ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.17, No.6, June 2006, pp.1459−1469
DOI: 10.1360/jos171459

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

面向服务软件体系结构的接口模型*

陈振邦, 王 戟+, 董 威, 齐治昌

(国防科学技术大学 计算机学院,湖南 长沙 410073)

An Interface Model for Service-Oriented Software Architecture

CHEN Zhen-Bang, WANG Ji+, DONG Wei, QI Zhi-Chang

(School of Computer, National University of Defense Technology, Changsha 410073, China)

+ Corresponding author: Phn: +86-731-4573637, Fax: +86-731-4575802, E-mail: jiwang@nudt.edu.cn, http://www.nudt.edu.cn

Chen ZB, Wang J, Dong W, Qi ZC. An interface model for service-oriented software architecture. *Journal of Software*, 2006,17(6):1459−1469. http://www.jos.org.cn/1000-9825/17/1459.htm

Abstract: Service interface description is one of the most important problems in service-oriented software architecture, and formal description methods are considered to be a promising approach for service interfaces. However, the existing interface models do not support transaction information description. By extending the model of Web Service Interface, this paper presents an interface model that can capture transaction information by means of fault handling and compensation. In this model, service interfaces can be specified at the signature, conversation and protocol levels. At each level, the compatibility and substitutivity are described formally. Furthermore, the interface specifications and corresponding verification methods are also presented at the conversation and protocol levels. The model supports precise transaction information description as well as flexible and effective verification for interface specifications.

Key words: service-oriented software architecture; interface; transaction; compensation; compatibility; substitutivity; interface specification

摘 要: 服务接口描述是面向服务软件体系结构描述的关键问题之一,形式化描述方法是保证服务接口描述准确性的重要手段.目前的接口模型并不支持事务信息的描述.通过对已有 Web 服务接口模型进行扩展,提出了通过错误处理和补偿支持事务信息描述的接口模型,能够分别在特征层、会话层和协议层 3 个层面对服务的接口进行描述,然后提出了服务接口在 3 个层面的相容性和可替换性条件,并针对会话接口和协议接口提出了接口规约及其验证方法.该接口模型不仅能够支持准确的事务信息描述,而且还支持灵活、有效的接口规约验证.

关键词: 面向服务软件体系结构;接口;事务;补偿;相容性;可替换性;接口规约

中图法分类号: TP311 文献标识码: A

* Supported by the National Natural Science Foundation of China under Grant Nos.60233020, 60303013, 90612009 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.2005AA113130 (国家高技术研究发展计划(863)); the National Grand Fundamental Research 973 Program of China under Grant No.2005CB321802 (国家重点基础研究发展规划(973)); the Program for New Century Excellent Talents in University under Grant No.NCET-04-0996 (新世纪优秀人才支持计划)

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

面向服务是目前软件工程技术发展的重要趋势[1,2].如何保证面向服务软件系统的高可信性成为一个新的挑战.

在面向服务的软件体系结构中,服务提供者向服务注册机构发布服务的接口信息.该信息描述了这个服务对于外界环境的要求和它能够对外界提供的保证.如何保证接口信息的准确性,对于面向服务系统的构建非常关键.形式化的接口描述是保证其准确性的有效手段,也是对其进行测试或验证的前提.因此,建立服务的形式化接口模型受到了人们的关注.面向构件,文献[3]提出了接口自动机描述接口之间的时序行为;文献[4]提出了时间接口,对具备时间约束的接口时序行为进行描述.以此为基础,文献[5,6]中提出了一种服务接口模型,在 3 个抽象层面描述接口信息,并在每个层面给出了接口的相容性和可替换性条件,提出了接口规范及其验证方法.然而,目前的服务接口模型在事务信息的描述上还存在着不足[3−6],并不支持接口层面的事务模型描述.事务性是面向服务系统的一个非常重要的特点,面向服务的事务模型与传统的原子事务模型有着很大的区别[7],前者往往在执行时间上比较长,而且很多操作不具备传统事务模型的原子性和可回滚性,对于长事务(long-running transactions)模型,需要提供补偿和错误处理机制对其发生的异常进行处理.

针对接口层面的事务信息,本文对文献[5]中的模型进行扩展,提出了具备事务描述能力的接口模型,它能够在特征、会话和协议 3 个层面上描述事务信息;然后,给出了具备事务特征的接口的相容性和可替换性条件,并专门针对会话和协议层的接口提出了新的接口规约及其验证方法.

本文第 1 节针对服务提出支持事务信息的服务接口模型,在特征、会话及协议 3 个层面描述服务接口,并给出相应的相容性和可替换性条件.第 2 节提出会话和协议层的接口规约及其验证方法.第 3 节给出与相关工作的比较.第 4 节对全文进行总结.

1 面向服务的接口模型

方法调用是面向服务系统中的基本动作,在接口描述中,每个服务可以支持或引发多个方法调用,因此可以用方法及其返回结果来代表服务中的基本动作,用动作和动作之间的关系来刻画服务接口之间的使用关系.根据接口描述中的事务信息,可以对动作的成功与否进行区分.如果一个动作会导致异常,那么在异常发生以后,需要引发接口描述中相应的错误处理动作,并且要对在异常发生之前这个动作引发的成功动作进行补偿,为了满足长事务模型的基本特点,补偿动作的引发顺序应该与需要补偿的动作的引发顺序相反.

图 1 是一个具备事务特征的供应链管理系统的基本结构图,其中包含了 6 个服务:商店(shop)、库存(store)、银行(bank)、运输(transport)、供应商(supplier)和邮局(post-office).客户能够调用商店提供的 SellItem 方法购买商品,同时可以调用它的补偿方法 Recede 退还商品.当客户购买商品时,商店首先检查库存是否充足:如果库存不足,则此次购买失败;如果库存充足,则从库存量中减去购买量.在库存检查完成后,如果库存不足或剩余库存量低于某个固定值,库存部门就会向供应商订购商品;如果库存充足,则商店可以调用银行提供的 ProcPay 方法使客户付款,此方法的补偿方法是 Compensate,可以退还客户的付款;如果付款成功,则商店可以调用运输服务提供的 ShipItem 方法把商品运送给客户,此方法的补偿方法是 WithDraw,可以把客户的商品退还给商店;如果以上几个步骤全部成功,则用户购买成功.如果购买失败,失败之前成功的步骤都需要进行补偿,对于失败的步骤需要进行错误处理,商店在购买失败以后会调用 Apologize 方法向客户道歉.此方法要调用邮局提供的 SendLetter 方法给客户寄去一封道歉信.

我们用 M 代表一个有穷的方法集合,O 代表一个有穷的结果集合,则上述系统的基本格局可以表示如下:

M={SellItem,Recede,ChkAvail,Apologize,SendLetter,ChkStore,RecoverStore,ProcPay,Compensate,ShipItem, WithDraw,GetOffer,Order},O={SOLD,NOTFOUND,OK,FAIL,REC}.

$\langle$SellItem,SOLD$\rangle\in M\times O$ 代表一个 SellItem 方法被调用所产生的动作,这个动作的含义是用户购买商品成功.

针对方法调用的因果信息、会话信息和时序信息,可以在 3 个抽象层次对接口描述进行建模:首先在特征层可以描述方法调用之间的直接因果关系;然后在会话层可以描述一个方法能引发的不同方法调用;最后在协

议层可以描述方法调用之间的时序关系.

