ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.17, No.6, June 2006, pp.1276−1286
DOI: 10.1360/jos171276

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

面向对象范型体系结构中构件行为相容性研究*

胡海洋[1,2+], 吕 建[1,2], 马晓星[1,2], 陶先平[1,2]

[1](计算机软件新技术国家重点实验室(南京大学),江苏 南京 210093)

[2](南京大学 计算机软件研究所,江苏 南京 210093)

Study on Behavioral Compatibility of Components in Software Architecture Using Object-Oriented Paradigm

HU Hai-Yang[1,2+], LÜ Jian[1,2], MA Xiao-Xing[1,2], TAO Xian-Ping[1,2]

[1](State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210093, China)

[2](Institute of Computer Software, Nanjing University, Nanjing 210093, China)

+ Corresponding author: Phn: +86-25-83593694, Fax: +86-25-83593283, E-mail: hhy@ics.nju.edu.cn, http://www.nju.edu.cn

Hu HY, Lü J, Ma XX, Tao XP. Study on behavioral compatibility of components in software architecture using object-oriented paradigm. *Journal of Software*, 2006,17(6):1276−1286. http://www.jos.org.cn/1000-9825/17/1276.htm

Abstract: Software architecture (SA) provides a high-level abstraction for component-based software development. It's important to specify the interaction behavior of the components, verify the compatibility among the components, and ensure the deadlock-freedom of the composition configuration at the architectural level. Many component-based software architectures are using object-oriented paradigm, in which component composition is implemented by method invocations over component interfaces. Concentrating on the component composition in this kind of SA, this paper formally specifies the components and their interaction behaviors, distinguishes the caller's behavior from the callee's in interaction, and then presents a set of rules to verify the behavioral compatibility on the interfaces between the components composed together. Finally, an example of e-commerce application is presented to illustrate the feasibility and pertinence of the approach.

Key words: component; component composition; software architecture; behavioral compatibility; deadlock-free

摘 要: 软件体系结构(SA)为基于构件的软件开发提供了一种高层次的抽象.如何有效描述体系结构中构件的对外交互行为、验证组装构件间的行为相容及保证整个体系结构行为无死锁是其中较为重要的研究内容.在基于面向对象范型这类重要的软件体系结构中,构件组装通常是通过接口方法调用加以实现.针对这样一类基于软件体系结构的构件组装问题,形式描述与定义了构件及其对外交互协议,分析了服务请求构件与服务提供构件所能展示的不同行为,给出了组装交互的构件在请求/提供接口上协议级行为相容的一组验证规则及相关

* Supported by the National Natural Science Foundation of China under Grant Nos.60403014, 60233010 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant Nos.2005AA113160, 2005AA119010, 2005AA113030 (国家高技术研究发展计划(863)); the National Grand Fundamental Research 973 Program of China under Grant No.2002CB312002 (国家重点基础研究发展规划(973))

Received 2006-01-10; Accepted 2006-03-13

定理.最后,通过一个电子商务应用实例来说明所提出途径的可用性和针对性.

关键词: 构件;构件组装;软件体系结构;行为相容;无死锁
中图法分类号: TP311 文献标识码: A

基于构件的软件体系结构(SA)设计与开发已成为软件工程研究与开发实践所关注的重点[1].这样构建的软件系统具有良好的适应性、灵活性和易维护性,能较大限度地支持软件复用,因而受到越来越多的软件研究与开发人员的关注[2−6].SA 提高了软件开发者对复杂软件系统的抽象等级,更多地关注软件系统全局性的设计与描述,而不是底层计算实体中算法与数据结构的选择.软件体系结构与基于构件的软件开发是密切相连的,它提供了一种自顶向下、基于构件的软件系统开发途径[1,6],将软件系统分解为一组构件及构件之间交互的关系,从而使得软件开发者可以从全局的角度去设计与分析系统,并克服了传统的基于构件的软件开发过程中采用自底向上途径的局限.

面向对象技术对于基于构件的软件体系结构的研究具有重要的作用.从理论和方法的角度看,Mary Shaw 在文献[2]中定义了具有面向对象风格(object-oriented style)的软件体系结构.在这种风格的软件体系结构中,构件表现为对象的形式,它们是对数据表示与操作的封装,对象可实现一个或多个接口,对象之间通过功能方法调用进行连接.现有一些关于体系结构的工作中也采用了面向对象的范型[7,8],在这些工作中,构件具有类型、实例的概念,构件间通过显式定义的接口方法进行组装交互.另一方面,从工程实践的角度看,现有较为成熟的构件系统如 CORBA,EJB/J2EE,COM/DCOM 中所采用的构件组装调用机制均是基于面向对象思想的,即通过接口调用方法来加以实施.因此,无论从理论与方法的角度,还是从工程实践的角度,如何保证基于面向对象范型的体系结构中构件组装交互的正确性具有较为重要的意义.

目前,与这样一类基于体系结构的构件组装交互正确性相关的研究主要包括以下两个方面:(1) 从应用的角度来看,目前较为成熟的构件技术(如 CORBA 等)通过接口描述语言(IDLs)来规范构件之间的交互[9],但 IDLs 一般仅能定义交互接口中符号级方面的内容,即接口中的参数个数、顺序和类型等方面的内容,而对于构件之间正确交互所必需的行为协议方面的规定则无法加以保证[10].因此,对于功能接口完全匹配的两个交互构件而言,其在运行时仍会因行为协议的不相容而引发失败.因此,文献[10]认为,构件间能够正确组装交互不仅需要定义符号级(signature level)的规范,还需要定义协议级(protocol level)的交互规范,即构件间接口操作调用次序的约束与规范.因此,在设计基于构件的软件体系结构过程中,需要考虑到如何通过某种有效途径来描述与验证 SA 中组装构件间正确交互所必需的行为相容性(behavioral compatibility),及体系结构中行为无死锁(deadlock-free)等方面的问题;(2) 从理论研究的角度来看,20 世纪 90 年代后期,对软件体系结构中构件间协议级行为相容的研究逐步成为研究者探讨的热点[3−5,11,12].上述已有工作主要围绕某种特定的体系结构描述语言(ADL)展开分析讨论,而并非专门针对采用面向对象范型的 SA 场景.因此,相关的研究成果在这样的 SA 中使用时受到了一定的限制.

本文的主要工作就是在汲取现有理论研究成果[4,11,12]的基础上,针对基于面向对象范型的体系结构中构件组装交互正确性的问题,形式化地描述与定义了构件及其对外交互协议,分析了服务请求构件与服务提供构件在连接接口上所能展示的不同行为,给出了组装交互的两构件在请求/提供接口上协议级行为相容的一组验证规则及定理,包含一方对另一方有请求行为的情形中,基于观察等价关系的两构件接口行为相容规则和基于行为迹关系的行为相容规则;双方互有请求行为的情形中,基于观察等价关系的两构件接口行为相容规则和基于行为迹关系的接口行为相容必要条件.并在此基础上,对多个构件组装成的体系结构行为无死锁的情形也进行了分析与讨论.最后,本文通过这组规则验证了一个具体实例,以说明所提出的规则的可用性和针对性.

