体系结构层状态型方面的建模和编织——基于 LTS 的方法

杨春花[1),2)] 王海洋[1)]

[1)](山东大学计算机科学与技术学院 济南 250100)

[2)](山东轻工业学院信息科学与技术学院 济南 250353)

摘 要 状态型方面是一种封装反应式关注点的方面,它需要专门的切点机制来跟踪系统的执行历史.然而当前在体系结构层还缺乏支持状态型方面建模的有效机制.针对这一问题,提出一种体系结构层状态型方面的建模和编织方法.它以标记转换系统(Labelled Transition System,LTS)为底层形式化,建立了用于识别系统运行轨迹中特定模式的切点机制,并定义了支持多种通知类型的编织运算.该方法当前已在一个基于 FSP 规范的原型编织工具上实现,并用一个实例说明了该方法的有效性.

关键词 方面;状态型方面;编织;标记转换系统;面向方面的软件开发

中图法分类号 TP311 **DOI 号**: 10.3724/SP.J.1016.2011.00342

Modeling and Weaving Architectural-Level Stateful Aspects——A LTS Based Approach

YANG Chun-Hua[1),2)] WANG Hai-Yang[1)]

[1)](*School of Computer Science and Technology, Shandong University, Jinan* 250100)

[2)](*School of Information Science and Technology, Shandong Institute of Light Industry, Jinan* 250353)

Abstract Stateful aspect is an aspect mechanism to encapsulate reactive concerns, which needs special pointcut mechanisms to track the system execution history. However, there are still no effective modeling mechanisms for stateful aspects at the architectural level to date. To address the issue, an approach to modeling and weaving stateful aspects at the architectural level is proposed. The approach has the Labeled Transition System(LTS) as the underlying formalism. In the approach, a pointcut mechanism for identifying certain patterns in the system trace is built and weaving operations that support several types of advices are defined. The approach has been implemented on a FSP based prototype tool. A case study shows the effectiveness.

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

1 引 言

面向方面的软件开发(Aspect-Oriented Software Development,AOSD)是一种高级关注点分离技术.它在传统软件技术之上,引入一阶实体-方面(Aspect)对横切关注点进行封装,通过方面的模块化处理,增强系统的可追踪性、可理解性和可维护性.

收稿日期:2009-08-15;最终修改稿收到日期:2010-11-14.本课题得到国家自然科学基金(60673130)资助.**杨春花**,女,1974 年生,博士研究生,主要研究方向为面向方面建模和软件工程.E-mail: jnych@126.com. **王海洋**,男,1965 年生,教授,博士生导师,主要研究领域为数据库、工作流和软件工程等.

状态型方面(stateful aspects)[1]是这样一类方面：其通知的触发条件不是单个的关联点，而是一组关联点的序列.换言之，该类方面的触发执行依赖于系统的计算历史.例如，某个额外的行为需要在执行一组事件后执行就是一个典型的状态型方面.

本质上说，状态型方面封装的是反应式(reactive)的关注点[2].这类关注点是软件设计和编程经常需要面临的一类关注点，它们强烈地依赖于系统的运行历史.而由于基本系统(the base system)一般提供基于过程的、面向对象的或基于组件的分解，并没有提供合适的表示和实现反应式系统的抽象形式[2]，造成这类关注点不能有效地模块化.

与常规的方面相比，状态型方面需要专门的切点机制来跟踪系统的执行历史和确定通知的触发点.当前，在代码层已经提出许多支持状态型方面编程的方法，典型的如 EAOP[1]、JAsCo[3]、DEP(Declarative Event Patterns)[4]等.这些方法采用基于有限状态机的切点机制对程序运行历史中的特定事件模式进行识别，当模式识别后，通知的代码被激发.然而在体系结构层，当前提出的面向方面的建模方法，如 DAOP/ADL[5]、ABC/ADL[6]、AAM 方法[7]、PRISMA[8]等，仅关注于常规方面的建模和编织，并没有提供支持状态型方面建模的有效机制.

针对上述问题，吸取代码层状态型方面建模的某些思想，提出了一种体系结构层状态型方面建模和编织的方法.它采用标记转换系统(Labelled Transition System，LTS)[9]为底层形式化，建立了识别系统运行迹中特定模式的切点机制，定义了支持多种通知类型的编织运算.该方法已在一个原型编织工具——LTSbAW① 上获得实现.

本文第 2 节给出方法原理；第 3 节是方法的实现和实例研究；第 4 节分析相关工作；第 5 节是结束语.

2　基于 LTS 的状态型方面的建模和编织

考虑图 1 所示的一个国际旅行社代理例子[10].该代理可以为其分支机构提供检索旅行目的地信息(queryInfo)以及订购旅行项目(book)的服务.为激励其分支机构提高业绩，该代理制定一项奖励措施：若在使用这些服务后最终订购了(book)一个旅行项目，将得到一个奖励(bonus).

奖励是一个方面，而且其是否应用依赖于系统执行轨迹.例如，若某次执行的轨迹为

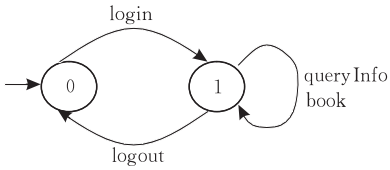

图 1　一个例子

login，queryInfo，queryInfo，queryInfo，logout，则奖励方面不能应用；反之，若运行轨迹中含有 book 动作时，如

login，queryInfo，book，queryInfo，logout

可应用.因此，奖励是一个状态型方面.

标记转换系统(Labelled Transition System，LTS)[9]是建模和分析体系结构行为的常用形式化，本文以 LTS 为基础进行了适当扩展，建立了支持状态型方面建模的切点机制，并定义了相应的编织运算.下面首先介绍 LTS 的相关概念，然后详细阐述本文方法.

2.1　LTS 相关概念

本文采用文献[11-12]中给出的 LTS 的相关概念，其定义如下：

设 $States$ 是一个状态全集，$Labels$ 是一个动作标记的全集.

定义 1.　一个标记转换系统(Labelled Transition System，LTS) P 是一个四元组 (S,L,T,s_{init})，其中，$S\subseteq States$ 是一个有限的状态集，$L\subseteq Labels$ 是一个有限的标记集，$T\subseteq(S\times L\times S)$ 是一个状态间的标记转换关系，$s_{init}\in S$ 是初始状态.

一个 LTS $P=(S,L,T,s_{init})$，对 $\forall s\in S$，若可以通过转换 $(s,\ell,s')\in T$ 到达状态 s'，则称 P 从状态 s 经过一步执行到达状态 s'，记为 $s\xrightarrow{\ell}s'$.设 $\sigma=\ell_1,\cdots,\ell_k(k\geq 0)$ 是集合 L 上一个标记序列，用 $s\xrightarrow{\sigma}s'$ 表示存在 $s_0,\cdots,s_k$ 使得 $s=s_0$，$s'=s_k$，和 $s_i\xrightarrow{\ell_{i+1}}s_{i+1}(0\leq i<k)$.用 $s\xrightarrow{\sigma}$ 表示 $\exists s'\in S\cdot s\xrightarrow{\sigma}s'$；并用 $s\rightarrow$ 表示 $\exists\ell\in L$，$s'\in S\cdot s\xrightarrow{\ell}s'$.

一个 LTS 的语义是从其初始状态开始可以执行的标记序列的集合.

定义 2(运行迹，trace).　已知 LTS $P=(S,L,T,s_{init})$，$\pi=\ell_1,\cdots,\ell_k(0\leq k)$ 是集合 L 上一个标记序列，若 $\exists s\in S\cdot s_{init}\xrightarrow{\pi}s$，则 π 是 P 的一个运行迹(trace).