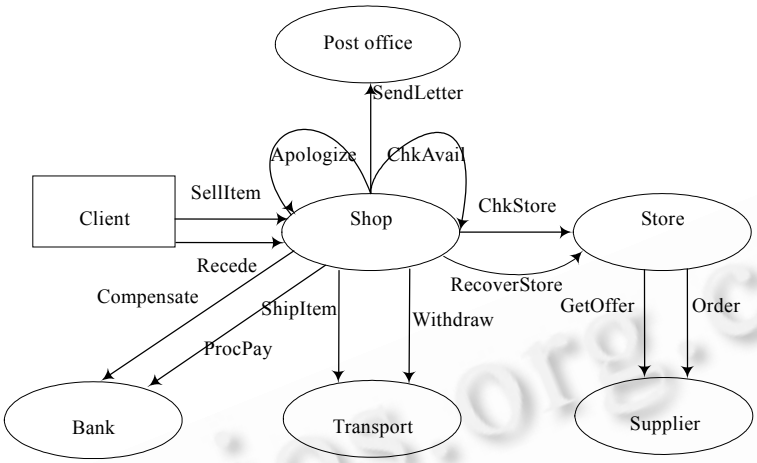

Fig.1 Supply chain management application with transactions

图 1 具备事务特征的供应链管理系统

1.1 特征接口

定义 **1**(特征接口,**signature interface**,简称 **SI**). 特征接口 P 是一个四元组(A,S,S_C,S_F),其中:

- $A\subseteq M\times O$ 是接口中可能出现的动作集;
- S: $A\to 2^A$ 是一个部分函数,给出由一个动作能够引发的动作集合;
- S_C: $A\to 2^A$ 是一个部分函数,给出对一个成功动作进行补偿能够引发的动作集合;
- S_F: $A\to 2^A$ 是一个部分函数,给出对一个异常动作进行错误处理能够引发的动作集合;
- $dom(S_C)\cup dom(S_F)=dom(S)$,并且 $dom(S_C)\cap dom(S_F)=\varnothing$.

下面给出特征接口中的不同动作类型.如果动作 $a\in dom(S)$,则 a 是一个本地动作,说明 a 是由接口中定义的方法被调用所产生的;如果方法 $m\in M$ 是某个本地动作的方法,则 m 是一个本地方法;如果动作 $a\in dom(S_C)$,则 a 是一个成功动作;如果动作 $a\in dom(S_F)$,则 a 是一个异常动作;如果动作 a'能够被某个本地动作引发,或被本地动作的补偿和错误处理所引发,则 a'是一个必需动作,可以用如下公式来表示:

$$required(a')=(\exists a\in dom(S).a'\in S(a))\vee(\exists b\in dom(S_C).a'\in S_C(b))\vee(\exists c\in dom(S_F).a'\in S_F(c)).$$

在面向服务的软件体系结构中,服务注册机构或服务请求者总希望能够获得尽量完整的服务接口描述.如果在服务的接口描述中,某个本地方法产生的动作是必需动作,但这个动作并没有在服务接口中出现,我们就认为这个接口描述是不完整的,例如,供应链管理系统的商店服务中没有库存检查成功动作.我们用良定义来保证接口描述的完整性.如果特征接口 P 满足以下两个条件,则称 P 是良定义的:1) 所有本地方法产生的必需动作都是本地动作;2) 补偿或错误处理不能引发异常动作.可以用如下公式来表示两个条件:

$$\forall a\in A,\exists b\in A.a=(m,o_1)\wedge required(a)\wedge b\in dom(S)\wedge b=(m,o_2)\Rightarrow a\in dom(S),$$

$$\forall a,b\in dom(S_F),c\in dom(S_C).a\notin S_F(b)\wedge a\notin S_C(c).$$

本文只讨论良定义的特征接口.在本地动作 a 引发的正常动作完成以后,如果 a 是异常动作,则 a 所引发的成功动作需要被补偿,异常动作和 a 本身需要进行错误处理.

例 1:良定义的特征接口.我们使用特征接口对供应链管理系统中的商店(shop)服务的接口进行描述,可以得到如下的特征接口:$P_{Shop}=(A_{Shop},S_{Shop},S_{C_{Shop}},S_{F_{Shop}})$,

$A_{Shop}=\{\langle$SellItem,SOLD$\rangle,\langle$SellItem,FAIL$\rangle,\langle$SellItem,NOTFOUND$\rangle,\langle$ChkAvail,OK$\rangle,\langle$ChkAvail,FAIL$\rangle,$
$\langle$Apologize,OK$\rangle,\langle$ChkStore,OK$\rangle,\langle$ChkStore,FAIL$\rangle,\langle$RecoverStore,OK$\rangle,\langle$ProcPay,OK$\rangle,\langle$ProcPay,FAIL$\rangle,$
$\langle$Compensate,OK$\rangle,\langle$Recede,OK$\rangle,\langle$ShipItem,OK$\rangle,\langle$ShipItem,FAIL$\rangle,\langle$WithDraw,OK$\rangle,\langle$SendLetter,OK$\rangle\}$;

S_{Shop}={⟨SellItem,SOLD⟩→{⟨ChkAvail,OK⟩,⟨ProcPay,OK⟩,⟨ShipItem,OK⟩},
⟨SellItem,FAIL⟩→{⟨ChkAvail,FAIL⟩,⟨ChkAvail,OK⟩,⟨ProcPay,FAIL⟩,⟨ProcPay,OK⟩,⟨ShipItem,FAIL⟩},
⟨ChkAvail,OK⟩→{⟨ChkStore,OK⟩},⟨ChkAvail,FAIL⟩→{⟨ChkStore,FAIL⟩},
⟨Recede,OK⟩→{⟨Compensate,OK⟩,⟨WithDraw,OK⟩,⟨RecoverStore,OK⟩},
⟨Apologize,OK⟩→{⟨SendLetter,OK⟩}};

$S_{C_{Shop}}$={⟨SellItem,SOLD⟩→{⟨Recede,OK⟩},⟨ChkAvail,OK⟩→{},⟨Apologize,OK⟩→{},⟨Recede,OK⟩→{}};

$S_{F_{Shop}}$={⟨SellItem,FAIL⟩→{⟨Apologize,OK⟩},⟨ChkAvail,FAIL⟩→{}}.

其中:动作⟨SellItem,SOLD⟩是一个成功动作;⟨SellItem,FAIL⟩是一个异常动作,它能够直接或间接引发的所有动作的集合是{⟨SellItem,FAIL⟩,⟨ChkAvail,FAIL⟩,⟨ChkStore,FAIL⟩,⟨ChkAvail,OK⟩,⟨ChkStore,OK⟩,⟨ProcPay,FAIL⟩,⟨ProcPay,OK⟩,⟨ShipItem,FAIL⟩,⟨Apologize,OK⟩,⟨SendLetter,OK⟩}.P_{Shop}是良定义的.

1.2 会话接口

特征接口用于描述动作之间的引发关系.但是,某些动作在不同情况下可能会引发不同的动作集合,比如在供应链管理系统中,购买失败可能是由库存不足造成的,也可能是由支付失败造成的,不同的原因代表不同的动作集.特征接口并不能描述这种情况,为此定义了会话接口.一个会话是一组动作的集合.我们用命题公式来表示不同的会话.

