构件系统建模及其动态演化一致性验证方法

郑 明[1] 李 彤[1,2] 林 英[1,2] 周小煊[1] 李 响[1] 明 利[1]
(云南大学软件学院 昆明 650500)[1] (云南省软件工程重点实验室 昆明 650500)[2]

摘 要 基于构件的软件开发已成为软件开发的主流方法,但针对构件系统动态演化后的一致性保持问题,目前尚缺乏统一的标准,为此提出一种验证构件系统动态演化一致性的方法。首先,应用进程代数构造构件模型,并在此基础上得到粗粒度的构件系统模型;然后,根据构件系统模型及其状态的变化,提出构件系统外部行为提取算法,并基于弱互模拟理论定义构件系统动态演化一致性的验证准则;最后,提取演化前后构件系统的行为,并将其转换成便于Pi演算自动工具MWB(Mobility Workbench)识别的格式,以进行行为一致性验证。案例研究表明,该方法是可行且有效的。

关键词 构件,构件系统,进程代数,弱互模拟,动态演化
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.11.012

Dynamic Evolution Consistency Verification Method for Component System Modeling

ZHENG Ming[1] LI Tong[1,2] LIN Ying[1,2] ZHOU Xiao-xuan[1] LI Xiang[1] MING Li[1]
(School of Software, Yunnan University, Kunming 650500, China)[1]
(Key Laboratory in Software Engineering of Yunnan Province, Kunming 650500, China)[2]

Abstract Component-based technology has become a main stream approach for software development, however, how to maintain a high level of consistency of the component system after dynamic evolution is still lacking a generally accepted standard. So a verification method was proposed to verify whether a component system is consistent after dynamic evolution. Firstly, component model was proposed based on process algebraic, and a complex component system model could be obtained by combination of the components model. Secondly, according to the component system model and its state changes, an algorithm for extracting the external behavior of the component system was proposed, and a verification standard of consistency of the component system was presented based on weak bisimulation theory. Finally, the behavior of the component system was obtained, and then these behaviors were converted into a format which is convenient for Mobility Workbench to identify and verify. The case study shows that the proposed approach is feasible and effective.

Keywords Component, Component system, Process algebraic, Weak bisimulation, Dynamic evolution

1 引言

随着软件开发技术的发展,基于构件式的软件开发方法已经比较成熟[1]。一般来说,基于构件式的软件工程(Component-Based Software Engineering, CBSE)是将所需的构件通过集成组装成最终所需的系统[2]。随着系统的维护以及用户需求的改变,构件的增加、删除和修改无法避免,演化后构件系统的行为是否偏离演化前构件系统的行为是判断构件系统动态演化正确与否的本质标准,也是保证动态演化实施可靠性的重要条件,即在动态演化前后,构件与其他构件之间可观察的交互行为必须保持一致。

为了验证软件系统演化前后的系统一致性问题,国内外学者针对面向动态演化的构件建模及演化前后系统行为一致性的验证做了大量的研究工作。文献[3]给出了基于构件行为协议的构件式软件系统中构件可替换的必要条件,至于替换后的构件系统与替换前的构件系统是否一致,其并没有给出相应的验证方法,因此本文提出了一种基于进程代数建模的构件系统演化前后一致性验证的方法。文献[4]基于Petri网形式化工具对构件进行建模,并在系统行为层面加入一致性约束,从而在保证原来系统功能行为正确的前提下增加了

到稿日期:2016-10-20 返修日期:2016-12-27 本文受国家自然科学基金(61379032,61262024,61462092),云南省教育厅科学研究基金(2014Y012),云南大学研究生科研创新基金项目(111)资助。

郑 明(1992-),男,硕士生,主要研究方向为软件演化、形式化方法等;李 彤(1963-),男,教授,博士生导师,CCF高级会员,主要研究方向为软件工程、信息安全;林 英(1973-),女,博士,副教授,主要研究方向为信息安全、软件工程等,E-mail:linying@ynu.edu.cn;周小煊(1993-),女,硕士生,CCF学生会员,主要研究方向为软件工程、软件演化、数据挖掘;李 响(1993-),男,硕士生,主要研究方向为领域软件工程、软件演化;明 利(1989-),女,硕士生,主要研究方向为软件动态演化、云计算。

灵活性,但它不具有将多个 Petri 子网构件行为计算组合成一个大的 Petri 网构件系统行为的标准并发操作,使得构件演化前后的一致性标准较低,很难准确地验证演化后系统的一致性。本文基于进程代数对构件和构件系统进行建模,使其不仅具有严格的形式化语义,同时还具备通过并发操作符将多个构件的行为计算组合成更粗粒度的构件系统的行为。文献[5-6]基于进程代数构建构件模型,形式化地描述了构件及其对外交互协议,虽然给出了演化前后行为一致性验证规则,但缺少对构件外部交互行为的提取以及具体的构件行为一致性标准,因此本文提出了一种能够对构件系统模型外部行为序列进行提取的算法和基于弱互模拟理论的一致性验证准则。文献[7]基于时间自动机模型对软件演化前后的状态行为进行建模,利用时间自动机模型验证工具 UPPAAL 对系统的安全性规约和活性规约,但时间自动机无法对构件系统内部的活动进行精细的刻画,例如不能有效地描述多个活动的并发执行等,虽然采用时间自动机模型也可以对构件之间的行为交互协议进行建模,但以时间自动机模型表示的构件的行为交互只能支持相关性质的检查,并不能支持等价理论,因而也就不支持构件之间的相似性分析。本文之所以采取进程代数来对构件进行建模,正是因为进程代数能够对构件系统内部多个活动的并发进行有效的支持,并能将多个构件的行为计算组合成总行为,在弱互模拟的理论上支持相似性的分析,因此能够支持构件系统动态演化前后的一致性验证。

综上,本文首先基于进程代数构造一种能够形式化定义构件及其对其他构件交互行为的构件模型,基于该构件模型得到更粗粒度的构件系统模型;其次,提出一种构件外部行为提取算法,并基于弱互模拟关系来定义构件系统动态演化前后的外部一致行为性准则;最后,利用 Pi 演算自动验证工具 MWB[8] 来验证构件系统演化前后的一致性保持。

本文的主要贡献在于:

(1)在提出构件模型的基础上,基于构件间的交互及组合关系构建了更粗粒度的便于演化前后行为一致性研究的构件系统模型。

(2)提出构件系统模型外部行为序列提取算法,实现了构件系统模型外部行为序列提取的自动化。

(3)基于 Milner 的弱互模拟理论[9] 提出了构件系统演化前后行为一致性的验证准则。

本文第 2 节概述了基于进程代数的构件建模和构件系统演化前后系统行为一致性验证方法的实施步骤;第 3 节详细阐述了基于进程代数的构件建模及基于构件的构件系统建模;第 4 节定义了构件系统外部行为进程代数表达式形式的提取算法,以及构件系统演化前后行为一致性的准则;第 5 节以一个在线车票预订构件系统为例说明如何使用本文提出的基于进程代数的建模方法和演化前后构件系统外部行为一致性的验证方法;最后总结全文并展望下一步研究方向。

2　基于进程代数的构件建模及一致性验证方法概览

进程代数最早由 Bergstra 和 Klop[10] 于 1982 年提出。进程代数以进程的交互作为基本活动来描述具有多个并行、并发活动进程的系统,能够有效地对基于构件的构件系统进行描述;同时,进程代数还可以对构件系统的状态及状态转换的动作序列进行刻画,并进一步对构件系统演化前后的行为进行一致性验证。本文以意大利学者 Bernardo 提出的进程代数 PA[8,10] 作为形式化工具来描述及分析验证构件系统演化前后的外部行为一致性问题,并提出一种从行为的角度来验证构件系统演化前后是否一致的验证方法,如图 1 所示。

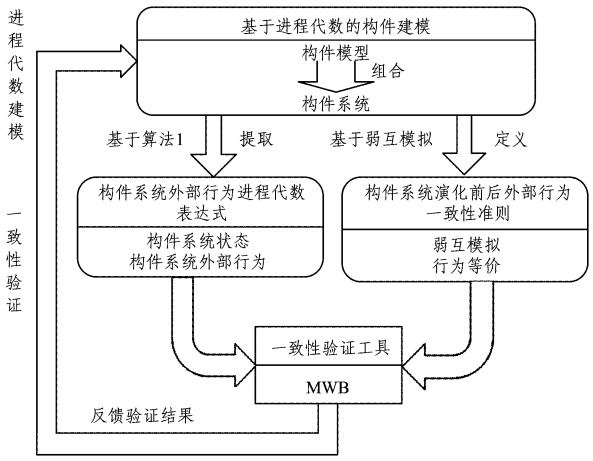

图 1　构件系统动态演化一致性验证流程

下面将详细阐述本文提出的基于进程代数的构件建模及构件组合的构件系统演化前后行为一致性验证方法。

3　基于进程代数的构件建模及构件系统建模

3.1　构件建模

文献[11]将行为协议引入构件模型,对构件间的请求/服务接口的相容性进行了深入研究。本文基于已有的研究,引入了构件系统的概念,通过构件间的连接关系将多个构件组装成构件系统,使得构件动态演化一致性的验证不再是基于局部的一个构件,而是多个构件构成的构件系统的总体验证。用于组装的构件模型如图 2 所示。接口有两种类型:请求服务接口、提供服务接口。构件的接口的个数为 $k(0\leqslant k\leqslant n)$,且存在服务接口到请求接口和请求接口到请求接口的内部连接关系。

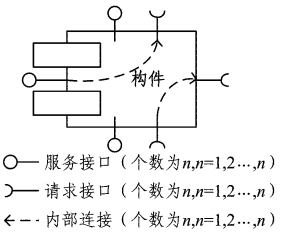

图 2　构件模型

定义 1(构件)　构件是一个六元组,$C=\langle N_C, I_C^P, I_C^R, IL_C, A_C, P_C\rangle$,其中:

(1)N_C 表示构件的名称,即构件的唯一标识。

(2)I_C^P 表示构件 C 对外提供服务的接口集合,$I_C^P=\{Ite_1, Ite_2, \cdots, Ite_n\}$,其中任意一个接口 $Ite_i\in I_C^P$ 表示构件给外界调用的服务接口,它包括一组服务提供接口方法操作 $Ite_i=\{a_1^i, a_2^i, \cdots, a_m^i\}$。

(3)I_C^R 表示构件 C 对外请求服务的接口集合,其中任何一个接口 $Ite_j \in I_C^R$ 表示构件对外所需的请求接口,它包含一组请求调用接口方法操作 $Ite_j=\{a_1^j,a_2^j,\cdots,a_{nj}^j\}$。

(4)IL_C 表示构件的内部连接关系。每一构件内部连接 $il_i \in IL_C$,形式化定义为 $il_i=\langle Ite_i,Ite_j\rangle$,其中,$Ite_i \in I_C^P \wedge I_C^R$,$Ite_j \in I_C^R$,称 Ite_i 为内部请求接口,Ite_j 为内部服务接口。一种情况是构件为响应服务接口 Ite_i 接收到的请求,从而通过请求接口 Ite_j 向其他构件发送新的请求;另一种情况是构件为响应请求接口 Ite_i 请求其他构件所返回的结果,从而通过接口 Ite_j 向其他构件发送新的请求。

(5)A_C 表示构件的有穷接口动作集。其包括服务、需求和内部动作集,分别用 A_C^P,A_C^R,A_C^H 表示,且三者互不相交。

(6)P_C 表示构件的行为协议,该协议通过 PA 形式化定义,描述了本构件对外交互行为的协议约束与规范,P_C 对应于一个二元组$\langle S_C,T_C\rangle$,其中:

1)S_C 表示构件交互行为的非空有限状态集,S_{Init} 为初始状态,S_{Fina} 为终止状态。

2)$T_C \subseteq S_C \times A_C \times S_C$ 表示构件对外交互行为的有限状态转换集,可用三元式$(s,a,s') \in T_C$ 表示,其中 $s,s' \in S_C$,$a \in A_C$ 记作 $s \xrightarrow{a} s'$,表示构件对外交互行为状态在动作 a 下从状态 s 转换到状态 s'。本文中的构件 C 不存在(s,a,s')和(s,a,s'')同时存在的情况,其中 $s,s',s'' \in S_C$,$a \in A_C$,$s' \neq s''$。

进程代数中的并行组合运算符“$\|$”[12]被用于从若干并发交互的子系统的行为描述中计算得到它们组合在一起的总行为[13],因此对于构件的对外连接集和行为协议的定义和描述,使用该推理规则可以给出多个构件组合后的行为描述,再结合构件系统的执行语义就可以得到构件系统的状态转换关系。多个构件组装成的构件系统定义如下,其中包含了构件系统的执行语义。

3.2 构件系统建模

定义 2(构件系统) 构件系统是一个四元组,$M=\langle N_M,C_M^{Set},L_M,S_M\rangle$,其中:

(1)N_M 为构件系统的名称。

(2)$C_M^{Set}=\{C|C=\langle N_C,I_C^P,I_C^R,IL_C,A_C,P_C\rangle\}$,是构件对象组成的集合。

(3)L_M 表示有穷的构件对外连接集,体现了构件系统中构件与构件之间的组装连接信息,其中每一对构件连接 $l_i \in L_M$ 定义为三元组 $l_i=\langle C_N,Ite_i,LC_i\rangle$,在 l_i 中:

1)C_N 为连接关系中的一个构件名称;

2)$Ite_i \in I_C^P \wedge I_C^R$,表示相连构件之间的接口;

3)LC_i 表示构件 C_N 与构件系统中相连接的构件。

对于两个相连的构件 $C_i=\langle N_{C_i},I_{C_i}^P,I_{C_i}^R,IL_{C_i},A_{C_i},P_{C_i}\rangle$ 和 $C_j=\langle N_{C_j},I_{C_j}^P,I_{C_j}^R,IL_{C_j},A_{C_j},P_{C_j}\rangle$,如果 C_i 请求调用 C_j 的提供服务接口 Ite,则这样的 Ite 可以称为 C_i 和 C_j 的共享请求/服务接口,用 $e(C_i,C_j)$表示;与其对应的构件间的接口方法操作集为 $a(C_i,C_j)$,表示接口 Ite 中的方法操作是需要同时执行的动作,即接口方法操作动作同步。

(4)S_M 表示构件系统的状态,$S_M=\{s_1,s_2,\cdots,s_n\}$($s_i \in S_{C_i}$,$0 \leqslant i \leqslant n$),$S_{Init}$ 为初始化状态,S_{Fina} 为终止状态。

定义 3(构件系统的执行语义) 设 S 和 S'是构件系统的两个不同的状态,其中 $S=\{s_1,s_2,\cdots,s_n\}$,$S'=\{s_1',s_2',\cdots,s_n'\}$,以下两种情况之一被满足时,构件系统的状态才可以通过动作 a 从状态 S 转换到状态 S'。

1)有动作 $a \in A_{C_i}^H$,且$(s_i,a,s_i') \in T_{C_i}$,同时,对于构件系统内的其他构件 C_j($1 \leqslant j \leqslant n, j \neq i$)而言,有 $s_j=s_j'$;

2)有动作 $a \in Ite, Ite \in e(C_i,C_j)$($1 \leqslant i,j \leqslant n, i \neq j$),如果$(s_i,a,s_i') \in T_{C_i}$且$(s_j,a,s_j') \in T_{C_j}$,同时,对构件系统内的其他构件 C_k($1 \leqslant k \leqslant n, k \neq i \neq j$),有 $s_k=s_k'$。

4 构件系统外部行为进程代数表达式形式提取算法及一致性验证准则

4.1 构件系统外部行为进程代数表达式形式提取算法

构件系统状态的转换是由一系列的构件动作引起的,包括内部不可见的动作和外部可观察的动作。这些动作可以抽象成构件系统的行为,它们确实是已经发生并引起了构件系统状态的变化,如何将这些导致构件系统状态转换的动作提取出来是庞大并且复杂的工作,为此,本文提出了一种基于深度优先遍历算法的构件系统外部行为进程代数表达式形式提取算法,该算法如下所示。

算法 1 构件系统外部行为进程代数表达式形式提取算法

```
Input:int Start;int End;Graph g
Output:String process//构件系统外部行为进程代数表达式形式
1. Begin
2.   Stack s;String process
3.   Init (g);//初始化图,设置图的顶点和边,顶点是构件系统的状态节点,边是导致状态转换的动作
4.   s.push(Start);
5.   While (!s.isEmpty())
6.   {int v=getAdjUnvisitedVertex(s.peek());//得到未被访问,与栈顶元素相连,且不在栈中的顶点
7.     If (v==-1)//如果不存在
8.      {s.pop();}
9.     Else {s.push();}
10.    If (!s.isEmpty() && End==s.peek())//一条图深度优先搜索的路径,即进程代数表达式形式
11.      {process=printStack(s);//从栈底到栈顶输出栈中两两元素的边,即动作
12.        process+=process+"+";
13.        s.pop();}}
14.  Return process;
15. End
```

在执行算法之前需将构件系统状态及状态转换的动作序列用图 Graph 来表示,其中图的节点为构件系统的状态节点,它包括了初始状态节点编号(Start)及结束状态节点编号(End),图的节点之间的边即为导致状态转换的动作。该算法首先初始化图 Graph,并将初始状态 Start 入栈。然后,算法进入 While 循环,在循环中查看栈顶状态节点 s 在图中是否有可以到达、没有入栈且没有从这个节点 s 出发访问过的状态节点(用 AdjUnvisitedVertex 函数来判断)。如果有,则

将找到的状态节点入栈;如果没有,弹出栈顶状态节点 s,当栈顶元素为终点状态时,打印输出的栈中元素,弹出栈顶状态节点。重复执行 While 循环直至栈为空,则图 Graph 中从开始状态节点出发到终点状态节点的所有的动作序列(即构件系统的外部行为)都被找到。

4.2　构件系统动态演化前后一致性验证准则

本文从外部行为的角度,提出构件系统演化一致性的验证准则。由于行为协议描述了从构件外部对构件行为的观察,因此在对构件进行演化时,若演化之后的构件在整个构件系统中依然满足用户规定的行为,则从构件外部观察,被演化的构件在演化前后并没有太大差异,即使演化前和演化后其实现方式发生了巨大的变化。由此可见,外部行为一致性的保持的关键在于演化后的构件系统依然能"模拟"用户要求的行为。由于构件系统内的构件内部是不可观察的,因此这种"模拟"是从外部观察的角度要求的一种"弱模拟",它只关注外部能观察到的"外部行为",而不是外部无法观察到的"内部行为"。

定义 4(弱互模拟)　PA 上的某二元关系 B 为弱互模拟关系,当且仅当:无论何时,若有 $(E_1,E_2)\in B$,则对所有操作 $a\in Act$ 而言,下列两个条件同时满足:

(1)若 $E_1\xrightarrow{a}E_1'$,则存在 E_2',$E_2\overset{\wedge}{\underset{a}{\mapsto}}E_2'$,且 $(E_1',E_2')\in B$;

(2)若 $E_2\xrightarrow{a}E_2'$,则存在 E_1',$E_1\overset{\wedge}{\underset{a}{\mapsto}}E_1'$,且 $(E_1',E_2')\in B$。