本文第 1 节概述所需用到的基础知识与基本概念.第 2 节给出扩展了行为协议的组装构件一般化模型,并对相关问题进行分析.有关构件间在交互接口上协议级行为相容及体系结构行为无死锁的分析在第 3 节给出.第 4 节通过具体实例来说明本文工作的特点.第 5 节是相关工作的比较和对本文的小结.

1 基本概念

与已有研究者的工作相似[4,7,11],本文用进程代数来形式化描述组装构件的对外行为协议,进而通过相关理论分析并验证构件间在交互接口上协议级行为相容的特性.本文采用意大利学者 M. Bernardo 提出的进程代数 PA[4,12]作为形式化工具来描述与验证构件的对外行为协议,其语法与语义规范定义如下:

定义 **1**. 系统中的进程可用如下的范式形式化定义:

$$E::=\underline{0}|a.E|E/L|E[\varphi]|E_1+E_2|E_1\|_S E_2|A.$$

其中,$\underline{0}$ 表示进程已停止,不能再执行任何动作(action);

"$a.E$"表示可执行动作 a,然后转化为进程 E;

"E/L"表示对于 E 中出现在 L 中相关动作将被屏蔽其可被外界观察的效果,即在运行时将其转化为进程的不可见动作 τ;

"$E[\varphi]$"表示对 E 中的相关执行动作 a 进行重标号(relabel),将其转化为 $\varphi(a)$;

"E_1+E_2"表示根据后续动作所从属的进程做出选择 E_1 或 E_2;

"$E_1\|_S E_2$"表示进程 E_1 与 E_2 对于从属于集合 S 里的动作同步执行,而对于其他动作则各自异步执行;

"A"表示常量,可进行这样的定义 $A \triangleq E$.

PA 的相关操作语义如下:

1. $a.E \xrightarrow{a} E$	2. $\dfrac{E \xrightarrow{a} E'}{E/L \xrightarrow{a} E'/L}$ 若 $a\notin L$	3. $\dfrac{E \xrightarrow{a} E'}{E/L \xrightarrow{\tau} E'/L}$ 若 $a\in L$
4. $\dfrac{E_1 \xrightarrow{a} E_1'}{E_1\|_S E_2 \xrightarrow{a} E_1'\|_S E_2}$ 若 $a\notin S$	5. $\dfrac{E_2 \xrightarrow{a} E_2'}{E_1\|_S E_2 \xrightarrow{a} E_1\|_S E_2'}$ 若 $a\notin S$	
6. $\dfrac{E_1 \xrightarrow{a} E_1', E_2 \xrightarrow{a} E_2'}{E_1\|_S E_2 \xrightarrow{a} E_1'\|_S E_2'}$ 若 $a\in S$	7. $\dfrac{E_1 \xrightarrow{a} E'}{E_1+E_2 \xrightarrow{a} E'}$	8. $\dfrac{E_2 \xrightarrow{a} E'}{E_1+E_2 \xrightarrow{a} E'}$
9. $\dfrac{E \xrightarrow{a} E'}{E[\varphi] \xrightarrow{\varphi(a)} E'[\varphi]}$	10. $\dfrac{E \xrightarrow{a} E'}{A \xrightarrow{a} E'}$ 若 $A \triangleq E$	

定义 **2**. 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,下列两条件同时满足:

- 若 $E_1 \xrightarrow{a} E_1'$,则存在 E_2',$E_2 \overset{\hat{a}}{\mapsto} E_2'$,且 $(E_1', E_2')\in B$;
- 若 $E_2 \xrightarrow{a} E_2'$,则存在 E_1',$E_1 \overset{\hat{a}}{\mapsto} E_1'$,且 $(E_1', E_2')\in B$.

弱互模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 $\overset{\sigma}{\mapsto} \equiv \overset{\tau^m}{\Rightarrow} \overset{\sigma}{\Rightarrow} \overset{\tau^n}{\Rightarrow}$.这里,$\overset{\sigma}{\Rightarrow} \equiv \xrightarrow{a_1} \ldots \xrightarrow{a_n}$ (若 $\sigma = a_1 \ldots a_n$); $\hat{a}$ 表示忽略动作中 τ 的影响,即若 $a=\tau$,则 $\hat{a}=\varepsilon$,这里 ε 表示空操作序列;否则 $\hat{a}=a$.

定义 **3**. 设进程 E 可定义为如下的有限状态变迁 L:$E\equiv S_0 \xrightarrow{a_1} S_1 \ldots \xrightarrow{a_{n-1}} S_{n-1} \xrightarrow{a_n} S_n$,进程 E 与该状态变迁 L 相对应的迹(trace)为 $\langle a_1,a_2,\ldots,a_n\rangle$;迹中的动作均能被外界所观察.

进程中产生的死锁反映了进程处于一种阻塞状态,即进程没有成功地终止,而又无法继续向下执行.无死锁(deadlock free)的进程可定义如下:

定义 **4**. 进程 E 是无死锁的,当且仅当对其状态变迁图中的任一非终止状态 s,存在着这样的一个可观察动作 a 和这样的一个状态 s',且有 $s \overset{a}{\Rightarrow} s'$.

2 基本模型

构件是 SA 的重要组成实体,一般来说有如下 3 类构件模型[6,8]:描述/分类模型、规约/组装模型和实现模型.描述/分类模型描述该构件在构件库中能被高效地分类、存储和检索的相关信息;规约/组装模型描述了构件的

功能和行为规约、构件应用的语境及构件间的交互与组装等信息;构件实现模型则描述了构件在源程序级的实现机制,如 Microsoft 的 COM/DCOM 模型、OMG 的 CCM 模型及 Sun 的 EJB 模型等.

在基于面向对象范型的体系结构中,构件规约/组装模型描述了构件对外服务功能接口、请求接口、与体系结构中其他构件实例的接口连接情况及对外交互的相关规范与约束等方面.本文参照了现有 ADLs 中对构件组装模型的定义,如 ABC[6,8],Darwin[13],Wright[11],LEDA[7],PADL[4]等,并在我们原有工作的基础上[14,15]给出基于面向对象范型的 SA 中扩展了行为协议的构件一般化组装模型.

定义 5. 扩展了行为协议的构件组装模型可表述为这样的一个五元组:$C=\langle I_C^P, I_C^R, A_C, L_C, P_C\rangle$,其中:

I_C^P 为构件 C 的对外提供接口集,$I_C^P=\{Ite_1, Ite_2, \ldots, Ite_n\}$.其中任何 $Ite_i\in I_C^P$,Ite_i 为 C 提供给外界调用的服务接口,它包含一组服务提供接口方法操作,即 $Ite_i=\{a_1^i, a_2^i, \ldots, a_{n_i}^i\}$;

I_C^R 为 C 的对外请求接口集,任何 $Ite_j\in I_C^R$,Ite_j 为 C 对外界所需的请求接口,它包含一组请求调用接口方法操作,即 $Ite_j=\{a_1^j, a_2^j, \ldots, a_{n_j}^j\}$;

A_C 为构件有穷接口动作集,包括需求、服务和内部动作集: A_C^R, A_C^P, A_C^H,且三者互不相交;

L_C 为有穷的构件对外连接集,表达了本构件与体系结构中其他构件间的组装连接信息;每一构件对外连接 $l_i\in L_c$ 形如 $l_i=\langle RIte_i, PIte_i, Ins_i, PL_i\rangle$.其中 $RIte_i\in I_C^R$ 为本构件对所连构件的一个请求接口;$PIte_i$ 为所连构件的一个提供接口;$RIte_i$ 与 $PIte_i$ 在语法上是完全匹配的;Ins_i 为所连接构件的实例;PL_i 为所连接构件实例所处位置;

P_C 为构件的行为协议,它通过 PA 进行形式化定义,描述了本构件对外交互行为协议约束与规范;P_C 对应于这样的(S_C, Γ_C):其中 S_C 为构件交互行为的有穷状态集,s_{Init}, s_{Fina} 分别为初始化、终止状态;$\Gamma_C\subseteq S_C\times A_C\times S_C$ 为对外交互行为的有穷状态转换集.

