软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2012,23(3):517−538 [doi: 10.3724/SP.J.1001.2012.04103]
©中国科学院软件研究所版权所有.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

跨组织多业务事务建模与验证方法*

袁 敏+, 黄志球, 胡 军

(南京航空航天大学 计算机科学与技术学院,江苏 南京 210016)

Modeling and Verification of Cross-Organizational Multi-Business Transactions

YUAN Min+, HUANG Zhi-Qiu, HU Jun

(College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China)

+ Corresponding author: E-mail: yuanmin@nuaa.edu.cn

Yuan M, Huang ZQ, Hu J. Modeling and verification of cross-organizational multi-business transactions. *Journal of Software*, 2012,23(3):517−538. http://www.jos.org.cn/1000-9825/4103.htm

Abstract: Service-Oriented transaction processing is a key technology to ensure the correctness of interaction and collaboration among business processes. For cross-organizational multi-business processes, an approach of modeling and verification of multi-business transactions is proposed in this paper. In the modeling approach, an extended Pi-calculus was proposed to describe business transactions coordination via introducing transaction semantics based on the connection between the process interactions and transaction membrane activities. On the other hand, the model checker is employed for checking whether or not the multi-business transactions satisfy the given properties by equal value transformation of the finite state automaton. Finally, the experimental results have demonstrated that it is an important means of ensuring correctness during the design and implementation of multi-business processes.
Key words: cross-organization; business process; transaction; Pi-calculus; verification

摘 要: 面向服务的事务处理是保障业务交互和协作正确性的关键技术.针对跨组织多业务流程,提供了一种支持多业务事务建模与验证的方法.在建模方法上,对 Pi-演算扩展了事务语义,将进程间的动作交互与跨组织膜活动相关联来刻画多业务事务协调行为.在验证过程中,基于等价自动机转换思路集成现有模型检验技术,验证多业务事务是否满足人们需要的各种性质.实验结果表明,该建模与验证方法能够有效地保障多业务流程在设计与实现过程中的正确性.
关键词: 跨组织;业务流程;事务;Pi-演算;验证
中图法分类号: TP311 **文献标识码**: A

面向服务计算是以 Web 服务为基础的分布式计算模式,旨在有效解决在分布、动态、异构环境下,数据、应用和系统集成的问题[1].Web 服务作为一种自主而开放的应用实体,具有松耦合、平台无关、互操作性强等特点[2],它正从最初的描述、发布和交互向新的阶段发展,从而支持健壮的业务交互与协作[3].为了保证它们获得正确、一致的执行结果,大多数工作流和 B2B 协作应用程序都需要支持复杂的、长时间运行的业务事务(business

* 基金项目: 国家高技术研究发展计划(863)(2009AA010307); 湖南省自然科学基金(09JJ3114)
收稿时间: 2011-01-28; 修改时间: 2011-06-10; 定稿时间: 2011-08-23

transaction).传统事务模型由于过于严格,已无法适用于松耦合、自治和开放的 Web 服务应用环境[4].实际业务中,事务协调过程包括前事务(pre-transaction)、主事务(main-transaction)和后事务(post-transaction)这 3 个阶段[5],它们在事务的原子性、一致性和隔离性等方面都有不同程度的放松.众多研究针对主事务阶段(main-transaction phase)中业务流程执行和事务处理过程提出了各种高级的事务模型,在这些面向服务的松弛事务模型中,都有一个重要的概念——“补偿(compensation)”,即通过反向操作取消已成功完成和提交的事务所产生的影响.然而,不是所有的事务都能逆转并完全消除已经执行的效果,作为事务处理前参与者之间交换试探性信息的前事务阶段(pre-transaction phase),它能够提高业务事务成功完成的机率,减少不必要的失败或补偿.所谓多业务事务MBT(multi-business transactions)[6]就是指在事务执行之前,为了提高事务的灵活性,业务之间主动交换各自状态信息的协调过程.它们允许客户对资源进行非阻塞的预定,放松了事务的隔离性要求.与多业务事务相关的一组业务流程又称多业务流程(multi-business processes).多业务事务的概念最早见于 W3C 的 THP(tentative hold protocol)协议[7],该协议是一个开放的、松耦合的、基于消息传递的事务协调协议,事务执行之前允许客户对资源进行尝试性地占有,并进行交换业务应用意图的会话,涉及事务提交的具体条款,例如价格和送货时间等.它定义了一个临时、非阻塞地资源占有的锁结构,并及时更新客户端和资源端各自的状态信息,能够最小化业务应用补偿次数.尝试性预定机制也能够提高资源的利用率,减少事务补偿的等待时间.

已有许多研究关注多业务事务协调机制,Park 等人通过对 THP 协议的资源预定上限和预定时效两个参数进行调整来优化资源分配策略,结合二阶段提交协议提出了自适应的协调框架[8].Limthanmaphon 等人将尝试性地占有方法和补偿方法相结合,将事务补偿的可能性降到最低[9].Younas 等人研究服务组合事务的性能,提出尝试性提交协议 TCP(tentative commit protocol),减少了网络通信延迟和事务处理时间[10].但这些研究中没有涉及多业务事务协调的正确性验证方法.形式化方法则可以帮助人们更好地理解 Web 服务行为特性,其语义模型也可以提供对服务业务流程推理的基础;基于模型检验的分析技术还可以允许自动地检验服务组合业务流程是否满足人们需要的各种性质,从而帮助消除设计缺陷,提高应用的可靠性.面对 B2B 电子商务跨组织多业务协作的需求,如何设计正确的跨组织多业务流程、建模和验证多业务事务,仍是一个亟待解决的重要问题.

本文针对跨组织多业务流程,提出了基于扩展事务语义的 Pi 演算的建模方法,并根据等价自动机转换思路,集成现有模型检验技术,给出了多业务事务相关性质的验证方法.

本文第 1 节进行相关工作的比较.第 2 节概述跨组织多业务事务的协调机制.第 3 节提出一种扩充的 Pi 演算用于描述跨组织多业务事务协调行为,并给出相应的语法和操作语义.第 4 节采用这种形式化语言对跨组织多业务事务进行建模.第 5 节介绍如何基于该模型进行模型检验的验证方法.第 6 节结合实例对本文的方法进行实验结果分析和方法评价.最后对全文内容进行总结.

1 相关工作

如何设计正确的服务组合业务流程,让它们相互协同工作并符合业务事务要求,是人们普遍关心的问题.已有众多研究采用形式化验证方法对 Web 服务流程进行了研究,如文献[11]基于编程统一理论 UTP(unifying theories of programming),研究 BPEL 语言中事务型 Web 服务的编程新特性,提出了一个精确的事务数学框架,并从代数法则证明的角度研究了补偿事务的精化关系.文献[12]基于作用域的、可编程补偿的角度研究了 BPEL 的异常处理和补偿机制,并给出了一个形式化语义模型.文献[13]提出了一种细胞膜演算的形式化方法,利用对象和细胞膜的反应过程,描述了 Web 服务事务异常处理和补偿过程,并分析检验了早期的 Web 服务事务处理(WS-transaction)规范.上述这些研究主要集中在业务流程执行和事务处理的主事务阶段,而没有涉及事务处理开始之前,即参与者交换试探性信息的前事务阶段.本文提出的方法充分考虑了前事务阶段的跨组织多业务流程的协调过程,能够对多业务事务的协调行为进行精确描述,并给出了与业务事务要求相关的系统性质验证方法,有效保障了跨组织多业务流程在设计与实现过程中的正确性.

在建模方法上,基于进程代数的形式化方法能够自然描述通信并发系统,作为移动进程代数的 Pi-演算[14]则具有可组合性和移动性,更适于描述开放动态的并发系统.已有许多研究将它们应用到 Web 服务及事务建模研

究中,如文献[15]把异常处理和补偿合并到事件处理的框架里,给出了 BPEL 的基于扩展 Pi-演算的形式语义,提出了基于 Pi-演算的事件演算模型.文献[16]提出了πt-calculus,对 Pi-演算作了扩展,加入事务模块控制算子以支持长事务补偿特性.文献[17]在传统的进程代数基础上加入补偿算子,提出支持描述事务处理的语言 StAC,是最先将补偿和进程控制紧密结合的进程演算之一.文献[18]提出的适用于建模业务事务中失效恢复机制的 cCSP 语言,则是在通信顺序进程 CSP(communicating sequential processes)框架上引入了支持顺序、选择、并行和补偿等复合操作算子.上述对 Web 服务事务的建模方法大都通过扩充新的操作算子来刻画事务语义,这在一定程度上也增加了演算自身的语言复杂性.本文没有新增特定的事务操作算子,而是引入“事务膜”的概念来表示事务的作用域,事务处理过程则可以由进程链接相对事务膜移动的膜活动来表示.提出了将跨组织膜活动与进程动作交互建立动态关联的方法,可以更加简洁自然地刻画多层作用域、异常处理和补偿等事务语义.

在模型的验证方法上,目前有一些直接支持 Pi-演算模型检验的工具,如基于 Tableau 的定理证明系统 MWB (mobility work bench)[19].然而,基于定理证明的方法不适合做时态方面的推理,例如不能直接验证用时态逻辑描述的时序性质,它更适于系统功能规范和参数化描述.另外一类检测工具就是基于等价自动机转换的思路,具有代表性的模型检验工具如 HAL[20],它将 Pi-演算移动进程模型转化成一种自定义的自动机模型来检测.与 HAL 类似,文献[21]也是基于 Pi-演算早迁移操作语义,将系统模型转换为标号迁移系统,然后直接将其语义解释到具体的模型验证引擎上.支持 Pi-演算的系统化验证工具和应用仍较少,现有的 Pi-演算模型检验工具在这些方面还有大量亟待解决的问题[22].本文沿用等价自动机转换的思路,但这里所提出的转换方法是基于 LTS(labeled transition system)与 KS(kripke structure)这两种结构的等价关系,并且进一步用代数方法来表示系统的功能特性和非功能特性,实现了扩充 Pi-演算进程模型到自动机模型的转换,使得该转换方法有更好的适用性,增强了模型检验工具的健壮性和可扩展性.

此外,我们先前的工作是本文的基础.文献[23]提出了一种扩展事务语义的 Pi-演算(记作 MPi-演算)及其互模拟等价关系,在此基础上,本文针对跨组织协调行为,提出了跨组织多业务事务的建模方法.另一方面,文献[24]提出了基于标准 Pi-演算的模型检验框架,本文则进一步完善了 MPi-演算模型的等价自动机转换方法,为支持扩充 Pi-演算的模型检验提供了一种有效的途径.

2 多业务事务协调

面向服务的业务流程一方面本身由 Web 服务组装而成,另一方面也可以当作 Web 服务来使用.也就是说,流程既可以作为服务请求者(client),也可以扮演服务提供者(vendor).在松耦合、自治和开放的 Web 服务应用环境,事务不能再集中统一控制和决策,而是由跨组织的业务参与者相互协商来完成,请求者无法长时间排他的锁定所需资源,提供者也不允许将它们的资源被开放环境中不可预测的业务活动长期占用.大多数现实生活中的业务场景都包含复杂的多对多的业务交互,例如,文献[6]中提到的旅行计划业务(如图 1 所示),它描述了旅行代理借助电子商务平台的相关服务资源来制定旅行计划的场景.旅行中机票、宾馆和出租车的预定是件比较繁琐的工作,一种方法是事先查询好各个项目然后一起预定,如首先查询机票的折扣,接着查找合适的房间.但若期间耗时太多,机票的报价会过时又需要重新查询等;另一种方法是依次即时预定各个项目,但先订了机票后可能会出现合适房间都已被预订满的情况,造成必须退订机票的麻烦等.这势必需要在业务协调过程中尽可能减少人为干预,前事务阶段具有代表性的应用通信协议——THP 协议为业务参与方在这些方面提供了更大的灵活性[5].如图 1 所示,它用一个协调器(travel coordinator)来代替客户与每个服务资源进行交互.将客户对各种资源的全部请求打包成一个虚拟的预定操作,即先尝试性预订所有资源,只有全部预定请求被满足时才进行确认消费.当被客户尝试性占有的资源变成无效

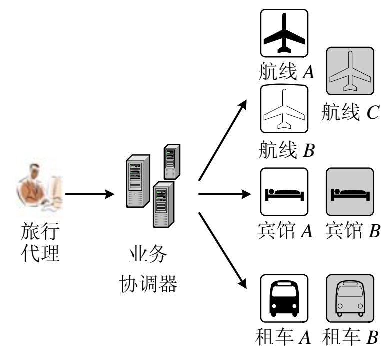

Fig.1 Travel plan business scenarios
图 1 旅行计划业务场景

时,服务提供方还会及时通知客户取消相应的尝试性预定,避免事后补偿的麻烦.这样一来,提高了业务事务协调的自动化水平,从而增加业务的灵活性,减少了运营成本.