定义 **2**(会话表达式). 一个动作集 A 上的会话表达式定义为

$$\omega :: \mathrm{T} \mid a \mid \omega_1 \sqcup \omega_2 \mid \omega_1 \sqcap \omega_2,$$

其中:T 是命题常元,代表没有动作需要引发;$a \in A$ 表示动作 a 需要被引发;$\omega_1 \sqcup \omega_2$ 表示 ω_1 和 ω_2 所代表的每个会话都可能被引发;$\omega_1 \sqcap \omega_2$ 表示每个会话必须包含 ω_1 和 ω_2 所代表的各一个会话.我们用 $\omega(A)$ 表示动作集合 A 上所有的会话表达式.

定义 **3**(会话接口,**conversation interface**,简称 **CI**). 会话接口 I 是一个四元组(A,E,E_C,E_F),其中

- $A \subseteq M \times O$ 是接口中可能出现的动作集;
- E: $A \to \omega(A)$是一个部分函数,给出一个动作能够引发的所有会话的会话表达式;
- E_C: $A \to \omega(A)$是一个部分函数,给出对一个成功动作进行补偿能够引发的所有会话的会话表达式;
- E_F: $A \to \omega(A)$是一个部分函数,给出对一个异常动作进行错误处理能够引发的所有会话的会话表达式;
- $dom(E_C) \cup dom(E_F) = dom(E)$,并且 $dom(E_C) \cap dom(E_F) = \varnothing$.

一个会话接口 $I=(A,E,E_C,E_F)$相应的特征接口($csi(I)$)可以计算为:$csi(I)=(A_s,S,S_C,S_F)$,$A_s=A$;如果 $a \in dom(E)$,则 $S(a)=\{a'|a'$在 $E(a)$中出现$\}$;如果 $a \in dom(E_C)$,则 $S_C(a)=\{a'|a'$在 $E_C(a)$中出现$\}$;如果 $a \in dom(E_F)$,则 $S_F(a)=\{a'|a'$在 $E_F(a)$中出现$\}$;否则 $S(a)$,$S_C(a)$和 $S_F(a)$都无定义.如果 $csi(I)$是良定义的,则称 I 是良定义的.会话接口 I 中动作 a 的类型与 a 在 $csi(I)$中的类型一致.本文只讨论良定义的会话接口.同一个会话表达式在不同详细程度的接口描述中可能代表不同的会话,所以会话表达式与接口描述相关.在给定的接口描述下,会话表达式代表的会话可以用会话集来表示.

定义 **4**(会话集). 给定一个会话接口 $I=(A,E,E_C,E_F)$,I 下的会话表达式对应的会话集可以用函数 $[\omega]_R$:$\omega(A) \to 2^{2^A}$ 计算,$[\omega]_R$ 可递归定义如下($[\omega]$:$\omega(A) \to 2^{2^A}$ 是正常会话集函数):

$$[\omega]_R=[\omega],\text{如果}(\omega=\mathrm{T}) \vee (\omega=a \wedge a \notin dom(E_F)),$$

$$[a]_R=\{\{a\} \cup s_c \cup s_f \mid s_c \in \phi(s),\ s \in [E(a)],\ s_f \in [E_F(a)]\},\text{如果 } a \in dom(E_F),$$

$$[\omega_1 \sqcup \omega_2]_R=[\omega_1]_R \cup [\omega_2]_R,\ [\omega_1 \sqcap \omega_2]_R=\{s_1 \cup s_2 \mid s_1 \in [\omega_1]_R \wedge s_2 \in [\omega_2]_R\}.$$

其中 $\phi(s)=\left\{ s \cup s_1 \cup s_2 \middle| s_1 \in \left\{ \bigcup S_a \in [E_C(a)] \right\}_{a \in s \wedge a \in dom(E_C)}, s_2 \in \left\{ \bigcup S_b \in [E_F(b)] \right\}_{b \in s \wedge b \in dom(E_F)} \right\}$,

$[\mathrm{T}]=\{\}$,

$[a]=\{\{a\}\}$,如果 $a\notin dom(E)$,

$[a]=\{\{a\}\cup s|s\in[E(a)]\}$,如果 $a\in dom(E)$,

$[\omega_1\sqcup\omega_2]=[\omega_1]\cup[\omega_2]$, $[\omega_1\sqcap\omega_2]=\{s_1\cup s_2|s_1\in[\omega_1]\wedge s_2\in[\omega_2]\}$.

如果一个动作不是本地动作,说明这个动作需要环境支持,或者说是一个还没有实现的动作,我们认为动作会马上返回,所以这个动作对应的会话集中只有一个会话,其中只包含这个动作本身;如果这个动作是一个成功动作,则会话集中的会话除了包含它自身之外,还包括它能够引发的会话集中的会话;如果是一个异常动作,则会话集中的会话除了包含自身和正常会话集中的会话以外,还应该包含补偿和错误处理会话集中的会话.

例 2:良定义的会话接口.在例 1 的基础上,我们使用会话接口对供应链管理系统中商店(shop)服务的接口进行描述,可以得到如下的会话接口 $I_{Shop}=(A_{Shop},E_{Shop},E_{C_{Shop}},E_{F_{Shop}})$($A_{Shop}$ 与例 1 相同).

$E_{Shop}=\{\langle\mathrm{SellItem,SOLD}\rangle\rightarrow\langle\mathrm{ChkAvail,OK}\rangle\sqcap\langle\mathrm{ProcPay,OK}\rangle\sqcap\langle\mathrm{ShipItem,OK}\rangle,$

$\langle\mathrm{SellItem,FAIL}\rangle\rightarrow\langle\mathrm{ChkAvail,FAIL}\rangle\sqcup(\langle\mathrm{ChkAvail,OK}\rangle\sqcap(\langle\mathrm{ProcPay,FAIL}\rangle\sqcup$

$(\langle\mathrm{ProcPay,OK}\rangle\sqcap\langle\mathrm{ShipItem,FAIL}\rangle))),$

$\langle\mathrm{ChkAvail,OK}\rangle\rightarrow\langle\mathrm{ChkStore,OK}\rangle,\langle\mathrm{ChkAvail,FAIL}\rangle\rightarrow\langle\mathrm{ChkStore,FAIL}\rangle,$

$\langle\mathrm{Apologize,OK}\rangle\rightarrow\langle\mathrm{SendLetter,OK}\rangle,$

$\langle\mathrm{Recede,OK}\rangle\rightarrow\langle\mathrm{Compensate,OK}\rangle\sqcap\langle\mathrm{WithDraw,OK}\rangle\sqcap\langle\mathrm{RecoverStore,OK}\rangle\};$

$E_{C_{Shop}}=\{\langle\mathrm{SellItem,SOLD}\rangle\rightarrow\langle\mathrm{Recede,OK}\rangle,\langle\mathrm{ChkAvail,OK}\rangle\rightarrow\mathrm{T},\langle\mathrm{Apologize,OK}\rangle\rightarrow\mathrm{T},\langle\mathrm{Recede,OK}\rangle\rightarrow\mathrm{T}\};$

$E_{F_{Shop}}=\{\langle\mathrm{SellItem,FAIL}\rangle\rightarrow\langle\mathrm{Apologize,OK}\rangle,\langle\mathrm{ChkAvail,FAIL}\rangle\rightarrow\mathrm{T}\}.$

⟨SellItem,SOLD⟩在例 1 中能够引发的 3 个动作必须出现在它能够引发的每个会话中.⟨SellItem,FAIL⟩必须在正常会话完成以后引发补偿和错误处理会话,例如:如果⟨ChkAvail,OK⟩和⟨ProcPay,FAIL⟩发生,则这个会话是 {⟨SellItem,FAIL⟩,⟨ChkAvail,OK⟩,⟨ChkStore,OK⟩,⟨ProcPay,FAIL⟩,⟨Apologize,OK⟩,⟨SendLetter,OK⟩}.$csi(I_{Shop})=P_{Shop}$.因为例 1 中的 P_{Shop} 是良定义的,所以 I_{Shop} 是良定义的.