① Be available at：http://code.google.com/p/ltsaspectweaving/downloads/list

定义 3(强模拟等价,strong bisimulation equivalence). 已知两个 LTS $P=(S_P,L_P,T_P,s_{initP})$和 $Q=(S_Q,L_Q,T_Q,s_{initQ})$且 $L_P=L_Q$,P 和 Q 是强模拟等价的,如果(S_P,S_Q)被包含进某个模拟关系 $R\subseteq S_P\times S_Q$,该关系对任意的 $\ell\in L_P$,$s_p\in S_P$,$s_q\in S_q$,有下式成立:

(1) $((s_p,s_q)\in R)\wedge(s_p\xrightarrow{\ell}s'_p)\Rightarrow(\exists s'_q\in S_q\cdot s_q\xrightarrow{\ell}s'_q\wedge(s'_p,s'_q)\in R)$.

(2) $((s_p,s_q)\in R)\wedge(s_q\xrightarrow{\ell}s'_q)\Rightarrow(\exists s'_p\in S_p\cdot s_p\xrightarrow{\ell}s'_p\wedge(s'_p,s'_q)\in R)$.

强模拟等价记为 $P\sim Q$.

定义 4(并发组合,parallel composition). 已知两个 LTS $P=(S_P,L_P,T_P,s_{initP})$和 $Q=(S_Q,L_Q,T_Q,s_{initQ})$,并发组合($\|$)是一个操作符且 $P\|Q$ 是 LTS $(S_P\times S_Q,L_P\cup L_Q,T,(s_{initP},s_{initQ}))$,其中 $T\subseteq((S_P\times S_Q)\times(L_P\cup L_Q)\times(S_P\times S_Q))$由图 2 所示规则导出.

$$\frac{P\xrightarrow{\ell}P'}{P\|Q\xrightarrow{\ell}P'\|Q}\ \ell\in L_P\wedge\ell\notin L_Q \qquad \frac{Q\xrightarrow{\ell}Q'}{P\|Q\xrightarrow{\ell}P\|Q'}\ \ell\notin L_P\wedge\ell\in L_Q$$

$$\frac{P\xrightarrow{\ell}P' \quad Q\xrightarrow{\ell}Q'}{P\|Q\xrightarrow{\ell}P'\|Q'}\ \ell\in L_P\wedge\ell\in L_Q$$

图 2 并发组合($\|$)的规则

2.2 基本系统定义

本文将封装业务关注点的系统称为基本系统(the base system).基本系统由组件构成,其相关定义如下.

定义 5(组件,component). 一个组件是一个 LTS $e=(S,L,T,s_{init})$,其中 L 是其与外部环境进行交互的通信动作集.

组件通过通信动作与外部环境进行交互,而组件之间通过共享的通信动作进行交互,交互遵循 LTS 之间的并发组合($\|$)规则.

定义 6(基本系统,the base system). 一个基本系统 P_B 是 n 个组件 $e_1,\cdots,e_n$ 的并发组合:$P_B=(e_1\|\cdots\|e_n)(n\in N)$.

一个合理的基本系统应当满足如下的语义特性:确定性和进展性.

定义 7(确定性). 一个 LTS $P=(S,L,T,s_{init})$是确定的(deterministic)当且仅当 $\forall\ell\in L\cdot(\exists(s,\ell,s_1)\in T\wedge(s,\ell,s_2)\in T\Rightarrow(s_1=s_2))$.

定义 8(进展性). 一个 LTS $P=(S,L,T,s_{init})$是可进展的(progressive)当且仅当 $\forall s\in S.(s_{init}\rightarrow s)\Rightarrow(s\rightarrow)$.

定义 9(良定义的,well-defined). 如果一个 LTS 是确定的和可进展的,则称该 LTS 是良定义的.

2.3 基于 *f*LTS 的切点机制

状态型方面的切点根据基本系统的运行轨迹来确定通知的触发点-关联点.本文对 LTS 概念进行扩展,提出了 fLTS概念和扩展的并发组合运算($\propto$),并以此为基础,给出了切点的定义.

定义 10. 一个 fLTS 是一个五元组 $P=(S,L,T,f,s_{init})$,其中,(S,L,T,s_{init})是一个 LTS,$f:T\rightarrow\{0,1\}$是一个函数.

fLTS 中的函数 f 是一个关联点标记函数.给定转换 t,若 $f(t)=1$,则该转换被标记以关联点,也就是通知的触发点.

对于一个 fLTS $P=(S,L,T,f,s_{init})$,定义一个辅助一元运算 ∂ 获得其 LTS:$\partial(P)=(S,L,T,s_{init})$.另外,定义一个扩展的并发组合操作($\propto$)实现对一个 LTS 和 fLTS 的并发组合.

定义 11(扩展的并发组合,extended parallel composition). 已知一个 LTS $P=(S_P,L_P,T_P,s_{initP})$,和一个 fLTS $Q=(S_Q,L_Q,T_Q,f_Q,s_{initQ})$,则扩展的并发组合($\propto$)是一个操作符且 $P\propto Q$ 是 fLTS $(S_P\times S_Q,L_P\cup L_Q,T_{PQ},f_{PQ},(s_{initP},s_{initQ}))$,其中 $T_{PQ}\subseteq((S_P\times S_Q)\times(L_P\cup L_Q)\times(S_P\times S_Q))$和 $f_{PQ}:T_{PQ}\rightarrow\{0,1\}$是由图 3 所示规则导出的.

$$\frac{P\xrightarrow{\ell}P'}{P\propto Q\xrightarrow{\ell}P'\propto Q\quad f_{pQ}(P\propto Q\xrightarrow{\ell}P'\propto Q)=0}\ \ell\in L_P\wedge\ell\notin L_Q$$

$$\frac{Q\xrightarrow{\ell}Q'}{P\propto Q\xrightarrow{\ell}P\propto Q'\quad f_{pQ}(P\propto Q\xrightarrow{\ell}P\propto Q')=f_Q(Q\rightarrow Q')}\ \ell\notin L_P\wedge\ell\in L_Q$$

$$\frac{P\xrightarrow{\ell}P' \quad Q\xrightarrow{\ell}Q'}{P\propto Q\xrightarrow{\ell}P'\propto Q'\quad f_{pQ}(P\propto Q\xrightarrow{\ell}P'\propto Q')=f_Q(Q\rightarrow Q')}\ \ell\in L_P\wedge\ell\in L_Q$$

图 3 扩展的并发组合($\propto$)的规则

定义 12(切点,pointcut). 已知一个基本系统 $P_B=(S_B,L_B,T_B,s_{initB})$,$P_B$ 上的切点(pointcut)是一个满足下列条件的 fLTS $P_{pc}=(S_{pc},L_{pc},T_{pc},f_{pc},s_{initpc})$:

(1) $\partial(P_{pc})$是良定义的;

(2) $L_{pc}\subseteq L_B$;

(3) $\partial(P_B\propto P_{pc})\sim P_B$.

其中,条件 1 保证切点 fLTS 的特性;条件 2 保证切点跟踪基本系统的 trace,没有额外的通信动作;条件 3 保证切点是基本系统的观察者,它和基本系统组合后得到的 LTS 和基本系统语义等价,不改变基本系统的行为.

依据定义 12，并吸取 EAOP[1] 中事件模式的思想，下面给出基本系统运行迹模式的概念（定义 13）以及根据此模式来获取切点的具体方法（定义 14 和定理 1）.