THP 协议具有多业务事务协调的非阻塞预定、尝试性占有和排他性消费的主要特点:(1) 非阻塞性(non-blocking):服务提供者也能够让自身资源非阻塞地被多个客户预定,并允许其拥有选择客户的权利;(2) 尝试性(tentative):服务请求者可以先尝试性预定整个业务流程所需的所有服务资源,最后一起提交整个事务;(3) 排他性(exclusive):服务资源一旦被消费,其他客户的预订将被取消,不会出现多个客户同时消费一个资源的情况.在THP 协议技术规范中,用时序图来描述各组件之间的交互,并针对"2 个客户(client)竞争 1 个资源(resource)"(记作 2C-1R 型)的典型业务,用一系列特定时序操作步骤详细地说明和实现了多业务事务协调的基本过程,如图 2所示[7]:(1) Step 1~Step 5:客户 1 申请预定一个服务资源,并获得尝试性占有该资源的许可;(2) Step 6~Step 10:客户 2 也申请和预定成功了同一服务资源;(3) Step 11~Step 14:客户 2 随后购买该资源,客户 1 会收到资源已消费的通知,并被要求取消原有的预定.

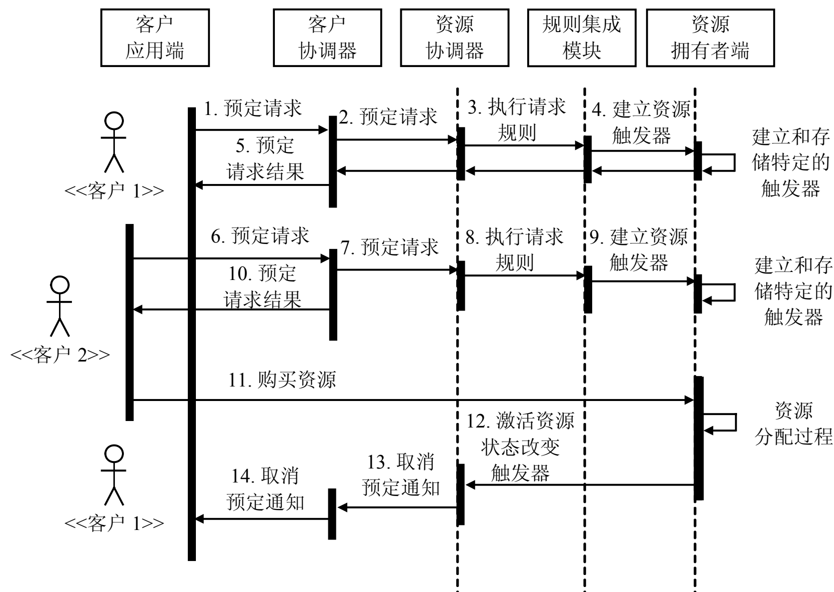

Fig.2 Steps needed to realize the tentative hold protocol
图 2 THP 协议栈的实现步骤

客户端和资源端都有自己的协调器,分别是客户端协调器(client coordinator,简称 CC)和资源端协调器(resource coordinator,简称 RC),它们负责双方各种消息的通信.其中,客户端协调器 CC 位于客户端应用程序(client application,简称 CA)一边,负责创建新的事务,转发客户端应用程序的请求,并能够将事务协调的过程和结果实时地发送给客户端应用程序 CA;资源协调器 RC 位于资源提供者(resource owner,简称 RO)一边,负责接收和响应来自客户端协调器 CC 的预定请求,通过与规则集成模块(rules integration module,简称 RIM)的会话来确定是否同意对方的预定请求,并统一调控所有申请成功的预定,决定何时向哪个发出取消预订的通知.受限于时序图的描述能力,图 2 中描述了 2 个业务流程(客户)与 1 个服务资源之间发送和接收消息的时间序列.事实上,它只对应该业务协调活动的一个序列,没能描述动态、并发的协调活动.相比而言,形式化方法能够对参与者之间的消息传递语义进行精确地刻画,建立多业务事务形式化模型,从而用模型检验技术进一步分析和验证多业务事务协调机制的正确性.

由于"2 个客户竞争 1 个资源"(2C-1R 型)的业务实现了多业务事务协调的基本过程,与 THP 协议技术规范类似,本文同样先针对这个典型业务来讨论如何建模和验证多业务事务.为了简化目标模型,这里参与多业务事

务协调的主要是客户端协调器 *CC* 和资源端协调器 *RC*,不同业务之间的事务协调可以看成是它们各自协调器之间的交互.图 3 给出了客户端协调器 CC 和资源端协调器 *RC* 之间相互通信的各种消息,表 1 则对应各种消息缩写符号的含义,包括非阻塞预定和排他性消费两个阶段:(1) 非阻塞预定:任意一个 *CC* 都可向某 *RC* 发出预定请求(*Hreq*),*RC* 将回应预定拒绝(*Hden*)或预定同意(*Hgra*);(2) 排他性消费:*RC* 接到某 *CC* 的消费请求(*Creq*),将先向其他预定成功的 *CC* 发出取消预定(*Hcan*)的通知,并等待他们都限制消费(*Cden*)后,才能同意该 *CC* 的消费请求(*Cgra*),最后让其完成消费任务(*Cons*).

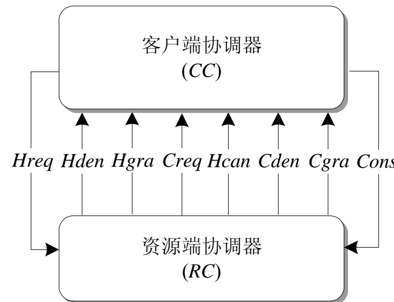

Fig.3 Messages passing diagram

图 3 消息传递图

Table 1 Abbreviations of denotations

表 1 符号的缩写形式

类型	名字	缩写形式
进程(processes)	Client coordinator/Resource coordinator	*CC*1/*CC*2/*RC*1/*RC*2
活动(activities)	Hold/Consume	H/C (*cons*)
消息(messages)	request/grant/denial/cancel of activities	*activities+req/gra/den/can*

3 跨组织协调行为的描述

Pi-演算(π-calculus)[14]是常用的进程代数之一,它具有自身算子可组合性和传送通道名所赋予的移动性两大特点,更适合于描述开放动态的通信并发系统.在面向服务的业务流程相关研究中,已经广泛采用它作为形式化描述语言.进程(process)和名字(name)是 Pi-演算的两个基本实体,而动作交互(interaction)则是它的基本原语.一方面,通过定义进程能够执行的基本动作来表明进程的行为能力;另一方面,在这些动作交互中,又通过传递消息名字的通信方式来实现进程的演化.Pi-演算与通信自动机之间最主要的区别在于改变进程网络连接的能力,这种对连接的改变就是进程的移动性(mobility).即在 Pi-演算中,进程的移动可以完全由其链接的移动来表示[25].这种移动性的表示方法经济、灵活,并且比较简单.为了能够描述事务处理及协调等信息,本文进一步用这种类型的移动性来表示进程相对事务作用域的移动,使其能更准确和自然地描述跨组织业务流程之间的交互行为和多业务事务的性质.

3.1 MPi-演算的语法

在文献[23]中,我们提出了一种扩充的 Pi-演算(记作 MPi-演算),引入“事务膜”的概念来表示事务的作用域.当不同事务作用域的进程交互时,链接发生了移动,穿越它们之间的事务膜,这种链接相对事务膜的移动被称作膜活动(membrane activity),事务处理过程可以由进程链接相对事务膜的移动来表示.这样,膜活动就描述了进程演化相对事务作用域位置不断变化的过程,它与 Pi-演算中的进程动作交互相关联,可以更加自然地刻画事务的作用域、异常处理和补偿等语义.下面首先介绍本文中所用到的 MPi-演算基本语法定义.

设$\mathcal{N}$为有序可数名字集,用 $x,y,z,\ldots$等小写字母表示名字集$\mathcal{N}$上的名字,进程表达式用 P,Q,R 等大写字母表示.如图 4 所示,MPi-演算的完整语法可用 BNF 语法给出.名字是最基本的概念,将数据值、数据变元、输入参

变元以及传递数据的通道不加区别,一律作为名字来处理.和型Σ表示在有限进程行为中的选择执行关系,P 被π护卫,表示 P 必须在π所代表的动作发生后才能开始,如果 P 为空进程'0',则表示不做任何动作通常将其略去;Pi-演算系统通过选择"+"、并行"|"、限制"v"、匹配"[]"、复制"!"等操作算子实现进程的演化.这里没有进程标识符 $A=_{def}P_A,\ldots$,因为复制算子与传递名字相配合已经具有与显式递归定义进程表达式的同样能力[25].

$$
\begin{aligned}
&P ::= \Sigma \mid P \mid P' \mid (vx)P \mid\ !P \mid M[\![P]\!],\\
&\Sigma ::= 0 \mid \pi\cdot P \mid \Sigma+\Sigma',\\
&\pi ::= \lambda\{\mu\} \mid \tau \mid [x=y]\pi,\\
&\lambda ::= \overline{x}\langle y\rangle \mid x(z),\\
&\mu ::= In(M) \mid Out(M) \mid 0 \mid \mu;\mu'.
\end{aligned}
$$

Fig.4 Syntax of MPi-calculus

图 4 MPi-演算基本语法

基本 Pi-演算的前缀式包括输出前缀 $\overline{x}\langle y\rangle$ 、输入前缀 $x(y)$和哑前缀τ,其中可观察的动作又用λ来代表.上述语法定义中,输入前缀和限制式是两个约束构造,它们都会约束名字,我们称 P 中约束出现的名字为受限名集(bound name),记作 $bn(P)$;未被约束的名字为自由名集(free name),记作 $fn(P)$;受限名集和自由名集的并集为 P 的名字集$\mathcal{N}$,记作 $n(P)$.同一个通道上的一对输入和输出动作称作互补关系,移动发生在互补关系的输入和输出的动作上,它们并发运行且都不被其他动作护卫.下面考虑 3 个进程并发复合的情形,举例说明 Pi-演算中如何用"链接的移动"来表示"进程的移动".这里用流图(flow graph)[25]来描述系统的结构,即组成部分之间的链接结构,其中,圆形表示进程,以两端带圆点的实线表示进程之间的链接名字和通信端口,椭圆框表示名字的作用域.

进程表达式和迁移关系如图 5 所示:(1) 迁移之前,我们假设名字 x 在 P 和 Q 中自由出现,名字 z 被限制在 P 和 R 中,这里用椭圆框表示名字 z 的作用域范围;(2) 迁移发生时,进程 P 通过 x 把指向 R 的一个链接 z 送给进程 Q;(3) 迁移之后,名字 z 变成被限制在 Q'和 R 中.此时,z 不在 P'中自由出现,P'将失去到 R 的链接.

$$(v\ z)(\overline{x}\langle z\rangle\cdot P \mid R) \mid x(y)\cdot Q \xrightarrow{\tau} P \mid (v\ z)(R \mid Q'\{z/y\}) .$$

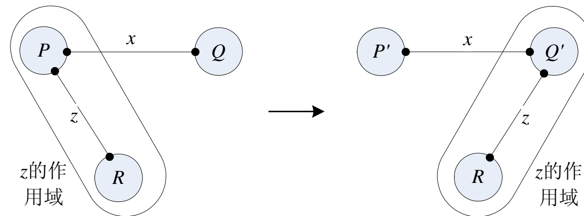

Fig.5 Flow graph in concurrent composition

图 5 并发复合的进程流图

MPi-演算允许在可观察动作λ上关联一个或多个膜活动,简记为$\{\mu\}$,它描述了进程交互时,事务作用域不断变化的过程.膜活动$\{\mu\}$的名字集为相关的事务膜进程名字集,记为 $n(\mu)$,它们在整个事务中自由出现,因此有$n(\mu)\subseteq fn(P)$.$In(M)$表示进程链接移入子事务膜 M;$Out(M)$表示进程链接移出当前事务膜 M;0 表示进程链接在当前事务膜内移动.此时可以省略$\{0\}$这部分,例如 $\overline{x}\langle y\rangle\{0\}$ 简记作 $\overline{x}\langle y\rangle$.多个膜活动用分号(;)顺序连接.膜活动既可以描述系统进程根据事务上下文的演化过程,也可以限定进程交互发生的事务作用域.$M[\![P]\!]$也是一种进程,我们称作名为 M 的事务膜(transaction membrane)进程,P 是在事务内部执行的进程.

