ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.20, No.6, June 2009, pp.1470−1486
doi: 10.3724/SP.J.1001.2009.03353

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

基于分布式协调模型的服务协作方法研究*

乔晓强+, 魏 峻, 黄 涛

(中国科学院 软件研究所 软件工程技术研究开发中心,北京 100190)

Service Collaboration Approach Based on Decentralized Mediation Model

QIAO Xiao-Qiang+, WEI Jun, HUANG Tao

(Technology Center of Software Engineering, Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China)

+ Corresponding author: E-mail: qiaoxiaoqiang@otcaix.iscas.ac.cn

Qiao XQ, Wei J, Huang T. Service collaboration approach based on decentralized mediation model. *Journal of Software*, 2009,20(6):1470−1486. http://www.jos.org.cn/1000-9825/3353.htm

Abstract: As services are developed by independent providers, it is impossible to predict all potential interactive possibility at development stage. In order to ensure the correct service collaboration, it is necessary to check the interaction compatibilities among participating services. This paper presents a top-down approach to improve the reusability of available services and the flexibility of service collaboration based on a decentralized mediation model. Through failure-equivalent projection, the collaboration process is transformed to decentralized processes, which facilitates a more efficient pair-wise compatibility check. A mediation-based method is proposed to check if the compatibilities can be achieved by using mediation mechanism. Algorithms of compatibility check and automatic mediator generation are also provided. This approach makes the coupling between available services and collaborative environment more looser, thus improving the flexibility of collaboration while preserving the autonomy of services.

Key words: service collaboration; session-oriented service; protocol compatibility; protocol mediation

摘 要: 由于服务是由彼此独立的提供商开发的,无法在开发阶段就预测到其潜在的所有交互可能,因此需要在实际协作时检查服务之间的兼容性,从而保障协作的正确性和一致性.提出了一种基于分布式协调模型的方法,以提高服务的可复用性和服务协作的灵活性.该方法通过失败等价(failure-equivalent)行为语义保持的投影规则,将协作流程转换为协作子流程,以实现分布式的成对兼容性检查,并提出基于适配的检查方法,检验服务是否可以通过适配机制满足兼容性的要求,同时给出了适配器自动生成的算法.适配机制的引入进一步降低了服务与协作环境的耦合关系,从而在保持服务组件自治特性的同时提高了服务协作的灵活性.

关键词: 服务协作;会话类服务;协议兼容性;协议适配

中图法分类号: TP311 **文献标识码**: A

* Supported by the National Natural Science Foundation of China under Grant No.60673112 (国家自然科学基金); the National Key Technology R&D Program of China under Grant No.2006BAH02A02 (国家科技支撑计划); the National High-Tech Research and Development Plan of China under Grant No.2006AA01Z19B (国家高技术研究发展计划(863))

Received 2008-01-11; Revised 2008-03-12; Accepted 2008-04-21

面向服务计算以可重用的服务为构造单元,为解决分布式应用集成提供了新型的计算范型[1],近年来得到了迅速的发展.服务作为自治的、自描述的和平台独立的模块化应用[2],基于接口描述来展现其功能,并通过SOAP[3],WSDL[4],UDDI[5]等服务描述、发布、发现和调用标准来提供协作的基础.单个的服务由于功能单一,不能提供完整的解决方案.为了实现复杂的业务逻辑,就必须对服务进行组合和协作.基于服务的组合技术为企业间的B2B分布式协作提供了有效的解决方案.实际业务中存在两类面向服务协作的应用场景[6]:一类是多个合作伙伴通过定义相互之间的消息交互流程,实现共同的业务目标.例如,RosettaNet组织所提出的面向电子商务的企业协作流程等标准;另一类则是单个企业实体作为服务请求者,建立组合服务的需求模型,并查找和集成其他企业可用的服务,从而为客户提供增值服务.

服务协作中一个重要的概念是服务的接口描述模型.服务的接口描述了服务的功能以及如何使用该功能的方式,为服务协作和匹配提供基础.文献[7]中将服务分为离散型的服务(discrete service)和会话式的服务(session-oriented service)两种类型.其中,前者表现为无状态的多个服务方法的简单罗列;而后者则表现为有状态的交互过程,方法的调用除了满足输入/输出以外,还必须满足一定的时序关系,即针对该服务的访问协议.

服务协作的关键问题在于如何保证服务交互的一致性和正确性.会话式服务的引入又为服务协作带来了新的影响,参与协作的服务不仅需要在单个方法的输入/输出层次满足协作需求(操作兼容性),在服务的访问协议层次也同样需要满足整体的协作需求,即服务方法调用的时序关系必须满足正确性和一致性的要求(协议兼容性).因此,参与协作的服务组件必须在单个方法正确调用的基础上满足协议兼容的特性.

针对上述问题,本文提出了一种自上而下的实现服务协作的解决方案,基于分布式的方法来实现服务组件的兼容性检查,并提出了基于适配的协议兼容性检查方法,采用适配机制以弥合服务组件在交互协议上的差异,从而提高了服务组件的利用率,并给出了相应的协议兼容性检查算法和适配器自动生成算法.

本文的第 1 节分析相关工作以及本文的贡献.第 2 节~第 4 节给出基于分布式协调模型的服务协作方法的具体过程,并结合实例对本文的方法进行验证.最后是结论和今后工作的展望.

1 相关工作

在服务协作的实现方法上,很多公司以及研究机构给出了自己的研究成果,并推出了服务组合系统产品.HP实验室开发的e-flow系统[8]提供了一个集成平台,以支持组合服务的定义、运行与监控等,其执行模型主要基于集中式的流程引擎.SELF-SERV[9]是由澳大利亚的新南威尔士大学实现的一个陈述性的服务协同框架,以P2P的方式实现组合服务的执行.文献[10]介绍了模型驱动的服务组合方法,使用统一建模语言UML进行高层次的抽象,并能直接映射到其他标准,如WS-BPEL.同时,该方法使用OCL(object constraint language)来表达业务规则和流程.上述这些实现服务组合的方法只是将参与协作的服务看作是离散型的服务组件,而没有考虑具有状态的会话类服务参与协作的情况,即没有考虑服务组件的方法调用之间的时序逻辑关系.

服务在协作环境中的行为兼容性与一致性方面也有很多研究工作,如文献[11]提出了基于行为描述服务,实现服务协作的方法.该方法使用有限状态机(finite state machine)来描述服务接口.在进行服务组合时,首先将组合服务定义和服务组件的接口状态机转化为 DPDL(deterministic propositional dynamic logic)描述,通过判定 DPDL 的可满足性来检查是否存在满足目标的组合服务.但该工作主要基于集中式的结构,服务交互只存在于服务组件与组合服务之间,没有讨论分布式交互的问题.文献[12]把服务建模为一个 Mealy 自动机,采用基于会话的方法,分析参与协作的服务的局部行为与全局行为.文献[13]采用有限状态机对会话类服务进行建模,并以此来分析服务组合过程中的兼容性,是一种自下而上的方法.然而,服务的自下而上的直接组合可能产生不期望的全局行为,无法满足协作目标.文献[14,15]分别使用 FSP(finite state process),CSP(communicating sequential processes)对 WS-BPEL 或 WS-CDL 进行形式化描述,文献[16]则使用 LTS(labeled transition system)描述全局流程与单个服务组件行为.3 种方法都致力于检查编排流程与实际服务之间的一致性问题,其不足之处主要在于,采用集中式的检查方法将协作目标与所有服务组件进行整体检查,众多相互之间没有关联关系的操作交织在一起考虑,增加了检查方法的复杂性.此外,上述工作在进行协作检查时,服务组件需要满足严格的兼容性与一

致性要求,这降低了服务的可复用性,也不适用于经常发生动态改变的服务计算环境.实际上,服务组件之间交互的差异性可以通过一些方法(例如适配机制)来实现有效的弥合.

本文中,我们提出了一种自上而下建立企业间服务协作的解决方案.首先构造集中式的协作目标流程,并映射到各个参与者以实现分布式的兼容性检查.其协作模型如图 1 所示.与前述工作相比,本文的贡献主要在于:

首先,将角色信息引入服务协作目标,并基于角色将协作目标映射为 P2P 的分布式协作子流程(如图 1 所示的抽象服务接口集合),以此作为服务组件的检验标准,与服务组件进行成对的兼容性检查,从而只考虑与服务组件相关的交互过程的一致性,降低了检查方法的复杂性;

其次,提出了基于适配的兼容性检查方法,检查服务组件是否可以通过适配机制满足一致性和兼容性的要求,从而提高了组件的可复用性.在满足可适配的前提下,给出了适配器的自动生成算法.最终,系统通过适配器的分布式交互来实现协作目标,并基于适配器与服务组件的交互来保证协作的正确性;

最后,适配器层的引入进一步降低了实际服务与协作环境的耦合关系,使得在不改变源服务组件实现的前提下满足应用协作的要求.而且在协作目标或者单个服务组件发生变化时,只需重新配置适配器即可再次实现有效的协作,降低了系统更新的代价.本文所提供的方法在保持服务的自治特性的同时,提高了企业间服务协作的灵活性.

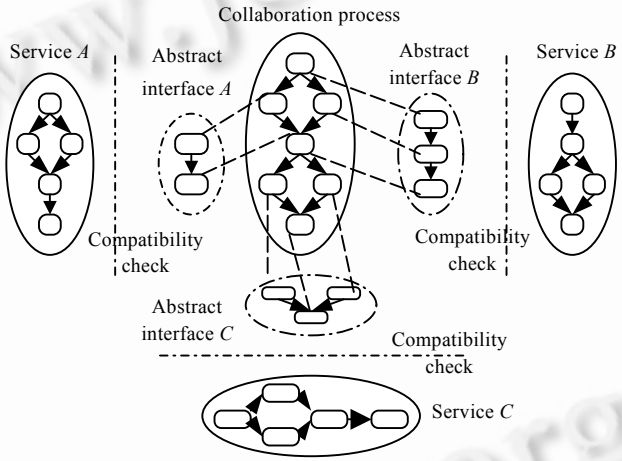


Fig.1 Distributed service collaboration model

图 1 分布式服务协作模型

2 全局视角的服务协作描述

企业协作目标或者组合服务需求模型通过集中式的服务协作流程来定义,从全局角度描述了参与协作的服务之间的公共交互消息和协作契约关系.本文使用CSP[17]语言来严格地描述服务协作流程.CSP主要通过实体之间外部可观察的交互行为来定义实体的行为,可以用来描述和分析并行系统通信.进程与事件是CSP中的重要概念,进程通过它能执行的通信事件来定义.一个进程所有可以执行的事件集合称为该进程的符号集,表示为αP.下面首先来介绍本文中所用到的CSP的操作符子集.

$$P=a\rightarrow P|\mu X\cdot F(X)|P\Box Q|P\sqcap Q|P;Q|P||Q|SKIP|STOP.$$

前缀(prefix):设有一事件为 a,有一进程为 P,则执行事件 a,然后按照进程 P 的描述来执行,用 $a\rightarrow P$ 表示.一个流程以前缀描述开始,则被称为是有前卫的(guarded).