1.3 协议接口

由于会话只是动作的集合,并没有刻画动作间的时序关系,因此定义了协议接口.在面向服务的系统中存在多种服务调用方式,包括顺序调用、并发调用等.例如,在供应链管理系统中,商店为了能让客户尽快地拿到商品,采取同时付款和送货的方式.针对这种特点,我们给出协议项来表示.

定义 5(协议项). 一个动作集 A 上的协议项定义如下:

$$term::\tau\mid a\mid a\sqcup b\mid a\sqcap b\mid a\boxplus b\mid a;b\mid\langle term\rangle.$$

其中:$a,b\in A$;τ表示没有动作需要被引发;a 表示需要引发动作 a;$a\sqcup b$ 表示引发动作是在 a 和 b 之间的非确定性选择;$a\sqcap b$ 表示动作 a 和 b 同时被引发,只有两个动作都完成以后协议项才算完成;$a\boxplus b$ 表示动作 a 和 b 同时被引发,a 或 b 的完成都可以使协议项完成;$a;b$ 表示只有动作 a 完成以后,动作 b 才能够被引发;⟨$term$⟩表示尖括号中的协议项是一个子事务,称为事务协议项.如果 $term$ 会引发异常动作,则仅仅由 $term$ 引发的动作需要被补偿或进行错误处理;如果 $term$ 不会引发异常,⟨$term$⟩的引发行为与 $term$ 相同,失去了事务协议项的意义,所以本文假设⟨$term$⟩中的 $term$ 必须包含异常动作.由于⟨$term$⟩是个独立的子事务,所以多层嵌套事务协议项的行为与单层相同,即⟨⟨$term$⟩⟩=⟨$term$⟩.我们用 $Term(A)$表示 A 上的所有协议项.

对于服务接口描述中含有时序的接口行为,可以用扩展协议自动机来表示动作的引发顺序,然后针对不同的本地动作,根据与其相关的各种行为,作本地动作到扩展协议自动机的映射.

定义 **6**(扩展协议自动机,**extended protocol automaton**,简称 **EPA**). 扩展协议自动机 G 是一个三元组 (A,L,δ),其中

- $A\subseteq M\times O$ 是 EPA 中可能出现的动作集;
- L 是位置集合,其中 $\perp\in L$ 是返回位置,$\boxtimes\in L$ 是异常位置;

- $\delta \subseteq (L\setminus\{\bot,\boxtimes\})\times Term(A)\times L$ 是位置的迁移关系.

在 EPA 中,一个位置是可终止的当且仅当从这个位置出发存在一条以⊥或⊠结尾的路径.

定义 7(协议接口,**protocol interface**,简称 **PI**). 协议接口 T 是一个四元组(G,R,R_C,R_F),其中

- G 是一个扩展协议自动机;
- R: $A\to L$ 是一个部分函数,给出一个动作的引发行为在 G 中的起始位置;
- R_C: $A\to L\setminus\{\boxtimes\}$是一个部分函数,给出一个成功动作的补偿行为在 G 中的起始位置;
- R_F: $A\to L\setminus\{\boxtimes\}$是一个部分函数,给出一个异常动作的错误处理行为在 G 中的起始位置;
- $dom(R_C)\cup dom(R_F)=dom(R)$,并且 $dom(R_C)\cap dom(R_F)=\varnothing$.

在协议接口 T 中,一个位置是可终止的当且仅当这个位置在 T 的 G 中能够终止,并且这个终止路径上可能引发的动作在 G 中对应的位置在 T 中也是可终止的.一个协议接口 $T=(G,R,R_C,R_F)$相应的特征接口($psi(T)$)可以计算为:$psi(T)=(A_s,S,S_C,S_F)$,$A_s=A$;如果 $a\in dom(R)$,则 $S(a)=sigl(R(a))$;如果 $a\in dom(R_C)$,则 $S_C(a)=sigl(R_C(a))$;如果 $a\in dom(R_F)$,则 $S_F(a)=sigl(R_F(a))$;否则,$S(a)$,$S_C(a)$和 $S_F(a)$都无定义,函数 $sigl:L\to 2^A$ 定义如下:

$$sigl(\bot)=\{\},sigl(\boxtimes)=\{\},sigl(q)=\bigcup\nolimits_{\exists(q,term,q')\in\delta}\sigma(term)\cup sigl(q'),$$

$$\sigma(\tau)=\{\},\sigma(a)=\{a\},\sigma(a\square b)=\{a,b\},\text{其中:}\square\in\{\sqcup,\sqcap,\boxplus,;\},\sigma([term])=\sigma(term).$$

协议接口 T 中动作 a 的类型与 a 在 $psi(T)$中的类型一致.

如果服务描述中的接口行为不是终止的,那么就会一直引发动作,使系统的资源耗尽,这不是人们所期望的结果.此外,本文暂不考虑子事务的嵌套和子事务被并发引发,这些要求和假设可以通过协议接口的良定义进行保证.如果协议接口 T 满足以下全部条件,则称 T 是良定义的:1) $psi(T)$是良定义的,并且 T 中每个本地动作的引发行为的起始位置是可终止的,每个补偿行为的起始位置是可终止的,每个错误处理行为的起始位置也是可终止的;2) 在补偿或错误处理中不引发事务协议项,事务协议项不能够递归引发或被并发引发.本文只讨论良定义的协议接口.协议接口的语义限于篇幅,简略阐述如下:

- 协议接口 T 的一次执行过程从某个本地动作 a 被引发开始.如果 a 是成功动作,则不需要记录动作的引发过程;如果 a 是异常动作,则需要记录动作的引发过程用于异常发生后的补偿和错误处理.为了保证补偿过程的正确性,记录的原则是先进后出.执行过程从 EPA 中 a 的正常行为的起始位置开始.
- 如果当前位置 q 在 EPA 中存在协议项为 $term$ 的迁移,并且本次执行过程是由异常动作引发的,就需要记录 $term$,然后根据它代表的不同引发方式来引发新的动作.在动作完成返回以后,从迁移的目标位置开始新的迁移.
- 如果当前位置 q 是返回位置,则需要进行返回动作,这是一个内部动作,返回位置可能是在某个并发分支中.这时,将根据引发分支的协议项的类型来确定返回操作,返回动作标志本分支正常完成.
- 如果当前位置 q 是异常位置,则说明有异常动作发生,也要开始返回.如果异常发生在某个并发分支中,则需要根据引发分支的协议项的类型来确定异常操作:如果是由"⊓"引发的并发分支,采取的策略是马上终止其他分支的动作,整个过程发生了异常;如果是由"⊞"引发的并发分支,采取的策略是:如果每个分支都发生异常,则认为整个过程发生异常,如果存在一个分支正确返回,则认为整个过程是成功的.
- 在执行完所有的正常引发动作返回以后,如果执行过程是由成功动作引发,则执行过程结束;如果是由异常动作引发,则需要根据前面记录的协议项进行补偿或错误处理:依次取出已经记录的协议项,根据不同的引发类型和其中的动作类型进行补偿或错误处理,起始位置是 T 的补偿函数或错误处理函数给出的记录动作的对应位置,当所有记录的协议项依次被处理以后,这一次执行过程才结束.