3.2 跨组织活动的语义

自动化业务流程的规约语言 BPEL 所规定的长期运行事务是假定作用域和它嵌套的所有作用域都被包含在单个流程中,并由一个组织机构的单个 BPEL 引擎来管理[26].BPEL 不支持分布于流程间甚至跨组织的多个 BPEL 引擎上运行的事务.多业务事务往往跨越组织边界,涉及多个业务流程间的通信合作.由于多业务事务处

于“前事务阶段”,并不包含具体的事务执行过程,此阶段各业务流程的组织边界就代表了事务的不同作用域,而膜活动则与跨组织的业务活动相对应.接下来我们就将跨组织活动与 Pi-演算系统中的进程交互建立动态关联,用进程间的动作交互和跨组织膜活动共同来刻画多业务事务协调行为.跨组织膜活动关注系统进程根据多业务事务上下文演化的过程,记录指向进程的链接发生移动时,穿越组织边界的各种活动.按链接移动的位置变化过程,跨组织业务活动可以表示为下面 3 种基本膜活动,这里仍然采用流图(flow graph)的变化来描述系统的变迁,并在流图的基础上增加了用实线框表示组织单元对应的事务作用域.事务内部的进程通过事务膜(transaction membrane)进程与外部进行通信,此时,代表通信端口的实心圆点则位于事务膜进程对应的实线框上.

定义 1(组织内部活动). 假设进程 P 和 Q 在同一个组织单元对应的事务作用域 M 内,它们发生交互,指向进程 R 的链接将从进程 P 移到进程 Q,链接在本地组织单元内部移动.活动对应的进程流图如图 6 所示,进程表达式为

$$M[\![(\nu\ z)(\overline{x}\langle z\rangle\{0\}\cdot P\,|\,R)\,|\,x(y)\{0\}\cdot Q]\!]\xrightarrow{\tau}M[\![P\,|\,(\nu\ z)(R\,|\,Q'\{z/y\})]\!].$$

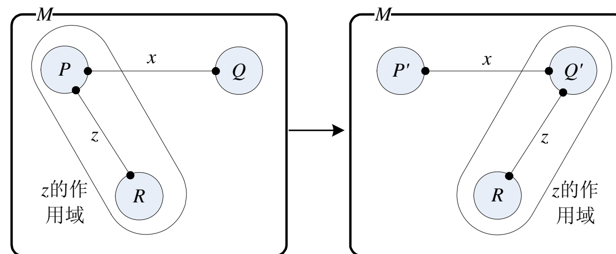

Fig.6 Flow graph of local activity

图 6 组织内部活动的进程流图

定义 2(进入下级组织). 假设进程 P 和 Q 的所在的组织单元分别为 $O1$ 和 $O2$,它们对应的事务作用域分别是 $M1$ 和 $M2$.如果 $O2$ 是 $O1$ 的下级组织,则 $M2$ 是 $M1$ 的子事务作用域,进程 P 与子事务 $M2$ 内的进程 Q 发生交互,指向进程 R 的链接将从进程 P 移到进程 Q 所在的事务,并移入子事务 $M2$ 的作用域.活动对应的进程流图如图 7 所示,进程表达式为

$$M1[\![(\nu\ z)(\overline{x}\langle z\rangle\{In(M2)\}\cdot P\,|\,R)\,|\,M2[\![x(y)\{In(M2)\}\cdot Q]\!]]\!]\xrightarrow{\tau}M1[\![P'\big|(\nu\ z)(R\,|\,M2[\![Q'\{z/y\}]\!])]\!],\ \text{if}\ z\notin fn(M2).$$

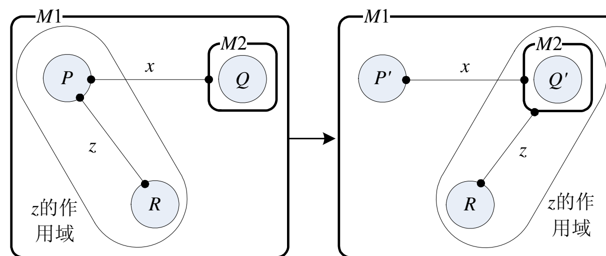

Fig.7 Flow graph of into the unit

图 7 进入组织的进程流图

定义 3(退出当前组织). 假设进程 P 和 Q 的所在的组织单元分别为 $O1$ 和 $O2$,它们对应的事务作用域分别是 $M1$ 和 $M2$.如果 $O2$ 是 $O1$ 的上级组织,则 $M2$ 是 $M1$ 的父事务作用域,事务 $M1$ 内的进程 P 与 Q 发生交互,指向进程 R 的链接将由当前事务的作用域移出,并从进程 P 所在的事务 $M1$ 移到进程 Q.活动对应的进程流图如图 8 所示,进程表达式为

$$M2[\![(\nu z)(M1[\![\overline{x}\langle z\rangle\{Out(M1)\}\cdot P]\!]\,|\,R)\big|x(y)\{Out(M1)\}\cdot Q]\!]\xrightarrow{\tau}M2[\![M1[\![P']\!]\,|\,(\nu z)(R\,|\,Q'\{z/y\})]\!].$$

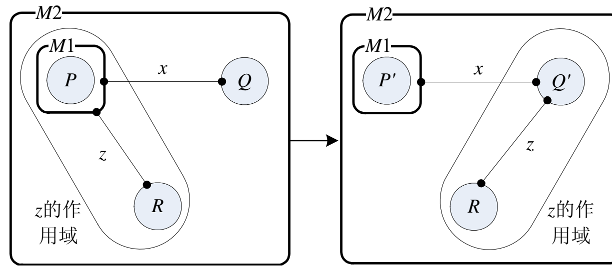

Fig.8 Flow graph of out of the unit

图 8 退出组织的进程流图

上面给出的是一个尽可能简洁的跨组织膜活动的基本规则,在此基础上,结合事务协调的各种上下文,可以衍生出其他具体的业务活动过程.例如,事务 $M1$ 和 $M2$ 为同一级组织单元对应的事务作用域,指向进程 R 的链接从事务 $M1$ 移动到了事务 $M2$,可以认为它先由事务 $M1$ 的作用域退出,再进入兄弟事务 $M2$ 的作用域.活动对应的进程流图如图 9 所示,进程表达式如下:

$$(\nu z)(M1[\![\overline{x}\langle z\rangle\{Out(M1);In(M2)\}\cdot P]\!]\,|\,R)\,|\,M2[\![x(y)\{Out(M1);In(M2)\}\cdot Q]\!]\xrightarrow{\tau}$$
$$M1[\![P']\!]\,|\,(\nu z)(R\,|\,M2[\![Q'\{z/y\}]\!]),\ \text{if}\ z\notin fn(M2)$$

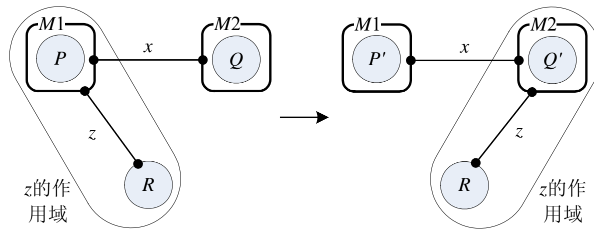

Fig.9 Flow graph of changing the unit

图 9 改换组织的进程流图

3.3 扩展的操作语义

以上规则只是描述跨组织业务活动的过程,只有当它们与进程动作相关联才具有进程演化的操作语义,Pi-演算的行为通常被解释到一个标号迁移系统上.与之类似,定义 4 给出一个关联跨组织膜活动的 MPi-演算行为对应的标号迁移系统(labeled transition system,简称 LTS).

定义 4(标号迁移系统). 它是一个四元组 $M=(S,I,A,\Delta)$:

- S 是一个有限的系统进程状态集;
- $I\subseteq S$ 表示初始状态集合;
- A 是标号集合,可观察动作 λ 与膜活动 $\{\mu\}$ 相关联的有限 $\lambda\{\mu\}$ 的集合,内部哑元动作 $\tau\notin A$;
- $\Delta\subseteq S\times(A\cup\{\tau\})\times S$,表示迁移关系,$\Delta$是一个 $\{\xrightarrow{\alpha}|\alpha\in A\cup\{\tau\}\}$ 迁移的集合,表示变迁在动作 α 上发生.元素 $(r,\alpha,s)\in\Delta$称为变迁,通常用 $r\xrightarrow{\alpha}s$ 表示.

MPi-演算扩展操作语义见表 2,其中省略了一些对称的语义以及与复制算子相关的操作语义:(1) 一类与α相关的操作语义,如 PREF 前缀语义,表示任何情况下,$\alpha\cdot P$ 经过α动作成为 P.与 Pi-演算操作语义不同的是,这里的α是关联跨组织膜活动的可观察动作$\lambda\{\mu\}$或内部哑动作 τ,(2) 另一类与互补动作相关的操作语义,如在 COMM-L 和 CLOSE-L 等语义中规定了互补的输入输出动作进程,同步转化为哑动作τ,相对整个交互系统而言,

此时发生的膜活动和两个组件 P 和 Q 之间的交互一样是不可观察的.它们对应的跨组织膜活动相同,表明了进程间的交互,除需要拥有共享的链接名字外,还受到与跨组织膜活动相关的事务作用域的约束.

Table 2 Extended reduction rules of MPi-calculus

表 2 MPi-演算扩展操作语义

$$\text{PREF: } \frac{-}{\alpha \cdot P \xrightarrow{\alpha} P} \qquad \text{MAT: } \frac{\pi \cdot P \xrightarrow{\alpha} P'}{[x=x]\pi \cdot P \xrightarrow{\alpha} P'} \qquad \text{SUM-L: } \frac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'}$$

$$\text{PAR-L: } \frac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q}\ bn(\alpha) \cap fn(Q) = \emptyset \qquad \text{RES: } \frac{P \xrightarrow{\alpha} P'}{(v\,y)P \xrightarrow{\alpha} (v\,y)P'} \quad y \notin n(\alpha)$$

$$\text{COMM-L: } \frac{P \xrightarrow{\bar{x}\langle y\rangle\{\mu\}} P' \quad Q \xrightarrow{x(z)\{\mu\}} Q'}{P|Q \xrightarrow{\tau} P'|Q'\{y/z\}} \qquad \text{OPEN: } \frac{P \xrightarrow{\bar{x}\langle y\rangle\{\mu\}} P'}{(v\,y)P \xrightarrow{(v\,z)\bar{x}\langle z\rangle\{\mu\}} P'\{z/y\}}\ y \neq x, z \notin fn((v\,y)P')$$

$$\text{CLOSE-L: } \frac{P \xrightarrow{(v\,z)\bar{x}\langle z\rangle\{\mu\}} P' \quad Q \xrightarrow{x(z)\{\mu\}} Q'}{P|Q \xrightarrow{\tau} (v\,z)(P'|Q')}\ z \notin fn(Q)$$

MPi-演算的语法以及扩展的操作语义增加了事务的标识符名字“M”,将动作前缀与事务膜活动“$\{\mu\}$”相关联,并用嵌套的事务膜符号“⟦⟧”描述各个进程所在事务作用域的关系,它们从名字、进程和动作交互等 3 个要素上为 Pi-演算引入了事务语义,从而提高了目标模型在事务特性上的分析能力.接下来,我们回到前面提到的“2 个客户竞争 1 个资源”(2C-1R 型)的典型业务(如图 2 所示),讨论如何用 MPi-演算来描述多业务事务中的跨组织协调行为.图 2 中的两个客户所代表的业务流程参与同一服务资源的竞争,它们属于不同的组织单元(organization unit).$CC1$ 和 $CC2$ 分别属于组织单元 $O1$ 和 $O2$,它们对同一服务资源发起尝试性占有申请,资源端协调器 RC 位于服务资源一边,属于组织单元 $O3$.图 10 是关于该多业务事务实例的 4 种不同建模方案,其中,图 10(a)、图 10(b)和图 10(c)前 3 种是采用 Pi-演算建模,图 10(d)则是采用 MPi-演算建模.图 10 中的流图(flow graph)仍用实线框表示事务作用域,与之对应组织单元的边界则用点划线框表示;其他图例含义与前面的流图一样,圆形表示进程,以两端带圆点的实线表示进程之间的链接名字和通信端口.

图 10(a)所示两个客户端 $CC1$ 和 $CC2$ 共用一个链接通道与资源端的 RC 进行通信.在该方案中,由于客户共用一个通道,当资源端 RC 收到一个客户发出的预定请求(*Hreq*),若预订请求被同意(*Hgra*),为了避免通道上的消息值被覆盖,则需要等到该客户消费(*Cons*)或取消预订(*Hcan*)之后才能响应其他客户.这与 THP 协议所规定的“非阻塞预定”不相符.