定义 13（运行迹模式）. 已知基本系统 $P_B=(S_B,L_B,T_B,s_{initB})$，若 $\exists' s_i,s_i,s'\in S,\ell_i\in L\wedge(\alpha_i\in L\wedge\alpha_i\neq\ell_i)(1\leq i\leq n)$ 使得 $s_{initB}\xrightarrow{\alpha_1^*}{}'s_1\xrightarrow{\ell_1}s_1\xrightarrow{\alpha_2^*}{}'s_2\xrightarrow{\ell_2}s_2\cdots\xrightarrow{\alpha_n^*}{}'s_n\xrightarrow{\ell_n}s'$ 成立，则称 $\ell_1\ell_2\cdots\ell_n$ 为基于 P_B 的运行迹的模式，其中 $\xrightarrow{\alpha_i^*}$ 表示连续经过若干个标记 α_i 的转换.

定义 14（识别运行迹模式的 fLTS）. 已知 $\ell_1\ell_2\cdots\ell_n$ 为基于基本系统 $P_B=(S_B,L_B,T_B,s_{initB})$ 的运行迹的模式，则 $P_p=(S_p,L_p,T_p,f_p,s_{initp})$ 是一个识别模式 $\ell_1\ell_2\cdots\ell_n$ 的 fLTS，其中

$L_p=L_B$，

$S_p=\{s_{initp},s_1,s_2,\cdots,s_{n-1}\}$，

$T_p=\{(s_{initp},\ell_1,s_1),(s_1,\ell_2,s_2),\cdots,(s_{n-2},\ell_{n-1},s_{n-1}),(s_{n-1},\ell_n,s_{initp})\}\cup\{(s_{initp},\alpha,s_{initp})\mid\alpha\in L_p-\{\ell_1\}\}\cup\{(s_1,\alpha,s_1)\mid\alpha\in L_p-\{\ell_2\}\}\cup\cdots\cup\{(s_{n-1},\alpha,s_{n-1})\mid\alpha\in L_p-\{\ell_n\}\}$，

$f_p:T_P\to\{0,1\}$ 满足

$(\exists t_p=(s_p,\ell_p,s_p')\in T_p\wedge s_p\neq s_p'\wedge f_p(t_p)=1)\wedge(\forall t\in T_p-\{t_p\}\wedge f_p(t)=0)$.

定理 1. 识别基本系统 P_B 运行轨迹模式的 fLTS 是 P_B 上的切点.

证明.

设 $\ell_1\ell_2\cdots\ell_n$ 为基于基本系统 $P_B=(S_B,L_B,T_B,s_{initB})$ 的运行轨迹的模式，fLTS $P_p=(S_p,L_p,T_p,f_p,s_{initp})$ 是识别模式 $\ell_1\ell_2\cdots\ell_n$ 的 fLTS.

由定义 14，容易得出

$$\partial(P_p)\text{是良定义的且 }L_p\subseteq L_B \tag{1}$$

设 $\partial(P_B\propto P_p)=(S_{Bp},L_{Bp},T_{Bp},s_{initBp})$，并定义关系 $R=\{\langle(s_B,s_P),s_B\rangle\mid(s_B,s_P)\in S_{Bp},s_B\in S_B,s_P\in S_P\}\subseteq S_{Bp}\times S_B$，则对 $\forall\langle(s_B,s_P),s_B\rangle\in R$，有下列式(2)、(3)成立：

对 $\forall(s_B,s_P)\xrightarrow{l}(s_B',s_P')\in T_{Bp}$，则必然 $\exists s_B'\in S_B$ 使得 $s_B\xrightarrow{l}s_B'\in T_B$，根据 R 定义知 $\langle(s_B',s_P'),s_B'\rangle\in R$；(2)

对 $\forall s_B\xrightarrow{l}s_B'\in T_B$，根据定义 14，必然 $\exists s_P'\in S_P$，使得 $s_P\xrightarrow{l}s_P'\in T_p$，则 $(s_B,s_P)\xrightarrow{l}(s_B',s_P')\in T_{Bp}$，且根据 R 定义知 $\langle(s_B',s_P'),s_B'\rangle\in R$. (3)

根据～定义和式(2)、(3)，得

$$\partial(P_B\propto P_p)\sim P_B \tag{4}$$

根据定义 12 和式(1)、(4)，得 P_p 为 P_B 上的切点. 证毕.

以图 1 所示基本系统为例，book logout 是其运行迹模式，图 4 是识别该模式的 fLTS，其中 $1\xrightarrow{\text{logout/JP}}0$ 表示转换 $1\xrightarrow{\text{logout}}0$ 的 f 函数值为 1，而其余转换的 f 函数值都为 0. 则根据定理 1，该 fLTS 是奖励方面的切点. 图 4 中每个状态上的环状转换在 ConcurrentEAOP[13] 中称为等待环（wait loops）.

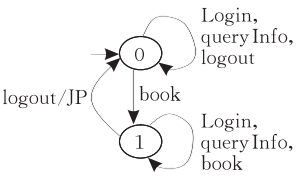

图 4　识别运行迹模式的 fLTS 示例

2.4　通知和方面定义

一个通知可以是一个观察者，它仅在特定事件发生后插入额外的动作而不改变基本系统的运行轨迹；也可以是一个辅助者，使得基本系统转到另外的状态[14]. 观察者的例子有 tracing，debugging，logging[15] 等，而辅助者方面如存取控制[7]、恢复方面[16] 等.

令 $WT=\{$bef-seq，aft-seq，choice，bef-branch，aft-branch，bypass，replace$\}$ 是一个预定义的通知类型的集合. 其中，顺序型（bef-seq 和 aft-seq）通知插入到关联点之前或之后；选择型（choice）通知在关联点被激发，并使基本系统转向一个特定状态；分支型（bef-branch 和 aft-branch）通知在关联点被激发，并有条件地使系统转向一个特定状态；旁路型（bypass）通知在关联点对应的源状态和目的状态之间加入通知；而替换型（replace）通知替换原有的关联点.

定义 15（通知，advices）. 一个通知（advice）是一个元组 $Ad=(type,S,L,T,s_{init},S_{final})$，其中 $type\in WT$ 是通知的类型，$(S,L,T,s_{init},S_{final})$ 是一个描述通知行为的扩展 LTS. $S_{final}=\{s\mid s\in S\wedge\neg\exists s'\in S\bullet(s,a,s')\in T\}$ 是一个终结状态集，且当类型 $type=[$bef-seq，aft-seq，bypass，replace，choice$]$ 时 $|S_{final}|=1$，而当类型 $type=[$bef-branch，aft-branch$]$ 时 $|S_{final}|=2$. 另外，$(S,L,T,s_{init},S_{final})$ 是确定的且除了终结状态之外的所有状态都是可进展的.

定义 16(方面元素,aspectlet). 一个方面元素(aspectlet)是一个元组(P_{pc},π,P_{ad}),其中P_{pc}是切点,P_{ad}是通知,π是基本系统的一个 trace,用来标识一个特定的状态,且当 $P_{ad}.type=$[bef-seq, aft-seq,bypass,replace]时,π为空;否则非空.

定义 17(方面,aspects). 一个方面是一个方面元素的集合$A=\{e|e$是一个方面元素$\}$.

2.5 方面编织

切点和基本系统的组合(∞)得到一个和基本系统等价的 fLTS,其所有关联点都已确定,可以织入通知.

对于一个 fLTS $P=(S,L,T,f,s_{init})$,定义辅助运算 $select(P)$获得其关联点集合:

$$select(P)=\{t|t\in T\ \wedge f(t)=1\}.$$

另外,已知两个状态集合S_1和S_2含有互相区分的状态,定义一个 $Union$ 函数返回S_1和S_2的合并状态集:

$$Union(S_1,S_2,\{s_{11}\rightarrow s_{21},\cdots,s_{1n}\rightarrow s_{2n}\})=$$
$$\{s|(\{s_{11},\cdots,s_{1n}\}\subseteq S_1\wedge\{s_{21},\cdots,s_{2n}\}\subseteq S_2)\wedge$$
$$(s=s_1\wedge s_1\in S_1-\{s_{11},\cdots,s_{1n}\})\wedge(s=s_2\wedge s_2\in S_2)\wedge$$
$$(s=s_{2x}\wedge s_{1x}\in\{s_{11},\cdots,s_{1n}\}\wedge x\in\{1,\cdots,n\})\}.$$

首先,定义一个微运算↵,将通知插入到一个 fLTS 的某个关联点.