协议接口中的一条动作路径是某个执行过程中每一步出现的动作集组成的路径,由一个动作可能引发多条动作路径.对于 EPA 中的一个位置 q,能够从 q 开始的动作路径称为 q-路径.通过 PI 的迁移语义,可以把 PI 中本地动作所引发的接口行为转换到一个标记迁移系统(labelled transition system,简称 LTS),用 $LTS(T,a)$来表示协议接口 T 中的本地动作 a 能够引发的接口行为所对应的 LTS.

例 3:良定义的协议接口.在例 1 的基础上,我们使用协议接口对供应链管理系统中的商店(shop)服务的接口进行描述,可得到如下协议接口 $T_{Shop}=(G_{Shop},R_{Shop}, R_{C_{Shop}}, R_{F_{Shop}})$.为了简单起见,用$(l,term,l')$表示 G_{shop} 中存在位置 l 到 l',且协议项是 $term$ 的位置迁移,在迁移前面的动作表示不同的位置函数,迁移中的源位置是动作在不同位置函数中的对应值.

$\{\langle \text{SellItem,SOLD}\rangle \xrightarrow{R} (q_0,\langle \text{ChkAvail,OK}\rangle,q_1),(q_1,\langle \text{ProcPay,OK}\rangle,q_2),(q_2,\langle \text{ShipItem,OK}\rangle,\perp);$

$\langle \text{SellItem,FAIL}\rangle \xrightarrow{R} (q_3,\langle \text{ChkAvail,FAIL}\rangle,\boxtimes),(q_3,\langle \text{ChkAvail,OK}\rangle,q_4),(q_4,\langle \text{ProcPay,FAIL}\rangle,\boxtimes),$
$(q_4,\langle \text{ProcPay,OK}\rangle,q_5),(q_5,\langle \text{ShipItem,FAIL}\rangle,\boxtimes);$

$\langle \text{ChkAvail,OK}\rangle \xrightarrow{R} (q_6,\langle \text{ChkStore,OK}\rangle,\perp);$

$\langle \text{ChkAvail,FAIL}\rangle \xrightarrow{R} (q_7,\langle \text{ChkStore,FAIL}\rangle,\boxtimes),\langle \text{Apologize,OK}\rangle \xrightarrow{R} (q_8,\langle \text{SendLetter,OK}\rangle,\perp);$

$\langle \text{Recede,OK}\rangle \xrightarrow{R} (q_9,\langle \text{Compensate,OK}\rangle,q_{10}),(q_{10},\langle \text{WithDraw,OK}\rangle,q_{11}),(q_{11},\langle \text{RecoverStore,OK}\rangle,\perp);$

$\langle \text{SellItem,SOLD}\rangle \xrightarrow{R_C} (q_{12},\langle \text{Recede,OK}\rangle,\perp),\langle \text{ChkAvail,OK}\rangle \xrightarrow{R_C} \perp,\langle \text{Recede,OK}\rangle \xrightarrow{R_C} \perp,$
$\langle \text{Apologize,OK}\rangle \xrightarrow{R_C} \perp;$

$\langle \text{SellItem,FAIL}\rangle \xrightarrow{R_F} (q_{13},\langle \text{Apologize,OK}\rangle,\perp),\langle \text{ChkAvail,FAIL}\rangle \xrightarrow{R_F} \perp\}.$

由$\langle$SellItem,FAIL$\rangle$动作可能引发如下的动作路径$\{\langle$ChkAvail,OK$\rangle\},\{\langle$ChkStore,OK$\rangle\},\ldots,\{\langle$ProcPay,OK$\rangle\},\ldots,\{\langle$ShipItem,FAIL$\rangle\},\ldots,\{\langle$Apologize,OK$\rangle\},\{\langle$SendLetter,OK$\rangle\},\ldots$(省略的是内部动作).$T_{Shop}$ 是良定义的.

1.4 相容性与可替换性

在接口层面判断服务组合后是否能正常工作,对于构建面向服务的系统非常重要.以供应链管理系统为例,我们需要判断商店服务与库存服务在接口层面能否组合在一起.首先,两个被组合的服务不能提供相同的动作供其他服务调用,即商店和库存不能同时提供购买商品的方法,否则顾客购买商品时就会引发不一致性;此外,还要求组合形成的服务接口也是完整的.

相容性 **1**(特征接口的相容性). 给定两个特征接口 $P_1=(A_1,S_1, S_{C_1}, S_{F_1})$和 $P_2=(A_2,S_2, S_{C_2}, S_{F_2})$,如果以下两个条件都成立,则 P_1 和 P_2 是相容的:1) $dom(S_1)\cap dom(S_2)=\varnothing$;2) $P_1\cup P_2=(A_1\cup A_2,S_1\cup S_2, S_{C_1}\cup S_{C_2}, S_{F_1}\cup S_{F_2})$是良定义的.用 $P_1\|P_2=P_1\cup P_2$ 表示两个相容的特征接口的组合结果.

相容性 **2**(会话接口的相容性). 给定两个会话接口 $I_1=(A_1,E_1, E_{C_1}, E_{F_1})$和 $I_2=(A_2,E_2, E_{C_2}, E_{F_2})$,如果以下两个条件都成立,则 I_1 和 I_2 是相容的:1) $csi(I_1)$与 $csi(I_2)$是相容的;2) $I_1\cup I_2=(A_1\cup A_2,E_1\cup E_2, E_{C_1}\cup E_{C_2}, E_{F_1}\cup E_{F_2})$是良定义的.用 $I_1\|I_2 = I_1\cup I_2$ 表示两个相容的会话接口的组合结果.

相容性 3(协议接口的相容性). 给定两个协议接口 $T_1=(G_1,R_1, R_{C_1}, R_{F_1})$和 $T_2=(G_2,R_2, R_{C_2}, R_{F_2})$,如果以下两个条件都成立,则 T_1 和 T_2 是相容的:1) $psi(T_1)$与 $psi(T_2)$是相容的,并且 $L_1\cap L_2=\{\perp,\boxtimes\}$;2) $T_1\cup T_2=(G_1\cup G_2,R_1\cup R_2, R_{C_1}\cup R_{C_2}, R_{F_1}\cup R_{F_2})$是良定义的,其中 $G_1\cup G_2=(A_1\cup A_2,L_1\cup L_2,\delta_1\cup\delta_2)$.用 $T_1\|T_2=T_1\cup T_2$ 表示两个相容的协议接口的组合结果.

因为在组合形成的新服务中,某些以前对于子服务接口的非本地动作会变成本地动作,所以不同层面的接口行为会发生改变.

定理 **1**. 对于 SI,CI 和 PI,相容的组合操作都是可交换和可组合的.

在系统设计或运行过程中,可能会用某个服务替换系统中的另一个服务,并且希望替换后的系统仍然能够正常运行.对于服务的接口,我们认为它的本地动作及其类型是这个服务对于外界环境的保证,而必需动作是对外界环境的假设.如果新服务所提供的保证不比原来的服务少,而对外界环境的假设不会更多,我们就认为新的服务对于原来的服务是可以替换的.