本文设定构件 C 的状态转换 Γ_C 中不存在如下情形:$((s,a,s'),(s,a,s'')\in\Gamma_C)\wedge(s,s',s''\in S_C, a\in A_C)\wedge s'\neq s''$;我们用状态转换序列来表示构件 C 的一个对外交互行为迹,形如:$s_0\xrightarrow{a_0}s_1\xrightarrow{a_1}\ldots\xrightarrow{a_{n-1}}s_{n-1}\xrightarrow{a_n}s_n$,表示构件对外交互行为状态从 s_0 开始,通过动作 a_0 转换到状态 s_1,又通过动作 a_1 转换到状态 s_2,如此类推.形如 $s_0=s_{Init}, s_n=s_{Fina}$ 的行为迹则称为 C 的一个完整对外行为迹,在本文的后续章节中,用 $trace_{co}(P_C)$ 表示构件 C 所有能产生的完整对外行为迹的集合.

构件自身所定义的对外交互行为协议需要是无死锁且可正确终止的.可正确终止的形式定义如下:

定义 6. 构件 $C=\langle I_C^P, I_C^R, A_C, L_C, P_C\rangle$,$P_C$ 是可正确终止的,当且仅当对于从 P_C 的初始状态 s_{Init} 可达的任意中间状态 s',若 $s'\neq s_{Fina}$,则存在 $\sigma\in A_C^*$,$s'\overset{\sigma}{\Rightarrow}s_{Fina}$.

对于体系结构中多个构件实例同时连接某构件的一个相同服务提供接口时,通常的做法是复制多个这样的服务构件实例[13],分别与这些请求构件实例组装连接.文献[12]中定义了扩展的 and/or 连接,用于描述多个构件实例同时连接某服务构件实例的一个服务提供操作,其实现方式是对该服务提供操作的行为协议进行了重新改写,对不同构件实例的连接操作口进行了重新标号,并定义了更为细致的、与多构件实例交互的协议规则,实质上仍是将其转化为多个构件实例同时连接同一服务构件实例的不同服务提供操作的情形进行验证.基于以上考虑,本文约定在这样的组装结构中,不存在两个构件实例同时连接某构件实例的一个相同服务接口的情形.

由于在描述与验证构件对外行为协议时,一般的做法是将连接子(connectors)视为与构件相同的实体,对其也需定义对外交互接口和相关的行为协议,因此在验证构件协议级行为相容的层面上,这样的连接子与 SA 中的构件并无区别[4,7],且本文也不涉及构件之间的具体连接细节,故本文没有额外引入对连接子(connectors)的定义与描述.

对于两相连构件 $C_i=\langle I_{C_i}^P, I_{C_i}^R, A_{C_i}, L_{C_i}, P_{C_i}\rangle$,$C_j=\langle I_{C_j}^P, I_{C_j}^R, A_{C_j}, L_{C_j}, P_{C_j}\rangle$,若 C_i 请求调用 C_j 的提供接口 Ite,则两者的组装交互行为可表示为 $P_{C_i}\|_{Ite}P_{C_j}$,即可通过验证 $P_{C_i}\|_{Ite}P_{C_j}$ 的相关性质来分析 C_i, C_j 依照各自所定义的行为协议 P_{C_i}, P_{C_j} 在接口 Ite 上能否组装交互成功,这样的 Ite 可称为 C_i, C_j 间的共享请求/提供接口.不失一般性,

本文设定相连构件除出现在共享的请求/提供接口中的方法动作同名以外,不同共享接口集内的接口名与接口中的方法名互不相交.相连两构件间的共享接口集可表示为 $e(C_i,C_j)$,它反映了相连构件间通过一个或多个请求/提供接口进行组装交互的关系,则与 $e(C_i,C_j)$相对应的构件间同步接口方法集为 $a(C_i,C_j)=\bigcup_{1\le k\le|e(C_i,C_j)|} Ite_k$ ($Ite_k\in e(C_i,C_j)$);对于不相连的构件间共享接口集 $e(C_i,C_j)=\varnothing$.$e(C_i,C_j)$与 $a(C_i,C_j)$反映了相连构件间通过请求/提供接口连接交互,且在接口方法上需同步运行的关系.

多个构件$\{C_1,C_2,\ldots,C_n\}$($C_i=\langle I_{C_i}^P, I_{C_i}^R, A_{C_i}, L_{C_i}, P_{C_i}\rangle$)可组装成一定形式的体系结构,体系结构的行为 P_M 可形式化表示为 $P_{C_1}\|_{a(C_1,C_2)} P_{C_2}\|_{a(C_1,C_3)\cup a(C_2,C_3)} P_{C_3}\|\ldots\|_{a(C_1,C_n)\cup\ldots\cup a(C_{n-1},C_n)} P_{C_n}$;$P_M$ 中的状态 s_M 为$(s_1,s_2,\ldots,s_n)$($s_i\in S_{C_i}$,$0\le i\le n$),即 $s_M\in(S_{C_1}\times S_{C_2}\times\ldots\times S_{C_n})$;$P_M$ 行为状态的转换有如下两种情形:

定义 **7**. 设 s 和 s'为 P_M 的两个不同状态,且 $s=(s_1,s_2,\ldots,s_n)$,$s'=(s_1',s_2',\ldots,s_n')$,当满足以下条件之一时,$P_M$ 可通过动作 a 从状态 s 到达状态 s',即 $s\xrightarrow{a}s'$:

对于动作 $a\in A_{C_i}^H$,在 Γ_{C_i} 中存在着一个这样的行为状态转换:(s_i,a,s_i'),同时对其他构件 $C_j(1\le j\le n,j\ne i)$,有 $s_j=s_i'$;

对于动作 $a\in Ite,Ite\in e(C_i,C_j)(1\le i,j\le n,i\ne j)$,若$(s_i,a,s_i')\in\Gamma_{C_i}\wedge(s_j,a,s_j')\in\Gamma_{C_j}$,同时对于其他构件 $C_k(1\le k\le n,k\ne i,j)$,有 $s_k=s_k'$.