定义 18. 已知一个 fLTS $P_M=(S_M,L_M,T_M,f_M,s_{initM})$,一个关联点 $t_{jp}=(s_{jp},\ell_{jp},s'_{jp})\in select(P_M)$,一个通知 $P_{Ad}=(type_{ad},S_{ad},L_{ad},T_{ad},s_{initad},S_{finalad})$,一个 trace π. 令 s''_{jp}是π确定的状态:$P_M\xrightarrow{\pi}s''_{jp}$. 则运算↵在 t_{jp} 处织入 P_{Ad} 并返回一个 fLTS: $P_M\text{↵}(t_{jp},\pi,Ad)=(S_C,L_C,T_C,f_C,s_{initC})$,其中,

$L_C=L_M\cup L'_{ad}$;

$$f'_C(t)=\begin{cases}f'_M(t), & t\in T'_M,\\ 0, & t\in T'_C-T'_M;\end{cases}$$

$s'_{initC}=s'_{initM}$;

S_C和T_C的取值依赖于通知P_{Ad}的类型:

(1)若 $type_{ad}=$bef-seq,则

$S_C=Union(S_{ad},S_M,\{s_{initad}\rightarrow s'_{jp}\})$;

$T_C=(T_M-\{t_{jp}\})\cup T_{ad}\cup\{(s_{adfinal},\ell_{jp},s'_{jp})\}$,

其中 $s_{adfinal}\in S'_{finalad}$;

(2)若 $type_{ad}=$aft-seq,则

$S_C=Union(S_{ad},S_M,\{s_{adfinal}\rightarrow s'_{jp}\})$;

$T_C=(T_M-\{t_{jp}\})\cup\{(s_{jp},\ell_{jp},s_{initad})\}\cup T_{ad}$,

其中 $s_{adfinal}\in S'_{finalad}$;

(3)若 $type_{ad}=$bypass,则

$S_C=Union(S_{ad},S_M,\{s_{initad}\rightarrow s_{jp},s_{adfinal}\rightarrow s'_{jp}\})$,

其中 $s_{adfinal}\in S_{finalad}$;

$T_C=T_M\cup T_{ad}$.

(4)若 $type_{ad}=$replace,则

$S_C=Union(S_{ad},S_M,\{s_{initad}\rightarrow s_{jp},s_{adfinal}\rightarrow s'_{jp}\})$,

其中 $s_{adfinal}\in S'_{finalad}$;

$T_C=(T_M-\{t_{jp}\})\cup T_{ad}$.

(5)若 $type_{ad}=$choice,则

$S_C=Union(S_{ad},S_M,\{s_{initad}\rightarrow s'_{jp},s_{adfinal}\rightarrow s''_{jp}\})$,

其中 $s_{adfinal}\in S'_{finalad}$;

$T_C=T_M\cup T'_{ad}$;

(6)若 $type_{ad}=$bef-branch,则

$S_C=Union(S_{ad},S_M,\{s_{initad}\rightarrow s_{jp},s_{adfinal2}\rightarrow s''_{jp}\})$;

$T_C=(T_M-\{t_{jp}\})\cup T_{ad}\cup\{(s_{adfinal1},\ell_{jp},s'_{jp})\}$,

其中 $s_{adfinal1},s_{adfinal2}\in S'_{finalad}$;

(7)若 $type_{ad}=$aft-branch,则

$S_C=Union(S_{ad},S_M,\{s_{adfinal1}\rightarrow s'_{jp},s_{adfinal2}\rightarrow s''_{jp}\})$,

其中 $s_{adfinal1},s_{adfina12}\in S_{finalad}$;

$T_C=(T_M-\{t_{jp}\})\cup\{(s_{jp},\ell_{jp},s_{initad})\}\cup T_{ad}$.

令函数 $clone(Ad)$返回一个通知 Ad 的克隆,且约定每次克隆得到的克隆通知都具有不同的状态名,而且克隆通知的状态名和动作名与基本系统的相区别.

一个方面元素的编织就是利用微运算↵,将其克隆通知插入到基本系统和切点组合得到的 fLTS 的所有关联点的过程.

定义 19. 已知一个方面元素$AL=(P_{pc},\pi,P_{ad})$和一个基本系统 LTS $P_B=(S_B,L_B,T_B,s_{initB})$,则运算$\angle_{AL}$将 AL 织入 P_B,得到一个 LTS:$P_B\angle_{AL}AL=\partial(\cdots((P_B\propto P_{pc})\text{↵}(t_{jp1},\pi,clone(P_{ad}))\text{↵}(t_{jp2},\pi,clone(P_{ad})))\text{↵}\cdots\text{↵}(t_{jpn},\pi,clone(P_{ad})))$,其中,$select(P_B\propto P_{pc})=\{t_{jp1},\cdots,t_{jpn}\}(n\geq1)$.

而方面的编织就是逐个织入其方面元素的过程.

定义 20. 已知一个基本系统 P_B和一个方面 $A=\{AL_1,\cdots,AL_n\}(n\geq1)$,则运算$\angle$将方面 A 织入 P_B,得到一个 LTS:

$$P_B\angle A=(\cdots(P_B\angle_{AL}AL_1)\cdots\angle_{AL}AL_n).$$

编织前,对基本系统的特性进行了一定限制,下面的定理 2 说明编织后得到的系统仍然保持这种特性.

定理 2(确定性和可进展性的保持). 设一个良定义的基本系统 P_B和一个方面 A,则 $P_B\angle A$ 也是良定义的.

方面编织可能会对基本系统产生各种影响. 人们一般期望两个基本系统 P_1和 P_2是语义等价的,则应用一个方面到 P_1和 P_2的结果(语义上来讲)也

是等价的[16]. 本文用 LTS 之间的强模拟等价关系表示这种语义等价关系. 下面的定理 3 说明编织后得到的系统仍然保持这种特性.

定理 3(语义等价性的保持). 给定两个基本系统 P_1 和 P_2，一个方面 A 可应用到 P_1 和 P_2，则 $P_1 \sim P_2 \Rightarrow P_1 \angle A \sim P_2 \angle A$.

定理 2 和定理 3 的证明见附录.

从以上两个定理看出，方面编织运算∠保持了系统的确定性、可进展性和语义等价性. 也就是说，编织后的系统和编织前的系统一样，可以参与进一步的并发组合(‖)运算或编织运算(∠). 因此，编织运算∠可以看成是与并发组合(‖)相对应的一种新运算符，编织运算∠支持 LTS 和方面的组合，而并发组合(‖)支持 LTS 和 LTS 的组合.

3　实例研究

3.1　实　现

我们开发了一个基于 FSP(Finite State Processes)[17]的原型编织工具——LTSbAW，实现本文的方法.