循环(recursive):若 $F(X)$是一个包含了名称 X 的有前缀的流程描述,则$\mu X\cdot F(X)$给出循环执行流程 X 的定义.

外部确定性选择(external deterministic choice):一个流程可以按照进程 P 的描述执行,也可以按照 Q 的描述执行,其选择由外部环境决定,则表示为 $P\Box Q$.

内部非确定性选择(internal non-deterministic choice):一个流程可以按照进程 P 的描述执行,也可以按照 Q 的描述执行,其选择由流程自身决定,则表示为 $P \sqcap Q$.

顺序(sequence):$P;Q$ 表示流程首先按照进程 P 的描述执行,如果成功,则继续按照 Q 的描述执行.

并发(parallel):$P \| Q$ 表示通常的并行操作,进程 P 和 Q 将基于符号表中的相同事件进行同步,即进程 P 和 Q 在事件集 $\alpha P \cap \alpha Q$ 上同步.

CSP 中还定义了两种特殊进程:*STOP* 与 *SKIP*.其中,*STOP* 表示流程死锁,*SKIP* 表示流程执行成功.

最后给出执行轨迹的概念.流程的一个轨迹(trace)给出了一个确定时刻所记录的已执行有限事件的记录.*traces*(*P*)则定义了一个流程所有可能的轨迹的集合.

在描述并发流程方面,CSP 具有很强的表达能力.首先,CSP 具有较为完整的分布计算环境中的代数演算能力,CSP 提供了丰富的运算操作,如非确定选择、并发运算、选择运算等,这些都体现了分布式程序的特征;其次,死锁机制为分析导致流程异常终止的失配提供了基础;最后,并发组合很好地刻画了服务之间的交互与通信.

接下来,我们给出引入角色信息的全局协作流程的定义.

定义 1(GSCP). 全局的服务协作流程定义为一个三元组 $GSCP=(M,R,P_{CHOR})$,其中:

1) M是协作流程中所有交互消息的集合.$M=\{m_i|1\le i\le n\}$,n是消息的数目;

2) R是参与协作流程的角色的集合.$R=\{r_i|1\le i\le l\}$,l是角色的数目;

3) P_{CHOR}定义了参与协作的角色之间的消息交互流程,基于CSP进行描述.P_{CHOR}的事件符号集是形如 $m(r_s,r_d)$的集合,其中,$r_s,r_d\in R$,$m\in M$,表示消息m由角色r_s发送到角色r_d.

为了保证服务协作流程的正确性,我们给出了以下限制条件:

1) 首先,要求消息的名称具有唯一性;

2) 其次,对于选择操作符,要求参与协作的一个角色拥有决策权,即由一个角色内部(非确定性)决定所选择的分支路径.因此,在集中式的服务协作流程中只存在非确定性选择操作符($\sqcap$),而且要求选择分支的起始活动具有相同的消息发送者.我们给出一个反例进行说明:$[m_1(r_1,r_2)\to m_3(r_2,r_1)\to SKIP]\sqcap[m_2(r_3,r_2)\to m_4(r_2,r_3)\to SKIP]$.

在该流程中,如果r_1决定是否发送消息m_1到r_2,而r_3也独立地决定是否发送消息m_2到r_2,那么,如果r_1和r_3都决定发送相应的消息给r_2,则r_2就无法确定下一步的操作.在这种情况下,需要一个全局的仲裁者来决定分支的选择.但是在基于服务的协作流程中,参与的各方以P2P的方式实现相互协作,而不存在一个全局的控制者.因此,我们将忽略这种容易引起歧义或者错误状态的情况.同时,这种限制也符合一般的应用场景需求.

3) 最后,循环结构应具有以下的形式:$\mu X\cdot(P\sqcap Q;X)$.

即循环结构 $Q;X$ 可以执行一次或者多次,但是必须具有至少一个进程 P 以提供退出循环结构的选择分支.

下面给出协作流程中活动之间的偏序关系的定义.

定义 2. 对于协作协议P_{CHOR},$\forall a_1,a_2\in\alpha(P_{CHOR})$,$\forall s\in traces(P_{CHOR})$,若$a_1,a_2$存在于$s$中,且总有$first(a_1,s)<first(a_2,s)$,则$\boldsymbol{a}_1<<\boldsymbol{a}_2$(或$\boldsymbol{a}_2>>\boldsymbol{a}_1$).这里,函数$first(a,s)$表示"如果事件$a$存在于执行轨迹$s$中,计算事件$a$在执行轨迹$s$中首次出现的序号".$a_1<<a_2$表明执行结构中,$a_1$在$a_2$之前.如果$\neg(a_1<<a_2)\ \wedge\neg(a_2<<a_1)$,则$a_1$和$a_2$之间不存在偏序关系,表示为$\boldsymbol{a}_1\infty\boldsymbol{a}_2$.

图 2 给出了一个网上书店的案例.该实例主要包含 3 个角色:用户(C)、网上书店(B)和书库(W).用户可以通过发送查询请求到网上书店来获取图书列表,还可以发送图书订单到网上书店.书店接收到用户发来的订单信息后,将详细的订单信息发送到书库.根据书库返回的图书存储信息,书店可以选择接受或者拒绝用户的订单.客户收到订单接受消息后,会执行付款操作并接收发票信息.书店则同时发送送货信息给书库.书库完成送货后,返回确认信息到网上书店.图中每个活动表示角色之间的消息交互事件.其消息交互流程的 CSP 描述为

$$\begin{aligned}P_{CHOR}=\mu X\cdot[&(Search\_Request(C,B)\to Quote(B,C)\to X)\sqcap(Book\_Order(C,B)\to Order\_Details(B,W)\to\\&Book\_Info(W,B)\to[(Order\_Acceptance(B,C)\to[(Payment(C,B)\to Payment\_Confirmation(B,C)\to\\&Invoice(B,C)\to SKIP)\|(Shipment\_Order(B,W)\to Shipment\_Confirmation(W,B)\to SKIP)])\sqcap\end{aligned}$$

(*Order_Rejection*(*B*,*C*)→*SKIP*)]→*SKIP*)].

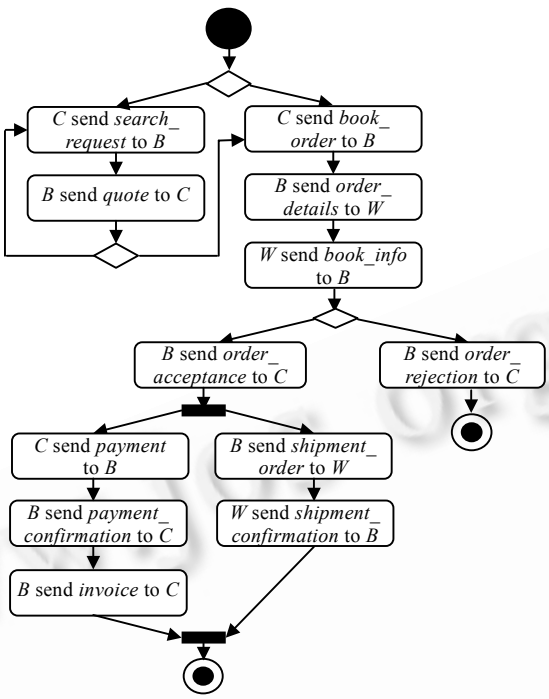


Fig.2 On-Line bookstore collaboration example

图 2 网上书店实例的协作流程

3 基于 GSCP 构建分布式协作流程

在本文的方法中,集中式的协作流程将转换为P2P的分布式协作流程,包含了各个角色的抽象服务接口.每个抽象服务接口定义了相应的角色需要在协作中所实现的交互模式.这一步是我们原有工作[18]的基础之上,进一步给出了转换规则的正确性验证方法.

定义 3(DSCP). 分布式协作流程是抽象服务接口的集合$DSCP=\{AP_i|1\le i\le n\}$,$n$是参与角色的数目.每个参与者所对应的抽象服务接口表示为$AP=(r,M_{R_AP},M_{S_AP},P_{AP})$.设$GSCP=(M,R,P_{CHOR})$,则有:

1) r 代表抽象服务接口对应的角色,$r\in R$;

2) $M_{R\_AP}\subseteq M$,包含了当前角色在协作中所收到的全部消息集合;

3) $M_{S\_AP}\subseteq M$,包含了当前角色在协作中所发送的全部消息集合;

4) P_{AP}定义了抽象服务接口的流程,描述了当前角色所应实现的交互事件序列.流程P_{AP}的符号集是事件$!m(r_s,r_d)$和$?m(r_s,r_d)$的集合,这里,"!"表示消息发送事件,而"?"表示消息接收事件,$m\in(M_{R\_AP}\cup M_{S\_AP})$.

分布式协作流程DSCP的行为定义为$P_{DC}=P_{AP1}||P_{AP2}||\ldots||P_{APn}$,$n$是角色的数目.在使用"||"连接两个流程$P$和$Q$时,消息的发送事件需要和相应的接收事件同步,即只有当P处于消息发送状态,而Q处于同一消息的接收状态时,两个流程执行一次消息通信动作.下面给出基于GSCP构建分布式协作流程的具体转换规则.