弱互摸拟关系的集合为观察等价(Observational Equivalences)关系,可表示为 $E_1\approx_B E_2$,其中 $\overset{\sigma}{\mapsto}\equiv\overset{\tau^m}{\Rightarrow}\Rightarrow\overset{\sigma}{\Rightarrow}\overset{\tau^n}{\Rightarrow}(\sigma=a_1\cdots a_n)$,$\hat{a}$表示忽略动作中 τ 的影响,即若 $a=\tau$,则$\hat{a}=\varepsilon$,ε 表示空操作序列,否则$\hat{a}=a$。

构件系统内部的构件演化导致构件系统的演化,如果演化前的构件系统的外部行为与演化后的构件系统外部行为是一种等价关系,由于构件系统的外部行为由构件系统内的多个构件外部行为计算组合而成,那么就可以保证演化后的构件系统 $M_{evolution}$ 中已完成演化的构件能够与构件系统 M 中未演化构件间的交互兼容,即这种演化后的构件体现的外部行为对用户来说并没有差别,从而保证了演化后的整个构件系统所体现的功能和用户的需求也并没有差异,那么就得到了构件系统演化前后行为一致性的准则。如一致性验证准则所示,凡是演化前后的构件系统的外部行为满足一致性验证准,则可认为演化后的构件系统与演化前的构件系统是保持一致的,动态演化是可以实施的。

一致性验证准则(构件系统):如果演化前的构件系统 M 的外部行为和演化后的构件系统 $M_{evolution}$ 的外部行为满足弱互模拟关系,那么这两个构件系统的外部行为等价,即演化后的构件系统与演化前的构件系统满足一致性关系,记为$M=M_{evolution}$。

基于构件组合的构件系统进行动态演化,其中一个构件的动态演化可以导致整个构件系统的演化,如果演化前构件系统 M 的外部行为与演化后的构件系统 $M_{evolution}$ 的外部行为满足定义 4,即它们之间是弱互模拟关系,那么就可以说演化前后构件系统的外部行为等价,也即演化后的构件系统中的构件与原构件系统中其他的构件之间的交互行为是兼容的,满足一致性保持,因此可以保证演化后的构件系统与演化前的构件系统满足一致性定义,即 $M=M_{evolution}$。

5　案例分析

5.1　网上在线列车订票的构件系统建模

为了验证本文所提的构件系统演化前后一致性验证方法确实是可行且有效的,以一个经典的网上应用系统——网上在线列车订票构件系统为例来具体说明。在使用网络进行列车订票时,有 9 类构件参与其中并组成了在线列车订票的构件系统(Online Train Ticket Booking System,OTTBS),如图 3 所示。

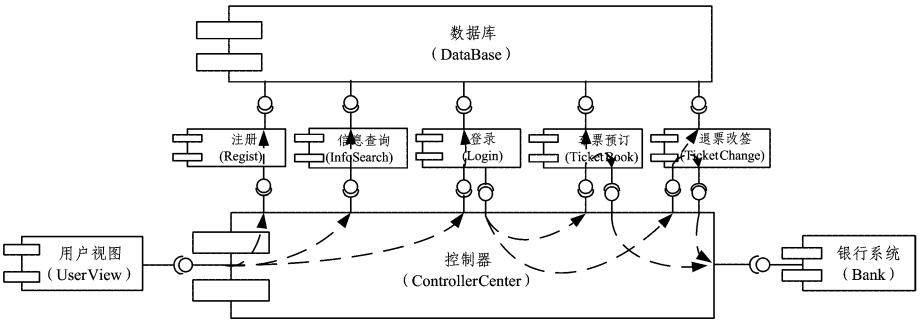

图 3　网上在线列车订票构件系统

图 3 中,9 类构件分别是:用户视图(UserView)、控制器(ControllerCenter)、用户注册(Regist)、用户登录(Login)、信息查询(InfoSearch)、车票预订(TicketBook)、退票与改签(TickeChange)、数据库(DataBase)和银行系统(Bank)。整个系统的运作过程大致如下:用户不需注册即可通过信息查询构件对列车的信息进行获取,但是如果要对车票进行预定或者退票和改签,则必须先进行注册并登录后,才可以使用相关的构件功能。用户登录之后选择车票预订或者退票和改签服务,ControllerCenter 构件会请求构件 Bank 完成用户在线支付相应款项或者退款操作。当用户需要使用退票或者改签功能时,TicketChange 构件会对构件 ControllerCenter 发出请求并查询用户已有的车票信息,其中构件 Regist、构件 Login、构件 InfoSearch、构件 TicketBook 和构件 TicketChange 需要请求调用铁路公司的数据库构件 DataBase,并与之实时互动。

由网上在线列车订票构件系统可知,整个构件系统共有 15 个对外连接,其中,构件 ControllerCenter 为响应从服务接

口 $Ite_{Controller\_User}$ 接收的请求，通过接口 $Ite_{Controller\_Regist}$，$Ite_{Controller\_InfoSearch}$ 和 $Ite_{Controller\_Login}$ 分别向构件 Regist、构件 InfoSearch和构件 Login 发送新的请求，故有内部连接〈$Ite_{Controller\_User}$，$Ite_{Controller\_Regist}$〉，〈$Ite_{Controller\_User}$，$Ite_{Controller\_InfoSearch}$〉和〈$Ite_{Controller\_User}$，$Ite_{Controller\_Login}$〉；构件 Regist、构件 InfoSearch 和构件 Login 为响应从服务接口 $Ite_{Controller\_Regist}$，$Ite_{Controller\_InfoSearch}$ 和 $Ite_{Controller\_Login}$ 接收的请求，分别通过接口 Ite_{Regist_Data}，$Ite_{InfoSearch\_Data}$ 和 Ite_{Login_Data} 向构件 DataBase 发送新的请求，故有内部连接〈$Ite_{Controller\_Regist}$，Ite_{Regist_Data}〉，〈$Ite_{Controller\_InfoSearch}$，$Ite_{InfoSearch\_Data}$〉和〈$Ite_{Controller\_Login}$，$Ite_{Login\_Data}$〉。当用户登录之后，需要使用购票服务时，构件 ControllerCenter为响应从服务接口 $Ite_{ControllerCenter\_Login}$ 接收的购票请求，通过接口 $Ite_{Controller\_TicketBook}$ 向构件 TicketBook 发送新的请求，故有内部连接〈$Ite_{ControllerCenter\_Login}$，$Ite_{Controller\_TicketBook}$〉，构件 TicketBook 为响应从服务接口 $Ite_{Controller\_TicketBook}$ 接收的购票请求，从而通过接口 $Ite_{TicketBook\_Data}$ 和接口 $Ite_{ControllerCenter\_TicketBook}$ 分别发送车次信息查询请求及扣款请求；构件 ControllerCenter 为了响应从服务接口 $Ite_{ControllerCenter\_TicketBook}$ 接收到的扣款请求，通过接口 $Ite_{Controller\_Bank}$ 向构件 Bank 发送新的请求，故有内部连接〈$Ite_{Controller\_TicketBook}$，$Ite_{ControllerCenter\_TicketBook}$〉，〈$Ite_{Controller\_TicketBook}$，$Ite_{TicketBook\_Data}$〉和〈$Ite_{ControllerCenter\_TicketBook}$，$Ite_{Controller\_Bank}$〉；同理，如果用户需要使用退票或者改签服务时，有内部连接〈$Ite_{ControllerCenter\_Login}$，$Ite_{Controller\_TicketChange}$〉，〈$Ite_{Controller\_TicketChange}$，$Ite_{ControllerCenter\_TicketChange}$〉，〈$Ite_{Controller\_TicketChange}$，$Ite_{TicketChange\_Data}$〉和〈$Ite_{ControllerCenter\_TicketChange}$，$Ite_{Controller\_Bank}$〉。