图 10(b)所示两个客户端 $CC1$ 和 $CC2$ 用各自的链接通道分别与资源端 RC 进行通信.在该方案中,客户可以独立地向资源端 RC 发出预定请求(*Hreq*),满足 THP 协议所规定的“非阻塞预定”要求.但是资源端 RC 则需要根据客户请求之间的先后关系以及不同的响应结果,用嵌套的循环逐一描述各种可能存在的情况,使得资源端 RC 的建模过程复杂.随着客户端数量的增加,目标模型会变得难于验证和分析.

图 10(c)所示两个客户端 $CC1$ 和 $CC2$ 用各自的链接分别与各自不同的资源端 $RC1$ 和 $RC2$ 进行通信.在该方案中,不同客户对资源端的请求和响应都是完全独立的,资源端的建模只需要考虑与其相对应的客户端的请求和响应,建模过程可以分而治之,简单而有效.但是资源端 RC 可以被多个客户同时预定成功(*Hgra*),也可以同时被消费(*Cons*),后者与 THP 协议所规定的“排他性消费”不相符.

图 10(d)所示的事务 $M1$,$M2$ 和 $M3$ 的作用域分别对应组织单元 $O1$,$O2$ 和 $O3$ 的边界,最外层的事务膜 MBT 表示整个多业务全局事务.两个客户端与资源端之间的通信由事务 $M1$,$M2$ 与 $M3$ 之间的通信来完成,并且代表通信端口的实心圆点不再直接连接表示客户端和资源端进程的圆圈,而是位于事务膜进程对应的实线框上.一方面,客户端可以用各自的链接通道独立地向对应的资源端实例 $RC1$ 和 $RC2$ 发出预定请求(*Hreq*),这满足 THP

协议所规定的“非阻塞预定”要求;另一方面,资源端实例 *RC*1 和 *RC*2 并没有完全隔离,仍共享所有与事务膜进程相连的通道,它们同在一个事务作用域中,即事务 *M*3 同一时刻只允许响应一个客户端,当资源端接到某客户端的消费请求(*Creq*)时,会向其他预定成功的客户端发出取消预定的通知(*Hcan*),这也满足 THP 协议所规定的“排他性消费”要求.

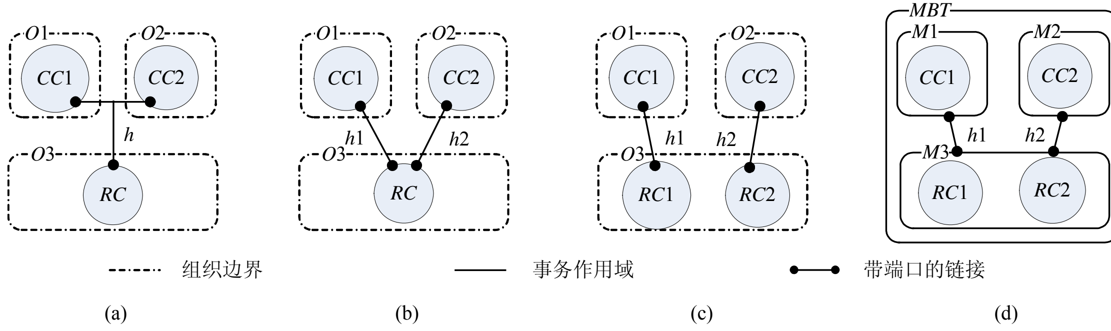

Fig.10 Flow graph of an example for type 2C-1R
图 10 2C-1R 型业务实例进程流图

4 跨组织多业务事务建模

通过上面关于多业务事务实例的 4 种不同建模方案的分析比较可以看出,MPi-演算不仅能够清楚地表示系统行为,而且更适合描述复杂的跨组织协调行为,是刻画跨组织多业务事务处理过程的有效方式.这里从名字、进程和动作交互等 3 个基本要素上给出用 MPi-演算建模多业务事务的 3 条基本规则.

规则 1. 需要通信的参与者之间引入通道,将通道和消息映射为 MPi-演算中的名字.

规则 2. 参与者作为 MPi-演算中一个独立的进程,整个多业务事务用各参与者进程的并发复合来表示.

规则 3. 参与者之间的通信用进程间的动作交互来表示,事务膜活动与进程的动作交互相关联,用进程链接相对事务膜的移动来刻画表示事务处理的过程.

下面根据以上规则对图 10(d)所示的流图进行形式化建模,各模块对应的 MPi-演算进程表达式如下.

(1) 客户端协调器 *CC*1

$\tilde{n}$ 是系统中所用到的消息名字集合,其中:名字符号的含义在表 1 中已经具体说明,$\{\mu1\}$和$\{\mu2\}$代表两个穿越事务膜 *M*1 和 *M*3 的跨组织膜活动,它们与进程动作交互相关联,表示当前事务作用域的变化.*CC*1 向 *RC*1 发出预定请求(*Hreq*),并等待 *RC*1 的响应.当 *RC*1 拒绝预定(*Hden*)时,*CC*1 返回到它的初始状态;当 *RC*1 同意预定(*Hgra*)时,*CC*1 可以继续向 *RC*1 发出消费请求(*Creq*),并等待来自 *RC*1 的消费确认通知(*Cgra*)之后,最终完成消费任务(*Cons*);在 *CC*1 发出预定请求(*Hreq*)、预定请求被同意(*Hgra*)和发出消费请求(*Creq*)的这些时段,*CC*1 都有可能收到来自资源端的取消预订(*Hcan*)通知,此时,*CC*1 都将限制自身的消费行为(*Cden*),然后返回初始状态.

$$\tilde{n}=\{Hreq,Hgra,Hden,Hcan,Creq,Cgra,Cden,Cons\}$$

$$\{\mu1\}=\{Out(M1);In(M3)\}$$

$$\{\mu2\}=\{Out(M3);In(M1)\}$$

$$\begin{aligned}CC1(\tilde{n})=M1[\![&\overline{h1}\langle Hreq\rangle\{\mu1\}.h1(msg)\{\mu2\}.([msg=Hden]CC1(\tilde{n})+[msg=Hgra]\\&(\overline{h1}\langle Creq\rangle\{\mu1\}.h1(msg2)\{\mu2\}([msg2=Cgra]\overline{h1}\langle Cons\rangle\{\mu1\}.CC1(\tilde{n})+\\&[msg2=Hcan]\overline{h1}\langle Cden\rangle\{\mu1\}.CC1(\tilde{n}))+h1(msg1)\{\mu2\}[msg1=Hcan]\\&\overline{h1}\langle Cden\rangle\{\mu1\}.CC1(\tilde{n}))+[msg=Hcan]\overline{h1}\langle Cden\rangle\{\mu1\}.CC1(\tilde{n}))]\!]\end{aligned}$$

(2) 客户端协调器 *CC*2

$\{\mu3\}$和$\{\mu4\}$则代表两个穿越事务膜 *M*2 和 *M*3 的跨组织膜活动,其关联的系统动作交互与 *CC*1 同出一辙,因此 *CC*2 的 MPi-演算进程表达式与 *CC*1 类似.

$$\{\mu 3\}=\{Out(M2);In(M3)\}$$
$$\{\mu 4\}=\{Out(M3);In(M2)\}$$
$$CC2(\tilde{n})=M2[\![\overline{h2}\langle Hreq\rangle\{\mu 3\}.h2(msg)\{\mu 4\}.([msg=Hden]CC2(\tilde{n})+[msg=Hgra](\overline{h2}\langle Creq\rangle\{\mu 3\}.h2(msg2)\{\mu 4\}([msg2=Cgra]\overline{h2}\langle Cons\rangle\{\mu 3\}.CC2(\tilde{n})+[msg2=Hcan]\overline{h2}\langle Cden\rangle\{\mu 3\}.CC2(\tilde{n}))+h2(msg1)\{\mu 4\}[msg1=Hcan]\overline{h2}\langle Cden\rangle\{\mu 3\}.CC2(\tilde{n}))+[msg=Hcan]\overline{h2}\langle Cden\rangle\{\mu 3\}.CC2(\tilde{n}))]\!]$$

(3) 资源端协调器 RC

$RC1$ 和 $RC2$ 同在事务膜 $M3$ 内,共享通道 $h1$ 和 $h2$,它们也有着类似的系统行为.以 $RC1$ 为例,$RC1$ 收到 $CC1$ 发来的预订请求($Hreq$)后,决定是拒绝预定($Hden$)或同意预定($Hgra$).如果拒绝预定($Hden$),$RC1$ 将返回自己的初始状态;如果同意预定($Hgra$),$RC1$ 将等待 $CC1$ 的消费请求($Creq$),并向 $CC2$ 发出取消预订的通知($Hcan$),在确认 $CC2$ 已经限制消费($Cden$)后,才同意 $CC1$ 的消费请求($Cgra$),并返回初始状态.这里,“排他性消费”是由资源端以广播方式向所有其他客户端发出取消预订通知($Hcan$)来实现的,而且客户端也不再区分是否预定成功,都将作出限制自身消费($Cden$)的响应.资源端不必记录每个客户端的状态,使得参与者之间满足松耦合、自治的要求.

$$RC1(\tilde{n})=M3[\![h1(msg)\{\mu 1\}.[msg=Hreq](\overline{h1}\langle Hden\rangle\{\mu 2\}.RC1(\tilde{n})+\overline{h1}\langle Hgra\rangle\{\mu 2\}.h1(msg1)\{\mu 1\}.[msg1=Creq]\overline{h2}\langle Hcan\rangle\{\mu 4\}.h2(msg2)\{\mu 3\}.[msg2=Cden]\overline{h1}\langle Cgra\rangle\{\mu 2\}.RC1(\tilde{n}))]\!]$$
$$RC2(\tilde{n})=M3[\![h2(msg)\{\mu 3\}.[msg=Hreq](\overline{h2}\langle Hden\rangle\{\mu 4\}.RC2(\tilde{n})+\overline{h2}\langle Hgra\rangle\{\mu 4\}.h2(msg1)\{\mu 3\}.[msg1=Creq]\overline{h1}\langle Hcan\rangle\{\mu 2\}.h1(msg2)\{\mu 1\}.[msg2=Cden]\overline{h2}\langle Cgra\rangle\{\mu 4\}.RC2(\tilde{n}))]\!]$$

(4) 整个多业务事务 MBT

整个多业务事务 MBT 的进程是各个进程模块的并发复合,其中,$h1$ 和 $h2$ 是全局事务的私有通道,用来连接需要通信的参与者,彼此交换消息名字.

$$MBT(\tilde{n})=(\nu\ h1,h2)MBT[\![CC1(\tilde{n})\mid CC2(\tilde{n})\mid RC1(\tilde{n})\mid RC2(\tilde{n})]\!].$$

5 MPi-演算进程的模型检验方法

形式化方法不仅是系统规范化描述的数学语言,更是实现其性质验证的技术和工具.符号模型检验器 NuSMV 是一个开源的且具有开放式结构的验证环境框架,常常被用作一个对新的验证算法进行实现、测试和比较的实验环境.它是基于 Kripke 结构的自动符号模型检验器,用于证明以时态逻辑描述的、有限状态系统的性质.另一方面,Pi-演算的语义模型是标号迁移系统(labeled transition system,简称 LTS).本文基于 LTS 与 KS 的映射关系,在语法层上实现 MPi-演算进程模型到自动机模型的转换.为了将现有的符号模型检验技术集成到 MPi-演算进程的模型检验方法中,关键步骤就是从基于标号迁移系统(LTS)的 MPi-演算模型形式化语义到基于 Kripke 结构的符号模型输入语言的转换,如图 11 所示,使用 MPi-演算对跨组织多业务事务进行形式化描述,并将 MPi-演算进程表达式转化为 NuSMV 的输入语言,进而集成开源的符号模型检验工具完成系统属性的验证.

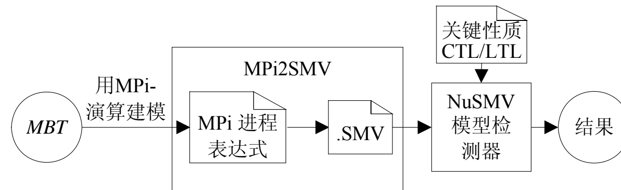

Fig.11 Framework of formal verification for MBT

图 11 多业务事务的形式化验证框架

本文第 3.3 节给出了 Mpi-演算进程对应的标号迁移系统的定义,而符号化模型检验工具 NuSMV 的输入语言(SMV 语言)是一个基于状态的有限状态自动机形式语言,它的行为被定义成 Kripke 结构.