规则 1. 假设集中式协作流程为$GSCP=(M,R,P_{CHOR})$,$\forall AP=(r,M_{R_AP},M_{S_AP},P_{AP})$,对于$P_{CHOR}$中每一个$m(r_s,r_d)$,如果$r_s=r$,则$m\in M_{S_AP}$;如果$r_d=r$,则$m\in M_{R_AP}$.

规则 1 用来建立各个角色在协作中所交换的消息集合.

规则 2. GSCP 投影规则(见表 1).

Table 1 Projection rules on GSCP

表 1 GSCP 投影规则

	Type	Rule
Rule 2.1	Atomic event	$proj(SKIP,r)=SKIP$ $proj(m(r_s,r_d),r)=SKIP$ $(r\neq r_s\wedge r\neq r_d)$ $proj(m(r_s,r_d),r)=!m(r_s,r_d)$ $(r=r_s)$ $proj(m(r_s,r_d),r)=?m(r_s,r_d)$ $(r=r_d)$
Rule 2.2	$P;Q$	$proj(P;Q,r)=proj(P,r);$ $(A_{syn\_s}\|\|A_{syn\_r});$ $proj(Q,r)$, here $A_{syn\_s}=\|\|_{j\le n}!m_{syn}(r,r_j),$ $r_j\in Head(Q)$ (if $r\in Tail(P)$) $A_{syn\_s}=SKIP$ (else) $A_{syn\_r}=\|\|_{i\le l}?m_{syn}(r_i,r),$ $r_i\in Tail(P)$ (if $r\in Head(Q)$) $A_{syn\_r}=SKIP$ (else)
Rule 2.3	$m(r_s,r_d)\rightarrow P$	Similar to $P;Q$
Rule 2.4	$P\|\|Q$	$proj(P\|\|Q,r)=proj(P,r)\|\|proj(Q,r)$
Rule 2.5	$P\sqcap Q$	$proj(P\sqcap Q,r)=(\|\|_{i\le n}!m_P(r,r_i));$ $proj(P,r)\sqcap(\|\|_{i\le n}!m_Q(r,r_i));$ $proj(Q,r)$ $(r=r_0)$ $proj(P\sqcap Q,r)=?m_P(r_0,r)\rightarrow proj(P,r)\,\square\,?m_Q(r_0,r)\rightarrow proj(Q,r)$ $(r\neq r_0)$
Rule 2.6	$\mu X\cdot(P\sqcap Q;X)$	$proj(\mu X\cdot(P\sqcap Q;X),r)=\mu X\cdot(proj(P\sqcap Q;X,r))$

在规则 2.2 中,设$Head(Q)=\{r_j|1\le j\le n\}$,是流程$Q$中所有起始活动的消息发送角色的集合;$Tail(P)=\{r_i|1\le i\le l\}$,则包括了流程$P$中所有结束活动的消息接收角色.$A_{syn_s}$和$A_{syn_r}$为增加的顺序同步流程.在规则 2.5 中,假设决策角色为r_0,而$\{r_i|r_i\neq r_0\wedge 1\le i\le n\}$则为所有在流程$P$或$Q$中拥有事件的角色.$m_P(r_0,r_i)$和$m_Q(r_0,r_i)$均为构造的选择同步消息,从$r_0$发送到每一个$r_i$,用来通知$r_i$角色$r_0$所选择的分支,用来保证分布式协作流程中的选择一致性.

实际上,这些规则还可以进一步简化.例如规则 2.2,如果$r=r_j$或者$r=r_i$,则相应的同步事件可以省略;对于规则 2.5,在一个选择分支中,选择决策可以通过顺序同步事件来传递,则相应的一些选择同步事件可以省略.

投影后得到的抽象服务接口的并发组合必须与集中式的P_{CHOR}流程有相同的活性(liveness)和安全性(safety),由于集中式协作流程中包含非确定性选择分支,依据文献[17],P_{CHOR}与P_{DC}不仅需要是执行轨迹等价(trace-equivalent),还需要是失败等价(failure-equivalent).

定理 1. 设集中式协作流程为$GSCP=(M,R,P_{CHOR})$,通过规则 1 和规则 2 所得到的分布式协作流程为$DSCP=\{AP_i|1\le i\le n\}$,则有$failures(P_{CHOR})=failures(P_{DC}\backslash\alpha A_{syn})$.其中,$failures(P)=\{s,X|s\in traces(P)\wedge X\in refusals(P/s)\}$,$refusals(P)$ 是可能引起流程P死锁的事件集合;n是角色的数目;而αA_{syn}是规则 2 中所加入的所有同步事件的集合;"\"是隐藏操作符,隐藏αA_{syn}中定义的事件集.为了方便证明定理 1,我们首先给出引理 1.

引理 1. 设$P_{CHOR}=(P_{CHOR1};P_{CHOR2})$,$P_{DC}=\|\|_{i\le n}P_{AP_i}$,$n$是角色的数目,以及$P_{AP_i}=proj(P_{CHOR},r_i)$.那么在$P_{DC}$中,对一个给定角色$r$,有

$$\{a>>b|a\in\alpha(proj(P_{CHOR2},r)),b\in\alpha(proj(P_{CHOR1},r_i))\},(i=1,2,\ldots,n) \quad (1)$$

证明:根据投影规则,我们有$P_{AP_i}=proj(P_{CHOR1};P_{CHOR2},r_i)=proj(P_{CHOR1},r_i);(A_{syn_si}\|\|A_{syn_ri});proj(P_{CHOR2},r_i)$.

对于每个给定角色r:

1) 如果$(r_i=r)$,则显然我们有$\{a>>b|a\in\alpha(proj(P_{CHOR2},r)),b\in\alpha(proj(P_{CHOR1},r))\}$,即公式(1)成立.

2) 如果$(r_i\neq r)$,则有以下 4 种情况:

(a) $r\in Head(P_{CHOR2})\wedge r_i\in Tail(P_{CHOR1})$

根据A_{syn_r}和A_{syn_si}的定义,其中总有一个对应的同步消息发送与接收事件对,即存在一个$a_x\in\alpha(A_{syn_r})$和$b_x\in\alpha(A_{syn_si})$,有$a_x=b_x$.于是有$\{a>>a_x=b_x>>b|a\in\alpha(proj(P_{CHOR2},r)),b\in\alpha(proj(P_{CHOR1},r_i))\}$,即公式(1)仍然成立.

(b) $r\in Head(P_{CHOR2})\wedge r_i\notin Tail(P_{CHOR1})$,此时有:

对于r_i在P_{CHOR1}中的任意事件b_i,总存在至少一个事件b_x,其消息接收角色属于$Tail(P_{CHOR1})$,且有$b_i<<b_x$.

根据情况(a)以及"<<"的传递性,我们仍可以得到公式(1).

(c) $r\notin Head(P_{CHOR2})\wedge r_i\in Tail(P_{CHOR1})$,与情况(b)同理,我们可以得到公式(1).

(d) $r\notin Head(P_{CHOR2})\wedge r_i\notin Tail(P_{CHOR1})$,同时应用情况(b)和情况(c),我们仍可以得到公式(1). □

接下来我们给出定理 1 的证明.

证明:利用归纳法来证明.设有流程P_{CHOR},其投影后所得到的分布式协作流程为P_{DC},而αA_{syn}是P_{DC}中的同步事件集合.假设P_{CHOR}中所包含的操作符数目为j.

步骤 1. 当j=1 时,$P_{CHOR}=m(r_1,r_2)\rightarrow SKIP$.

对于r_1, $P_{AP_1}=!m(r_1,r_2)\rightarrow SKIP$,对于$r_2$, $P_{AP_2}=?m(r_1,r_2)\rightarrow SKIP$,则

$$P_{AP_1}\parallel P_{AP_2}=!m(r_1,r_2)\rightarrow SKIP\|?m(r_1,r_2)\rightarrow SKIP=m(r_1,r_2)\rightarrow SKIP.$$

同时有$\alpha A_{syn}=\{\}$,因此,$P_{DC}\backslash\alpha A_{syn}=(P_{AP_1}\parallel P_{AP_2})\backslash\alpha A_{syn}=P_{AP_1}\parallel P_{AP_2}$.很明显,我们有

$failures(P_{AP_1}\parallel P_{AP_2})=\{(\langle\rangle,\{\surd\}),(\langle m(r_s,r_d)\rangle,\{m(r_s,r_d)\}),(\langle m(r_s,r_d),\surd\rangle,\{X|X\subseteq\alpha(P_{AP_1}\parallel P_{AP_2})\})\}=failures(P_{CHOR})$.

步骤 2. 假设当$j\leq l(l>1)$时,上述结论仍然成立.

步骤 3. 当$j=l+1$ 时,设有流程P_{CHOR1}和P_{CHOR2},它们中的操作符数目之和为l,且P_{DC1},P_{DC2}分别为其投影后所得到的分布式协作流程,而αA_{syn1}和αA_{syn2}分别是P_{DC1}和P_{DC2}中的同步事件集合.

1) 如果$P_{CHOR}=(P_{CHOR1};P_{CHOR2})$,

$$failures(P_{CHOR})=\{(s,X)|(s,X\cup\{\surd\})\in failures(P_{CHOR1})\}\cup\{(s^\frown t,X)|s^\frown\{\surd\}\in traces(P_{CHOR1})\wedge(t,X)\in failures(P_{CHOR2})\},$$

$$P_{DC}\backslash\alpha A_{syn}=\{[proj(P_{CHOR1},r_1);proj(P_{CHOR2},r_1)]\|\ldots\|[proj(P_{CHOR1},r_n);proj(P_{CHOR2},r_n)]\}.$$

根据引理 1 的结论,我们可以得到:

$$\{[proj(P_{CHOR1},r_1);proj(P_{CHOR2},r_1)]\|\ldots\|[proj(P_{CHOR1},r_n);proj(P_{CHOR2},r_n)]\}=$$
$$\{[proj(P_{CHOR1},r_1)\|\ldots\|proj(P_{CHOR1},r_n)];[proj(P_{CHOR2},r_1)\|\ldots\|proj(P_{CHOR2},r_n)]\},$$

又$\alpha A_{syn}\supseteq\alpha A_{syn1}\cup\alpha A_{syn2}$,所以有$P_{DC}\backslash\alpha A_{syn}=(P_{DC1}\backslash\alpha A_{syn1});(P_{DC2}\backslash\alpha A_{syn2})$,则

$$failures(P_{DC}\backslash\alpha A_{syn})=\{(s,X)|(s,X\cup\{\surd\})\in failures(P_{DC1}\backslash\alpha A_{syn1})\}\cup$$
$$\{(s^\frown t,X)|s^\frown\{\surd\}\in traces(P_{DC1}\backslash\alpha A_{syn1})\wedge(t,X)\in failures(P_{DC2}\backslash\alpha A_{syn2})\}.$$

根据归纳假设,我们有$failures(P_{DC}\backslash\alpha A_{syn})=failures(P_{CHOR})$.