5.2 网上在线列车订票的构件系统形式化描述与一致性验证

限于篇幅，本节根据定义 1 仅对构件 ControllerCenter 和构件 Bank 进行描述。

构件 ControllerCenter 的描述如下：

$ControllerCenter=\langle I^N_{SC}, I^P_{SC}, I^R_{SC}, IL_{SC}, A_{SC}, P_{SC}\rangle$

$I^N_{SC}: ControllerCenter$；

$I^P_{SC}: \{Ite_{Controller\_User}, Ite_{ControllerCenter\_Login}, Ite_{ControllerCenter\_TicketBook}, Ite_{ControllerCenter\_TicketChange}\}$；

$Ite_{Controller\_User}=\{login, logout, regist, regist\_r, getTrainInfo, getTrainInfo\_r, buytickey, buyTicket\_success, buyTicket\_failurechangeTicket, changeTicket\_r\}$；

$Ite_{ControllerCenter\_Login}=\{buytickey, buyTicket\_success, buyTicket\_failure, changeTicket, changeTicket\_r\}$；

$Ite_{ControllerCenter\_TicketBook}=\{recoup, recoup\_r\}$；

$Ite_{ControllerCenter\_TicketChange}=\{recoup, recoup\_r\}$；

$I^R_{SC}: \{Ite_{Controller\_Regist}, Ite_{Controller\_Login}, Ite_{Controller\_InfoSearch}, Ite_{Controller\_TicketBook}, Ite_{Controller\_TicketChange}, Ite_{Controller\_Bank}\}$；

$Ite_{Controller\_Regist}=\{regist, regist\_r, checkrepeat, checkrepeat\_r\}$；

$Ite_{Controller\_Login}=\{login, logout, checkaccount, checkaccount\_r, loginfailure, loginsuccess\}$；

$Ite_{Controller\_InfoSearch}=\{getTrainInfo, getTrainInfo\_r\}$；

$Ite_{Controller\_TicketBook}=\{buyTicket, buyTicket\_success, buyTicket\_failure\}$；

$Ite_{Controller\_TicketChange}=\{changeTicket, changeTicket\_r\}$；

$Ite_{Controller\_Bank}=\{recoup, recoup\_r, refund, refund\_r\}$；

$IL_{SC}: \{\langle Ite_{Controller\_User}, Ite_{Controller\_Regist}\rangle, \langle Ite_{Controller\_User}, Ite_{Controller\_InfoSearch}\rangle, \langle Ite_{Controller\_User}, Ite_{Controller\_Login}\rangle, \langle Ite_{ControllerCenter\_TicketBook}, Ite_{Controller\_Bank}\rangle, \langle Ite_{ControllerCenter\_TicketChange}, Ite_{Controller\_Bank}\rangle, \langle Ite_{ControllerCenter\_Login}, Ite_{Controller\_TicketBook}\rangle, \langle Ite_{ControllerCenter\_Login}, Ite_{Controller\_TicketChange}\rangle\}$；

$A_{SC}: \{A^P_{SC}, A^R_{SC}, A^H_{SC}\}$；

$A^P_{SC}: \{login, logout, regist, regist\_r, getTrainInfo, getTrainInfo\_r, buytickey, buyTicket\_success, buyTicket\_failure, changeTicket, changeTicket\_r, recoup, recoup\_r\}$；

$A^R_{SC}: \{regist, regist\_r, login, logout, getTrainInfo, getTrainInfo\_r, buyTicket, buyTicket\_success, buyTicket\_failure, changeTicket, changeTicket\_r, recoup, recoup\_r, refund, refund\_r, checkrepeat, checkrepeat\_r, checkaccount, checkaccount\_r, loginfailure, loginsuccess\}$；

$A^H_{SC}: \Phi$；

$P_{SC}: \{P_{SC} \triangleq P[ControllerCenter]_{Init}$

$P[ControllerCenter]_{Init} \triangleq regist.\ P[ControllerCenter]_1$

$P[ControllerCenter]_1 \triangleq regist\_r.\ P[ControllerCenter]_2$

$P[ControllerCenter]_2 \triangleq login.\ P[ControllerCenter]_3$

…

$P[ControllerCenter]_i \triangleq logout.\ P[ControllerCenter]_{Fina} P[ControllerCenter]_{Fina} \triangleq \underline{0}\}$。

构件 Bank 的描述如下：

$Bank=\langle I^N_B, I^P_B, I^R_B, IL_B, A_B, P_B\rangle$

$I^N_B: Bank$；

$I^P_B: \{Ite_{Controller\_Bank}\}$；

$Ite_{Controller\_Bank}=\{recoup, recoup\_r, refund, refund\_r, buyTicket\_success, buyTicket\_failure\}$；

$I^R_B: \Phi$；

$IL_B: \Phi$；

$A_B: \{A^P_B, A^R_B, A^H_B\}$；

$A^P_B: \{recoup, recoup\_r, refund, refund\_r, buyTicket\_success, buyTicket\_failure\}$；

$A^R_B: \Phi$；

$A^H_B: \Phi$；

$P_B: \{P_B \triangleq P[Bank]_{Init}$

$P[Bank]_{Init} \triangleq recoup.\ P[Bank]_1 + refund.\ P[Bank]_1$；

$P[Bank]_1 \triangleq recoup\_r.\ P[Bank]_2 + refund\_r.\ P[Bank]_2$；

$P[Bank]_2 \triangleq buyTicket\_success.\ P[Bank]_3 + buyTicket\_failure.\ P[Bank]_3$