定义 5(Kripke 结构). 假定 AP 为非空原子命题(atomic propositions)的集合,Kripke 结构是一个四元组 $M_k=(S_k,I_k,R,L)$,其中:

- S_k 为有限状态集合;
- $I_k\subseteq S_k$ 表示初始状态集合;
- $R\subseteq S_k\times S_k$ 表示状态间的迁移关系,即$\forall s\in S_k,\exists s'\in S_k$,使得$(s,s')\in R$;
- $L:S_k\to 2^{AP}$ 为标号函数,它把状态映射到 AP 的幂集,即对每个状态 $s\in S_k,L(s)$返回在该状态成立的原子命题的集合.

MPi-演算进程表达式与符号模型检验工具 NuSMV 的输入语言之间的映射关系可以归结到标号迁移系统 LTS 与 Kripke 结构之间的转换.LTS 和 KS 这两种结构非常相似,都可以看作是状态自动机的一种.LTS 中的迁移标号是用来标注动作的发生,描述导致状态变化的动作;而 KS 中的标号函数则是用来标注状态的变化,描述状态如何被迁移修改.现有 LTS 和 KS 这两种结构之间的转换方法各种各样,本文采用的从 LTS 到 KS 的转换方法就是受到文献[27]的启发,这里的转换思想是:将 LTS 中的迁移一分为二,为标号对应的每个可见动作在 KS 中构造一个新的状态,即 LTS 中的变迁关系$(r,\lambda\{\mu\},s)$在 KS 结构中,记$\lambda\{\mu\}$为 a.用两个变迁关系$(r,(r,a,s))$和$((r,a,s),s)$表示,我们将这些新增的状态(r,a,s)称作“与动作相关的状态”,而 LTS 中原有的状态 r,s 等则称作“与进程相关的状态”.

定义 6(从标号迁移系统 *LTS* 到 Kripke 结构 *KS* 的转换). 给定标号迁移系统 $M=(S,I,A,\Delta)$,并且符号$\perp$不在 A 中出现,那么 Kripke 结构可以定义为 $M_k=(S_k,I_k,R,L)$,其中:

- $S_k = S\cup\{(r,a,s)\mid a\in A, r\in S, s\in S, r\xrightarrow{a}s\}$;
- $I_k=I$;
- $R=\{(r,s)\mid r\xrightarrow{\tau}s\}\cup\{(r,(r,a,s))\mid r\xrightarrow{a}s\}\cup\{((r,a,s),s)\mid r\xrightarrow{a}s\}$;
- $AP=A\cup\{\perp\}$,对于 $r,s\in S,a\in A:L(r)=\{\perp\},L(s)=\{\perp\}$且 $L((r,a,s))=\{a\}$.

文献[28]引入一种 L^2TS(doubly labeled transition systems)结构分别表示 LTS 和 KS 两种结构,论证了 LTS 中的迁移标号与 KS 中相邻状态的标号语义上的一致性,并从行为等价的角度解释了这种从 LTS 到 KS 结构转化方法的正确性.另外,对比 LTS 和 KS 两种结构的定义,分析各元素的构造过程可以得到定理 1.

定理 1. 对任意一个标号迁移系统 $M=(S,I,A,\Delta)$,存在一个相应的 Kripke 结构 $M_k=(S_k,I_k,R,L)$,反之亦然.

证明:

(1) 充分性.由定义 6 可以得出,对于一个标号迁移系统 $M=(S,I,A,\Delta)$,可以构造出一个 KS 结构 $M_k=(S_k, I_k,R,L)$,从 LTS 到 KS 的转换目标是“在 KS 中如何表示 LTS 的可见动作标号信息”.这里,通过将每个可见的迁移分成两个连续的迁移,新增加(r,a,s)的状态可以来表示可见动作标号含义,即 LTS 中的迁移(r,a,s)在 KS 结构中用两个迁移关系$(r,(r,a,s))$和$((r,a,s),s)$表示.

(2) 必要性.可以从上述的反过程来分析,从 KS 到 LTS 的转换目标则是“从 KS 中如何还原 LTS 的可见动作标号信息”,将每个状态分成两个相邻的状态.这里,为 KS 结构中每个状态 $s\in S_k$ 构造一个伴随状态 s',目标 LTS 的有限状态集合 $S=S_k\cup\{s'|s\in S_k\}$;KS 结构中原子命题 AP 的幂集对应到 LTS 的可见动作集合,即 $A=2^{AP}\cup\{\perp\}$;LTS 的迁移关系Δ则由两部分组成:在 KS 中,与进程相关的每个状态 $s\in S_k$,对应到 LTS 的迁移关系$(s,\perp,s')$和$(s',L(s),s)$;而与动作相关的状态(r,a,s),若 $L(r)=L(s)$对应 LTS 的迁移关系(r,τ,s),若 $L(r)\neq L(s)$时则是$(r,L(s),s)$.总之,对于一个 KS 结构 $M_k=(S_k,I_k,R,L)$,也可以构造出一个标号迁移系统 $M=(S,I,A,\Delta)$. □

定理 1 进一步说明了这两种结构之间存在一对一的映射关系,互为充要条件.因而我们有如下结论:

推论 1. 采用 Kripke 结构的 SMV 语言描述形式,具有与 MPi-演算进程等价的表达能力.

这个推论也可以从另外一个方面去理解,NuSMV 的输入语言(SMV 语言)被设计用于描述有限状态系统,状态变量的初始状态用“init()”语句来定义,状态的迁移用“next()”语句来实现.即确定状态变量下一时刻的状态,可以看成 MPi-演算中进程之间的迁移关系,因而采用 Kripke 结构的 SMV 语言与 MPi-演算进程所描述的系统可以具有同样的反应式,是等价的有限状态迁移系统.

如何将标准 Pi-演算进程表达式转换为 SMV 语言的描述形式见文献[24],我们在文中给出了输入、输出和哑动作等 3 种基本动作前缀对应的 SMV 脚本样式,并提出了用 SMV 语言编码 Pi-演算进程表达式的一系列规则.在 SMV 中:“与进程相关的状态”集合用变量 state 表示,它的枚举值用 *Pi* 表示;“与动作相关的状态”集合用非空动作的通道名变量(例如 *y_msg*)表示,它的枚举值对应通道上传输的消息名字(例如 x,z);τ用于表示进程外部不可见的内部动作,直接将 LTS 中的状态对应到 *KS* 的状态即可;MPi-演算引入了与进程动作交互相关联的膜活动$\{\mu\}$,因此,在 SMV 中需增加一个有界数组变量 *u_val*[1...*n*]来表示跨组织膜活动序列.

表 3 所示从 MPi-演算进程中的可见动作前缀式到 SMV 语言脚本的转换过程:

(1) 对于输出前缀 $\overline{y}\langle z\rangle\{\mu\}$,*P*1 对应状态 s_1,*P*2 对应状态 s_2.当前状态是 *P*1 时,*state*=*P*1 条件式满足,会同时触发变量 *state* 和 *y* 的变迁,即 $(s_1,(s_1,\overline{y}\langle z\rangle\{\mu\},s_2))\wedge((s_1,\overline{y}\langle z\rangle\{\mu\},s_2),s_2)$,这里的输出动作被写成“赋值表达式” *y_msg*:=*z* 和 *y_msg*:=μ的形式,与目标状态一起作为状态迁移的结果,其含义是进程发出消息 *z* 的同时发生了跨组织膜活动$\{\mu\}$;

(2) 对于输入前缀 $y(z/x)\{\mu\}$,当状态从 *P*1 变迁到 *P*2 时,条件式 *state*=*P*1,*y_msg*=*z* 和 *u_val*[1]=μ必须同时成立,即$(s_1,(s_1,y(x)\{\mu\},s_2))\wedge((s_1,y(x)\{\mu\},s_2),s_2)$,这里的输入动作被写成“条件表达式”*y_msg*=*z* 和 *u_val*[1]=μ的形式,与源状态一起作为状态迁移的前提条件,其含义是进程接收一个消息,若消息为 *z* 且发生了相应的跨组织膜活动$\{\mu\}$,则表现为另一个进程.

Table 3 From the action prefixes of MPi-calculus to SMV scripts

表 3 从 MPi-演算动作前缀式到 SMV 脚本的转换

MPi-演算动作前缀	从 LTS 到 KS	SMV 代码程序
$\overline{y}\langle z\rangle\{\mu\}$	$\forall s_1,s_2\in S,\forall\overline{y}\langle z\rangle\{\mu\}\in A.(s_1\xrightarrow{\overline{y}\langle z\rangle\{\mu\}}s_2)\in\varDelta\Rightarrow$ $s_1,s_2,(s_1,\overline{y}\langle z\rangle\{\mu\},s_2)\in S_k\wedge(s_1,(s_1,\overline{y}\langle z\rangle\{\mu\},s_2))\in R\wedge$ $((s_1,\overline{y}\langle z\rangle\{\mu\},s_2),s_2)\in R\wedge L(s_1)=\{\perp\}\wedge L(s_2)=\{\perp\}\wedge$ $L((s_1,\overline{y}\langle z\rangle\{\mu\},s_2))=\{\overline{y}\langle z\rangle\{\mu\}\}$	VAR *state*: {*P*1,*P*2,…}; *next*(*y_msg*):=*case* *y_msg*: {*z*,…}; *state*=*P*1: *z*; *u_val*: array 1...*n* 1: *y_msg*; of {*u*,…,*null*}; *esac*; ASSIGN *next*(*state*):=*case* *next*(*u_val*[1]):=*case* *state*=*P*1: *P*2; *state*=*P*1: *u*; 1: *state*; 1: *u_val*[1]; *esac*; *esac*;
$y(z/x)\{\mu\}$	$\forall s_1,s_2\in S,\forall y(x)\{\mu\}\in A.(s_1\xrightarrow{y(x)\{\mu\}}s_2)\in\varDelta\Rightarrow\lhd$ $s_1,s_2,(s_1,y(x)\{\mu\},s_2)\in S_k\wedge(s_1,(s_1,y(x)\{\mu\},s_2))\in R\wedge$ $((s_1,y(x)\{\mu\},s_2),s_2)\in R\wedge L(s_1)=\{\perp\}\wedge$ $L(s_2)=\{\perp\}\wedge L((s_1,y(x)\{\mu\},s_2))=\{y(x)\{\mu\}\}$	VAR *state*: {*P*1,*P*2,…}; *u_val*: array 1...*n* of {*u*,…,*null*}; ASSIGN *next*(*state*):=*case* *state*=*P*1 & *y_msg*=*z* & *u_val*[1]=*u*: *P*2; 1: *state*; *esac*;

6 原型实现与实例分析

模型检验的分析技术可以允许自动地检验业务事务是否满足人们需要的各种性质,从而帮助消除设计缺陷,提高软件及业务流程的可信程度.根据本文所提出的 MPi-演算进程模型检验方法,我们设计了原型系统,针对实例验证了事务相关性质,并对文中的方法进行了定性和定量的评价与分析.

6.1 原型实现及示例

图 12 为 MPi2SMV 的系统架构,实现一个从 MPi-演算模型到 SMV 程序代码的自动转换工具.它包括 3 大组件:MPi-演算文本解析器、转换适配器和 SMV 程序产生器:(1) MPi-演算文本解析器用于把 MPi-演算表达式的文本转化为内存表示形式,分为词法分析、语法分析和语义分析(即规则转换)这 3 部分,采用 ANTLR (another tool for language recognition)[29]基于特定语法,产生词法分析程序和语法分析程序,规则转换用 Java 程序语言编写,其处理动作嵌入到词法分析和语法分析的文件中.(2) 转换适配器将 MPi-演算的内存表示形式转换为 SMV

程序的内存表示形式.(3) SMV 程序产生器将 SMV 程序内存表示方式转换为 SMV 程序代码文本.(4) MPi2SMV 的输入为 MPi-演算表达式文件和基于 LTL/CTL 的性质描述文件.输出也为两个文件,它们是 SMV 程序代码文件和 MPi-演算标识轨迹文件.SMV 程序代码文件为 MPi-演算表达式对应的 SMV 源程序,其中包含了性质描述文件中的内容,作为 NuSMV 检测工具的输入;MPi-演算标识轨迹文件用来标识进程状态迁移过程信息,可以辅助分析模型检验时所产生的反例.