至于前缀操作符“→”可以看作是顺序操作符“;”的特例,因此具有相同的推导过程.

2) 如果$P_{CHOR}=(P_{CHOR1}\|P_{CHOR2})$,

$$failures(P_{CHOR})=\{(s,X\cup Y)|s\in(\alpha P_{CHOR1}\cup\alpha P_{CHOR2})^*\wedge(s\uparrow\alpha P_{CHOR1},X)\in failures(P_{CHOR1})\wedge$$
$$(s\uparrow\alpha P_{CHOR2},Y)\in failures(P_{CHOR2})\}.$$

这里,↑是执行轨迹的限制操作符,$s\uparrow A$ 即轨迹 s 中只取集合 A 所定义的事件.

$$P_{DC}=[proj(P_{CHOR1},r_1)\|proj(P_{CHOR2},r_1)]\|\ldots\|[proj(P_{CHOR1},r_n)\|proj(P_{CHOR2},r_n)]$$
$$=proj(P_{CHOR1},r_1)\|proj(P_{CHOR2},r_1)\|\ldots\|proj(P_{CHOR1},r_n)\|proj(P_{CHOR2},r_n)$$
$$=[proj(P_{CHOR1},r_1)\|\ldots\|proj(P_{CHOR1},r_n)]\|\ldots\|[proj(P_{CHOR2},r_1)\|proj(P_{CHOR2},r_n)]$$
$$=P_{DC1}\|P_{DC2},$$
$$failures(P_{DC}\backslash\alpha A_{syn})=failures((P_{DC1}\backslash\alpha A_{syn1})\|(P_{DC2}\backslash\alpha A_{syn2}))$$
$$=\{(s,X\cup Y)|s\in((\alpha P_{DC1}-\alpha A_{syn1})\cup(\alpha P_{DC2}-\alpha A_{syn2}))^*\wedge$$
$$(s\uparrow(\alpha P_{DC1}-\alpha A_{syn1}),X)\in failures(P_{DC1}\backslash\alpha A_{syn1})\wedge$$
$$(s\uparrow(\alpha P_{DC2}-\alpha A_{syn2}),Y)\in failures(P_{DC2}\backslash\alpha A_{syn2})\}.$$

因为$\alpha P_{CHOR1}=\alpha P_{DC1}-\alpha A_{syn1}$和$\alpha P_{CHOR2}=\alpha P_{DC2}-\alpha A_{syn2}$以及归纳假设,有$failures(P_{DC}\backslash\alpha A_{syn})=failures(P_{CHOR})$.

3) 如果$P_{CHOR}=(P_{CHOR1}\sqcap P_{CHOR2})$,

$$failures(P_{CHOR})=failures(P_{CHOR1})\cup failures(P_{CHOR2}).$$

这里,我们假设决策角色为r_1,则有

$$P_{DC}=[proj(P_{CHOR1}\sqcap P_{CHOR2},r_1)]\|\ldots\|[proj(P_{CHOR1}\sqcap P_{CHOR2},r_n)]$$
$$=[((\|_{1<i\leq n}!m_{CHOR1}(r_1,r_i));proj(P_{CHOR1},r_1))\sqcap((\|_{1<i\leq n}!m_{CHOR2}(r_1,r_i));proj(P_{CHOR2},r_1))]\|$$
$$\|_{1<i\leq n}[(?m_{CHOR1}(r_1,r_i))\rightarrow proj(P_{CHOR1},r_i))\Box(?m_{CHOR2}(r_1,r_i))\rightarrow proj(P_{CHOR2},r_i))]$$
$$=[(\|_{1<i\leq n}m_{CHOR1}(r_1,r_i));proj(P_{CHOR1},r_1)\|\ldots\|proj(P_{CHOR1},r_n)]\sqcap$$
$$[(\|_{1<i\leq n}m_{CHOR2}(r_1,r_i));proj(P_{CHOR2},r_1)\|\ldots\|proj(P_{CHOR2},r_n)],$$

则

$$P_{DC}\backslash\alpha A_{syn}=\{[proj(P_{CHOR1},r_1)\|\ldots\|proj(P_{CHOR1},r_n)]\sqcap[proj(P_{CHOR2},r_1)\|\ldots\|proj(P_{CHOR2},r_n)]\}\backslash\alpha A_{syn}$$

$$=(P_{DC1}\sqcap P_{DC2})\backslash\alpha A_{syn}=(P_{DC1}\backslash\alpha A_{syn1})\sqcap(P_{DC2}\backslash\alpha A_{syn2}).$$

因此,

$$failures(P_{DC}\backslash\alpha A_{syn})=failures(P_{DC1}\backslash\alpha A_{syn1})\cup failures(P_{DC2}\backslash\alpha A_{syn2})$$
$$=failures(P_{CHOR1})\cup failures(P_{CHOR2})=failures(P_{CHOR}).$$

$\mu X\cdot(P_{CHOR1}\sqcap(P_{CHOR2};X))$具有与上述同样的推理过程. □

定理 1 证明,通过转换规则 1 和转换规则 2 建立起来的分布式协作子流程拥有与集中式协作流程等价的行为.原有的一些工作流方面的角色投影方法[19,20]只是通过简单的去除无关的消息来实现投影,并不能保证得到的分布式流程和集中式流程具有相同的行为.我们基于消息传递来刻画活动节点,更符合服务计算环境的分布式交互的特点,而且投影规则保证了投影前后的流程具有失败等价性,从而更加严格和完备.

在我们的例子中,集中式协作流程分解为 3 个抽象服务接口,分别为 P_{AP}^{C}, P_{AP}^{B}, P_{AP}^{W}.图 3 给出了书店角色的抽象接口 P_{AP}^{B},虚线方框表示添加的同步事件.其 CSP 描述如下:

$$P_{AP}^{B}=\mu X\cdot[(?Search\_Request(C,B)\rightarrow!Quote(B,C)\rightarrow X)\Box(?Book\_Order(C,B)\rightarrow!Order\_Details(B,W)\rightarrow$$
$$?Book\_info(W,B)\rightarrow[(!Order\_Acceptance(B,C)\rightarrow[(?Payment(C,B)\rightarrow!Payment\_Confirmation(B,C)\rightarrow$$
$$?m_{pa}(C,B)\rightarrow!Invoice(B,C)\rightarrow SKIP)\|(?m_{ac}(C,B)\rightarrow!Shipment\_Order(B,W)\rightarrow$$
$$?Shipment\_Confirmation(W,B)\rightarrow SKIP)])\sqcap(!m_{re}(B,W)\rightarrow!Order\_Rejection(B,C)\rightarrow SKIP)]\rightarrow SKIP)].$$

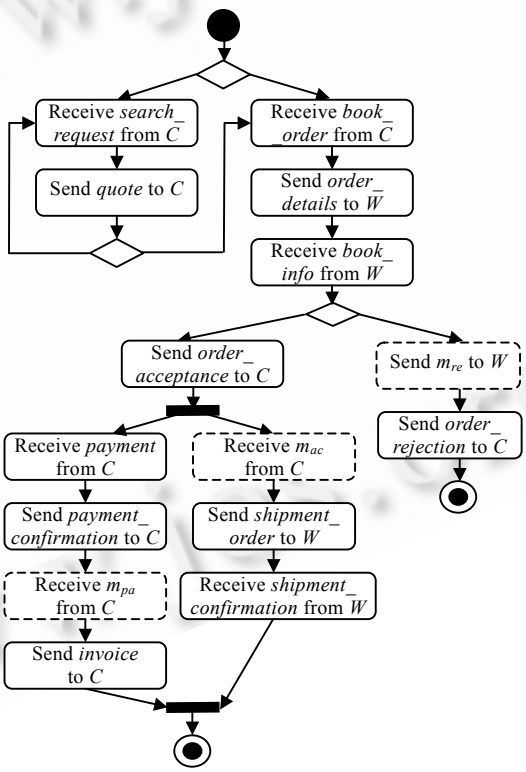


Fig.3 Abstract service interface of bookstore

图 3 网上书店的抽象服务接口

4 基于适配实现服务的协作

4.1 服务接口的兼容性

在建立了分布式服务协作流程之后,需要对 DSCP 和实际服务的行为接口进行兼容性检查,以保证正确的

服务协作关系.这里,我们主要考虑协议兼容性.首先给出服务组件接口的定义.

定义 4(SI). 服务组件接口是一个三元组$SI=(M_{R_SI},M_{S_SI},P_{SI})$,其中:

1) M_{R_SI}是服务所有的接收消息集合,M_{S_SI}是服务所有的发送消息集合;

2) P_{SI}是服务组件的接口协议.P_{SI}的符号集是事件!m和?m的集合,$m\in(M_{R\_SI}\cup M_{S\_SI})$.

在给出服务接口的兼容性定义之前,首先定义抽象服务接口P_{AP}的对偶流程,即:对P_{AP}进行转向变换,包括接收操作改为发送操作,反之亦然(同时,内部选择符Π和外部选择符$\square$也需要互相转换);忽略抽象流程中的角色信息,与其他角色之间的同步事件集合(αA_{syn})则看作是内部事件并加以隐藏.上述操作的目的是使抽象服务接口和实际服务形成交互的双方.这样所得到的P_{AP}的对偶流程定义为P_{OA}.