精化 **1**(特征接口的精化). 给定两个 SI:$P_1=(A_1,S_1, S_{C_1}, S_{F_1})$和 $P_2=(A_2,S_2, S_{C_2}, S_{F_2})$,如果以下条件全部成立,则 P_2 精化 $P_1(P_2\preccurlyeq P_1)$:1) 所有 P_1 的本地动作在 P_2 中都是本地动作;2) 所有 P_1 的成功动作在 P_2 中都是成功动作; 3) 所有 P_1 的异常动作在 P_2 中都是异常动作;4) 对于所有 $a,a'\in A_1\cup A_2$,如果 a 是 P_1 的本地动作,并且 P_2 中的函数$\zeta_2\in\{S_2, S_{C_2}, S_{F_2}\}$在 a 上引发的动作集中包含 a',则 P_1 中与 P_2 对应的动作函数$\zeta_1\in\{S_1, S_{C_1}, S_{F_1}\}$在 a 上引发的动作集也包含 a';5) 对于所有 P_2 中的非本地动作 a,如果 a 在 P_2 中是必需动作,则 a 在 P_1 中也是必需动作.

精化 **2**(会话接口的精化). 给定两个 CI:$I_1=(A_1,E_1,E_{C_1},E_{F_1})$和 $I_2=(A_2,E_2,E_{C_2},E_{F_2})$,如果以下条件全部成立,则 I_2 精化 $I_1(I_2\preccurlyeq I_1)$:1) $csi(I_2)\preccurlyeq csi(I_1)$;2) 对于 I_1 的所有本地动作 a,如果 a 能够在 I_2 中引发会话 t,则 a 能够在 I_1 中引发一个包含 t 的会话;3) 对于 I_2 的所有本地动作 a,如果 a'存在于 a 引发的某个会话中且 a'不是本地动作,则 a'在 $csi(I_1)$中是必需动作.

精化 **3**(协议接口的精化). 给定两个 PI:$T_1=(G_1,R_1,R_{C_1},R_{F_1})$和 $T_2=(G_2,R_2,R_{C_2},R_{F_2})$,如果以下条件全部成立,则 T_2 精化 $T_1(T_2\preccurlyeq T_1)$:1) $psi(T_2)\preccurlyeq psi(T_1)$;2) 对于 T_1 的所有本地动作 a,a 在 T_2 中引发的接口行为对应的 LTS 与 a 在 T_1 中引发的接口行为对应的 LTS 存在弱模拟关系*.

定理 **2**. 给定 3 个会话接口 I_1,I_2 和 I_3,I_1 与 I_3 是相容的,I_2 与 I_3 也是相容的,如果 $I_2\preccurlyeq I_1$,则 $I_2\|I_3\preccurlyeq I_1\|I_3$.

定理 **3**. 给定 3 个协议接口 T_1,T_2 和 T_3,T_1 与 T_3 是相容的,T_2 与 T_3 也是相容的,如果 $T_2\preccurlyeq T_1$,则 $T_2\|T_3\preccurlyeq T_1\|T_3$.

2 接口规约及验证

为了保证服务接口的准确性,需要对服务接口的某些关键性质进行验证.不同的性质可以在接口的不同层面进行验证,对于某些时序性质(例如在供应链管理系统中,要求购买商品时,商品发送必须发生在付款之后),需要在协议接口层进行验证;而付款成功和库存不足不可能同时发生,则可以在会话层进行验证.

2.1 会话规约及其验证

规约 **1**(会话规约,**conversation specification**,简称 **CS**). 一个会话接口 I 的会话规约 CS 如下定义:

$$CS :: a\mapsto\Diamond D \mid a\mapsto\Box D \mid a\nrightarrow D \mid a\nrightarrow\Diamond D \mid a\nrightarrow\Box D,$$

其中:$a\in dom(E)$,$D\subseteq A$,$a\mapsto\Diamond D$ 表示动作 a 可能引发包含 D 的会话;$a\mapsto\Box D$ 表示动作 a 引发的所有会话中都包含 D;$a\nrightarrow D$ 表示 D 不属于 a 引发的任何会话中;$a\nrightarrow\Diamond D$ 表示 D 中所有动作都不会被 a 引发;$a\nrightarrow\Box D$ 表示 a 可以引发不包含 D 中任何动作的会话.它们可以进行如下验证:1) $a\mapsto\Diamond D$ 成立当且仅当$\exists S\in[a]_R.D\subseteq S\backslash\{a\}$;2) $a\mapsto\Box D$ 成立当且仅当$\forall S\in[a]_R.D\subseteq S\backslash\{a\}$;3) $a\nrightarrow D$ 成立当且仅当$\forall S\in[a]_R.D\not\subseteq S$;4) $a\nrightarrow\Diamond D$ 成立当且仅当$\forall S\in[a]_R.D\cap S=\varnothing$; 5) $a\nrightarrow\Box D$ 成立当且仅当$\exists S\in[a]_R.D\cap S=\varnothing$.

例 4:会话规约的验证.库存(store)的会话接口 $I_{Store}=(A_{Store},E_{Store},E_{C_{Store}},E_{F_{Store}})$,其中:

$E_{Store}=\{\langle\text{ChkStore,OK}\rangle\to\text{T}\sqcup(\langle\text{GetOffer,OK}\rangle\sqcap\langle\text{Order,OK}\rangle),\langle\text{ChkStore,FAIL}\rangle\to\text{T},\langle\text{RecoverStore,OK}\rangle\to\text{T}\}$;

$E_{C_{Store}}=\{\langle\text{ChkStore,OK}\rangle\to\langle\text{RecoverStore,OK}\rangle,\langle\text{RecoverStore,OK}\rangle\to\text{T}\}$;

$E_{F_{Store}}=\{\langle\text{ChkStore,FAIL}\rangle\to\langle\text{GetOffer,OK}\rangle\sqcap\langle\text{Order,OK}\rangle\}$.

与例 2 中的商店接口进行组合,得到的组合服务接口 $I_{Shop}\|I_{Store}$.对 $I_{Shop}\|I_{Store}$ 可以验证如下几条规约:

- $\langle\text{SellItem,FAIL}\rangle\mapsto\Diamond\{\langle\text{RecoverStore,OK}\rangle\}$表示购买失败可能会导致库存恢复;
- $\langle\text{SellItem,FAIL}\rangle\mapsto\Box\{\langle\text{GetOffer,OK}\rangle\}$表示购买失败总会导致库存部门向供应商订购商品;
- $\langle\text{SellItem,FAIL}\rangle\mapsto\Box\{\langle\text{Apologize,OK}\rangle\}$表示商店对于购买失败总会向客户道歉;
- $\langle\text{SellItem,FAIL}\rangle\nrightarrow\{\langle\text{ChkStore,FAIL}\rangle,\langle\text{ProcPay,OK}\rangle\}$表示不可能付款成功和库存不足同时发生.

除了第 2 条以外,上述几条规约都成立.通过检查发现:在库存充足并且库存剩余量也充足的情况下,付款失败会导致购买失败,而此时库存部门不必向供应商订购商品.

定理 **4**. 给定一个会话接口 I_1,如果 I_1 满足 $a\nrightarrow D$ 或 $a\nrightarrow\Diamond D$(其中 a 是 I_1 的本地动作),则对于所有会话接口 I_2,如果 $I_2\preccurlyeq I_1$,则 I_2 满足相应的会话规约.

推论 **1**. 给定 3 个会话接口 I_1,I_2 和 I_3,I_1 与 I_3 是相容的,I_2 与 I_3 也是相容的,且 $I_2\preccurlyeq I_1$,如果 $I_1\|I_3$ 满足 $a\nrightarrow D$ 或 $a\nrightarrow\Diamond D$(其中 a 是 $I_1\|I_3$ 的本地动作),则 $I_2\|I_3$ 满足相应的会话规约.