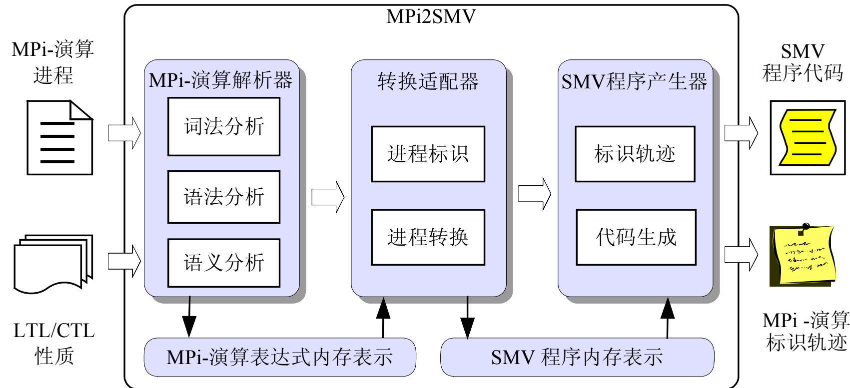

Fig.12 Architecture of MPi2SMV

图 12 MPi2SMV 的系统架构

在上面关于“2 个客户竞争 1 个资源”(2C-1R 型)的典型业务实例中,假设某一资源对外开放提供服务,客户通过 THP 协议访问该资源,存在下面两个类似的场景:

场景 1:一个客户向服务资源发出预订请求(*Hreq*)之后:

(a) 该客户收到“拒绝预定(*Hden*)”通知后,再次发出同样的预定请求(*Hreq*);

(b) 该客户收到“同意预定(*Hgra*)”通知并进行消费后,再次发出同样的预定请求(*Hreq*);

(c) 该客户在没有收到响应通知之前,再次发出同样的预定请求(*Hreq*).

以上 3 种情况中的再次发出同样预定请求,这都属于“重复请求”,对于这种来自同一客户端的重复请求是允许发生的.

场景 2:一个客户向服务资源发出预订请求(*Hreq*)之后,客户收到了“同意预定(*Hgra*)”通知后,在还没有发出消费请求之前,再次发出同样的预定请求.这可能会导致其他用户一直申请不到资源而出现“饿死”的现象,这种情况通常被称作恶意请求.

上述这两种场景非常相似,都是同一客户端对相同服务资源的请求.在开放、动态的互联网环境下,为了加强资源共享,提高客户对资源申请的成功率,允许客户端对资源的重复请求;但是在较复杂的多业务流程设计中,由于人为疏忽或其他偶然因素,可能会将允许重复请求的业务逻辑定义成允许恶意请求,结果不仅没有增强资源共享性,反而可能被某些客户利用,故意长期占用资源.下面对以上重复请求和恶意请求这两种业务逻辑,通过模型检验技术,分析如何验证多业务事务是否满足人们关注的性质.使用 CTL 或 LTL 逻辑对重复请求和恶意请求的性质定义如下:

(1) 重复请求(a).客户端收到“拒绝预定”通知后,不能再次发出预定请求.用计算树逻辑 CTL 描述为:

性质 1. SPEC AG ($h1_msg=Hden \rightarrow !EF\ h1_msg=Hreq$).

(2) 重复请求(b).客户端收到“同意预定”通知并进行消费后,不能再次发出预定请求.用计算树逻辑 CTL 描述为:

性质 2. SPEC AG ($h1_msg=Cons \rightarrow !EF\ h1_msg=Hreq$).

(3) 重复请求(c).客户端在没有收到响应通知之前,不能再次申请.因为该性质涉及两次发出预定请求的时间点,采用线性时序逻辑 LTL 更合适,描述如下:

性质 3. LTLSPEC G ((*h*1_*msg*=*Hgra*|*h*1_*msg*=*Hden*)→(!(*h*1_*msg*=*Hreq*) *S* *h*1_*msg*=*Hreq*)).

这里把任意响应通知描述成客户收到 *Hgra* 或 *Hden* 两个消息的输入动作,事实上,客户还可能收到来自资源端的其他消息,如 *Hcan*,*Cgra* 等.跨组织膜活动总是与动作交互相关联,因而基于交互的膜活动能够灵活地以动作、交互和进程等不同粒度来刻画系统性质,使得性质描述更简单和准确.

- 用膜活动“*u_val*[2]=*In_M*1”可以表示客户端 *CC*1 收到的任意响应通知,这些动作都与“进入事务膜 *M*1”的活动相关联;
- 用膜活动“*u_val*[1]=*Out_M*3”可以表示资源端 *RC* 发出的任意响应通知,这些动作都与“退出事务膜 *M*3”的活动相关联;
- 用膜活动序列“*u_val*[1]=*Out_M*3 & *u_val*[2]=*In_M*1”可以表示所有发生在进程 *RC* 和 *CC*1 之间,从事务膜 *M*3 到 *M*1 的任意响应通知.

显然,上述 3 种膜活动表达式所刻画的任意“响应通知”含义仍存在细微的差别,但这里它们都可以用来描述重复请求(c),因此也可以用下面的性质表示:

性质 4. LTLSPEC G ((*u_val*[2]=*In_M*1)→(!(*h*1_*msg*=*Hreq*) *S* *h*1_*msg*=*Hreq*)).

(4) 恶意请求.客户端收到同意预定通知后到消费请求之前,不能再次发出预定请求.用 LTL 描述为:

性质 5. LTLSPEC G (*h*1_*msg*=*Creq*→(!(*h*1_*msg*=*Hreq*) *S* *h*1_*msg*=*Hgra*)).

这里用 LTL 描述性质时,除了使用常用的将来时态算子(FUTURE)外,如 *G*(globally),*U*(until),*F*(finally)和 *X*(next),还用到了过去时态算子(PAST)中的 *S*(since),其含义为,性质 $p\ S\ q$ 在 t 时刻为真的条件是,q 在 t'时刻($t'\leqslant t$)和 p 在 t''期间($t'<t''\leqslant t$)为真.即,自从过去某时刻 t',性质 q 满足之后至 t 时刻,性质 p 一直保持.

与其他事务提交协议类似,在跨组织多业务事务中应用 THP(tentative hold protocol)协议需要进一步考察系统的安全性和活性[30].安全性描述的是某危险事件在并发系统执行中永远不会发生,而活性描述的是某必需事件在执行过程中一定会发生.

定义 7(安全性(safety requirements)).

(1) 排他性(excludability). 一个资源不可能被两个客户同时消费.

性质 6. SPEC AG !(*h*1_*msg*=*Cons* & *h*2_*msg*=*Cons*).

(2) 响应性(responsibility). 客户的预定请求一定会被响应.

性质 7. SPEC AG (*h*1_*msg*=*Hreq*→*AF*(*u_val*[2]=*In_M*1)).

定义 8(活性(liveness requirements)).

(1) 非阻塞性(non-blocking). 一个资源可以同意多个客户的预订请求.

性质 8. SPEC AG !(*h*1_*msg*=*Hgra* & *h*2_*msg*=*Hgra*).

(2) 非平凡性(non-triviality). 每个客户都有可能进入消费状态.

性质 9. SPEC EF(*h*1_*msg*=*Cons*|*h*2_*msg*=*Cons*).

验证的硬件环境为:CPU Pentium Dual E2180 2.00GHz,内存 1GB;软件环境为:Windows XP SP2,NuSMV 2.4.3.将文中业务实例的 MPi-演算模型通过 MPi2SMV 工具转换成 SMV 语言描述.在 SMV 代码中,除 MPi-演算模型中全部名字状态变量以外,还另外声明了进程状态变量 *Pi*(从 *P*1 到 *P*42),它们共同组成了 MPi-演算模型的标号迁移系统(LTS)的有限状态集合,这些进程状态标识信息记录在 MPi2SMV 工具自动产生的标识轨迹文件中,对于 MPi-演算中一些不方便文本输入的符号,这里采用类似 Pi-演算经典的检验工具 MWB[19]的输入格式,见表 4.进一步用 NuSMV 读入该 SMV 程序代码文件(mbt.smv)并执行后,显示整个系统产生的状态数为 171 460 379($2^{27.3533}$)个,其中有 35 151($2^{15.1013}$)个可达状态,如图 13 所示.

Table 4 State marking for MPi-calculus processes

表 4 MPi-演算进程状态标识

Agent *MBT*
MBT=P1,
P1=MBT[[CC1 | CC2 | RC1 | RC2]],
P2=CC1 | CC2 | RC1 | RC2,

Agent *CC*1
CC1=P4,
P4=M1[['h1<Hreq>{Out_M1;In_M3}.P3]],
P5='h1<Hreq>{Out_M1;In_M3}.P3,
P3=h1(msg){Out_M3;In_M1}.([msg=Hden]CC1+[msg=Hgra]P6
+[msg=Hcan]P7),
P6=P8+P9,
P8='h1<Creq>{Out_M1;In_M3}.P10,
P10=h1(msg2){Out_M3;In_M1}.([msg2=Cgra]P11+[msg2=Hcan]P12),
P11='h1<Cons>{Out_M1;In_M3}.CC1,
P12='h1<Cden>{Out_M1;In_M3}.CC1,
P9=h1(msg1){Out_M3;In_M1}.[msg1=Hcan]P13,
P13='h1<Cden>{Out_M1;In_M3}.CC1,
P7='h1<Cden>{Out_M1;In_M3}.CC1,

Agent *CC*2
CC2=P15,
P15=M2[['h2<Hreq>{Out_M2;In_M3}.P14]],
P16='h2<Hreq>{Out_M2;In_M3}.P14,
P14=h2(msg){Out_M3;In_M2}.([msg=Hden]CC2+[msg=Hgra]P17
+[msg=Hcan]P18),
P17=P19+P20,
P19='h2<Creq>{Out_M2;In_M3}.P21,
P21=h2(msg2){Out_M3;In_M2}.([msg2=Cgra]P22+[msg2=Hcan]P23),
P22='h2<Cons>{Out_M2;In_M3}.CC2,
P23='h2<Cden>{Out_M2;In_M3}.CC2,
P20=h2(msg1){Out_M3;In_M2}.[msg1=Hcan]P24,
P24='h2<Cden>{Out_M2;In_M3}.CC2,
P18='h2<Cden>{Out_M2;In_M3}.CC2,

Agent *RC*1
RC1=P26,
P26=M3[[h1(msg){Out_M1;In_M3}.[msg=Hreq]P25]],
P27=h1(msg){Out_M1;In_M3}.[msg=Hreq]P25,
P25=P28+P29,
P28='h1<Hden>{Out_M3;In_M1}.RC1,
P29='h1<Hgra>{Out_M3;In_M1}.P30,
P30=h1(msg1){Out_M1;In_M3}.[msg1=Creq]P31,
P31='h2<Hcan>{Out_M3;In_M2}.P32,
P32=h2(msg2){Out_M2;In_M3}.[msg2=Cden]P33,
P33='h1<Cgra>{Out_M3;In_M1}.RC1,

Agent *RC*2
RC2=P35,
P35=M3[[h2(msg){Out_M2;In_M3}.[msg=Hreq]P34]],
P36=h2(msg){Out_M2;In_M3}.[msg=Hreq]P34,
P34=P37+P38,
P37='h2<Hden>{Out_M3;In_M2}.RC2,
P38='h2<Hgra>{Out_M3;In_M2}.P39,
P39=h2(msg1){Out_M2;In_M3}.[msg1=Creq]P40,
P40='h1<Hcan>{Out_M3;In_M1}.P41,
P41=h1(msg2){Out_M1;In_M3}.[msg2=Cden]P42,
P42='h2<Cgra>{Out_M3;In_M2}.RC2

```
NuSMV Interactive
NuSMV > read_model -i d:\smv\mbt.smv
NuSMV > go
NuSMV > print_reachable_states
######################################################################
system diameter: 61
reachable states: 35151 (2^15.1013) out of 1.71461e+008 (2^27.3533)
######################################################################
NuSMV > show_vars -s
state : (MBT, P2, P1)
h1_msg : (null, Cden, Cons, Cgra, Creq, Hcan, Hgra, Hden, Hreq)
u_val[1] : (null, In_M2, Out_M2, In_M1, Out_M3, In_M3, Out_M1)
u_val[2] : (null, In_M2, Out_M2, In_M1, Out_M3, In_M3, Out_M1)
h2_msg : (null, Cden, Cons, Cgra, Creq, Hcan, Hgra, Hden, Hreq)
CC1Process.state : (P7, P13, P9, P12, P11, P10, P8, P6, CC1, P3, P5, P4)
CC2Process.state : (P18, P24, P20, P23, P22, P21, P19, P17, CC2, P14, P16, P15)
RC1Process.state : (P33, P32, P31, P30, P29, RC1, P28, P25, P27, P26)
RC2Process.state : (P42, P41, P40, P39, P38, RC2, P37, P34, P36, P35)
NuSMV >
```

Fig.13 Reachable states and state variables

图 13 可达状态及状态变量

图 14 给出了上述性质的模型检验结果.值得一提的是,图中显示性质的序号与文中的编号不一致.NuSMV 中把 CTL 性质和 LTL 性质的检测结果分开放置,每条性质下方的括号中有 3 项内容,分别表示该性质的逻辑类型、检测结果和是否存在反例以及对应的反例编号等,我们对结果分析如下.