定义 5(严格协议兼容). 给定抽象服务接口P_{AP}的对偶流程P_{OA},如果$P_{OA}\|P_{SI}$不存在死锁,那么我们认为,P_{OA}和实际服务接口P_{SI}是严格协议兼容的.

接口协议的失配则主要包括两种类型[21−23]:未声明的消息接收(unspecified reception)和死锁(deadlock).前者是指一方发送消息而另一方并未准备接收该消息;后者则指交互的双方或任意一方停止于非终止状态.并发组合$P_{OA}\|P_{SI}$可以用来检查上述提到的两种类型的失配,是因为在CSP描述的流程中,接收事件和发送事件需要同步,即一方处于发送消息状态,另一方必须同时处于接收该消息的状态.这样,无论是一方发送消息但另一方未准备接收该消息,还是任何一方未能成功结束,并发组合$P_{OA}\|P_{SI}$都会进入死锁状态.

4.2 适配机制的重要性

企业所提供的服务会应用在不同的协作场景中,或者通过相互交互实现企业间的协作,或者以组合增值服务形式提供给客户,很难保证其服务在不同的协作环境中都能满足严格的兼容性.而且,由于服务的自治性及其环境变化的动态性,它们会演化并改变其接口协议.这样,最初的协作流程就会失效.当发生变化时,通过改变协作流程定义或者替换服务组件实现新的协作关系,不仅代价高而且效率低下.实际上,引起交互失配的协议差异可以通过适配机制来弥补,其引入不仅能够提高服务组件的可用率,而且还可以提高服务协作的灵活性.

定义 6(可适配协议兼容). 如果存在流程$P_{Mediator}$使得$P_{OA}\|P_{Mediator}\|P_{SI}$不存在死锁,那么我们认为,$P_{OA}$和实际服务接口$P_{SI}$是可适配协议兼容的.

引起协议失配的主要原因是多方面的[24],包括顺序失配、冗余消息、缺失消息、消息合成与分解失配等等.在针对组件以及服务的适配研究方法中,文献[24,25]只是分析了服务接口之间的协议失配形式.文献[22]的重点在于适配过程和系统实现,对于适配检查和适配器的自动构造并没有给出严格的分析和论证.文献[21]使用有限状态机对组件接口协议进行建模,其局限性在于描述方法的表达能力不足,例如,无法表述确定性选择、非确定性选择、流程的并发组合等等,因此需要额外的映射关系描述来完成适配器的构建.而且,该方法基于状态集合进行多步的迭代化简求解,其算法复杂度为$O(n\times m)$,其中,n和m分别为两个协议的状态数目.而我们通过CSP的推演规则对交互协议进行死锁检查,其复杂度可为$O(n+m)$.文献[26]使用Pi-Calculus对接口协议建模,具有相当的表达力,然而该方法通过逐步添加活动的方式来实现适配器的构造,在每一步都需要对已有交互结构进行一次死锁检查,算法依然复杂而且效率很低.我们的方法采用CSP这种表达力更强、更有效的形式化方法,提高协议和适配器的描述能力,从而无须附加额外的映射关系描述.同时,建立交互协议的异步通信机制来检查和构建适配器,只需对整体交互结构作一次死锁检查,方法更有效且易于实现.此外,算法在兼容性检查的基础上能够计算出所有不满足可适配兼容性的交互路径,反馈给开发者以实现有效的调整.

我们通过建立P_{OA}和P_{SI}之间的协调模型来实现可适配的协议兼容性检查和适配器的自动生成.

4.3 建立协调模型

P_{OA}和P_{SI}之间的协调模型从两个方面来刻画:描述静态依赖关系的数据映射以及描述其动态行为的异步交互模型.下面分别给出其定义.

数据映射通过P_{OA}和P_{SI}之间的数据依赖关系来建立,即一方所需要接收的消息如何通过另一方的发送消息来构造.

定义 7. 数据映射包含两类集合:$Map_{\langle SI,OA\rangle}$和$Map_{\langle OA,SI\rangle}$.其中,$Map_{\langle SI,OA\rangle}=\{synth_i(m_r,M_S)|m_r\in M_{R\_OA},M_S\subseteq M_{S\_SI},1\le i\le n\}$,给出了从$M_{S_SI}$到$M_{R_OA}$的映射规则集合.$M_S$是与$m_r$有依赖关系的发送消息集合,*synth*函数给出了基于M_S来构造消息m_r的方法.

同样地,$Map_{\langle OA,SI\rangle}$是从$M_{S_OA}$到$M_{R_SI}$的映射规则集合.

图 4 给出了实际的网上书店服务组件的接口.这里,我们假设在协作目标流程中,用户最初发送的 *book_order* 消息包括了 *order_info* 以及 *shipment_info*,而实际服务起初只接收 *order_info* 消息,而在付款成功后再接收送货消息 *shipment_info*.从图 3 和图 4 中可以看出,抽象流程与实际服务之间的失配主要包括:冗余的消息(*shipment_ confirmation*)、缺失的消息(*payment_confirmation*)、消息分解(抽象协议的 *book_order* 包含了实际服务的 *order_ info* 和 *shipment_info*)以及顺序失配(*payment* 和 *shipment*).表 2 给出了网上书店的抽象接口与实际服务之间的数据依赖关系.

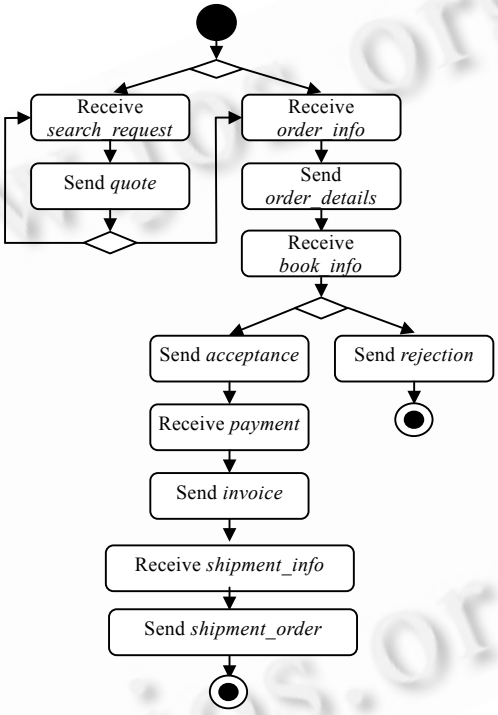


Fig.4 Concrete service interface of bookstore

图 4 网上书店服务组件接口

Table 2 Data dependencies of bookstore

表 2 网上书店的数据依赖关系

Abstract interface messages		Concrete service messages
search_request	←	*search_request*
quote	←	*quote*
book_order	←	*order_info*, *shipment_info*
order_details	←	*order_details*
book_info	←	*book_info*
order_acceptance	←	*acceptance*
order_rejection	←	*rejection*
payment	←	*payment*
shipment_order	←	*shipment_order*
shipment_confirmation	←	*NULL*
payment_confirmation	←	*NULL*
invoice	←	*invoice*

我们根据数据依赖关系构造数据映射.例如,实际服务的 *order_info* 和 *shipment_info* 通过抽取抽象接口中 *book_order* 消息的相应部分得到,对于缺少的 *payment_confirmation* 消息,则通过构造一个常量的消息来实现.

采用适配机制时,适配器在交互双方之间起到数据缓存、抽取、转换以及协调消息交互顺序的作用.适配器可以接收任何一方发送的消息,并且在需要的时候将消息发送到相应的接收方.实际上,适配器的存在使得同步的交互方式异步化了.因此,我们将建立P_{OA}和P_{SI}的异步交互模式来分析协调模型的动态行为.下面给出P_{OA}和P_{SI}的异步交互模型.

定义 8. P_{OA}和P_{SI}之间的异步交互模型为$(\|_{i\le n}Pipe_m_i)//[P_{OA_AC}\|P_{SI_AC}]$.

1) $Pipe_m_i$是消息通道,用来存储双方协议所交互的数据.对于数据映射中$(synth(m_r,M_S))$的每个接收消息m_r,都存在一个对应的消息通道$Pipe_m_r$,其行为描述如下:

$$Pipe\_m_r=[(\|_{i\le n}left?m_{si})\rightarrow synth\rightarrow right!m_r\rightarrow SKIP]\Box[close\rightarrow SKIP].$$

这里,$m_{si}\in M_S$,n是m_r所依赖的发送消息的数目.消息通道用来缓存消息,它通过自己的*left*信道接收消息,并通过其*right*信道发送消息.*synth*是数据映射定义中的*synth*函数的相应实现.因为相对应的消息可能存在于选择分支中,因此在一次具体的交互过程中可能并不接收消息(选择了其他分支),$[close\rightarrow SKIP]$的含义就是在交互过程全部完成时关闭消息通道.如果消息存在于循环结构中,则其相应的行为描述为

$$Pipe\_m_r=\mu X\cdot([(\|_{i\le n}left?m_{si})\rightarrow synth\rightarrow right!m_r\rightarrow X]\Box[close\rightarrow SKIP]).$$

2) P_{OA_AC}和P_{SI_AC}通过将P_{OA}和P_{SI}中的事件进行如下替换来构造:

- $\forall !m\in\alpha P_{OA}$(或$\alpha P_{SI}$),若$m$在数据映射中有相应的映射关系定义,则$!m\Rightarrow(\|_{i\le n}Pipe_m_i.left!m)$,$n$是所有依赖于$m$的消息数目;否则,$!m\Rightarrow SKIP$;
- $\forall ?m\in\alpha P_{OA}$(或$\alpha P_{SI}$),若$m$在数据映射中有相应的映射关系定义,则$?m\Rightarrow Pipe_m.right?m$;否则,$?m\Rightarrow STOP$.

3) 这里,符号"//"是从属操作符."//"符号左边部分的作用如同$P_{OA_AC}\|P_{SI_AC}$的附属流程,附属流程中的事件只有在主流程$(P_{OA_AC}\|P_{SI_AC})$允许其发生的时候才可以发生.

4.4 基于协调模型的可适配兼容性检查方法

接下来我们将证明,当$(\|_{i\le n}Pipe_m_i)//[P_{OA_AC}\|P_{SI_AC}]$没有死锁时,$P_{OA}$和$P_{SI}$将满足可适配的协议兼容.