…

$P[Bank]_i \triangleq logout.\ P[Bank]_{Fina}$

$P[Bank]_{Fina} \triangleq \underline{0}\}$。

根据定义 2 中构件系统的定义,在线列车订票的构件系统(Online Train Ticket Booking System,OTTBS)描述如下:

$OTTBS=\langle N_M, C_M^{Sa}, L_M, S_M\rangle$

$N_{OTTBS}: OTTBS$;

$C_{OTTBS}^{Sa}: \{UserView, ControllerCenter, Regist, Login, InfoSearch, TicketBook, TickeChange, Bank, DataBase\}$;

$L_{OTTBS}: \{\langle UserView, Ite_{Controller\_User}, ControllerCenter\rangle$;

$\langle Regist, Ite_{Controller\_Regist}, ControllerCenter\rangle$;

$\langle Login, Ite_{Controller\_Login}, ControllerCenter\rangle$;

$\langle InfoSearch, Ite_{Controller\_InfoSearch}, ControllerCenter\rangle$;

$\langle TicketBook, Ite_{Controller\_TicketBook}, ControllerCenter\rangle$;

$\langle TicketChange, Ite_{Controller\_TicketChange}, ControllerCenter\rangle$;

$\langle Bank, Ite_{Controller\_Bank}, ControllerCenter\rangle$;

$\langle Regist, Ite_{Regist\_Data}, DataBase\rangle$;

$\langle Login, Ite_{Login\_Data}, DataBase\rangle$;

$\langle InfoSearch, Ite_{InfoSearch\_Data}, DataBase\rangle$;

$\langle Login, Ite_{ControllerCenter\_Login}, ControllerCenter\rangle$;

$\langle TicketBook, Ite_{TicketBook\_Data}, DataBase\rangle$;

$\langle TicketChange, Ite_{TicketChange\_Data}, DataBase\rangle$;

$\langle TicketBook, Ite_{ControllerCenter\_TicketBook}, ControllerCenter\rangle$;

$\langle TicketChange, Ite_{ControllerCenter\_TicketChange}, ControllerCenter\rangle;\}$;

$S_{OTTBS}: \{S_{UserView}, S_{Regist}, S_{InfoSearch}, S_{Login}, S_{DataBase}, S_{TicketBook}, S_{TicketChange}, S_{Bank}, S_{ControllerCenter}\}$。

根据 OTTBS 并结合定义 3 中的执行语义得到 OTTBS 的状态转换图,如图 4 所示。

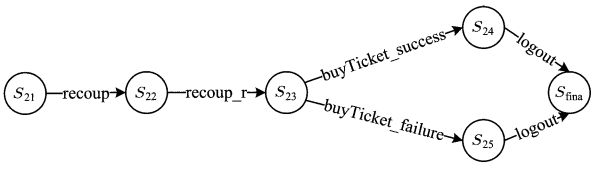

图 4　演化前构件系统状态转换图(部分)

由于银行系统进行升级,需要对构件 Bank 进行动态演化,即增加用户信用机制:构件 Bank 从用户的银行系统信用账户上扣除所需支付的车费后执行检查账户余额是否足够的内部动作,接着继续针对检查后的结果生成相应的银行系统通知消息,该生成动作也是内部动作,即如果余额足够或者信用账户可以透支,那么该银行系统消息会随着后续的外部动作 *buyTicket_success* 的执行推送给用户;如果余额不足并且信用账户不可以继续透支,则该银行系统消息会随着后续的外部动作 *buyTicket_failure* 的执行推送给用户,由于生成银行系统消息的内部动作包括直接先生成分类消息再分类和先分类消息再生成这两种不同的生成银行系统消息的方式,因此构件 Bnak 的动态演化方式有两种。

第一种演化方式:构件 Bank 演化后的构件 Evolution-Bank$_1$ 的描述如下(用 τ_1 来表示检查账户余额的内部动作,用 τ_2 来表示生成分类银行系统消息的内部动作):

$EvolutionBank_1=\langle I_{EB1}^N, I_{EB1}^P, I_{EB1}^R, IL_{EB1}, A_{EB1}, P_{EB1}\rangle$

$I_{EB1}^N: EvolutionBank_1$;

$I_{EB1}^P: \{Ite_{Controller\_EvolutionBank1}\}; Ite_{Controller\_EvolutionBank1}=\{recoup, recoup\_r, refund, refund\_r, buyTicket\_success, buyTicket\_failure\}$;

$I_{EB1}^R: \Phi$;

$IL_{EB1}: \Phi$;

$A_{EB1}: \{A_{EB1}^P, A_{EB1}^R, A_{EB1}^H\}$;

$A_{EB1}^P: \{recoup, recoup\_r, refund, refund\_r, buyTicket\_success, buyTicket\_failure\}$;

$A_{EB1}^R: \Phi$;

$A_{EB1}^H: \{\tau_1, \tau_2\}$;

$P_{EB1}: \{P_{EB1} \triangleq P[EvolutionBank_1]_{Init}$

$P[EvolutionBank_1]_{Init} \triangleq recoup.\ P[EvolutionBank_1]_1 + refund.\ P[EvolutionBank_1]_1$;

$P[EvolutionBank_1]_1 \triangleq recoup\_r.\ P[EvolutionBank_1]_2 + refund\_r.\ P[EvolutionBank_1]_2$;

$P[EvolutionBank_1]_2 \triangleq \tau_1.\ P[EvolutionBank_1]_3$

$P[EvolutionBank_1]_3 \triangleq \tau_2.\ P[EvolutionBank_1]_4$

$P[EvolutionBank_1]_4 \triangleq buyTicket\_success.\ P[EvolutionBank_1]_5 + buyTicket\_failure.\ P[EvolutionBank_1]_5$

…

$P[EvolutionBank_1]_i \triangleq logout.\ P[EvolutionBank_1]_{Fina}$

$P[EvolutionBank_1]_{Fina} \triangleq \underline{0}\}$。

限于篇幅,本文不再对演化后的构件系统进行描述,第一种演化方式动态演化后的构件系统 $OTTBS_{Evolution1}$ 的状态转换图如图 5 所示。

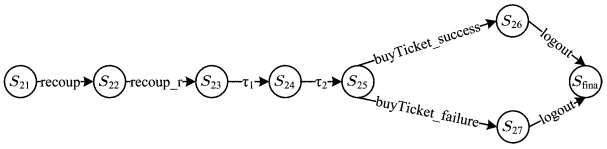

图 5　第一种演化方式演化后的构件系统状态转换图(部分)