(1) 性质 1(序号 000)和性质 2(序号 001)的检测结果都为 False,它们对应的反例 *T*1 和 *T*2(由于篇幅的限制,这里没有列出反例的内容)说明,客户端收到拒绝预定通知后或收到同意预定通知并进行消费后,这两种情况下都可以再次发出预定请求.这与资源端允许客户发出重复请求是相符的.

(2) 性质 3(序号 006)的检测结果为 True,表示客户端在没有收到响应通知之前,不能再次发出预定请求.显然,这与系统允许重复请求是相违背的,因为客户发出的请求消息可能会由于网络延迟而被资源端遗漏,这种情况下应该允许客户重复发出请求.分析原因不难发现,性质 3 中对客户可能收到的任意“响应通知”的描述只限于 $Hgra$ 或 $Hden$ 两种响应,忽略了其他可能存在的响应消息.相比而言,性质 4(序号 007)中用膜活动“$u_val[2]=In_M1$”来表示该客户端收到的任意响应通知更加简洁而准确.该性质检测结果为 False,存在反例 $T5$ 表示此时系统允许客户发出重复请求.

(3) 性质 5(序号 008)的检测结果为 True,表示客户端收到同意预定通知后到消费请求之前,不能再次发出预定请求,这与系统中禁止客户恶意请求是相符的.

(4) 性质 6(序号 002)的检测结果为 True,表示同一个资源不能被两个客户同时消费,这与 THP 协议所规定的“排他性消费”是相符合的;性质 7(序号 003)关于响应性的检测结果却为 False,存在反例 $T3$ 表示客户的预定请求不一定会被资源服务端及时地响应,因为 THP 协议并没有规定资源端对客户请求进行响应的时间期限,这也提醒我们在采用 THP 协议时,可以根据需要另外设置响应请求时效的系统参数.

(5) 性质 8(序号 004)的检测结果都为 False,存在反例 $T4$ 表示一个资源可以同意多个客户的预订请求,这与 THP 协议所规定的非阻塞预定是相符合的;性质 9(序号 005)的检测结果为“True”,表示每个客户会终止,且不希望总是消费不成功,它们各自有可能进入消费成功的状态.

```
NuSMV Interactive
NuSMV > show_property
**** PROPERTY LIST [ Type, Status, Counterex. Trace ] ****
--------------------------  PROPERTY LIST  -------------------------
000 : AG (h1_msg = Hden -> !(EF h1_msg = Hreq))
        [CTL          False          T1  ]
001 : AG (h1_msg = Cons -> !(EF h1_msg = Hreq))
        [CTL          False          T2  ]
002 : AG !(h1_msg = Cons & h2_msg = Cons)
        [CTL          True           N/A ]
003 : AG (h1_msg = Hreq -> AF u_val[2] = In_M1)
        [CTL          False          T3  ]
004 : AG !(h1_msg = Hgra & h2_msg = Hgra)
        [CTL          False          T4  ]
005 : EF (h1_msg = Cons | h2_msg = Cons)
        [CTL          True           N/A ]
006 :  G ((h1_msg = Hgra | h1_msg = Hden) -> (!(h1_msg = Hreq) S h1_msg = Hreq))

        [LTL          True           N/A ]
007 :  G (u_val[2] = In_M1 -> (!(h1_msg = Hreq) S h1_msg = Hreq))
        [LTL          False          T5  ]
008 :  G (h1_msg = Creq -> (!(h1_msg = Hreq) S h1_msg = Hgra))
        [LTL          True           N/A ]
NuSMV >
```

Fig.14 Result of checking property list

图 14 性质的检测结果

采用模型检验分析技术来检验系统是否满足各种性质,它们既可以是用户需要系统满足的性质(如性质 1~性质 5 的检测结果说明,文中的多业务事务实例系统满足允许重复请求和禁止恶意请求的性质,验证了业务流程设计中不存在混淆两种请求含义的情况,为客户公平地参与资源竞争提供了可靠的保证),也可以是系统本身拥有的属性(如性质 6~性质 9 的检测结果表明,该系统满足非平凡性、排他性消费和非阻塞预定等特性,而不满足响应性的要求,则说明模型检验方法能够帮助人们发现规范中存在的问题).

6.2 方法评价与分析

现实生活中的多业务事务往往涉及多个资源提供者,如图 1 所示旅行计划业务场景中,整个旅行计划需要预定机票、宾馆和出租车这 3 种服务资源.随着业务参与者的并发分量增加,系统的有穷状态模型状态数量往往呈指数增长,那么上述模型检验方法就会面临状态空间爆炸的问题.为了能够有效地应用模型检验方法,除了

在模型检测的过程应用不同方法以提高效率、缓解内存空间外,还有许多研究的目标是减少模型本身的复杂性,主要的方法是通过抽象和分解把复杂系统的验证转化成模型检验可以处理的问题[31].抽象方法[32]是抽掉原来模型中与待验证性质无关的信息,用尽可能少的状态刻画系统的运作过程;而分解方法[33]则是将一个系统分成若干部分,先验证各子系统的局部性质,然后把这些性质组合起来获得系统的全局性质.

图 1 旅行计划业务场景中只出现了一个旅行代理,省略了可能参与这些资源竞争的其他客户.从参与者自身角度来看,业务协调只是发生在自己和别人之间双方的竞争.即要么自己获得资源,广播通知其他参与者取消预订,要么别人获得资源,自己的预订被取消.另一方面,THP 协议中协调器具有开放、自治和松耦合的特性,换句话说,每个客户端协调器 *CC* 不关心其他具体还有多少客户端参与同一资源的竞争,而是将它们全部视为其他客户.对应的资源端协调器也不必记录每个客户端的状态,而只要区别当前客户和其他客户即可.综合以上两个方面来看,“2 个客户竞争 *m* 个资源”(记作 2C-*m*R 型)业务可以被看成是“多个客户竞争多个资源”(记作 *n*C-*m*R 型)业务的特例,2C-*m*R 型业务保留了 *n*C-*m*R 型业务具备的事务相关全部性质.接下来以图 1 所示旅行计划业务场景为例,讨论如何用本文的方法对其进行建模与验证.

由于多业务事务是发生在前事务阶段的协调行为,因而建模时不需要考虑服务之间的事务依赖性.例如,机票、宾馆和出租车这 3 种不同服务之间既不存在资源竞争关系,执行结果也不会相互影响.这样一来,我们可以采用分解的方法,利用业务是由不同类别服务资源组合而成的结构特点对系统进行分解,每种服务的相关协调行为单独作为一个子系统,先检验各子系统的性质,然后综合推导出整个系统的性质:(1) 图 15 是预定机票子系统的进程流图,表示旅行代理从 3 个供选航班中选择一个合适的航班.其中有两个客户端作用域 *M*1 和 *M*2,分别代表该旅行代理和其他代理,3 个供选航班资源虽然可能属于不同组织单元,但它们同属于一个原子事务,因此,这里将所有资源端实例都放在同一个事务作用域 *M*3 内.数字 1 和数字 2 表示两个客户端,小写字母 *a*,*b*,*c* 用来区分 3 个资源.一个资源有两个资源端实例,每个资源端实例对应一个客户端实例(如航班 Flight *A* 的实例是 *Ra*1 和 *Ra*2,分别对应客户端实例 *Ca*1 和 *Ca*2).(2) 两个客户端与资源端之间的通信由事务 *M*1,*M*2 与 *M*3 之间的通信来完成,客户端实例可以通过各自的链接独立地向对应的资源端实例发出预定请求(*Hreq*),这满足协议规定的非阻塞预定要求.(3) 所有资源端实例可以共享全部与事务膜进程相连的通道,当某资源端实例接到消费请求(*Creq*)时,会向其他预定同一资源的客户端实例发出取消预定的通知(*Hcan*),这也满足协议规定的排他性消费要求.与 2C-1R 型业务不同的是,由于是多个资源中选择一个资源,这时还需向已消费成功的客户端的其他实例发出取消预定的通知.例如,*Ra*1 接到 *Ca*1 的消费请求时,资源端会向 *Ca*2,*Cb*1 和 *Cc*1 发出取消预定的通知.(4) 同理可以建模预定宾馆和出租车的子系统,子系统模型中都描述了资源尝试性预定机制,模型组合后规定系统一起提交整个事务,即可满足协议的尝试性占有要求.通过类似第 6.1 节的性质检验,我们可以验证这些子系统具有与 2C-1R 型业务相同的系统性质.(5) 因为所有子系统都包含了原系统中与多业务事务相关的全部语义,在各个子系统之间不存在相互影响的情况下,可以使用模型分解方法中的简单组合策略,即若要对原系统全局性质进行模型检验,可以转化为验证各子系统在任意环境下都满足的性质[33].

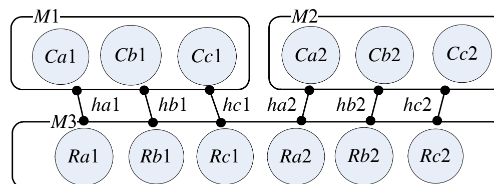

Fig.15 Flow graph of a example for type 2C-*m*R

图 15 2C-*m*R 型业务实例进程流图

业务类型中的 *m*R 表示客户需要多个资源,采用如上所述的分解方法对多次资源消费过程进行分而治之,可以有效地降低模型检验的复杂度.这样一来,每次资源消费的子系统模型的规模则是影响整个复杂多业务事务模型检测效率的关键因素.而子系统又是描述从多个同类资源中选择一个资源消费的过程,因此,同类资源的

数量则形成了子系统模型状态空间的决定因素.每个资源都可能被选择,一旦被某客户消费,则需要向另一个客户和该客户的其他实例发出取消预定的通知,这些协调行为语义对于每个资源都是一样的.随着同类资源的并发分量增加,也会导致系统状态数明显增加.这里仍可以利用模型分解的方法将每个资源相关的协调行为视为一个更小的子系统,进一步缩减模型检验时的问题规模.

表 5 给出了随着同类资源数量增加,业务子系统的 MPi-演算形式化语义所对应的系统状态空间参数变化情况,其中:变量数(number of variable)是指系统的状态变量数;系统直径(system diameter)即搜索深度,可以理解为从初始状态到最远可达状态之间的距离;可达状态(reachable state)和有效迁移(fair transitions)的数量则反映了系统状态空间规模的大小,也直接影响到模型检验的效率.图 16 是这些系统参数受资源数量影响相应的曲线图,从图中分析可以得到:图 16(a)系统状态变量数和图 16(b)搜索深度随着资源数量的增加呈线性增长;图 16(c)可达状态数和图 16 (d)有效迁移数这两个指标随资源数量的增加也接近线性增长,在一定程度上降低了模型检验的复杂性.