3 行为相容的分析与验证

3.1 构件间协议级的行为相容

由 PA 定义对外交互协议的两个构件,设它们之间通过接口 *Ite* 组装交互,则其在 *Ite* 上的动作状态变迁是同步发生的,若在此接口上存在着行为不相容的情形,双方的行为将不能同步一致,从而使其中的一方或双方陷入无法继续向下执行的状态.因此,对于组装交互的两个构件在其同步交互接口 *Ite* 上,若不会因为发生行为的不一致而产生死锁的情形,则可认为两者在该接口上是行为相容的.对构件间接口行为相容的定义如下:

定义 **8**. 相连构件 $C_i=\langle I_{C_i}^P, I_{C_i}^R, A_{C_i}, L_{C_i}, P_{C_i}\rangle$,$C_j=\langle I_{C_j}^P, I_{C_j}^R, A_{C_j}, L_{C_j}, P_{C_j}\rangle$间通过接口 *Ite* 组装交互,则 C_i,C_j 在接口 *Ite* 上同步交互时其行为相容当且仅当:对于 $P_{C_i}\|_{Ite}P_{C_j}$ 的初始状态(s_{Init}^i,s_{Init}^j),有$(s_{Init}^i,s_{Init}^j)\overset{\sigma}{\Rightarrow}(s_{Fina}^i,s_{Fina}^j)$,$\sigma\in(A_{C_i}\cup A_{C_j})^*\wedge\sigma\uparrow Ite\ne\langle\cdot\rangle$.

在上述定义中用到了受限操作"↑","$\sigma\uparrow I$"表示仅保留行为迹σ中出现在集合 I 中的操作名,如$\langle a,b,c,c,d\rangle\uparrow\{a,c\}=\langle a,c,c\rangle$.

一般说来,对于构件 C_1,C_2 通过 C_2 提供的服务接口 *Ite* 进行组装交互的情形中,存在着如下两种行为协议不相容的情形:(1) 服务请求方的请求得不到满足.设构件 C_i 在执行完 a_k 操作后需执行 a_{k+1} 操作,而若构件 C_j 仅能提供 b_{k+1} 操作($a_{k+1}\ne b_{k+1}$),且双方均需要对方的同步才能进入下一状态,故此时双方陷入无法继续往下执行的状态,即构件之间的交互行为不相容;(2) 服务请求方的请求尽管得到满足,但不符合服务提供方的规范要求,即调用后使得服务提供方陷入无法继续往下执行的状态.如 C_i 执行完操作 a_k 后就已结束在接口上的交互,但 C_j 还需执行 a_{k+1} 操作后在接口上的交互才结束,这样将使得 C_j 陷入等待同步操作 a_{k+1} 的过程中.

另一方面,对于构件 $C_j=\langle I_{C_j}^P, I_{C_j}^R, A_{C_j}, L_{C_j}, P_{C_j}\rangle$而言,它可提供多个服务接口 $I_{C_j}^P=\{Ite_1,Ite_2,\ldots,Ite_n\}$供外界多个构件$\{C_1,C_2,\ldots,C_n\}$组装调用.本文设定:在这样的构件行为协议定义中,不同服务接口之间的行为互不干扰.构件 C_i 调用 C_j 提供的服务接口 Ite_i 与其组装交互,显然,C_i 只关注 C_j 在接口 Ite_i 上所表现出的服务行为协议是否和自身在该接口上的请求行为协议相容;同样,对 C_j 而言,它也仅关注 C_i 在调用接口 Ite_i 时所表现出的请求行为是否符合 C_j 的行为协议映射在此接口上的规范要求.

设构件 C_i,C_j 通过 C_j 提供的服务接口 *Ite* 组装交互,若 C_j 能满足 C_i 在此接口上的所有请求行为,且 C_i 在此

接口上的请求行为同时符合 C_j 自身接口行为的完整规范,则可认为 C_i 与 C_j 在此接口上协议级行为相容.

定理 **1**. 相连构件 $C_i=\langle I_{C_i}^P, I_{C_i}^R, A_{C_i}, L_{C_i}, P_{C_i}\rangle, C_j=\langle I_{C_j}^P, I_{C_j}^R, A_{C_j}, L_{C_j}, P_{C_j}\rangle$间通过 C_j 的提供接口 *Ite* 组装交互,若下列两条件同时满足,则 C_i,C_j 在接口 *Ite* 上行为相容:

$(P_{C_i} \|_{Ite} P_{C_j})/D1 \approx_B P_{C_i}/D2$,这里,$D1=A_{C_i}\cup A_{C_j}-Ite, D2=A_{C_i}-Ite$;

$trace_{co}(P_{C_i})\uparrow Ite \subseteq trace_{co}(P_{C_j})\uparrow Ite$;

由于受篇幅所限,定理证明从略.对于相连构件 C_i,C_j 通过 C_j 提供的服务接口 *Ite* 组装交互,也可通过它们在 *Ite* 所能表现出的行为迹关系来判断两者是否在该接口上行为相容.

定理 **2**. 相连构件 $C_i=\langle I_{C_i}^P, I_{C_i}^R, A_{C_i}, L_{C_i}, P_{C_i}\rangle, C_j=\langle I_{C_j}^P, I_{C_j}^R, A_{C_j}, L_{C_j}, P_{C_j}\rangle$间通过 C_j 的提供接口 *Ite* 组装交互,对$\forall tr1\in trace_{co}(P_{C_i})\uparrow Ite$,当 C_i 依据 $tr1$ 与 C_j 运行交互时,$\exists tr2\in trace_{co}(P_{C_j})\uparrow Ite$,$C_j$ 依据 $tr2$ 确定运行,且 $tr1=tr2$,则 C_i,C_j 在接口 *Ite* 上行为相容.

由于受篇幅所限,定理证明从略.对于较一般情形,相连构件 $C_i=\langle I_{C_i}^P, I_{C_i}^R, A_{C_i}, L_{C_i}, P_{C_i}\rangle, C_j=\langle I_{C_j}^P, I_{C_j}^R, A_{C_j}, L_{C_j}, P_{C_j}\rangle$,若 C_i 需调用 C_j 提供的多个接口 $Ite_1,Ite_2,\ldots,Ite_n$,则 C_i,C_j 之间的行为相容转化为验证:对于 $P_{C_i} \|_I P_{C_j}$ 的初始状态(s_{Init}^i, s_{Init}^j),是否有$(s_{Init}^i, s_{Init}^j) \overset{\sigma}{\Rightarrow} (s_{Fina}^i, s_{Fina}^j)$,$\sigma\in (A_{C_i}\cup A_{C_j})^* \wedge \sigma\uparrow I\neq\langle\cdot\rangle$,这里,$I=\bigcup_{1\le k\le n} Ite_k$.这样的验证同样可以通过定理 1 和定理 2 来进行.

定理 1 和定理 2 给出了构件之间在组装交互接口上的一种偏序行为相容关系,即在组装过程中,仅一方对另一方有服务请求行为;而在其他一些场景中,构件之间可能互有服务请求行为.设 C_1,C_2 之间通过相互调用对方的提供接口 Ite_1,Ite_2 组装交互,则需使得在各自的提供接口上,双方能同时满足对方的请求行为,且彼此的服务请求行为符合对方行为协议映射在相关接口上的规范;这样使得 C_1,C_2 在接口 Ite_1,Ite_2 同步组装交互最终不会使其中的一方或双方陷入无法继续向下执行的状态.