FSP 是一种进程代数风格的语言，支持 LTS 建模. 在 FSP 中，一个进程 P 对应了一个 LTS，该进程及其局部进程构成了该 LTS 的状态集，而 P 是该 LTS 的起始状态. 例如，图 5 是图 1 所示基本系统的 FSP 描述. 进程 B 由局部进程 $B1$ 构成，(“—>”)和(“|”)是动作前缀和选择操作符.

```
B=(login->B1),
B1=(book->B1|queryInfo->B1|logout->B).
```

图 5　基本系统的 FSP 描述示例

LTSbAW 在 FSP 规范的基础上进行扩展，支持方面描述. 图 6 是 LTSbAW 支持的 Bonus 方面的描述. 其中，Aspect 是保留字，Bonus 是方面的标识. $A1$ 是一个方面元素，PC 和 AD 分别是其切点和通知，BeforeSeq 代表 AD 的类型 Bef-Seq. 进程 PC 描述了图 4 所示的 fLTS. 注意，在 PC 的描述中，其所有的等待环(wait Loops)由 LTSbAW 动生成.

```
Aspect Bonus{
  DEF A1=<BeforeSeq,PC,AD>
  PC=(book->PC1),
  PC1=(logout/JP->PC).
  AD=(bonus->EndAD).
}
```

图 6　Bonus 方面描述

将基本系统和方面规范输入 LTSbAW 后，可得到编织后的组合系统描述，它仍然满足 FSP 规范，可在支持 FSP 语法的工具——LTSA[17]上进行模型展示和分析. 图 7 是将图 5 和图 6 的规范输入 LTSbAW 后得到的组合系统描述，而图 8 是其在 LTSA 中的展示结果，从中可以清晰地看出 Bonus 编织后系统的行为.

```
B_PC=(login->B1_PC),
B1_PC=(logout->B_PC|queryInfo->B1_PC|book->
       B1_PC1),
B1_PC1=(book->B1_PC1|queryInfo->B1_PC1|bonus->
        AD_1_New1),
AD_1_New1=(logout->B_PC).
```

图 7　Bonus 方面编织后的组合系统描述

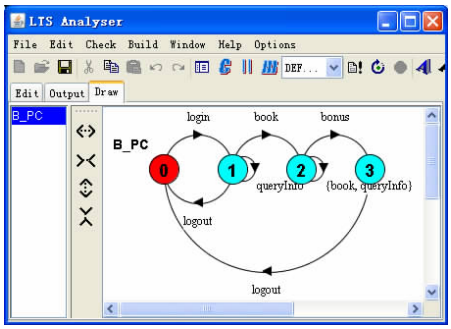

图 8　Bonus 方面编织后系统在 LTSA 中展示

3.2　实例分析

下面以我们建立的一个电子商务系统为例说明本文方法的应用以及与其它方法的分析比较结果.

该系统实现顾客网上购物功能，图 9 是其基本系统描述. 它通过 4 个组件的交互实现系统的业务功能：顾客(Customer)、商店(Shop)、支付中心(PaymentCenter)和商品发送中心(DeliveryCenter). 顾客在登录(login)系统后，可以浏览商品(browseProducts)，浏览促销广告(browsePromotions)，确认收货(confirmDelivery)，申请退款(refund)，加入购物车(addToCart)，然后可通过购买(buy)，确认订单(confirmOrder)和确认付款(confirmPayment)后完成一次购买过程. 支付中心在顾客确认付款后接收顾客付款，然后在发送商品成功(deliverySuccess)和顾客确认收货(confirmDelivery)后付款给商店或在发送商品失败(deliveryFail)和顾客申请退款(refund)时退款给顾客.

该系统需要面临 3 种方面需求：

(1) 顾客对促销广告的敏感度评定. 根据顾客进入系统后的行为轨迹评定其对促销广告的敏感程度：高敏感度顾客，进入系统后首先浏览促销广告，

然后进行其它操作;中等敏感度顾客,进入系统后首先浏览商品,然后浏览促销广告;不敏感顾客,从不浏览促销广告.

(2)订单日志. 每当顾客确认订单或取消订单时,记录日志.

(3)付款限时. 从发送商品成功到顾客确认收货需要时间限制,超过时间限制且客户还未确认收货时,系统自动付款给商家.

```
Customer=(login->Customer1),
Customer1=(browseProducts->Customer1 | browsePromotions->Customer1 | addToCart->Customer2
    | browseCart->Customer2 | confirmDelivery->Customer1 | refund->Customer1
    | logout->Customer),
Customer2=(browse->Customer1 | editCart->Customer2 | buy->Customer3 | logout->Customer),
Customer3=(backToCart->Customer2 | confirmOrder->Customer4 | logout->Customer),
Customer4=(confirmPayment->Customer5 | cancelOrder->Customer2 | logout->Customer),
Customer5=(browse->Customer1 | logout->Customer).
Shop=(confirmPayment->startDelivery->Shop).
PaymentCenter=(confirmPayment->paymentFromClients->PaymentCenter1),
PaymentCenter1=(deliverySuccess->PaymentCenter2 | deliveryFail->PaymentCenter3),
PaymentCenter2=(confirmDelivery->paymentToShop->PaymentCenter),
PaymentCenter3=(refund->paymentToClients->PaymentCenter).
DeliveryCenter=(startDelivery->DeliveryCenter1),
DeliveryCenter1=(deliverySuccess->DeliveryCenter | deliveryFail->DeliveryCenter).
||ECommerceSys=(Customer||PaymentCenter||DeliveryCenter||Shop).
```

图 9 电子商务系统的基本系统描述

根据上述方面需求,采用本文的方法对方面进行建模,得到如图 10 所描述的 5 个方面. 其中 Highpromotionpronecustomer, Mediumpromotionpronecustomer 和 Promotioninsensitivecustomer 用于实现需求 R1. 以 Highpromotionpronecustomer 为例. 其切点 *PC* 用于跟踪高敏感度顾客的执行轨迹:首先执行 login 进入状态 *PC*1,然后执行 browsePromotions 进入状态 *PC*2,在状态 *PC*2 可以执行其它动作,最后通过 logout 退出系统,此时触发通知. 若在状态 *PC*2 执行了除 browsePromotions 之外的其它动作,则转回到状态 *PC*.

```
Aspect Highpromotionpronecustomer{
  DEF A1=<BeforeSeq,PC,AD>
    PC=(login->PC1),
    PC1=(browsePromotions->PC2
      | browseProducts->PC | addToCart->PC
      | buy->PC | logout->PC | login->PC
      | browseCart->PC | confirmDelivery->PC
      | refund->PC | browse->PC | editCart->PC
      | backToCart->PC | confirmOrder->PC
      | confirmPayment->PC | cancelOrder->PC),
    PC2=(logout/JP->PC).
AD=(highpromotionpronecustomer->EndAD).
}
Aspect Mediumpromotionpronecustomer{
  DEF A1=<BeforeSeq,PC,AD>
    PC=(login->PC1),
    PC1=(browseProducts->PC2
      | browsePromotions->PC | addToCart->PC
      | buy->PC | logout->PC | login->PC
      | browseCart->PC | confirmDelivery->PC
      | refund->PC | browse->PC | editCart->PC
      | backToCart->PC | confirmOrder->PC
      | confirmPayment->PC | cancelOrder->PC),
    PC2=(browsePromotions->PC3),
    PC3=(logout/JP->PC).
AD=(mediumpromotionpronecustomer->EndAD)
}
```

```
Aspect Promotioninsensitivecustomer{
  DEF A1=<BeforeSeq,PC,AD>
  PC=(login->PC1 | browsePromotions->PC),
  PC1=(browsePromotions->PC
    | browseProducts->PC2 | addToCart->PC2
    | buy->PC2 | logout->PC2 | login->PC2
    | browseCart->PC2 | confirmDelivery->PC2
    | refund->PC2 | browse->PC2 | editCart->PC2
    | backToCart->PC2 | confirmOrder->PC2
    | confirmPayment->PC2 | cancelOrder->PC2),
PC2=(logout/JP->PC | browsePromotions->PC).
AD=(promotioninsensitivecustomer->EndAD).
}
Aspect LogOrder{
DEF A1=<AfterSeq,PC,AD>
PC=(confirmOrder/JP->PC | cancelOrder/JP->PC).
AD=(logOrder->EndAD).
}

Aspect TimeLimits{
  DEF A1=<AfterSeq,PC1,AD1>
    PC1=(deliverySuccess/JP->PC1).
    AD1=(starttimer->EndAD1).
DEF A2=<Bypass,PC2,AD2>
    PC2=(confirmDelivery/JP->PC2).
    AD2=(timeout->EndAD2).
}
```

图 10 各方面描述

LogOrder 和 TimeLimits 方面分别实现需求(2)和(3). LogOrder 方面用切点 *PC* 标识动作 confirmOrder 和 cancelOrder 为关联点,并在其后顺序织入通知 *AD*,记录日志. TimeLimits 方面含有 *A*1

和 $A2$ 两个方面元素，前者在关联点 deliverySuccess 之后，顺序织入通知 $AD1$，用于启动定时器(start-timer)；而后者在关联点 confirmDelivery 织入一个旁路型通知 $AD2$，用于限时(timeout).