Table 5 Subsystem scales for different number of resources

表 5 不同数量资源的子系统规模

资源数	变量数	系统直径	可达状态	有效迁移
1	9	61	$2^{15.1013}$	$2^{17.4232}$
2	11	77	$2^{16.2197}$	$2^{18.5416}$
3	13	81	$2^{16.4649}$	$2^{18.7868}$
4	15	85	$2^{16.6934}$	$2^{19.0154}$
5	17	89	$2^{16.9071}$	$2^{19.2291}$
6	19	93	$2^{17.1075}$	$2^{19.4294}$
7	21	97	$2^{17.2960}$	$2^{19.6179}$
8	23	101	$2^{17.4737}$	$2^{19.7956}$

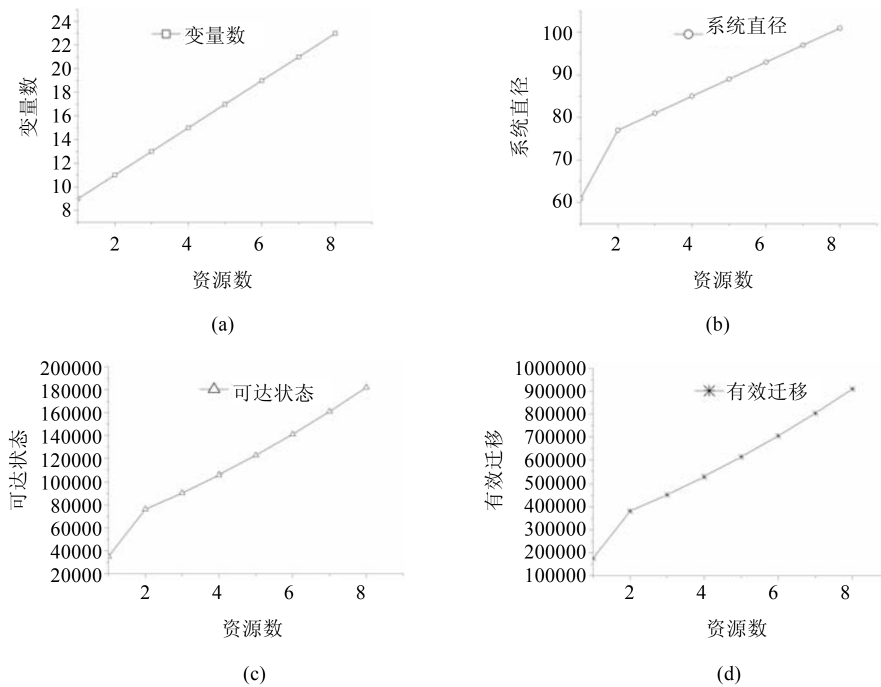

Fig.16 Effect of the number of resources on the state space

图 16 资源数量对状态空间的影响

如上所述,基于模型分解的组合模型检验方法利用系统的组合结构特点对问题进行分而治之,是一种有效的应对状态爆炸问题的验证方法.先检验各子系统的性质,然后运用综合推理的方法导出系统整体性质,避免了直接对整个系统状态空间进行验证所面临的状态爆炸问题.上述实例的各个子系统之间不存在相互影响,因而能够使用简单组合策略来验证各子系统在任意环境下都满足的性质.对于流程结构更复杂的业务系统,它们各子系统之间可能存在相互影响,我们仍可以采用条件假设的组合策略来实现问题的分而治之,如假设/保证组合策略[33,34].即若两个子系统之间存在相互影响,为了验证它们并发组合系统满足性质α和β,可以转换为验证一个子系统是否在满足性质α的所有环境下都满足另一性质β的问题.

7 结束语

面向服务的跨组织多业务事务发生在松耦合、自治和开放的 SOA 环境,业务流程参与者之间跨组织协调行为的复杂性与不一致性也更为突出.为了对多业务流程设计的正确性进行有效验证,保证跨组织多业务事务获得正确的执行和一致的结果,本文从 Pi-演算的名字、进程和动作交互这 3 个要素引入了事务语义,将进程的动作交互与跨组织膜活动建立动态关联,提出跨组织多业务事务建模方法.并基于等价自动机转换思路提出了从标号迁移系统 LTS 到 Kripke 结构的转换方法,能够有效地集成现有各类模型检验技术,实现对跨组织多业务流程及事务的验证.文中主要针对 2 个客户竞争 1 个资源的典型业务实例给出了建模与验证各阶段的具体实现方法,对于复杂多业务事务系统,由于其模型的状态数量往往随参与者并发分量的增加而呈指数增长,为了缓解状态空间爆炸问题,最后用实例说明了通过抽象或分解方法可以有效地减少模型本身的复杂性,把复杂业务系统的性质验证问题转化成简单业务系统来处理.

本文方法在建模阶段没有考虑多业务事务协调行为的时间和优先级的因素,如资源预定时效以及资源分配策略等,进一步的工作是在建模过程中引入时间和优先级等非功能属性的描述,在模型转换过程中采用与本文类似的方法,通过引入新的状态变量类型支持对应属性的模型检验.另一方面,验证方法所使用的实例仍较为简单,仅用于说明该方法的可行性和有效性.在下一阶段,我们把不同类型的抽象、分解等优化方法与多业务流程的控制模式相结合,以此降低模型检验的时间和空间复杂度,增强验证方法的适用性.此外,我们还将在现有工作基础上提高验证工具的自动化支持程度,利用进程标识轨迹文件进一步分析反例信息.

致谢 感谢审稿专家提出的宝贵意见和建议,并向对本文工作给予支持和建议的同行表示感谢.

References:

[1] He JF, Jin Z, Li XD. The preface of special issue on service-oriented computing. Journal of Software, 2007,18(12):2965−2966 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/18/2965.htm [doi: 10.1360/josl82965]

[2] Yu J, Han YB. Service-Oriented Computing—Principles and Applications. Beijing: Tsinghua University Press, 2006. 178−191 (in Chinese).

[3] Curbera F, Khalaf R, Mukhi N, Tai S, Weerawarana S. The next step in Web services. Communications of the ACM, 2003,46(10): 29−34. [doi: 10.1145/944217.944234]

[4] Dalal S, Temel S, Little M, Potts M, Webber J. Coordinating business transactions on the Web. IEEE Internet Computing, 2003, 7(1):30−39. [doi: 10.1109/MIC.2003.1167337]

[5] Papazoglou MP. Web services and business transactions. World Wide Web, 2003,6(1):49−91. [doi: 10.1023/A:1022308532661]

[6] Srinivasan K, Malu PG, Moakley G. Automatic multibusiness transactions. IEEE Internet Computing, 2003,7(3):66−73. [doi: 10.1109/MIC.2003.1200303]

[7] Roberts J, Collier T, Malu P, Srinivasan K. Tentative hold protocol part 2: Technical specification. 2001. http://www.w3.org/TR/tenthold-2/

[8] Park J, Choi KS. An adaptive coordination framework for fast atomic multi-business transactions using Web services. Decision Support Systems, 2006,42(3):1959−1973. [doi: 10.1016/j.dss.2006.05.004]

[9] Limthanmaphon B, Zhang YC. Web service composition transaction management. In: Schewe KD, Williams H, eds. Proc. of the 15th Australasian Database Conf. (ADC 2004). Sydney: Australian Computer Society, Inc., 2004. 171−179.

[10] Younas M, Chao KM, Lo CC, Li YS. An efficient transaction commit protocol for composite Web services. In: Proc. of the 20th Int'l Conf. on Advanced Information Networking and Applications (AINA 2006). Washington: IEEE Computer Society, 2006. 591−596. [doi: 10.1109/AINA.2006.84]

[11] He JF. Transaction calculus. In: Proc. of the 11th IEEE High Assurance Systems Engineering Symp. (HASE 2008). Washington: IEEE Computer Society, 2008. 4−12. [doi: 10.1109/HASE.2008.67]

[12] Qiu ZY, Wang SL, Pu GG, Zhao XP. Semantics of BPEL4WS-like fault and compensation handling. In: Fitzgerald J, Hayes IJ, Tarlecki A, eds. Proc. of the13th Int'l Symp. of Formal Methods Europe (FM 2005). LNCS 3582, Berlin, Heidelberg: Springer-Verlag, 2005. 350−365. [doi: 10.1007/11526841_24]

[13] Qi ZW, You JY. The formal specification and verification of transaction processing in Web services by membrane calculus. Chinese Journal of Computers, 2006,29(7):1137−1141 (in Chinese with English abstract).

[14] Milner R, Parrow J, Walker D. A calculus of mobile processes. Part I. Information and Computation, 1992,100(1):1−40. [doi: 10.1016/0890-5401(92)90008-4]

[15] Mazzara M, Lucchi R. A framework for generic error handling in business processes. Electronic Notes in Theoretical Computer Science, 2004,105(10):133−145. [doi: 10.1016/j.entcs.2004.05.002]

[16] Bocchi L, Laneve C, Zavattaro G. A calculus for long-running transactions. In: Najm E, Nestmann U, Stevens P, eds. Proc. of the 6th IFIP Int'l Conf. on Formal Methods for Open Object-Based Distributed Systems (FMOODS 2003). LNCS 2884, Belin, Heidelberg: Springer-Verlag, 2003. 124−138. [doi: 10.1007/978-3-540-39958-2_9]

[17] Butler M, Ferreira C. An operational semantics for StAC, a language for modelling long-running business transactions. In: Nicola RD, Ferrari GL, Meredith G, eds. Proc. of the 6th Int'l Conf. on Coordination Models and Languages (COORDINATION 2004). LNCS 2949, Belin, Heidelberg: Springer-Verlag, 2004. 87−104. [doi: 10.1007/978-3-540-24634-3_9]

[18] Ripon SH. Process algebraic support for Web service composition. ACM SIGSOFT Software Engineering Notes, 2010,35(2):1−7. [doi: 10.1145/1734103.1734118]

[19] Victor B, Moller F. The mobility workbench—A tool for the Pi-calculus. In: Dill DL, ed. Proc. of the 6th Int'l Conf. on Computer Aided Verification (CAV'94). LNCS 818, Belin, Heidelberg: Springer-Verlag, 1994. 428−440.

[20] Ferrari GL, Gnesi S, Montanari U, Pistore M. A model-checking verification environment for mobile processes. ACM Trans. on Software Engineering and Methodology, 2003,12(4):440−473. [doi: 10.1145/990010.990013]

[21] Liu Y, Müller S, Xu K. A static compliance-checking framework for business process models. IBM Systems Journal, 2007,46(2): 335−361. [doi: 10.1147/sj.462.0335]

[22] Van de Aalst WMP. Pi-Calculus versus Petri nets: Let us eat "humble pie" rather than further inflate the "Pi hype". BP Trends, 2005,3(5):1−11.

[23] Yuan M, Huang ZQ, Cao ZN, Xiao FX. An extended π-calculus and its transactional bisimulation. Journal of Computer Research and Development, 2010,47(3):541−548 (in Chinese with English abstract).

[24] Yuan M, Huang ZQ, Li X, Yan Y. Towards a formal verification approach for business process coordination. In: Proc. of the 8th IEEE Int'l Conf. on Web Services (ICWS 2010). Washington: IEEE Computer Society, 2010. 361−368. [doi: 10.1109/ICWS. 2010.100]

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

[26] Alves A, Arkin A, Askary S, Bloch B, Curbera F, Goland Y. Web services business process execution language (WSBPEL Version 2.0). 2007.

[27] Guessarian I, De Nicola R, Vaandrager F. Action versus state based logics for transition systems. In: Guessarian I, ed. Proc. of the LITP Spring School on Theoretical Computer Science. LNCS 469, Belin, Heidelberg: Springer-Verlag, 1990. 407−419. [doi: 10.1007/3-540-53479-2_17]

[28] de Nicola R, Vaandrager F. Three logics for branching bisimulation. Journal of the ACM, 1995,42(2):458−487. [doi: 10.1145/ 201019.201032]

[29] Parr T. The Definitive ANTLR Reference: Building Domain-Specific Languages. Lewisville: Pragmatic Bookshelf, 2007.

[30] Gray J, Lamport L. Consensus on transaction commit. ACM Trans. on Database Systems, 2006,31(1):133−160. [doi: 10.1145/1132863.1132867]

[31] Lin HM, Zhang WH. Model checking: Theories, techniques and applications. Acta Electronica Sinica, 2002,30(12A):1907−1912 (in Chinese with English abstract).

[32] Clarke EM, Grumberg O, Long DE. Model checking and abstraction. ACM Trans. on Programming Languages and Systems, 1994, 16(5):1512−1542. [doi: 10.1145/186025.186051]

[33] Grumberg O, Long DE. Model checking and modular verification. ACM Trans. on Programming Languages and Systems, 1994, 16(3):843−871. [doi: 10.1145/177492.177725]

[34] Wen YJ, Wang J, Qi ZC. Compositional model checking and compositional refinement checking of concurrent reactive systems. Journal of Software, 2007,18(6):1270−1281 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/18/1270.htm [doi: 10.1360/jos181270]

附中文参考文献:

[1] 何积丰,金芝,李宣东.面向服务的计算专刊前言.软件学报,2007,18(12):2965−2966. http://www.jos.org.cn/1000-9825/18/2965.htm [doi: 10.1360/josl82965]

[2] 喻坚,韩燕波.面向服务的计算——原理和应用.北京:清华大学出版社,2006.178−191.

[13] 戚正伟,尤晋元.基于细胞膜演算的 Web 服务事务处理形式化描述与验证.计算机学报,2006,29(7):1137−1141.

[23] 袁敏,黄志球,曹子宁,肖芳雄.一种扩充的π-演算及事务性等价关系研究.计算机研究与发展,2010,47(3):541−548.

[31] 林惠民,张文辉.模型检测:理论、方法与应用.电子学报,2002,30(12A):1907−1912.

[34] 文艳军,王戟,齐治昌.并发反应式系统的组合模型检验与组合精化检验.软件学报,2007,18(6):1270−1281. http://www.jos.org.cn/1000-9825/18/1270.htm [doi: 10.1360/jos181270]

袁敏(1977－),男,湖南郴州人,博士生,CCF 会员,主要研究领域为服务计算,软件工程,形式化验证.

胡军(1973－),男,博士,副教授,CCF 会员,主要研究领域为软件工程,软件验证,嵌入式系统软件.

黄志球(1965－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件工程,形式化方法,服务计算,知识工程.