定义 **9**. 相连构件 $C_i=\langle I_{C_i}^P, I_{C_i}^R, A_{C_i}, L_{C_i}, P_{C_i}\rangle, C_j=\langle I_{C_j}^P, I_{C_j}^R, A_{C_j}, L_{C_j}, P_{C_j}\rangle$,相互间通过调用对方的接口 Ite_1, Ite_2 组装交互$((Ite_1\in I_{C_i}^P \cap I_{C_j}^R)\wedge(Ite_2\in I_{C_i}^R \cap I_{C_j}^P))$,则 C_i,C_j 在接口 Ite_1,Ite_2 上协议级行为相容,当且仅当对于 $P_{C_i} \|_{Ite_1\cup Ite_2} P_{C_j}$ 的初始状态(s_{Init}^i, s_{Init}^j),有$(s_{Init}^i, s_{Init}^j) \overset{\sigma}{\Rightarrow} (s_{Fina}^i, s_{Fina}^j)$,$\sigma\in (A_{C_i}\cup A_{C_j})^* \wedge \sigma\uparrow(Ite_1\cup Ite_2)\neq\langle\cdot\rangle$.

设 C_i,C_j 相互间通过调用对方的接口 Ite_1,Ite_2 组装交互,$Ite_1\in I_{C_i}^P \cap I_{C_j}^R \wedge Ite_2\in I_{C_i}^R \cap I_{C_j}^P$,$Ite_1=\{a_1,a_2,\ldots,a_n\}$, $Ite_2=\{b_1,b_2,\ldots,b_n\}$,则 C_i,C_j 间互有请求场景下的行为不相容情形,除了前面分析的两种情形以外,还有如下的一种情形:两者组装交互时,C_i,C_j 间经过一系列同步操作$\langle c_1,c_2,\ldots,c_n\rangle$($c_i\in Ite_1\cup Ite_2$)后,$C_i$ 提供 a_i,C_j 提供 b_i,双方同时在等待着对方来调用自身提供的方法,从而使得双方均陷入无法继续向下执行的状态.

基于以上分析,若构件 C_i,C_j 在相互调用的接口 Ite_1,Ite_2 上所能展示的任何可能行为均没有能同步一致,则显然 C_i,C_j 在接口 Ite_1,Ite_2 上的同步调用不能顺利实施,即 C_i,C_j 在接口 Ite_1,Ite_2 上行为不相容.现给出此情形下构件间行为相容成立的必要条件:

定理 **3**. 相连构件 $C_i=\langle I_{C_i}^P, I_{C_i}^R, A_{C_i}, L_{C_i}, P_{C_i}\rangle, C_j=\langle I_{C_j}^P, I_{C_j}^R, A_{C_j}, L_{C_j}, P_{C_j}\rangle$,相互间通过调用对方的接口 Ite_1, Ite_2 组装交互$((Ite_1\in I_{C_i}^P \cap I_{C_j}^R)\wedge(Ite_2\in I_{C_i}^R \cap I_{C_j}^P))$,若 $trace_{co}(P_{C_i})\uparrow(Ite_1\cup Ite_2)\cap trace_{co}(P_{C_j})\uparrow(Ite_1\cup Ite_2)=\varnothing$,则 C_i,C_j 在接口 Ite_1,Ite_2 上行为不相容.

现考虑另外一种情形,即若构件 C_i,C_j 在相互调用的接口 Ite_1,Ite_2 上的同步行为能同时满足双方在接口 Ite_1,Ite_2 上的规范要求,则显然 C_i,C_j 在组装交互过程中不会出现上述分析的行为不相容情形.现给出此情形下构件间行为相容的规则:

定理 **4**. 相连构件 $C_i=\langle I_{C_i}^P, I_{C_i}^R, A_{C_i}, L_{C_i}, P_{C_i}\rangle, C_j=\langle I_{C_j}^P, I_{C_j}^R, A_{C_j}, L_{C_j}, P_{C_j}\rangle$,相互间通过调用对方的接口 Ite_1,Ite_2 组装交互$((Ite_1\in I_{C_i}^P \cap I_{C_j}^R)\wedge(Ite_2\in I_{C_i}^R \cap I_{C_j}^P))$,若下列两条件能同时满足,则 C_i,C_j 在接口 Ite_1,Ite_2 上行为相容:

$$(P_{C_i} \|_{Ite_1 \cup Ite_2} P_{C_j})/(A_{C_i} \cup A_{C_j} - (Ite_1 \cup Ite_2)) \approx_B P_{C_i} / (A_{C_i} - (Ite_1 \cup Ite_2));$$

$$(P_{C_i} \|_{Ite_1 \cup Ite_2} P_{C_j})/(A_{C_i} \cup A_{C_j} - (Ite_1 \cup Ite_2)) \approx_B P_{C_j} / (A_{C_j} - (Ite_1 \cup Ite_2)).$$

定理证明从略.定理 4 是一个较为严格的规则,它反映了需要进行组装交互的两构件满足这样的条件:由条件 1、条件 2 即得 $P_{C_i}/(A_{C_i} - (Ite_1 \cup Ite_2)) \approx_B P_{C_j}/(A_{C_j} - (Ite_1 \cup Ite_2))$,即 P_{C_i}, P_{C_j} 及 $P_{C_i} \|_{Ite_1 \cup Ite_2} P_{C_j}$ 三者在接口 Ite_1,Ite_2 上所能展示的行为须是完全一致和匹配的.

3.2 体系结构的行为分析

对多个构件 $\{C_1, C_2, \ldots, C_n\}$ $(C_i = \langle I_{C_i}^P, I_{C_i}^R, A_{C_i}, L_{C_i}, P_{C_i} \rangle)$ 组装成的体系结构,其行为 $P_M \equiv P_{C_1} \|_{a(C_1,C_2)} P_{C_2} \|_{a(C_1,C_3) \cup a(C_2,C_3)} P_{C_3} \| \cdots \|_{a(C_1,C_n) \cup \ldots \cup a(C_{n-1},C_n)} P_{C_n}$,则行为 P_M 无死锁表现为: P_M 中不存在交互接口上的行为不相容情形,即构件间在所有共享接口上的同步行为不会使其中的若干个 P_{C_i} 陷入无法继续向下执行的状态.即对 P_M 的初始状态 $(s_{Init}^1, s_{Init}^2, \ldots, s_{Init}^n)$,有 $(s_{Init}^1, s_{Init}^2, \ldots, s_{Init}^n) \overset{\sigma}{\Rightarrow} (s_{Fina}^1, s_{Fina}^2, \ldots, s_{Fina}^n)$, $\sigma \in \left(\bigcup_{1 \le i \le n} A_{C_i}\right)^* \wedge \sigma \uparrow \left(\bigcup_{1 \le i,j \le n, i \ne j} a(C_i, C_j)\right) \ne \langle \cdot \rangle$.令 $P^i \equiv P_{C_1} \|_{a(C_1,C_2)} P_{C_2} \|_{a(C_1,C_3) \cup a(C_2,C_3)} P_{C_3} \| \cdots \|_{a(C_1,C_{i-1}) \cup \ldots \cup a(C_{i-2},C_{i-1})} P_{C_{i-1}} \|_{a(C_1,C_{i+1}) \cup \ldots \cup a(C_{i-1},C_{i+1})} P_{C_{i+1}} \| \cdots \|_{a(C_1,C_n) \cup \ldots \cup a(C_{n-1},C_n)} P_{C_n}$,则 $P_M \equiv P_{C_i} \|_{I_i} P^i$.这里,$I_i = \bigcup_{1 \le k \le n, k \ne i} a(C_k, C_i)$,则验证 P_M 是否行为无死锁,则可转化为验证 P_{C_i},P^i 在 I_i 上是否行为相容.显然,若同时对所有 P_{C_i} 而言均有 P_{C_i},P^i 在 $I_i (1 \le i \le n)$ 上行为相容,则体系结构行为 P_M 无死锁.在实际验证的过程中并不需要验证所有的构件,而只需验证那些对体系结构中其他构件有请求行为的构件即可.