由于各方面编织后得到的组合系统状态数较多，限于篇幅，没有列出. 通过对编织结果进行分析，它们都满足相应的方面需求.

EAOP[1]是一种典型的状态型方面编程方法，其 ConcurrentEAOP[13]模型通过在并发组合的基本系统和方面中添加同步事件来实现方面编织. ConcurrentEAOP 也采用 FSP 描述，因此可以和本文方法进行实现对比.

基于上述电子商务系统实例中的 5 个方面以及 ConcurrentEAOP 中的 Consistency 方面[13]，用本文的方法和 ConcurrentEAOP 方法进行了分别实现，表 1 是两种实现的对比分析结果. 可以看出，本文方法方面编织得到的组合系统的状态数远远少于 ConcurrentEAOP，而且提供了比 ConcurrentEAOP 更强的方面建模能力.

表 1　本文方法与 CurrentEAOP 方法的比较结果

方面	CurrentEAOP		本文方法	
	可否建模	组合系统状态数	可否建模	组合系统状态数
Highpromotionpronecustomer	√	516	√	144
Mediumpromotionpronecustomer	√	735	√	195
Promotioninsensitivecustomer	√	525	√	153
LogOrder	√	235	√	69
TimeLimits	×		√	57
Consistency	√	10	√	2

由于基于的底层机制不同，很难将本文方法与其余体系结构层的面向方面建模方法进行实现对比. 我们分析了 AAM[7]、DAOP/ADL[5]、ABC/ADL[6]、AO-ADL[18]等几个典型方法对上述实例中 5 个方面的支持能力.

前 3 个顾客促销广告敏感度方面是状态型方面，由于这些建模方法没有提供相应的跟踪系统执行轨迹的切点机制，因此很难对它们建模. LogOrder 和 TimeLimits 方面的切点依赖于系统的单个动作，这在上述这些方法中都能找到相应的支持机制. 从通知类型上看，LogOrder 方面的为 AfterSeq，这在 ABC/ADL 和 AO-ADL 中可以用 After 通知类型来实现，在 DAOP/ADL 可以用 Output Aspects 来实现，在 AAM 中可以用组合指令实现. TimeLimits 方面的通知类型为 Bypass，这在 DAOP/ADL 可以用 event aspects 来实现，在 AAM 中可以用组合指令实现，而在 ABC/ADL 和 AO-ADL 中没有找到相应的支持机制.

通过对上述实例的研究可以看出：本文方法可以有效地建模状态型方面和常规方面，而且支持比其它建模方法更丰富的通知类型. 由于编织后组合系统的状态数较少，将给以后在组合模型上的分析工作带来效率方面的优势.

3.3　讨　论

该方法不仅适用于状态型方面，而且支持常规方面的建模和编织. 已知基本系统 $P_B=(S_B,L_B,T_B,s_{initB})$，设 fLTS $P=\{\{s\},L_B,\{s\xrightarrow{\ell/JP}s,s\xrightarrow{\alpha}s\},s\}$，其中 $\ell\in L_B,\alpha\in L_B\wedge\alpha\neq\ell$. 显然 P 是 P_B 的切点，且在 P 和 P_B 的组合 $P\propto P_B$ 中所有标记 ℓ 的转换都为关联点，可以在其上进行通知编织. 因此，该方法可以建模每当动作 ℓ 发生时进行某种操作的方面.

方面应用的一个重要问题的是方面干扰. 已知两个方面 A_1 和 A_2，和一个基本系统 P，若单独编织每个方面都是可行和正确的，而两个方面都编织时不可行或不正确，则存在方面干扰. 这里的可行指可以进行编织，正确则指达到了应有的效果(如获得应有的系统属性等).

方面干扰可以分为语法层的干扰和语义层的干扰. 若 $P\angle A_1$ 和 $P\angle A_2$ 可行，而 $(P\angle A_1)\angle A_2$(或 $(P\angle A_2)\angle A_1$)不可行，则称存在语法干扰. 这种干扰主要是由于 A_1(或 A_2)的编织使得 A_2(或 A_1)的切点失效.

当编织 $(P\angle A_1)\angle A_2$ 可行(即不存在语法干扰)，而 A_2 的编织却改变了 A_1 的应用效果时，例如 A_1 的编织使 $(P\angle A_1)$ 获得了属性 φ，而 $(P\angle A_1)\angle A_2$ 违反了属性 φ，称存在语义干扰.

语法层的干扰可通过编织运算∠本身进行检测，而语义层的干扰，则可以应用工具，如 LTSA 等，对 $(P\angle A_1)\angle A_2$ 予以分析处理.

4 相关工作

当前体系结构层面向方面的建模方法主要关注于常规方面的建模问题.从对方面的处理方式上可将这些方法分为3类.第1类方法在传统的基于组件的体系结构描述中引入方面机制对横切关注点模块化,根据方面中的切点信息通过织入的方式实现方面和组件组合.DAOP/ADL[5]、ABC/ADL[6]、基于XYZ/ADL[19]的AO-ADL[18]等都属于这类方法.第2类方法将方面视为一种体系结构视图,通过方面和基本体系结构模型的合并,得到系统的集成视图.典型的如AAM方法[7]、PCS框架[20]等.第3类方法将方面视为基本的构造块,通过搭积木的方式形成复杂的系统,如PRISMA[8]、Architecture Views of Aspects[21]等.上述这些方法都没有提供跟踪系统运行轨迹的切点机制,因此无法建模状态型方面.

本文的方法属于第1类,它与DAOP/ADL、ABC/ADL和XYZ/ADL方法的主要区别是:(1)提供了跟踪系统运行轨迹的切点机制,可以建模状态型方面;(2)支持更多的通知类型,而ABC/ADL,XYZ/ADL和DAOP/ADL支持类似于AspectJ的before、after、around通知类型,不支持本文的替换和旁路型通知;(3)基于不同的形式化.本文的方法基于LTS,而ABC/ADL和DAOP/ADL基于XML机制,XYZ/ADL基于时序逻辑语言XYZ/E.

编程层状态型方面建模的相关研究包括:EAOP[1]和JAsCo[3]利用有限状态机识别一组系统允许的执行trace,当特定的trace模式出现时,执行通知行为.声明事件模式(DEPs)[4]方法用事件模式声明系统执行中的一系列事件,并指定当模式被识别时采取的特定动作.每个事件模式对应一个事件分析器——一个下推式自动机.这些方法在编织时需要在程序中切点自动机事件出现的地方添加额外的指令来引起切点自动机的状态转换.本文方法的切点思想和这些方法类似,但是实现机制不同.它通过切点和基本系统的组合来确定关联点,其优点是不需要添加额外指令,从而产生的组合系统的状态数少.

文献[16]提出一种状态型方面的建模方法.它用Mealy自动机建模切点,通知是在关联点状态和目标状态之间增加的转换.该方法适用于反应式系统.本文方法的提出受到此方法的启发.与之不同的是,本文方法针对一般的软件系统,而且除了可以建模文献[16]中提出的两种方面之外,还提供了对其它类型方面的支持.