第二种演化方式构件:Bank 演化后的构件 Evolution-Bank$_2$ 的描述如下(用 τ_1 来表示检查账户余额的内部动作,用 τ_2 和 τ_3 来表示分类生成银行系统消息的内部动作):

$EvolutionBank_2=\langle I_{EB2}^N, I_{EB2}^P, I_{EB2}^R, IL_{EB2}, A_{EB2}, P_{EB2}\rangle$

$I_{EB2}^N: EvolutionBank_2$;

$I_{EB2}^P: \{Ite_{Controller\_EvolutionBank1}\}; Ite_{Controller\_EvolutionBank2}=\{recoup, recoup\_r, refund, refund\_r, buyTicket\_success, buyTicket\_failure\}$;

$I_{EB2}^R: \Phi$;

$IL_{EB2}: \Phi$;

$A_{EB2}: \{A_{EB2}^P, A_{EB2}^R, A_{EB2}^H\}$;

$A_{EB2}^P: \{recoup, recoup\_r, refund, refund\_r, buyTicket\_success, buyTicket\_failure\}$;

$A_{EB2}^R: \Phi$;

$A^{H}_{EB2}:\{\tau_1,\tau_2,\tau_3\}$;

$P_{EB2}:\{P_{EB2}\triangleq P[EvolutionBank_2]_{Init}$

$P[EvolutionBank_2]_{Init}\triangleq recoup.\ P[EvolutionBank_2]_1+refund.\ P[EvolutionBank_2]_1$;

$P[EvolutionBank_2]_1\triangleq recoup\_r.\ P[EvolutionBank_2]_2+refund\_r.\ P[EvolutionBank_2]_2$;

$P[EvolutionBank_2]_2\triangleq \tau_1.\ P[EvolutionBank_2]_3$

$P[EvolutionBank_2]_3\triangleq \tau_2.\ P[EvolutionBank_2]_4+\tau_3.\ P[EvolutionBank_2]_4$

$P[EvolutionBank_2]_4\triangleq buyTicket\_success.\ P[EvolutionBank_2]_5+buyTicket\_failure.\ P[EvolutionBank_2]_5$

…

$P[EvolutionBank_2]_i\triangleq logout.\ P[EvolutionBank_2]_{Fina}$

$P[EvolutionBank_2]_{Fina}\triangleq \underline{0}\}$。

第二种演化方式动态演化后的构件系统 $OTTBS_{Evolution2}$ 的状态转换图如图 6 所示。

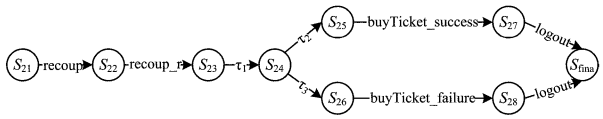

图 6 第二种演化方式演化后的构件系统状态转换图(部分)

利用图 4—图 6 中的构件系统状态转换图及算法 1 得到演化前和两种不同动态演化方式演化后的构件系统外部行为进程代数表达式形式,如图 7—图 9 所示。

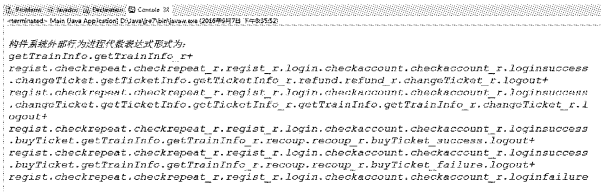

图 7 演化前构件系统外部行为进程代数表达式

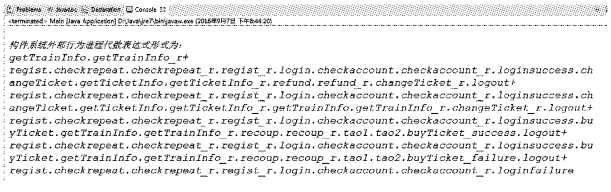

图 8 第一种演化方式演化后构件系统外部行为进程代数表达式

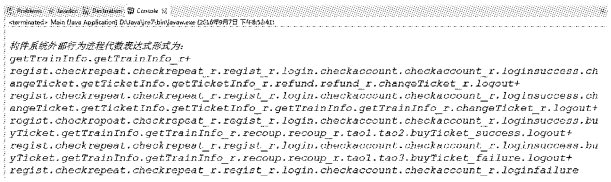

图 9 第二种演化方式演化后构件系统外部行为进程代数表达式

由定义 4 可知,第一种演化方式演化后的构件系统 $OTTBS_{Evolution1}$ 与演化前的构件系统 OTTBS 之间为弱互模拟关系;而第二种演化方式演化后的构件系统 $OTTBS_{Evolution2}$ 与演化前的构件系统 OTTBS 之间则不满足弱互模拟关系。由于证明过程过于繁琐,将演化前后构件系统外部行为的进程代数表达式形式转换成 Pi 演算自动验证工具 MWB 能识别的格式并将其导入工具以进行验证,结果分别如图 10 和图 11 所示。结果表明 $OTTBS_{Evolution1}$ 与 OTTBS 之间确实为弱互模拟关系,而 $OTTBS_{Evolution2}$ 则不是,通过一致性验证准则可知构件系统 $OTTBS_{Evolution1}$ 与 OTTBS 之间是等价关系,因此第一种演化方式演化后的构件系统中的构件 $EvolutionBank_1$ 与原构件系统中其他构件之间的交互行为是兼容的,即满足一致性保持。因此,可以保证演化后的构件系统 $OTTBS_{Evolution1}$ 与演化前的构件系统 OTTBS 是满足一致性保持的。而 $OTTBS_{Evolution2}$ 与 OTTBS 不是等价关系,因此第二种演化方式演化后的 $EvolutionBank_1$ 与 OTTBS 中的其他构件的交互行为必定不兼容,即不满足一致性保持,该动态演化方式不能实施。