定理 2. 当且仅当$(\|_{i\le n}Pipe_m_i)//[P_{OA_AC}\|P_{SI_AC}]$不存在死锁时,$P_{OA}$和$P_{SI}$是可适配协议兼容的.

设R_P_{OA}和R_P_{SI}分别是P_{OA}和P_{SI}的对偶流程,首先将P_{OA_AC}与R_P_{OA}合并,P_{SI_AC}与R_P_{SI}合并,其方法是:

对于P_{OA}(或P_{SI})中的每个事件a,R_P_{OA}(或R_P_{SI})中相应的事件为a',假设在P_{OA_AC}(或P_{SI_AC})中对应转换事件为a''.我们将使用操作符"$\rightarrow$"来连接a'和a''(若a'为接收事件,有$a'\rightarrow a''$;否则有$a''\rightarrow a'$).合并后的新协议分别定义为$P_{OA\cup AC}$和$P_{SI\cup AC}$.

证明:充分条件.对于按照上述方法构造的$P_{OA\cup AC}$和$P_{SI\cup AC}$,我们有$\alpha P_{OA}=\alpha R_P_{OA}$,$\alpha P_{SI}=\alpha R_P_{SI}$以及$\alpha P_{OA\cup AC}=\alpha R_P_{OA}\cup\alpha P_{OA_AC}$,$\alpha P_{SI\cup AC}=\alpha R_P_{SI}\cup\alpha P_{SI_AC}$.

设$P=P_{OA}\|(\|_{i\le n}Pipe_m_i)//[P_{OA\cup AC}\|P_{SI\cup AC}]\|P_{SI}=(\|_{i\le n}Pipe_m_i)//[P_{OA}\|P_{OA\cup AC}\|P_{SI\cup AC}\|P_{SI}]$(‖满足结合率),则$traces(P)\uparrow\alpha P_{OA}=traces(P_{OA}\|R_P_{OA})$.同样地,还有$traces(P)\uparrow\alpha P_{SI}=traces(P_{SI}\|R_P_{SI})$和$traces(P)\uparrow(\alpha P_{OA_AC}\cup\alpha P_{SI_AC})=traces((\|_{i\le n}Pipe_m_i)//[P_{OA_AC}\|P_{SI_AC}])$.

因为$(\|_{i\le n}Pipe_m_i)//[P_{OA_AC}\|P_{SI_AC}]$,$P_{OA}\|R_P_{OA}$和$P_{SI}\|R_P_{SI}$都不存在死锁.

另外,由于$\alpha P=\alpha P_{OA}\cup\alpha P_{SI}\cup\alpha P_{OA_AC}\cup\alpha P_{SI_AC}$,因此,$P$的执行轨迹中不存在死锁(*STOP*)事件,即$(\|_{i\le n}Pipe_m_i)//[P_{OA\cup AC}\|P_{SI\cup AC}]$是$P_{OA}$和$P_{SI}$的有效协调流程.

必要条件.我们将$P_{OA\cup AC}$和$P_{SI\cup AC}$转换成等价的非确定性选择的标准形式,即$P_{OA\cup AC}=P_{o1}\Pi P_{o2}\Pi\ldots\Pi P_{on}$,$P_{SI\cup AC}=P_{s1}\Pi P_{s2}\Pi\ldots\Pi P_{sl}$,$P_{o1}$,$P_{o2}$(或$P_{s1}$,$P_{s2}$)等称作流程$P_{OA\cup AC}$(或$P_{SI\cup AC}$)的分支子流程,则

$$P_{OA\cup AC}\|P_{SI\cup AC}=(P_{o1}\|P_{s1})\Pi(P_{o2}\|P_{s1})\Pi(P_{o1}\|P_{s2})\Pi(P_{o2}\|P_{s2})\Pi\ldots\Pi(P_{on}\|P_{sl}),$$

其中,$(P_{o1}\|P_{s1})$,$(P_{o2}\|P_{s1})$等是$P_{OA\cup AC}$和$P_{SI\cup AC}$之间的一条实际的交互路径,其集合则构成了$P_{OA\cup AC}$和$P_{SI\cup AC}$之间完整的交互路径集合.

我们可以看到,R_P_{OA}和R_P_{SI}是P_{OA}和P_{SI}的对偶流程,与P_{OA}和P_{SI}具有相同的结构而且可以无缝交互.因

此,$P_{OA\cup AC}$和$P_{SI\cup AC}$的并发组合结构实际上包含了P_{OA}和P_{SI}的所有交互路径的完整集合,因此,如果$(\|_{i\le n}Pipe_m_i)//[P_{OA_AC}\|P_{SI_AC}]$中存在死锁,相应地,在$(\|_{i\le n}Pipe_m_i)//[P_{OA\cup AC}\|P_{SI\cup AC}]$的某条执行轨迹中存在*STOP*事件,则表明$P_{OA}$和$P_{SI}$无法通过协调的方式实现正确的交互,即不满足可适配的协议兼容性. □

下面我们给出具体的可适配兼容性检查算法.按照定理 2 证明中的方法,我们将P_{OA_AC}和P_{SI_AC}转换成等价的非确定性选择的标准形式,然后我们对P_{OA_AC}和P_{SI_AC}的并发组合结构中的每一条交互路径进行检验.如果所有交互路径的检验结果都不存在死锁,P_{OA_AC}和P_{SI_AC}的并发组合就是无死锁的.

算法 1. 可适配检查算法.

输入:P_{OA_AC}的分支子流程p_1和P_{SI_AC}的分支子协议p_2,消息通道$(\|_{i\le n}Pipe_m_i)$;

输出:Boolean *freeOfDeadLock*.

```
1.  while (true) do
2.    result_p1, p1' ←move(p1),result_p2, p2' ←move(p2)
3.    if (result_p1=STOP∨result_p2=STOP) return false end if
4.    if (result_p1=NoMove∧result_p2=NoMove) return false end if
5.    if (result_p1=SKIP∧result_p2=SKIP) return true end if
6.    p1:= p1' ,p2:= p2'
7.  end while
8.  procedure result_p,p'          move(p)
9.    q:=firstSubProtocol(p)
10.   if (isSequence(q))           hasMoved=false;
11.     while empty(q)=false do
12.       result_q,q'←move(q)
13.       if (result_q=SKIP)       p:=removeHead(p,q),q:=firstSubProtocol(p),hasMoved:=true
14.       else if result_q=STOP return (STOP,union(q',p,";"))
15.       else if (hasMoved=true∨result_q=Move) return (Move,union(q',p,";"))
16.       else return (NoMove,union(q',p,";"))
17.       end if
18.     end while
19.     return (SKIP,NULL)
20.   else if isParalle(q)
21.     for all subprotocol qi do
22.       result^i:=move(qi)
23.     end for
24.     if all result^i=SKIP return (SKIP,NULL)
25.       else if exist result^i=STOP return (STOP,union(qi,"||"))
26.       else if exist result^i=Move return (Move,union(qi,"||"))
27.       else return (NoMove,p)
28.     end if
29.   else if isExternalChoice(q)
30.     if exist qi isChoosed(qi)=true return move(qi) else return (NoMove,p) end if
31.   else if isSendActivity(q)
32.     if canSend(q)
33.       p:=removeHead(p,q),      result,p'←move(p)
```

34. **if** *result*=*NoMove* **return** (*Move*,*p*) **else return** (*result*,*p*′) **end if**
35. **else return** (*NoMove*,*p*)
36. **end if**
37. **else if** *isReceiveActivity*(*q*)
38. **if** *canReceive*(*q*) *p*:=*removeHead*(*p*,*q*), *result*,*p*′←***move***(*p*)
39. **if** *result*=*NoMove* **return** (*Move*,*p*) **else return** (*result*,*p*′) **end if**
40. **else return** (*NoMove*,*p*)
41. **end if**
42. **else if** *q*=*SKIP* **return** (*SKIP*,*NULL*)
43. **else if** *q*=*STOP* **return** (*STOP*,*NULL*)
44. **end if**
45. **end procedure**

在该算法中,输入为两个协议的选择分支子协议.算法的基本思想是,函数*move*用来模拟协议的执行,返回结果*p*′是当前执行到的事件之后的流程.而*result*则有 4 种类型:*SKIP*,*STOP*,*NoMove*和*Move*.这样,如果两个子协议都返回*SKIP*,说明两者都已成功执行完毕,则其交互不存在死锁;如果任何一方返回*STOP*,或者两者都无法再继续执行(返回*NoMove*),则其交互存在死锁.函数*canReceive*和*canSend*分别用来检查相应的消息通道的状态是否可以写入数据以及是否可以读取数据.一次检验的时间复杂度为$O(m+n)$,m和n分别是两个子协议的事件数目.假设P_{OA_AC}和P_{SI_AC}的子协议数目分别为s_1和s_2,则整个算法的时间复杂度为$O(s_1\times s_2\times(n+m))$.因为协议的分支数目相对于状态数目而言很小,因此算法复杂度可以表示为$O(n+m)$.

在实现检查算法的过程中,每一次的匹配不成功(算法返回 false),表明了在当前的交互路径中存在不兼容的情况.我们可以将所有不兼容的交互路径返回给开发者,指明在哪些情况下无法基于适配进行正确的交互.开发者以此为参照,通过对产生失配的路径以及失配节点的分析,对协作流程或者实际服务进行调整,这样更加有效地利用了算法的结果.

4.5 适配器的自动生成

根据定理2的证明结果,我们将基于$(\|_{i\le n}Pipe_m_i)//[P_{OA\cup AC}\|P_{SI\cup AC}]$实现适配器自动构造算法.可以看到,适配器主要包含了 3 个部分:$P_{OA\cup AC}$中的R_P_{OA}部分实现与其他的适配器的通信,P_{OA_AC}基于消息信道*Pipe*实现消息的存取.$P_{SI\cup AC}$则实现与实际服务的交互,$(\|_{i\le n}Pipe_m_i)$则实现消息的存储、抽取、转换.适配器接收一方的数据在消息通道中缓存,并转换成另一方所需要的消息格式,并在可以正确接收的状态下发送给对方.适配器主要通过这种数据的缓存与转换机制来弥合之间的交互差异.