在需求层,文献[22]提出基于继承性消息顺序图(Hierarchical Message Sequence Charts,HMSCs)的语义编织算法,其切点指定一个出现在基本HMSC中的部分行为,它在编织后被替换为通知的行为.由于切点表示了基本系统执行的部分语义,因此该方法的方面在一定程度上可看成一种状态型方面.其编织是基于HMSC组合上的转换规则,这一点类似于本文基于LTS组合上的编织.但是该方法不能处理本文的顺序、分支和选择型方面.

另外,文献[2]提出用一个状态机分解语言对基本系统进行分解,用模块规范来表示状态型方面,从而实现方面和基本模块之间更好的耦合.但是该方法没有提供形式化的描述.文献[10]将状态型方面引入工作流,提出一个基于有限状态自动机的编织策略将历史跟踪信息代码织入工作流.

5 结束语

针对当前体系结构层缺乏支持状态型方面建模有效机制的问题,本文提出了一种基于LTS的状态型方面建模和编织的方法,建立了识别系统运行轨迹模式的fLTS切点机制,定义了支持顺序、选择、分支、替换、旁路型通知的编织运算,基于FSP开发了支持该方法的原型编织工具,并结合一个电子商务实例与其它方法做了对比分析.结果表明,该方法可以有效地对状态型方面和其它常规方面进行建模.

未来的研究工作包括进一步丰富本文的编织模型使之支持方面间的组合以及对方面间的语义干扰进行分析.

参 考 文 献

[1] Douence R, Fradet P, Sudholt M. Composition, reuse and interaction analysis of stateful aspects//Proceedings of the 3th International Conference on Aspect-Oriented Software Development. Lancaster, UK, 2004: 141-150

[2] Cottenier T, van den Berg A, Elrad T. Stateful aspects: The case for aspect-oriented modeling//Proceedings of the 10th Aspect-Oriented Modeling Workshop. Vancouver, Canada, 2007: 7-14

[3] Vanderperren W, Suvee D, Cibrán M A, De Fraine B. Stateful aspects in JAsCo//Proceedings of the Software Composition Workshop at the European Joint Conferences on Theory

and Practice of Software. Edinburgh, Scotland. LNCS 3628. Berlin, Heidelberg: Springer-Verlag, 2005: 167-181

[4] Walker R J, Viggers K. Implementing protocols via declarative event patterns//Proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering. Newport Beach, USA, 2004: 159-169

[5] Pinto M, Fuentes L, Troya J M. A dynamic component and aspect-oriented platform. The Computer Journal, 2005, 48(4): 401-420

[6] Wang Xiao-Guang, Feng Yao-Dong, Mei Hong. ABC/ADL: An XML-based software architecture description language. Journal of Computer Research and Development, 2004, 41(9): 1521-1531(in Chinese)
(王晓光，冯耀东，梅宏. ABC/ADL：一种基于 XML 的软件体系结构描述语言. 计算机研究与发展，2004，41(9)：1521-1531)

[7] France R, Ray I, Georg G, Ghosh S. Aspect-oriented approach to early design modeling. IEE Proceedings Software, 2004, 151(4): 173-185

[8] Prez J, Ramos I, Jan J, Letelier P, Navarro E. PRISMA: Towards quality, aspect oriented and dynamic software architectures//Proceedings of the 3rd IEEE International Conference on Quality Software (QSIC 2003). Dallas, Texas, USA, 2003: 59-66

[9] Keller R M. Formal verification of parallel programs. Communications of the ACM, 1976, 19(7): 371-384

[10] Braem M, Gheysels D. History-based aspect weaving for WS-BPEL using padus//Proceedings of the 5th European Conference on Web Services (ECOWS'07). Halle, Germany, 2007: 159-167

[11] Uchitel S, Kramer J, Magee J. Behaviour model elaboration using partial labelled transition systems//Proceedings of the 11th ACM SIGSOFT Syposium on the Foundations of Software Engineering (FSE03). Helsinki, Finland, 2003: 19-27

[12] Bergstra J A, Ponse A, Smolka S A. Handbook of Process Algebra. Amsterdam: Elsevier, 2001

[13] Douence R, Botlan D L, Noyé J, Südholt M. Concurrent aspects//Proceedings of the Generative Programming and Component Engineering Conference (GPCE 2006). Portland, Oregon, 2006: 79-88

[14] Clifton C, Leavens G. Observers and assistants: A proposal for modular aspect-oriented reasoning//Proceedings of the 2002 Foundations of Aspect-Oriented Languages Workshop (FOAL 2002). 2002: 33-44

[15] Kiczales G, Lamping J, Mendhekar A, Maeda C, Lopes C, Loingtier J, Irwin J. Aspect-oriented programming//Proceedings of the l1th European Conference on Object-Oriented Programming (ECO0P'97). Jyvaskyla, Finland. LNCS 1241. Berlin, Heidelberg: Springer-Verlag, 1997: 220-242

[16] Altisen K, Maraninchi F, Stauch D. Aspect-oriented programming for reactive systems: Larissa, a proposal in the synchronous framework. Science of Computer Programming, Special issue: Foundations of Aspect-Oriented Programming, 2006, 63(3): 297-320

[17] Magee J, Kramer J. Concurrency-State Models & Java Programs. Chichester: John Wiley & Sons, 1999

[18] Yang Jing-Zhong, Rong Mei, Zhang Guang-Quan. Aspect-oriented software architecture description language AO-ADL. Computer Engineering, 2008, 34(10): 80-82(in Chinese)
(杨敬中，戎玫，张广泉. 面向方面的软件体系结构描述语言 AO-ADL. 计算机工程，2008，34(10)：80-82)

[19] Zhu Xue-Yang. The dual software architecture description framework XYZ/ADL. Journal of Computer Research and Development, 2007, 44(9): 1485-1494(in Chinese)
(朱雪阳. 双重软件体系结构描述框架 XYZ/ADL. 计算机研究与发展，2007，44(9)：1485-1494)

[20] Kande M. A concern-oriented approach to software architecture[Ph. D. dissertation]. Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland, 2003

[21] Katara M, Katz S. Architectural views of aspects//Proceedings of the 2nd International Conference on Aspect-Oriented Software Development Presented at Aspect-Oriented Software Development (AOSD 2003). Boston, Massachusetts, USA, 2003: 1-10

[22] Jacques K, Loïc H, Jean-Marc J. Semantic-based weaving of scenarios//Proceedings of the 5th International Conference on Aspect-Oriented Software Development. Bonn, Germany, 2006: 27-38

附　录.

定理 2(确定性和可进展性的保持).　设一个良定义的基本系统 P_B 和一个方面 A，则 $P_B \angle A$ 也是良定义的.

证明.　设 $AL=\langle P_{PC}, \pi, P_{Ad}\rangle \in A$ 为 A 的任意方面元素. 首先看 $P_B \angle_{AL} AL$ 的确定性和可进展性.

由 P_B 是良定义的，得 P_B 满足确定性和可进展性. 由切点的定义知 P_{PC} 满足确定性和可进展性. 设 $P_M = P_B \propto P_{pc}$. 则根据切点和运算 $\propto$ 的定义，可以得出 P_M 也满足确定性和可进展性.

由通知的定义知，P_{Ad} 满足确定性且除终结状态外的其余状态都满足可进展性. 又根据编织定义，$P_B \angle_{AL} AL$ 是将通知 P_{Ad} 的克隆按照通知类型对应的运算规则织入 P_M 的过程. 下面证明 bef-seq 类型的通知编织能保持确定性和可进展性，其余类型的通知编织的证明以此类推.