定理 5. 对多个构件 $\{C_1, C_2, \ldots, C_n\}$ $(C_i = \langle I_{C_i}^P, I_{C_i}^R, A_{C_i}, L_{C_i}, P_{C_i} \rangle)$ 组装成的体系结构,其行为 P_M 为 $P_{C_1} \|_{a(C_1,C_2)} P_{C_2} \|_{a(C_1,C_3) \cup a(C_2,C_3)} P_{C_3} \| \cdots \|_{a(C_1,C_n) \cup \ldots \cup a(C_{n-1},C_n)} P_{C_n}$;对于任意对体系结构中其他构件有请求行为的构件 C_i 而言,若均有 P_{C_i},P^i 在 I_i 上行为相容,则体系结构的行为 P_M 无死锁.

定理证明从略.定理 5 说明了由多个构件组成的体系结构其行为 P_M 若无死锁,则其中任何对其他构件有请求行为的构件需与环境中其他构件组成的整体之间行为相容.但对于构件组装结构中无环形回路的体系结构而言,定理 5 所需条件可以适当放宽.设与其中某构件 C_k 相连的构件集为 $\{C_k^1, C_k^2, \ldots, C_k^m\}$,则对其中任两个构件: C_k^i, C_k^j $(1 \le i,j \le m, i \ne j)$,彼此之间互不直接或间接相连(否则结构将存在环形回路), $e(C_k^i, C_k^j) = \varnothing$,即 $P_{C_k^i} \|_\phi P_{C_k^j}$ 实质为两个构件各自行为的独立运行,从而对 $P_{C_k} \|_{I_k} (P_{C_k^1} \|_\phi \ldots \|_\phi P_{C_k^m})$ 行为相容的验证(这里,$I_k = \bigcup_{1 \le i \le m} a(C_k, C_k^i)$),则转化为验证 P_{C_k} 与 $P_{C_k^1}, \ldots, P_{C_k}$ 与 $P_{C_k^m}$ 之间在不同交互接口上的行为相容.

4 示 例

本文通过一个电子商务的具体例子[16]来说明文中验证规则的特点.在利用网络购书时,有 3 类构件参与其中:网上书店(BookShop)、图书中介(BookBroker)、网络银行(Bank).在运行过程中,BookShop 在 BookBroker 进行注册(register),当有用户需要图书(getABook)时,BookBroker 将查询注册的 BookShop 处是否有现货(inStock),若有,BookBroker 将进行定货(order),并让 BookShop 送货(deliver)给 User,并在 BookShop 的银行帐户上存入(deposit)书金.它们的组装结构、各自的 IDL 提供接口以及构件描述如图 1~图 3 所示.

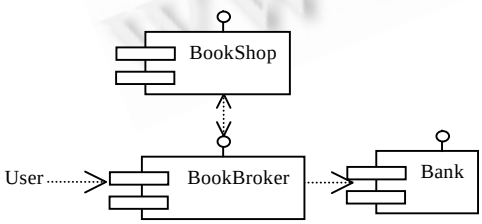

Fig.1 An e-commerce instance

图 1 电子商务的示例

在如图 1 所示的构件组装应用中,构件 Bank,BookShop 都仅与构件 BookBroker 相连,Bank,BookShop 之间不相连,且 Bank 对其他构件没有请求行为,BookBroker 和 BookShop 间互有请求行为.对于构件 BookBroker 而言,只要验证 BookBroker 与构件 Bank、构件 BookBroker 与构件 BookShop 之间是否行为相容即可.构件 BookBroker 与构件 Bank 之间为仅其中一方对另一方有请求行为,即需验证 P_{BB},P_{BA} 在

Bank 的提供接口 $Ite_{BankAccount}$ 上是否行为相容,因此可用定理 1 和定理 2 进行分析;而构件 BookBroker 与构件 BookShop 之间为双方之间相互均有请求行为,即需验证 P_{BB},P_{BS} 同时在 BookShop 的提供接口 Ite_{Book_Shop},BookBroker 的提供接口 Ite_{Broker_Shop} 上是否行为相容,因此需用定理 3 和定理 4 进行分析.

在验证过程中可以发现:$(P_{BB}\|_{Ite_1}P_{BA})/D1\approx_B P_{BB}/D2$,$Ite_1=Ite_{BankAccount}$,且 $trace_{co}(P_{BB})\uparrow Ite_{BankAccount}=\{\langle login,deposit,deposit\_r,logout\rangle\}\subseteq trace_{co}(P_{BA})\uparrow Ite_{BankAccount}$,这里,$D1=(A_{BA}\cup A_{BB})-Ite_{BankAccount}=\{inStock,inStock\_r,order,deliver,deliver\_r,getABook,getABook\_r,register,unregister\}$,$D2=A_{BB}-Ite_{BankAccount}=\{inStock,inStock\_r,order,deliver,deliver\_r,getABook,getABook\_r,register,unregister\}$,故构件 BookBroker 与 Bank 在后者的提供接口 $Ite_{BankAccount}$ 上行为相容.验证构件 BookBroker 与 BookShop 之间的协议级行为相容,则有:(1) $(P_{BB}\|_{Ite_2\cup Ite_3}P_{BS})/D3\approx_B P_{BB}/D4$,$Ite_2=Ite_{Broker\_Shop}$,$Ite_3=Ite_{Book\_Shop}$,这里,$D3=(A_{BS}\cup A_{BB})-Ite_{Book\_Shop}-Ite_{Broker\_Shop}=\{login,deposit,deposit\_r,logout,getABook,getABook\_r\}$,$D4=A_{BB}-Ite_{Book\_Shop}-Ite_{Broker\_Shop}=\{login,deposit,deposit\_r,logout,getABook,getABook\_r\}$;(2) $(P_{BB}\|_{Ite_2\cup Ite_3}P_{BS})/D3\approx_B P_{BS}/D5$.这里,$D5=A_{BS}-Ite_{Book\_Shop}-Ite_{Broker\_Shop}=\varnothing$.故 BookBroker 与 BookShop 同时在 BookShop 的提供接口 Ite_{Book_Shop},BookBroker 的提供接口 Ite_{Broker_Shop} 上行为相容.对于构件 BookShop 而言,需验证 P_{BS},$P_{BB}\|_{Ite_1}P_{BA}$ 同时在 BookShop 的提供接口 Ite_{Book_Shop},BookBroker 的提供接口 Ite_{Broker_Shop} 上是否行为相容,显然有:(1) $(P_{BS}\|_{Ite_2\cup Ite_3}(P_{BB}\|_{Ite_1}P_{BA}))/D6\approx_B(P_{BB}\|_{Ite_1}P_{BA})/D7$.这里,$Ite_1=Ite_{BankAccount}$,$Ite_2=Ite_{Broker\_Shop}$,$Ite_3=Ite_{Book\_Shop}$,$D6=(A_{BS}\cup A_{BB}\cup A_{BA})-Ite_{Book\_Shop}-Ite_{Broker\_Shop}$,$D7=(A_{BA}\cup A_{BB})-Ite_{Book\_Shop}-Ite_{Broker\_Shop}$;(2) $(P_{BS}\|_{Ite_2\cup Ite_3}(P_{BB}\|_{Ite_1}P_{BA}))/D6\approx_B P_{BS}/D8$,$D8=A_{BS}-Ite_{Book\_Shop}-Ite_{Broker\_Shop}$.故在这样的应用中行为无死锁.