定理 **5**. 给定两个会话接口 I_1 和 I_2,I_1 与 I_2 是相容的,如果 $I_1\|I_2$ 满足 $a\nrightarrow D$ 或 $a\nrightarrow\Diamond D$(其中 a 是 $I_1\|I_2$ 的本地动作),则 I_1 或 I_2 满足相应的会话规约.

---

* 弱模拟关系的主要思想可参考文献[12],唯一的区别是内部动作:文献[12]中只有一个内部动作 τ;而本文中存在多个内部动作.

2.2 协议规约及其验证

规约 **2**(协议规约,**protocol specification**,简称 **PS**). 一个协议接口 T 的协议规约 PS 定义为

$$PS :: a\mapsto \mathbf{A}\gamma \mid a\mapsto \mathbf{E}\gamma,\ \gamma :: \Box P \mid \Diamond P \mid P_1 \mathrm{U} P_2,\ P :: D \mid \neg D.$$

其中:$a\in dom(R)$,$D\subseteq A$.假设 t 是 T 中一条由 a 引发的动作路径 $t=a_1a_2\ldots a_n\ldots$:

- 对于 t 中的所有动作集 a_i,如果 $a_i\cap D\neq\varnothing$,则 a_i 满足 D,否则 a_i 满足 $\neg D$;如果 a_i 满足 P,则 $a_i\vDash P$,否则 $a_i\nvDash P$;
- 如果 t 中存在某个动作集 a_i 满足 P,则 t 满足 $\Diamond P$;如果 t 中所有的动作集 a_i 都满足 P,则 t 满足 $\Box P$;如果 t 中存在某个动作 a_j 满足 P_2,并且所有动作集 $a_i(i<j)$都满足 P_1,则 t 满足 $P_1\mathrm{U}P_2$;
- 如果动作 a 能够引发的所有动作路径都满足 γ,则 $a\mapsto\mathbf{A}\gamma$成立;如果动作 a 能够引发满足 γ的动作路径,则 $a\mapsto\mathbf{E}\gamma$成立;$q\vDash\mathbf{A}\gamma(q\in L)$表示所有 q-路径都满足 γ,$q\vDash\mathbf{E}\gamma$表示存在 q-路径满足 γ.

由于在面向服务的系统中可能存在动作相互引发的问题,并且扩展协议自动机中也可能存在回路,使得 PI 的本地动作可能会引发无穷路径,并且它引发的接口行为对应的 LTS 也会是一个无穷标记迁移系统.

针对不能引发事务协议项的成功动作,可类似文献[5]给出的推理规则和算法来解决无穷路径的问题.其基本思想是,通过 EPA 中的迁移关系,针对具体需要验证的规约进行反向循环推理,直到不能得到新的断言公式为止,若此时在断言公式集合中存在表明规约满足的公式,则表明协议接口满足此规约.

对于异常动作或能够引发事务协议项的成功动作,它们会引发补偿或错误处理行为,因此需要记录动作的引发过程.此外,发生异常而造成的并发行为的同步也会影响动作的调用顺序.本文假设需要检验的协议接口中不存在动作的相互引发,并且扩展协议自动机中不存在回路,这样就不会引发无穷动作路径.对于任意需要验证的规约 $PS=a\mapsto\mathbf{A}\gamma$或 $PS=a\mapsto\mathbf{E}\gamma$,由 a 引发的动作路径是有限的,$LTS(T,a)$是一棵树,每个叶节点代表一条路径,首先通过 $LTS(T,a)$得到所有由 a 引发的动作路径的集合 TS,然后通过算法 2(*JudgeSpec* 算法)对规约进行验证,算法的基本思想是通过搜索来验证规约的正确性.对于 $\mathbf{A}\gamma$,需要验证 $LTS(T,a)$中的所有路径都满足 γ,对于 $\mathbf{E}\gamma$,只要验证存在一条路径满足 γ即可.如果不满足 $\mathbf{A}\gamma$,可以返回一条路径显示不满足的原因;如果 $\boldsymbol{E}\gamma$满足,也可以返回一条路径显示 γ的例证.一条路径是否满足公式 γ可以通过算法 1(*JudgeTrace* 算法)进行验证.*JudgeTrace* 的算法复杂度是 $O(n)$,其中 $n=|t|$是输入路径的长度.*JudgeSpec* 的算法复杂度是 $O(n\cdot k)$,其中:$n=max_{t\in TS}|t|$是输入路径集合中的最长路径长度;$k=|S|$是输入路径集合中的路径数量.*JudgeSpec* 算法的输入路径中不包含内部动作.

算法 **1**. *JudgeTrace*(t,γ).
输入:路径 $t=a_1a_2\ldots a_n$,性质γ.
输出:T(如果 t 满足γ),F(否则).
变量:布尔变量 b_1

```
1   b1=F;
2   for each ai of t  do
3      if   γ=□P and ai⊭P then return F;
4      if   γ=◇P and ai⊨P then return T;
5      if   γ=P1UP2 and ai⊨P2 then
6         if   not b1 then return T;
7      if   γ=P1UP2 and ai⊭P1 then
8         b1=T;
9   end
10  if γ=□P then return T;
11  return F;
```

算法 **2**. *JudgeSpec*(PS,TS).
输入:性质 PS,PS 中的动作能够引发的路径集合 TS.
输出:T(如果 TS 满足 PS),F(否则).

```
1   for each t of TS do
2      if   PS=a↦Aγ and not JudgeTrace(t,γ) then
3         print t;
4         return F;
5      end
6      if   PS=a↦Eγ and JudgeTrace(t,γ) then
7         print t;
8         return T;
9      end
10  end
11  if PS=a↦Aγ then return T;
12  return F;
```

例 5:协议规约的验证.使用协议接口对供应链管理系统各个部分的接口进行描述,然后在组合后的协议接口上验证某些协议规约.表 1 中是协议规约和验证结果.其中协议规约的含义是:1) 购买成功总是在库存充足的情况下发生;2) 购买成功总会导致库存向供应商下订单;3) 付款可能在库存检查成功之前发生;4) 付款可能在库存检查成功之前发生;5) 商店总会在购买失败后送出道歉信;6) 购买失败可能会导致退款发生在付款之前.

定理 **6**. 给定一个协议接口 T_1,如果 T_1 满足 $a\mapsto\mathbf{A}\Box D$ 或 $a\mapsto\mathbf{A}\Box\neg D$(其中 a 是 T_1 的本地动作),那么对于所有

协议接口 T_2,如果 $T_2 \preccurlyeq T_1$,则 T_2 满足相应的协议规约.

推论 2. 给定 3 个协议接口 T_1,T_2 和 T_3, T_1 与 T_3 是相容的,T_2 与 T_3 也是相容的,且 $T_2 \square T_1$,如果 $T_1 \| T_3$ 满足 $a \mapsto \mathbf{A}\square D$ 或 $a \mapsto \mathbf{A}\square\neg D$ (其中 a 是 $T_1 \| T_3$ 的本地动作),则 $T_2 \| T_3$ 满足相应的协议规约.