因此,适配器自动生成算法的主要思路是合并P_{OA_AC}与R_P_{OA}以及P_{SI_AC}和R_P_{SI}.这里,我们省略具体的算法描述.将该算法应用到本文的例子中,得到网上书店的适配器协议的$P_{OA\cup AC}$和$P_{SI\cup AC}$,部分如下:

$$P_{OA\cup AC}=\mu X\cdot[(?S\_R\rightarrow P_{SR}.left!(S\_R)\rightarrow P_{Qu}.right?(Qu)\rightarrow!Qu\rightarrow X)\Box(?B\_O\rightarrow(P_{OI}.left!(B\_O)\|P_{SI}.left!(B\_O))\rightarrow P_{OD}.right?(O\_D)\rightarrow!O\_D\rightarrow?B\_I\rightarrow P_{BI}.left!(B\_I)\rightarrow[(P_{Ac}.right?(Ac)\rightarrow!O\_A\rightarrow[(?Pa\rightarrow P_{Pa}.left!(Pa)\rightarrow P_{PC}.right?(P\_C)\rightarrow!P\_C\rightarrow P_{In}.right?(In)\rightarrow!In\rightarrow SKIP)\|(P_{SO}.right?(S\_O)\rightarrow!S\_O\rightarrow?S\_C\rightarrow SKIP)])\Box(P_{Re}.right?(Re)\rightarrow!O\_R\rightarrow SKIP)]\rightarrow SKIP)],$$

$$P_{SI\cup AC}=\mu X\cdot[(P_{SR}.right?(S\_R)\rightarrow!S\_R\rightarrow?Qu\rightarrow P_{Qu}.left!(Qu)\rightarrow X)\Box(P_{OI}.right?(O\_I)\rightarrow!O\_I\rightarrow?O\_D\rightarrow P_{OD}.left!(O\_D\rightarrow P_{BI}.right?(B\_I)\rightarrow!B\_I\rightarrow[(?Ac\rightarrow P_{Ac}.left!(Ac)\rightarrow P_{Pa}.right?(Pa)\rightarrow!Pa\rightarrow?In\rightarrow P_{PC}.left!(P\_C)\rightarrow P_{PC}.left!(In)\rightarrow P_{SI}.right?(S\_I)\rightarrow!S\_I\rightarrow?S\_O\rightarrow P_{SO}.left!(S\_O)\rightarrow SKIP)\Box(?Re\rightarrow P_{Re}.left!(Re)\rightarrow SKIP)]\rightarrow SKIP)].$$

我们以上述协议为例来具体描述适配器是如何起到协调作用的.

例如,协议$P_{OA\cup AC}$中的$[?Pa\rightarrow P_{Pa}.left!(Pa)\rightarrow\ldots\|P_{SO}.right?(S_O)\rightarrow!S_O\rightarrow\ldots]$部分以并发的方式与其他适配器通信来处理*shipment*和*payment*消息,而相应的$P_{SI\cup AC}$中的$[P_{Pa}.right?(Pa)\rightarrow!Pa\rightarrow\ldots?S_O\rightarrow P_{SO}.left!(S_O)]$部分

则顺序地与实际服务进行*shipment*和*payment*的消息交换.

5 原型实现及实例验证

根据本文所提出的基于分布式协调模型的服务协作方法,我们进行了原型系统的实现,并针对实例进行了验证.原型系统架构如图 5 所示,包括了开发环境部分(collaboration management)和相应的协作参与者所应实现的部分.其中,开发环境部分主要基于中国科学院软件研究所开发的业务流程设计平台 OnceBPD 来实现.OnceBPD 基于 Eclipse 框架,提供面向服务的业务流程的建模、模型转换、流程部署与管理的功能.我们扩展了 OnceBPD 的相关功能模块来支持分布式协调的服务协作方法.我们还开发了协作流程执行系统 OnceBPEL 以支持基于 BPEL 语言的业务流程的解析与执行.在我们的实例验证中,OnceBPEL 系统作为参与协作的服务提供者.

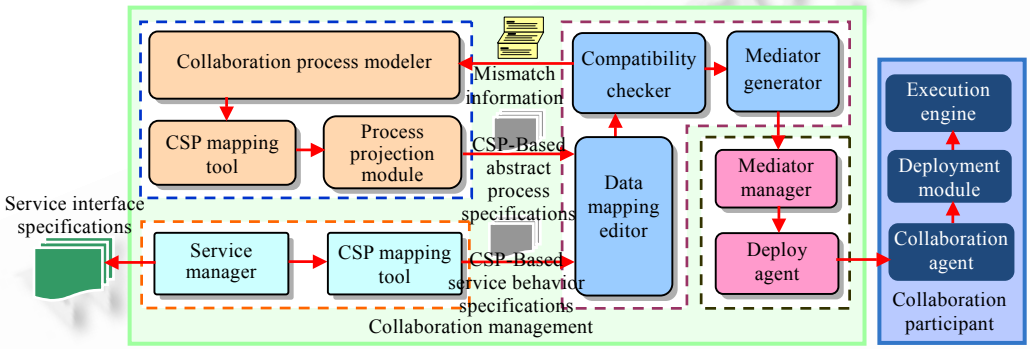


Fig.5 Architecture of prototype implementation

图 5 原型实现的系统架构

协作管理(collaboration management)中的 Collaboration Process Modeler 主要实现复杂业务流程的图形化建模与管理.CSP Mapping Tool 将业务流程模型转换为 CSP 描述,Process Projection Module 基于角色投影将集中式业务流程转换为分布式协作流程.

Service Manager 负责管理参与协作的服务,服务的行为接口通过 BPEL 抽象流程来描述.BPEL 抽象流程可以有效地描述服务交互协议而且不展现内部细节.BPEL 抽象流程将通过 CSP Mapping Tool 转换成为 CSP 描述形式,其转换规则见表 3.

Table 3 Correspondence between BPEL activities and CSP

表 3 BPEL 活动与 CSP 的映射规则

BPEL	⟨receive *varabile*="*x*"…/⟩		⟨reply *varabile*="*x*"…/⟩	⟨invoke *inputvariable*="*x*" *outputvariable*="*y*"…/⟩		⟨empty…/⟩	⟨terminate…/⟩
CSP	$?x$		$!x$	$!x \rightarrow ?y$		*SKIP*	*STOP*
BPEL	⟨sequence⟩ *activity* P_1 … *activity* P_n ⟨sequence/⟩	⟨flow…⟩ *activity* P_1 … *activity* P_n ⟨flow/⟩	⟨pick…⟩ ⟨*onMessage variable*="x_1"…⟩ *activity* P_1 ⟨*onMessage*/⟩ … ⟨*onMessage variable*="x_n"…⟩ *activity* P_n ⟨*onMessage*/⟩ ⟨pick/⟩		⟨switch…⟩ ⟨case…⟩ *activity* P_1 ⟨case…/⟩ … ⟨case…⟩ *activity* P_n ⟨case…/⟩ ⟨switch/⟩	⟨while…⟩ *activity P* ⟨while/⟩	
CSP	$P_1;\ldots;P_n$	$P_1 \| \| \ldots \| \| P_n$	$(?x_1 \rightarrow P_1) \Box \ldots \Box (?x_n \rightarrow P_n)$		$P_1 \sqcap \ldots \sqcap P_n$	$\mu X \cdot (P; X \sqcap SKIP)$	

Data Mapping Editor 用来可视化地编辑消息之间的映射规则.*synth* 函数通过 XSLT 语言来实现数据的抽取与转换.Compatibility Checker 处理抽象流程与实际服务接口的兼容性检查.根据检查的结果,或者将无法适配的路径信息返回到 Collaboration Process Modeler,或者由 Mediator Generator 模块自动地构造适配器.

Mediator Manager管理生成的适配器,并将其转换为相应的实现形式(如可执行BPEL流程、Java类等).在我们的实验系统中,OnceBPEL系统作为服务的提供者,因此,适配器将以可执行BPEL流程的形式存在($P_{OA \cup AC} \|$

$P_{SI\cup AC}$部分将实现为*flow*结构的两个并发活动).Deploy Agent实现适配器的分布式部署,通过与服务参与者的执行引擎通信,将适配器部署到相对应服务的执行环境中.

服务参与者的执行引擎需要提供相应的部署接口,或者如我们图中的 Collaboration Paticipant 部分所示那样提供 Deploy Agent 服务,以与服务协作管理环境通信,接收适配器的信息并实现部署.

我们通过本文中网上书店的例子来展示系统应用的具体流程.在开发阶段,首先在 Collaboration Process Modeler 中实现协作流程的建模,如图 6 所示.