An interface provided by Bank : *BankAccount*:

```
interface BankAccount{
  void login(in string accountNO);
  float getBalance();
  string deposit(in float amount);
  string withdraw(in float amount);
  void logout();
};
```

An interface provided by BookBroker: *Broker_User*:

```
interface Broker_User{
  boolean getABook(
      in string author,
      in string title,
      in float maxprice,
      in string addr,
      out date when);
};
```

An interface provided by BookShop: *Book_Shop*:

```
interface Book_Shop{
  struct BookRef {string ISBN; float price;}
   boolean inStock(in string title, in string author);
   void order(in BookRef b, out account a,
              out string purchaseID);
   date deliver(in string purchaseID,
              in string rcpt,
              in string addr};
};
```

An interface provided by BookBroker: *Broker_Shop*:

```
interface Broker_Shop{
  void register (in Bookshop b);
  void unregister (in Bookshop b);
  };
```

Fig.2 IDL interfaces provided by components Bank, BookShop and BookBroker

图 2 构件 Bank,BookShop,BookBroker 各自提供的 IDL 接口

5 相关工作比较与小结

目前,在构件的行为协议研究领域较为重要的工作有:基于化学抽象机模型(chemical abstract machine),文献[3]定义了构件对环境的需求,给出了构件能正常对外交互所需的必要条件,但未对保证构件之间交互成功的行为相容性进行探讨;基于 CSP[17]的 SOFA[18]则给出了软件系统中构件协议级可替换的必要条件,包含两构件组装的可替换和构件框架中的构件可替换,但未对如何保证构件间行为相容及整个体系结构行为无死锁进行分析与研究;Darwin[13]给出了基于π演算[19]描述构件交互协议的方式,但没有给出验证构件间协议级行为相容

的方法;LEDA[7]研究两个构件之间协议级的行为相容关系和行为可继承的关系,但没有涉及如何验证多个构件组装而成的体系结构行为无死锁;基于进程代数 PA,文献[4,12]对上述两方面进行了一定的阐述,由于它并不是专门针对采用面向对象范型的体系结构的,故构件间的交互没有接口的概念,也没有区分进行组装的两构件中服务请求方与服务提供方在连接交互接口上可实施的不同行为,而要求交互双方的行为完全等价,从而使得相关结论在采用面向对象范型的体系结构中使用时受到了一定的限制;文献[16]则在传统 CORBA 的 IDL 接口描述语言中扩充了基于π演算的协议描述机制,使得验证构件间协议级行为相容成为可能,但它没有给出相关的验证规则以保证构件交互的成功.

$BookShop=\langle I^P_{BS}, I^R_{BS}, A_{BS}, L_{BS}, P_{BS}\rangle$

I^P_{BS} :{Ite_{Book_Shop}};Ite_{Book_Shop}={*inStock, inStock_r, order, deliver, deliver_r*}

I^R_{BS} :{Ite_{Broker_Shop}};Ite_{Broker_Shop}={*register, unregister*}

A_{BS}:($A^R_{BS}, A^P_{BS}, A^H_{BS}$);

A^R_{BS} :{*register, unregister*}; A^H_{BS} :ϕ;

A^P_{BS} :{*inStock, inStock_r, order, deliver, deliver_r*};

L_{BS} : {$\langle Ite_{Broker\_Shop}, Ite_{Broker\_Shop}$, *BookBroker_Inst, BookBroker_Host*$\rangle$};

P_{BS}:{$P_{BS} \triangleq P[Shop]_{Init}$

$P[Shop]_{Init} \triangleq register.P[Shop]_1$

$P[Shop]_1 \triangleq inStock.inStock\_r.P[Shop]_2$

$P[Shop]_2 \triangleq order.deliver.deliver\_r.P[Shop]_1 + unregister.P[Shop]_{Fina}$

$P[Shop]_{Fina} \triangleq \underline{0}$};

$BookBroker=\langle I^P_{BB}, I^R_{BB}, A_{BB}, L_{BB}, P_{BB}\rangle$

I^P_{BB} :{Ite_{Broker_User}, Ite_{Broker_Shop}};Ite_{Broker_User}={*getABook, getABook_r*};Ite_{Broker_Shop}={*register, unregister*};

I^R_{BB} :{Ite_{Book_Shop}, $Ite_{BankAccount}$};Ite_{Book_Shop}={*inStock, inStock_r, order, deliver, deliver_r*};

$Ite_{BankAccount}$={*login, getBalance, getBalance_r, deposit, deposit_r, withdraw, withdraw_r,logout*}

A_{BB}:($A^R_{BB}, A^P_{BB}, A^H_{BB}$);

A^R_{BB} :{*inStock, inStock_r, order, login, deposit,deposit_r, logout, deliver, deliver_r*}; A^H_{BB} :ϕ;

A^P_{BB} :{*getABook, getABook_r, register, unregister*};

L_{BB}:{$\langle Ite_{Book\_Shop}$, Ite_{Book_Shop}, *BookShop_Inst, BookShop_Host*$\rangle$,$\langle Ite_{BankAccount}$, $Ite_{BankAccount}$, *Bank_Inst, Bank_Host*$\rangle$};

P_{BB}:{$P_{BB} \triangleq P[Broker]_{Init}$

$P[Broker]_{Init} \triangleq register.P[Broker]_1$

$P[Broker]_1 \triangleq getABook.P[Broker]_2$

$P[Broker]_2 \triangleq inStock.inStock\_r.getABook\_r.P[Broker]_3$

$P[Broker]_3 \triangleq order.P[Broker]_4 + unregister.P[Broker]_{Fina}$

$P[Broker]_4 \triangleq login.deposit.deposit\_r.logout.deliver.deliver\_r.P[Broker]_1$

$P[Broker]_{Fina} \triangleq \underline{0}$};