根据运算($\lrcorner$)的定义，bef-seq 类型的通知编织的主要是通过：

(1) 首先引入了一个 $clone(P_{Ad})$，并通过操作 $Union(S_{ad}, S_M, \{s_{initad} \rightarrow s_{jp}\})$ 将其起始状态 s_{initad} 和关联点 $t_{jp}=(s_{jp}, \ell_{jp}/JP, s'_{jp})$ 的状态 s_{jp} 合并为一个状态；

(2) 然后删除了转换 t_{jp}，并增加了转换 $(s_{adfinal}, \ell_{jp}, s'_{jp})$.

首先看确定性. 由于 P_{Ad} 满足确定性，所以 $clone(P_{Ad})$ 也满足确定性. 虽然状态 s_{initad} 合并为 s_{jp}，但由于 $clone(P_{Ad})$ 中的动作名区别于基本系统的动作，因此，不会引起状态 s_{jp}

的不确定性.同理,由于转换 $t_{jp}=(s_{jp},\ell_{jp}/\mathrm{JP},s'_{jp})$ 已经删除,所以向状态 s'_{jp} 新引入的转换 $(s_{adfinal},\ell_{jp},s'_{jp})$ 也不会引起 s'_{jp} 不确定性.除此之外,其余的状态和转换都分别取自基本系统和 $clone(P_{Ad})$,由于二者的动作集不同,因此不会引起不确定性.

再来看可进展性,由于 P_M 满足可进展性,其所有状态都是可进展的.$clone(P_{Ad})$ 中除了终结状态 $s_{adfinal}$ 外,其余状态都是可进展的.操作 $Union(S_{ad},\ S_M,\ \{s_{initad}\rightarrow s_{jp}\})$ 使得 $clone(P_{Ad})$ 中除终结状态 $s_{adfinal}$ 外其余状态在 $P_B\angle_{AL}AL$ 保持可进展性.而新引入的转换 $(s_{adfinal},\ell_{jp},s'_{jp})$ 使得 $clone(P_{Ad})$ 中的终结状态 $s_{adfinal}$ 也变为可进展.除此之外,$P_B\angle_{AL}\ AL$ 中的其余状态来自 P_B,它们的可进展性没有改变.

综上所述,A 的任意方面元素 AL 的编织 $P_B\angle_{AL}\ AL$ 满足确定性和可进展性.由此可得,$P_B\angle A$ 满足确定性和可进展性,是良定义的. 证毕.

定理 3(语义等价性的保持). 给定两个基本系统 P_1 和 P_2,一个方面 A 可应用到 P_1 和 P_2,则 $P_1\sim P_2\Rightarrow P_1\angle A\sim P_2\angle A$.

证明.

只要证明对于方面的任意元素 $AL=\langle P_{PC},\ \pi,\ P_{Ad}\rangle\in A$ 有 $P_1\angle_{AL}\ AL\sim P_2\angle_{AL}\ AL$ 成立,则 $P_1\angle A\sim P_2\angle A$ 成立.

设 $P_1=(S_1,\ L_1,\ T_1,\ f_1,\ s_{init1})$,$P_2=(S_2,\ L_2,\ T_2,\ f_2,\ s_{init2})$,$P_{pc}=(S_{pc},\ L_{pc},\ T_{pc},\ f_{pc},\ s_{initpc})$,$P_{M1}=P_1\propto P_{pc}=(S_{M1},\ L_{M1},\ T_{M1},\ f_{M1},\ s_{initM1})$,$P_{M2}=P_2\propto P_{pc}=(S_{M2},\ L_{M2},\ T_{M2},\ f_{M2},\ s_{initM2})$.

由 $P_1\sim P_2$ 和 $\partial(P_{M1})\sim P_1$ 和 $\partial(P_{M2})\sim P_2$,可得 $\partial(P_{M1})\sim\partial(P_{M2})$.这意味着对于任意的 $s_{M1}\in S_{M1}$ 和 $(s_{M1},\ell,s'_{M1})\in T_{M1}$,必然存在 $s_{M2}\in S_{M2}$ 和 $(s_{M2},\ell,s'_{M2})\in T_{M2}$ 满足

$$s_{M2}\sim s_{M1}\wedge s'_{M2}\sim s'_{M1} \tag{1}$$

同时,根据$\propto$运算的定义,f 函数值也一同参与运算,因此,对于任意的 $s_{M1}\in S_{M1}$ 和 $(s_{M1},\ell/\mathrm{JP},s'_{M1})\in T_{M1}$,则必然存在 $s_{M2}\in S_{M2}$ 和 $(s_{M2},\ell/\mathrm{JP},s'_{M2})\in T_{M2}$ 满足

$$s_{M2}\sim s_{M1}\wedge s'_{M2}\sim s'_{M1} \tag{2}$$

根据式(1)和(2),可得 $\partial(P_{M1})\sim\partial(P_{M2})$ 且 $select(P_{M1})\sim select(P_{M2})$,即 P_{M1} 和 P_{M2} 的关联点依附的状态互相等价.

AL 的编织是在 P_{M1} 或 P_{M2} 上进行的转换,下面证明每种类型的通知编织操作保持 P_{M1} 和 P_{M2} 之间的语义等价性.

对于顺序、选择、替换和旁路型通知的编织,由于此编织对 P_{M1} 和 P_{M2} 的关联点依附的等价的状态都执行相同的操作,因此不会打破 P_{M1} 和 P_{M2} 的等价性.

对于分支型通知编织,设状态 s'_{M1} 和 s'_{M2} 分别是 π 在 P_{M1} 和 P_{M2} 中标识的状态:$P_{M1}\xrightarrow{\pi}s'_{M1}$,$P_{M2}\xrightarrow{\pi}s'_{M2}$.由于 $\partial(P_{M1})\sim\partial(P_{M2})$,则 $s'_{M1}\sim s'_{M2}$.因此,分支型编织对 P_{M1} 和 P_{M2} 的关联点依附的等价的状态以及状态和 s'_{M1} 和 s'_{M2} 都执行相同的操作,也不会打破 P_{M1} 和 P_{M2} 的等价性.

综上所述,$P_1\angle AL\sim P_2\angle AL$. 证毕.

YANG Chun-Hua, born in 1974, Ph. D. candidate. Her current research interests include aspect oriented modeling and software engineering.

WANG Hai-Yang, born in 1965, professor, Ph. D. supervisor. His current research interests include database, workflow, and software engineering.

Background

Aspect oriented software development (AOSD) is an emerging paradigm for promoting the separation of crosscutting concerns throughout the software lifecycle. The crosscutting concerns are encapsulated in aspects. An aspect is composed of advices, which will be inserted to the base system through a process of weaving. The positions where advices would apply are called join points, which are specified by pointcuts.

Stateful aspect is an aspect whose advices are triggered by a sequence of join points instead of a single join point. Stateful aspect is a recent research topic in AOSD, whose aim is to encapsulate reactive concerns. Such concerns commonly exist in the software design and implementation. Due to the history-sensitive crosscutting characteristics, they cannot be expressed using the custom aspect concept.

Stateful aspect needs special pointcut mechanisms to track the system execution history and the corresponding approach to determine the join points, which makes its weaving more complicate. Currently, although many aspect-oriented approaches have been proposed to modularize, represent and compose crosscutting concerns at the architecture level, they do not provide adequate mechanisms for modeling and weaving stateful aspects.

This paper presents an approach to modeling and weaving stateful aspects at the architectural level. The approach has the Labeled Transition System(LTS) as the underlying formalism. In the approach, a pointcut mechanism for identifying certain patterns in the system trace is built and weaving operations that support several types of advices are defined. The approach has been implemented on a FSP based prototype tool. This work is supported by the National Natural Science Foundation of China under grant No. 60673130.