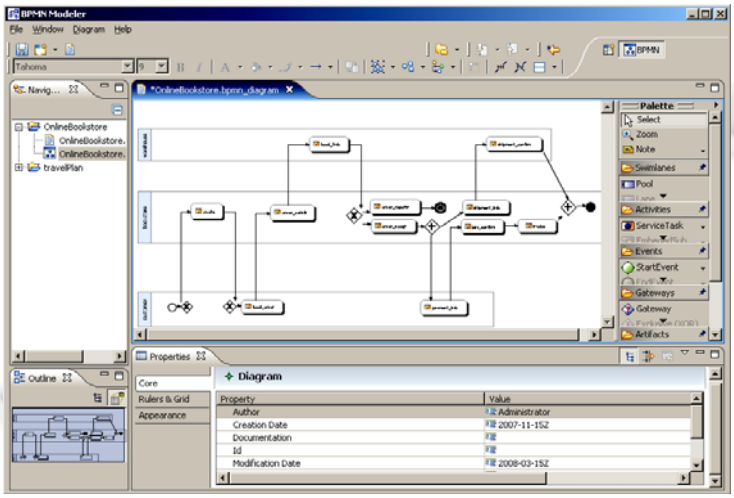


Fig.6 Global collaboration process of on-line bookstore

图 6 网上书店的集中式协作流程建模

集中式协作流程将依据角色信息自动地转换为分布式协作流程,相应部分的 CSP 描述如本文中各部分所列(省略了具体的消息内容).抽象流程与实际服务的数据映射规则通过 Data Mapping Editor 进行可视化的编辑.接下来,Compatibility Checker 将检查是否存在有效的适配器.在例子中,书店的实际服务可以通过适配的方式来实现协作,因此,Mediator Manager 依据服务接口信息将适配器转换为相应的 BPEL 流程.我们在图 7 中给出了书店服务对应适配器的 BPEL 流程片段.该片段展示了如何从 *book_order* 中抽取 *order_info* 消息,并实现实际服务的调用.从图中还可以看出,Pipes 实现为 *flow* 的子活动,它们与主流程的交互通过 *link* 元素来实现.

```
<process name="Bookstore_Mediator" targetNamespace="urn:samples:Onlinebookstore"
        ...
        xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/">
  <flow name="MainFlow">
    <sequence name="AbstractProcessSequence">
      ...
      <receive name="ReceiveBookOrder" partnerLink="bookstoreFrontEnd"
               portType="tns:bookstoreServicePT" operation="bookorder"
               variable="bookOrderVar">
        <source linkName="bookorder-to-pipes"/>
      </receive/>
      ...
    </sequence/>
    <sequence name="ConcreteServiceSequence">
      <invoke name="invokeorderBook" partnerLink="support:BookstoreServicePLT"
              portType="support:BookstoreServicePT" operation="orderBook"
              inputVariable="orderInfoVar">
        <target linkName="pipes-to-orderinfo"/>
      </invoke>
      ...
    </sequence/>
    <flow name="pipesFlow">
      <sequence name="bookorderPipe">
        <target linkName="bookorder-to-pipes"/>
        <source linkName="pipes-to-orderinfo"/>
        ...
        <assign>
          <copy>
            <from variable="bookOrderVar" part="orderInfo"/>
            <to variable="orderInfoVar"/>
          </copy>
          ...
        </assign>
      </sequence/>
    </flow/>
  </flow>
</process>
```

Fig.7 BPEL process of bookstore mediator

图 7 书店服务对应适配器的 BPEL 流程

在运行时,适配器与相对应的实际服务交互来进行应用数据的交换,而协作流程则通过适配器之间的分布式交互来实现.

6 结束语

本文提出了一种自上而下的实现企业间服务协作的解决方案.首先建立集中的服务协作模型,并基于角色转换为分布式的协作子流程集合.转换后得到的各个角色所对应的抽象流程,基于协调模型实现可适配的服务接口兼容性检查,并给出了抽象流程和实际服务接口之间适配器的自动生成算法.我们提出的方法在保持了服务自治特性的基础上提高了企业间服务协作的灵活性.

我们将在以下几个方面展开进一步的工作:首先会研究在运行时,当可用服务或者服务协作目标流程发生变化时的动态调整与管理机制,即如何响应这些动态变化,并调整处于运行状态的协作实例,使其由原有的协作结构迁移到新的协作结构.此外,如何利用自动模式匹配(schema matching)技术以提高建立协调模型的自动化程度,这是我们需要考虑的另一个方向.

References:

[1] Papazoglou MP, Georgakopoulos D. Service oriented computing. Communications of the ACM, 2003,46(10):24−28.

[2] Tsalgatidou A, Pilioura T. An overview of standards and related technology in Web services. Distributed and Parallel Databases, 2002,12(2-3):135−162.

[3] W3C. Simple object access protocol (SOAP). 2000. http://www.w3.org/2000/xp/Group

[4] W3C. Web services description language (WSDL). 2002. http://www.w3.org/2002/ws/desc

[5] OASIS. Universal description, discovery, and integration (UDDI). http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm

[6] Benatallah B, Medjahed B, Bouguettaya A, Elmagarmid A, Beard J. Composing and maintaining Web-based virtual enterprises. In: Proc. of the 1st Workshop on Technologies for E-Services (TES 2000). 2000. 155−174.

[7] Christophides V, Hull R, Karvounarakis G, Kumar A, Tong G, Xiong M. Beyond discrete e-services: Composing session-oriented services in telecommunications. In: Casati F, Georgakopoulos D, Shan MC, eds. Proc. of the 2nd Workshop on Technologies for E-Services (TES 2001). LNCS 2193, Berlin, Heidelberg: Springer-Verlag, 2001. 58−73.

[8] Casati F, Ilnickiand S, Jin LJ, Krishnamoorthy V, Shan MC. Adaptive and dynamic service composition in eFlow. In: Wangler B, Bergman L, eds. Proc. of the 12th Int'l Conf. on Advanced Information Systems Engineering (CaiSE 2000). LNCS 1789, Berlin, Heidelberg: Springer-Verlag, 2000. 13−31.

[9] Sheng QZ, Benatallah B, Dumas M, Mak EOY. SELF-SERV: A platform for rapid composition of Web services in a peer-to-peer environment. In: Proc. of the 28th Int'l Conf. on Very Large Data Bases (VLDB 2002). Hong Kong: Morgan Kaufmann Publishers, 2002. 1051−1054.

[10] Orriens B, Yang J, Papazoglou MP. Model driven service composition. In: Orlowska ME, Weerawarana S, Papazoglou MP, Yang J, eds. Proc. of the 1st Int'l Conf. on Service Oriented Computing (ICSOC 2003). LNCS 2910, Berlin, Heidelberg: Springer-Verlag, 2003. 75−90.

[11] Berardi D. Automatic service composition: Models, techniques and tools [Ph.D. Thesis]. Rome: University of Roma, 2005.

[12] Bultan T, Fu X, Hull R, Su JW. Conversation specification: A new approach to design and analysis of e-service composition. In: Hencsey G, White B, eds. Proc. of the 12th Int'l Conf. on World Wide Web (WWW 2003). New York: ACM, 2003. 403−410.

[13] Zhang WT, Peng Y, Chen JL. Interface compatibility and composition of session-oriented e-service. Chinese Journal of Computers, 2006,29(7):1047−1056 (in Chinese with English abstract).

[14] Foster H, Uchitel S, Magee J, Kramer J. Model-Based analysis of obligations in Web service choreography. In: Proc. of the IEEE Int'l Conf. on Internet & Web Applications and services (ICIW 2006). Washington: IEEE Computer Society, 2006. 149−156.

[15] Yeung WL. Mapping WS-CDL and BPEL into CSP for behavioural specification and verification of Web services. In: Bernstein A, Gschwind T, Zimmermann W, eds. Proc. of the 4th European Conf. on Web Services (ECOWS 2006). Washington: IEEE Computer Society, 2006. 297−305.

[16] Busi N, Gorrieri R, Guidi C, Lucchi R, Zavattaro G. Choreography and orchestration: A synergic approach for system design. In: Benatallah B, Casati F, Traverso P, eds. Proc. of the 3rd Int'l Conf. on Service Oriented Computing (ICSOC 2005). LNCS 3826, Berlin, Heidelberg: Springer-Verlag, 2005. 228−240.

[17] Hoare CAR. Communicating Sequential Processes. New Jersey: Prentice Hall International, 1985.

[18] Qiao XQ, Wei J. A decentralized services choreography approach for business collaboration. In: Proc. of the IEEE Int'l Conf. on Services Computing (SCC 2006). Washington: IEEE Computer Society, 2006. 190−197.

[19] van der Aalst WMP, Weske M. The P2P approach to inter-organizational workflows. In: Dittrich KR, Geppert A, Norrie MC, eds. Proc. of the 13th Int'l Conf. on Advanced Information Systems Engineering (CaiSE 2001). LNCS 2068, Berlin, Heidelberg: Springer-Verlag, 2001. 140−156.

[20] Qiao XQ, Wei J, Huang T. A decentralized approach for inter-enterprise business process collaboration. In: Doumeingts G, Muller J, Morel G, Vallespir B, eds. Proc. of the 2nd Interoperability for Enterprise Software and Applications Conf. (I-ESA 2006). Berlin, Heidelberg: Springer-Verlag, 2006. 309−319.

[21] Yellin DM, Storm RE. Protocol specification and component adaptors. ACM Trans. on Programming Languages and Systems, 1997, 19(2):292−333.

[22] Nezhad H, Benatallah B, Martens A, Curbera F, Casati F. Semi-Automated adaptation of service interactions. In: Proc. of the 16th Int'l Conf. on World Wide Web (WWW 2007). New York: ACM, 2007. 993−1002.

[23] Bordeaux L, Salaun G, Berardi D, Mecella M. When are two Web services compatible? In: Shan MC, Dayal U, Hsu M, eds. Proc. of the 5th Workshop on Technologies for E-Servcies (TES 2004). LNCS 3324, Berlin, Heidelberg: Springer-Verlag, 2004. 15−28.

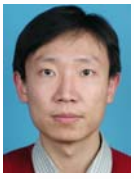
[24] Benatallah B, Casati F, Grigori D, Nezhad HRM, Toumani F. Developing adapters for Web services integration. In: Pastor O, Cunha JF, eds. Proc. of the 17th Int'l Conf. on Advanced Information Systems Engineering (CaiSE 2005). LNCS 3520, Berlin, Heidelberg: Springer-Verlag, 2005. 415−429.

[25] Cimpian E, Mocan A. WSMX process mediation based on choreographies. In: Bussler C, *et al*., eds. Proc. of the BPM 2005 Workshop. LNCS 3812, Berlin, Heidelberg: Springer-Verlag, 2005. 130−143.

[26] Bracciali A, Brogi A, Canal C. A formal approach to component adaptation. Journal of Systems and Software, 2005,74(1):45−54.

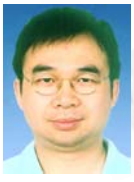
附中文参考文献:

[13] 张文涛,彭泳,陈俊亮.会话类 E-Service 的接口兼容和服务组合分析.计算机学报,2006,29(7):1047−1056.



乔晓强(1977－),男,河南焦作人,博士生,主要研究领域为网络分布计算,工作流技术,服务协作理论.

黄涛(1965－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为软件工程和方法学,分布式对象技术.



魏峻(1970－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为软件工程,软件理论,网络分布计算.