$Bank=\langle I^P_{BA}, I^R_{BA}, A_{BA}, L_{BA}, P_{BA}\rangle$

I^P_{BA} :{$Ite_{BankAccount}$}; $Ite_{BankAccount}$={*login, getBalance, getBalance_r, deposit, deposit_r, withdraw, withdraw_r, logout*};

I^R_{BA} :ϕ;

A_{BA}:($A^R_{BA}, A^P_{BA}, A^H_{BA}$);

A^R_{BA} :ϕ; A^H_{BA} :ϕ

A^P_{BA} :{*login, getBalance, getBalance_r, deposit, deposit_r, withdraw, withdraw_r, logout*};

L_{BA}:ϕ;

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

$P[Bank]_{Init} \triangleq login.P[Bank]_1$

$P[Bank]_1 \triangleq getBalance.getBalance\_r.P[Bank]_1 + deposit.deposit\_r.P[Bank]_1 + withdraw.withdraw\_r.P[Bank]_1 + logout.P[Bank]_{Fina.}$

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

Fig.3 Specifications of components Bank, BookShop and BookBroker

图 3 构件 Bank, BookShop, BookBroker 的描述

传统机制对构件间的组装交互一般关注于接口的语法级匹配,而对行为协议级方面则没有过多涉及.20 世纪 90 年代中期以来,对软件体系结构中组装构件间协议级行为相容的研究逐步成为研究者关注的热点.本文针对基于面向对象范型的软件体系结构展开工作,形式化地描述与定义了构件及其对外交互协议,分析了服务请

求构件与服务提供构件在交互接口上所能展示的不同行为,给出了连接交互的两构件在请求/提供接口上协议级行为相容的一组验证规则及相关定理,并通过相关实例说明所提出规则的可用性和针对性.今后的工作将围绕软件体系结构研究的一些新领域展开[20,21],如对动态体系结构性能的描述与验证,构件间组装调用的 Qos、构件组装连接类型等语义方面的考虑,以及在现有相关支持工具[22]基础上研制与实现适合本文所需的构件间协议级行为相容验证的原型工具,其内容包括数据结构、用户界面、功能模块等方面,并将其集成到我们的构件组装框架与支撑系统[14,15]中,这些都将有待进一步的努力.

References:

[1] Heineman GT, Councill WT. Component-Based Software Engineering. Boston: Addison-Wesley, 2001.

[2] Shaw M, Garlan D. Software Architecture: Perspectives on an Emerging Discipline. New Jersey: Prentice Hall, 1996.

[3] Inverardi P, Wolf AL. Formal specification and analysis of software architectures using the chemical abstract machine model. IEEE Trans. on Software Engineering, 1995,21(4):373−386.

[4] Bernardo M, Ciancarini P, Donatiello L. Architecting families of software systems with process algebras. ACM Trans. on Software Engineering and Methodology, 2002,11(4):386−426.

[5] Aalst WMP, Hee KM, Toorn RA. Component-Based software architectures: A framework based on inheritance of behavior. Science of Computer Programming, 2002,42:129−171.

[6] Mei H, Chen F, Feng YD, Yang J. ABC: An architecture based, component oriented approach to software development. Journal of Software, 2003,14(4):721−732 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/14/721.htm

[7] Canal C, Pimentel E, Troya JM. Compatibility and inheritance in software architectures. Science of Computer Programming, 2001,41:105−138.

[8] Mei H, Chang JC, Yang FQ. Software component composition based on ADL and middleware. Science in China (Series F), 2001,44(2):136−151.

[9] Szyperski C, Gruntz D, Murer S. Component Software: Beyond Object-Oriented Programming. 2nd. Massachusetts: Addison-Wesley Professional, 2002.

[10] Vallecillo A, Hernandez J, Troya JM. New issues in object interoperability. In: Proc. of the ECOOP 2000 Workshop on Object Interoperability. LNCS 1964, France, 2000. 256−269.

[11] Allen R, Garlan D. A formal basis for architectural connection. ACM Trans. on Software Engineering and Methodology, 1997, 6(3):213−249.

[12] Aldini A, Bernardo M. On the usability of process algebra: An architectural view. Theoretical Computer Science, 2005, 335:281−329.

[13] Magee J, Kramer J. Dynamic structure in software architectures. ACM SIGSOFT Software Engineering Notes, 1996,21(6):3−14.

[14] Lu J, Tao XP, Ma XY, Hu H, Xu F, Cao C. On agent-based software model for internetware. Science in China (Series E), 2005, 35(12):1233−1253 (in Chinese with English abstract).

[15] Hu HY, Yang M, Tao XP, Lu J. Research and implementation of late assembly technology in cogent. Acta Electronica Sinca, 2002,30(12):1823−1827 (in Chinese with English abstract).

[16] Canal C, Fuentes L, Troya JM, Vallecillo A. Extending CORBA interfaces with π-calculus for protocol compatibility. In: Mitchell R, ed. Proc. of the Technology of Object-Oriented Languages and Systems. Washington: IEEE Computer Society. 2000. 208−225.

[17] Brookes SD, Hoare CAR, Roscoe AW. A theory of communicating sequential processes. Journal of the ACM, 1984,31(3):560−599.

[18] Plasil F, Visnovsky S. Behavior protocols for software components. IEEE Trans. on Software Engineering, 2002,28(11): 1056−1076.

[19] Milner R. Communicating and Mobile Systems: The π-Calculus. Cambridge: Cambridge University Press, 1999.

[20] Dashofy EM, Hoek AVD, Taylor RN. A comprehensive approach for the development of modular software architecture description languages. ACM Trans. on Software Engineering and Methodology, 2005,14(2):199−245.

[21] Mei H, Cao DG. ABC-S^2C: Enabling separation of crosscutting concerns in component-based software development. Chinese Journal of Computers, 2005,28(12):2036−2044 (in Chinese with English abstract).

[22] Aldini A, Bernardo M. TwoTowers 4.0: Towards the integration of security analysis and performance evaluation. In: Haverkort B, ed. Proc. of the 1st Int'l Conf. on the Quantitative Evaluation of Systems.Washington: IEEE Computer Society, 2004. 336−337.

附中文参考文献:

[6] 梅宏,陈峰,冯耀东,杨杰.ABC:基于体系结构、面向构件的软件开发方法.软件学报,2003,14(4):721−732. http://www.jos.org.cn/1000-9825/14/721.htm

[14] 吕建,陶先平,马晓星,胡昊,徐峰,曹春.基于 Agent 的网构软件模型研究.中国科学 E 辑(信息科学),2005,35(12):1233−1253.

[15] 胡海洋,杨玫,陶先平,吕建.Cogent 后组装技术研究与实现.电子学报,2002,30(12):1823−1827.

[21] 梅宏,曹东刚.ABC-S^{2}C:一种面向贯穿特性的构件化软件关注点分离技术.计算机学报,2005,28(12):2036−2044.

胡海洋(1977－),男,江苏宝应人,博士生,主要研究领域为构件,中间件技术.

马晓星(1975－),男,博士,副教授,主要研究领域为软件体系结构,Internet 软件技术.

吕建(1960－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件形式化与自动化,软件 Agent 技术及其应用.

陶先平(1970－),男,博士,教授, CCF 高级会员,主要研究领域为移动 Agent,软件协同技术.