定理 7. 给定两个协议接口 T_1 和 T_2,T_1 与 T_2 是相容的,如果 $T_1 \| T_2$ 满足 $a \mapsto \mathbf{A}\square D$ 或 $a \mapsto \mathbf{A}\square\neg D$(其中 a 是 $T_1 \| T_2$ 的本地动作),则 T_1 或 T_2 满足相应的协议规约.

Table 1 Protocol specifications and verification results

表 1 协议规约和验证结果

No.	Protocol specification	Result
1	⟨SellItem,SOLD⟩↦**A**◇{⟨ChkAvail,OK⟩}	True
2	⟨SellItem,SOLD⟩↦**A**◇{⟨Order,OK⟩}	False
3	⟨SellItem,SOLD⟩↦**E**¬{⟨ChkStore,OK⟩}U{⟨ProcPay,OK⟩}	False
4	⟨SellItem,FAIL⟩↦**E**¬{⟨ChkStore,OK⟩}U{⟨ProcPay,OK⟩}	False
5	⟨SellItem,FAIL⟩↦**A**◇{⟨SendLetter,OK⟩}	True
6	⟨SellItem,FAIL⟩↦**E**¬{⟨ProcPay,OK⟩}U{⟨Compensate,OK⟩}	False

3 相关工作比较

文献[8]使用有限状态进程(finite state process,简称 FSP)对服务组合语言 BPEL4WS(BPEL)[9]进行形式化,然后使用模型检验方法验证组合语言的正确性,以保证通过 BPEL 构建的系统的可靠性.文献[10]使用 Guarded 有限状态机对单个服务接口建模,然后将模型转换到 Promela 语言,并在 SPIN 上进行验证,以保证接口行为的正确性.这两种方法在对接口层面的事务模型的描述能力上存在不足,它们对 BPEL 中错误处理和补偿行为的描述不够精确,例如,它们不能处理 BPEL 中的多层作用域,同时也没有考虑 BPEL 中默认的错误处理和补偿机制.基于面向构件的接口理论,本文提出的方法充分考虑了面向服务系统中的长事务模型的特点,能够对接口层面的事务行为进行精确描述.

文献[11]使用补偿通信顺序进程(compensating CSP,简称 cCSP)来描述长事务模型.补偿通信顺序进程是通信顺序进程的一种变体,它提供了补偿进程对来描述补偿行为,同时也通过标准的中断进程、中断处理对和产生中断进程对错误发生及错误处理行为进行描述,并且还提供了事务块进程来表示独立的子事务.文献[11]中处理并发进程中分支进程产生错误的策略与本文相同,都认为分支进程的错误会导致并发进程的错误.然而,文献[11]中并没有对并发进程中的分支进程发生错误后的进程同步策略进行讨论,而这一点在事务处理中非常重要.本文给出了并发进程中异常发生以后的同步策略,提出了选择性并发机制及其异常处理策略.

文献[5]是本文的基础,提出了特征、会话和协议 3 个抽象层面的接口描述方法,但它并没有考虑接口层面的事务特征,只能描述动作之间的正常引发关系(只包含正常动作函数).本文对每层的接口模型进行扩充,分别加入了错误处理和补偿,区分了动作的类型.本文在会话层重新定义了会话集函数,在协议层增加了顺序项和事务协议项,给出了满足长事务模型的语义模型.同时,本文重新给出了各层接口模型的相容性和可替换性条件.文献[5]在会话层提出了会话规约,本文在此基础上对会话规约的语法进行了扩充,加入了 4 种规约模式及其验证方法.本文也对文献[5]中的协议规约的语法进行了扩充,加入了分支存在算子和线性存在算子,并使 U 算子公式的构造方式更加一般化.在新的协议接口的定义下,针对不同的动作引发情况提出了相应的验证方法.

4 结 论

保证面向服务系统可靠的形式化方法是面向服务的软件方法学研究的热点之一.本文针对面向服务的软件体系结构,提出了支持事务信息描述的多层服务接口模型,在此基础上给出了接口的相容性和可替换性条件,并专门对会话接口和协议接口提出了接口规约及其验证方法.本文中的接口模型反映了服务接口中的事务特征,能够在面向服务的软件设计中保证接口的准确性和接口行为的正确性,同时也能够应用于面向服务软件体系结构中的服务发布和组合.

今后的工作包括针对具体服务组合描述语言(例如 BPEL)和异常嵌套与并发子事务的接口模型应用与扩展,以及相应的面向服务的形式验证.

References:

[1] Yang FQ. Thinking on the development of software engineering technology. Journal of Software, 2005,16(1):1−7 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/16/1.htm

[2] Cervanted H, Hall RS. Technical concepts of service orientation. In: Zoran S, Ajantha D, eds. Service-Oriented Software System Engineering: Challenges and Practices. Idea Group Publishing, 2005. 1−47.

[3] de Alfaro L, Henzinger TA. Interface automata. In: Proc. of the 9th Annual ACM Symp. on the Foundations of Software Engineering (FSE 2001). ACM Press, 2001. 109−120.

[4] de Alfaro L, Henzinger TA, Stoelinga M. Timed interfaces. In: Sangiovanni-Vincentelli A, Sifakis J, eds. Proc. of the 2nd Int'l Conf. on Embedded Software (EMSOFT 2002). LNCS 2491, Berlin: Springer-Verlag, 2002. 108−122.

[5] Beyer D, Chakrabarti A, Henzinger TA. Web service interfaces. In: Proc. of the 14th Int'l World Wide Web Conf. (WWW 2005). Chiba: ACM Press, 2005. 148−159.

[6] Beyer D, Chakrabarti A, Henzinger TA. An interface formalism for Web services. In: Proc. of the 1st Int'l Workshop on Foundations of Interface Technologies (FIT 2005). San Francisco: Elsevier, 2005.

[7] Little M. Transactions and Web services. Communication of the ACM, 2003,46(10):49−54.

[8] Foster H, Uchitel S, Magee J, Kammer J. Compatibility verification for Web service choreography. In: Proc. of the 3rd IEEE Int'l Conf. on Web Services (ICWS 2004). San Diego: IEEE Press, 2004. 738−741.

[9] Andrews T, Curbera F, Dholakia H, Goland Y, Klein J, Leymann F, Liu K, Roller D, Smith D, Thatte S, Trickovic I, Weerawarana S. Business process execution language for Web services. Version 1.1. BEA Systems, Int'l Business Machines Corporation, Microsoft Corporation, 2003.

[10] Fu X, Bultan T, Su J. Analysis of interacting BPEL web services. In: Proc. of the 13th Int'l World Wide Web Conf. (WWW 2004). New York: ACM Press, 2004. 621−630.

[11] Butler M, Ripon S. Executable semantics for compensating CSP. In: Proc. of the 2nd Int'l Workshop on Web Services and Formal Methods. LNCS 3670, Versailles: Springer-Verlag, 2005. 243−256.

[12] De Wachter B, Genon A, Massart T, Meuter C. The formal design of distributed controllers with $_d$SL and spin. Formal Aspects of Computing, 2005,17(2):177−200.

附中文参考文献:

[1] 杨芙清.软件工程技术发展思索.软件学报,2005,16(1):1−7. http://www.jos.org.cn/1000-9825/16/1.htm

陈振邦(1981－),男,湖北武汉人,博士生,主要研究领域为软件体系结构,面向服务的软件工程,软件验证.

董威(1976－),男,博士,副教授,主要研究领域为高可信软件工程技术.

王戟(1969－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为高可信软件技术,软件方法学,软件工程.

齐治昌(1942－),男,教授,博士生导师,主要研究领域为软件工程.