeling, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6210 LNCS, pp. 151-199, 2010.

[5] R. Douence, P. Fradet, and M. Südholt, "A framework for the detection and resolution of aspect interactions", In *Proceedings of the ACM SIGPLAN/SIGSOFT Conference on Generative Programming and Component Engineering(GPCE '02)*, London, UK, Lecture Notes in Computer Science 2487, pp. 173–188,2002.

[6] N. Ubayashi, T. Tamai, "Aspect-Oriented Programming with Model Checking", in *Proceedings of the 1st international conference on Aspect-oriented software development (AOSD'02)*, pp.148-154,2002.

[7] D. Xu, I. Alsmadi, W. Xu, " Model Checking Aspect-Oriented Design Specification", in *Proceedings of the 31st IEEE International Computer Software and Applications Conference (COMPSAC'07)*, pp.491– 500,2007.

[8] F. Mostefaoui and J. Vachon, "Design-level Detection of Interactions in Aspect-UML models using Alloy", *Journal of Object Technology, Special Issue: Aspect-Oriented Modeling*, vol. 6, no. 7, pp.137–165, 2007.

[9] Z. Altahat, T. Elrad, "Detection and verification of semantic interaction in AOSD", in *Proceedings of 6th International Conference on Information Technology: New Generations*, pp.807-812, 2009.

[10] M. Goldman, E. Katz, S. Katz, "MAVEN: modular aspect verification and interference analysis", *Formal Methods in System Design - FMSD* , vol. 37, no. 1, pp. 61-92, 2010.

[11] R. Milner, Communication and Concurrency, *Prentice-Hall International Series in Computer Science*, 1989.

[12] R. Allen and D. Garlan, "A formal basis for architectural connection", *ACM Transactions on Software Engineering and Methodology*, vol. 6, pp.213-249, 1997.

[13] R. Cleaveland, J. Parrow, and B. Steffen, "The Concurrency Workbench: A Semantics based Verification Tool for Finite-state Systems", in *Proceedings of the Workshop on Automated Verification Methods for Finite-state Systems*, LNCS, 407, 1989.

[14]D. Balzarotti, A. Castaldo D'Ursi, L. Cavallaro, and M. Monga, "Slicing AspectJ woven code", in *Proceedings of Foundations of Aspect-Oriented Languages workshop (FOAL2005)*, 2005.

[15]K. Mehner, M. Monga, G. Taentzer, "Interaction Analysis in Aspect-Oriented Models", in *Proceedings of the 14th IEEE International Requirements Engineering Conference*, pp.66-75, 2006.

[16]M. Aksit, A. Rensink, T. Staijen, "A graph-transformation-based simulation approach for analysing aspect interference on shared join points", in *Proceedings of the 8th ACM International Conference on Aspect-Oriented Software Development,* pp.39-50, 2009.

[17] G. Mussbacher, J. Whittle, D. Amyot, "Modeling and detecting semantic-based interactions in aspect-oriented scenarios", *Requirements Engineering*, vol. 15, no. 2, pp. 197-214, 2010.

[18] Chunhua Yang, Haiyang Wang," Formal Models for Architecture Aspects and Their Weaving", *Journal of Software*, vol. 3, no. 9 : Special Issue: Selected Best Papers of ISECS 2008 - Track on Software, pp.52-59, 2008.

[19] Mahmoud O. Elish, Mojeeb Al-Khiaty, Mohammad Alshayeb," Investigation of Aspect-Oriented Metrics for Stability Assessment: A Case Study", *Journal of Software*, vol. 6, no. 12: Special Issue: Parallel and Distributed Data Processing, pp.2508-2514, 2011.

[20] Lichen Zhang," Aspect-Oriented Development Method for Non-Functional Characteristics of Cyber Physical Systems Based on MDA Approach", *Journal of Software*, vol. 7, no. 3, pp. 608-619, 2012.

Chunhua Yang 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 Professor of School of Information Science of Shandong Light Industry, Shandong, China. Her research interests include aspect oriented technologies, business process, and web services.

© 2013 ACADEMY PUBLISHER