图 10 第一种演化方式演化前后构件系统外部行为一致性验证结果

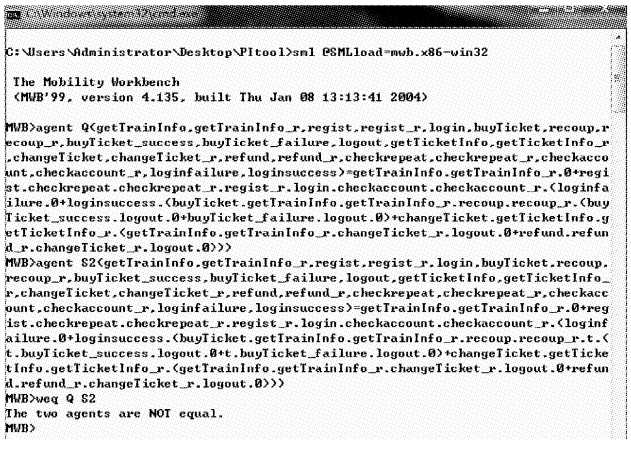

图 11 第二种演化方式演化前后构件系统外部行为一致性验证结果

结束语 针对构件动态演化后导致构件系统行为偏移演化前的构件系统行为的问题,即构件系统演化前后一致性保持的问题,本文基于进程代数提出了一种构件模型来形式化地描述构件外部交互行为,通过构件与构件的组合得到构件系统的模型,并通过构件系统外部行为提取算法提取了构件系统行为;同时基于进程代数中的弱互模拟理论提出了构件系统演化前后行为一致性的验证准则;在此基础上,将提取的

(下转第 113 页)

[10] DENG D,ZHANG W,LU S. Efficient concurrency-bug detection across inputs[C]// Acm Sigplan International Conference on Ojbect Oriented Prgramming Systems Languages & Applications. ACM,2013:785-802.

[11] LUCIA B,CEZE L,STRAUSS K. ColorSafe:architectural support for debugging and dynamically avoiding multi-variable atomicity violations[J]. ACM Sigarch Computer Architecture News,2010,38(3):222-233.

[12] GAO Q,ZHANG W,CHEN Z,et al. 2ndStrike:toward manifesting hidden concurrency typestate bugs[J]. ACM SIGPLAN Notices,2012,47(4):239-250.

[13] MUZAHID A,QI S,TORRELLAS J. Vulcan:Hardware support for detecting sequential consistency violations dynamically[C]// 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). IEEE,2012:363-375.

[14] DEMILLO R A,LIPTION R J,SAYWARD F G. Hints on Test Data Selection:Help for the Practicing Programmer[J]. Computer,1978,11(4):34-41.

[15] HAMLET R G. Testing Programs with the Aid of a Compiler[J]. IEEE Transactions on Software Engineering,1977,SE-3(4):279-290.

[16] ANDREWS J H,BRIAND L C,Labiche Y. Is mutation an appropriate tool for testing experiments? [C]// International Conference on Software Engineering,2005 (ICSE 2005). IEEE,2005:402-411.

[17] ZHONG H,ZHANG L,MEI H. An experimental study of four typical test suite reduction techniques[J]. Information & Software Technology,2008,50(6):534-546.

(上接第 86 页)

构件系统动作序列以 Pi 演算自动验证工具 MWB 工具的格式载入并进行对比,以判断演化前后的构件系统是否满足一致性保持;最后通过具体实例的分析,表明本文的构件系统动态演化一致性验证方法不仅能验证演化前后构件系统的一致性,也能检测出演化前后构件系统的不一致性,确实是可行且有效的。

本文基于进程代数中弱互模拟理论定义了一致性准则,通过提取演化前后构件系统的外部行为来对其一致性进行验证,主要考虑了封闭简单场景模式下构件系统演化前后的一致性验证问题,因此本文未来的工作将主要考虑开放复杂多场景模式下构件系统演化一致性的验证,并就构件系统动态演化过程的一致性问题进行研究。

参 考 文 献

[1] 王映辉. 构件式软件技术[M]. 北京机械工业出版社,2012.

[2] YANG F Q,MEI H,LI K Q. Software Reuse and Software Component Technology[J]. Acta Electronica Sinica,1999,27(2):68-75. (in Chinese)
杨芙清,梅宏,李克勤. 软件复用与软件构件技术[J]. 电子学报,1999,27(2):68-75.

[3] PLASIL F,VISOVSKY S. Behavior protocols for software components[J]. IEEE Transactions on Software Engineering,2002,28(11):1056-1076.

[4] LUO Y,LI X Y,GUAN L W,et al. Study on Behaavior Consistency of System on Component Evolution[J]. Computer Science,2008,35(1):266-270. (in Chinese)
罗毅,李兴宇,关连伟,等. 构件演化中的系统行为一致性的研究[J]. 计算机科学,2008,35(1):266-270.

[5] SHEN L M,MA C,WANG T. Research on behavioral consistency of component dynamic evolution based on process algebra[J]. Application Research of Computers,2009,26(4):1345-1348. (in Chinese)
申利民,马川,王涛. 基于进程代数的构件动态演化行为一致性研究[J]. 计算机应用研究,2009,26(4):1345-1348.

[6] MA C,SHEN L M,WANG T. Behavior Consistency Verification Method Based on Component Dynamic Evolution[J]. Computer Engineering,2010,36(6):80-83. (in Chinese)
马川,申利民,王涛. 基于构件动态演化的行为一致性验证方法[J]. 计算机工程,2010,36(6):80-83.

[7] ZHOU Y,HUANG Y K,HUANG Z Q,et al. Towards an Approach of Consistency Verification for Online Software Evolution in Open Environments[J]. Journal of Software,2015,26(4):747-759. (in Chinese)
周宇,黄延凯,黄志球,等. 一种开放环境下软件在线演化一致性验证方法[J]. 软件学报,2015,26(4):747-759.

[8] VICTOR B,MOLLER F. The Mobility Workbench—a tool for the π-calculus[C]// International Conference on Computer Aided Verification. Springer Berlin Heidelberg,1994:428-440.

[9] MILNER R. Communicating and mobile systems:the pi calculus[M]. Cambridge University Press,1999.

[10] BERGSTRA J A,KLOP J W. Fixed point semantics in process algebras[J]. Stichting Mathematisch Centrum Informatica,1982:1-21.

[11] HU H Y,LV J,MA X X,et al. Study on Behavioral Compatibility of Components in Software Architecture Using Object-Oriented Paradigm[J]. Journal of Software,2006,17(6):1276-1286. (in Chinese)
胡海洋,吕建,马晓星,等. 面向对象范型体系结构中构件行为相容性研究[J]. 软件学报,2006,17(6):1276-1286.

[12] BERNARDO M,CIANCARINI P,DONATIELLO L. Architecting families of software systems with process algebras[J]. ACM Transactions on Software Engineering and Methodology (TOSEM),2002,11(4):386-426.

[13] DAI F,LI T,XIE Z W,et al. Towards an algebraic semantics of software evolution process models[J]. Journal of Software,2012,23(4):846-863. (in Chinese)
代飞,李彤,谢仲文,等. 一种软件演化过程模型的代数语义[J]. 软件学报,2012,23(4):846-863.