一种服务消息交互行为的元建模方法

周文博[1,2] 刘洪佳[3] 刘 磊[1] 张 鹏[1,2,4] 吕 帅[1,2,4]

(吉林大学计算机科学与技术学院 长春 130012)[1]

(符号计算与知识工程教育部重点实验室(吉林大学) 长春 130012)[2]

(辽阳石化公司研究院 辽阳 111000)[3] (吉林大学数学学院 长春 130012)[4]

摘 要 为了提高服务消息接口的规范性和交互行为的正确性,提出了一种服务消息交互的元建模方法。基于工作流模型对服务进行建模,通过对消息操作模式予以分析,给出了接口形式化表示和接口相容性检查方法。采用推理规则和递归函数刻画消息传递的语义,讨论了服务交互时各种环境的变化情况。实例分析表明,该方法可以规范服务接口模式,有效地对消息的交互情景进行建模,进而保障服务建模的可靠性。

关键词 服务交互,元建模,工作流模型,消息接口,形式语义

中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.04.006

Meta-modeling Approach of Message Interaction in Service

ZHOU Wen-bo[1,2] LIU Hong-jia[3] LIU Lei[1] ZHANG Peng[1,2,4] LV Shuai[1,2,4]

(College of Computer Science and Technology, Jilin University, Changchun 130012, China)[1]

(Key Laboratory of Symbolic Computation and Knowledge Engineering (Jilin University), Ministry of Education, Changchun 130012, China)[2]

(Research Institute of PetroChina Liaoyang Petrochemical Company, Liaoyang 111000, China)[3]

(College of Mathematics, Jilin University, Changchun 130012, China)[4]

Abstract To improve the normalization of message interfaces and the correctness of interactive behaviors in service, a meta-modeling approach of service message interaction was proposed. Firstly, workflow model is utilized to model service. Then by analyzing the message operation pattern, a method about interface representing and compatibility checking is given. Finally, inference rules and recursive functions are used to describe the semantics of message transmission, and the change of environments is discussed at the same time. Case analysis shows that the proposed method can normalize service interface patterns, model message interaction situation effectively, and ensure the reliability of service modeling.

Keywords Service interaction, Meta-modeling, Workflow model, Message interface, Formal semantic

1 引言

随着云计算、物联网、大数据等新技术的不断发展,一切皆服务的思想引起了人们的广泛重视。服务科学作为一门新兴的复合交叉型学科,已超出了传统服务的固有定义,对各行各业产生了深远的影响。尤其在计算机领域,软件即服务[1]、计算即服务等一系列新概念大大扩展了服务的内涵,使得软件理论与技术应用中的服务计算得到了飞速的发展。

为了完善服务计算相关的理论基础,许多学者对服务的元建模、计算模式进行了深入的研究。Broyd 等人基于分布式系统的 FOCUS 理论,提出了一种新的服务形式化模型,并针对服务和组件给出了基本的技术规范和开发步骤[2]; Riccobene 等人提出了一种为面向服务的应用建模的形式化框架,包括基于模型的精确可执行语言 SCA-ASM 和用于早期服务设计评估的工具,使得服务组件可以进行离线模拟和评估,或作为抽象实现与其他组件实现一起执行[3]; Belkhir 等人提出了参数自动机的形式化模型,探讨了其在服务组合方面的应用,促进了服务组合的理论研究[4]。由于不同方面的服务具有不同的形式,将形式化方法引入到服务建模、服务交互的过程中,有助于规范服务和统一概念,避免二义性。

服务交互是具有动态性质的服务组合。对于服务组合问题,不同科研工作者从不同方面给出了相应的方法。Chen 等

到稿日期:2015-11-30 返修日期:2016-02-27 本文受国家自然科学基金(61300049),教育部高等学校博士学科点专项科研基金(20120061120059),中国博士后科学基金(2011M500612),吉林省重点科技攻关项目(20130206052GX),吉林省自然科学基金项目(20150101054JC),吉林省青年科研基金(20140520069JH)资助。

周文博(1991-),男,硕士生,CCF 学生会员,主要研究方向为服务计算与软件形式化,E-mail:zhouwb14@mails.jlu.edu.cn;刘洪佳(1983-),女,助理工程师,主要研究方向为软件工程;刘 磊(1960-),男,教授,博士生导师,主要研究方向为语义网与本体工程、软件安全与云计算;张 鹏(1986-),男,博士,讲师,主要研究方向为软件形式化(通信作者);吕 帅(1981-),男,博士,讲师,主要研究方向为智能规划与自动推理。

人针对面向服务计算模式中信息共享和过程中介的难题，结合语义 Web 和社会化网络技术，显式地定义了服务间的交互关系（简称服务关系），并将它们应用到服务注册中心的组织和构建中，便利了服务计算过程[5]。Wang 等人提出了一种面向大规模功能个性化需求的服务组合方法，其支持服务的大规模个性化定制，达到优化的成本有效性和客户满意度[6]。然而，这些研究往往简化了服务交互接口的具体概念，更没有从消息与接口相结合的角度考虑服务交互的建模问题。服务的优点主要是动态性与透明性，而消息接口的规范定义是服务可以动态绑定的前提，透明封装的服务业务流程也主要依靠接口与外界交互。目前还较为缺乏关于消息接口的形式化建模与交互规范。

本文的贡献在于：1）基于工作流模型对服务进行建模；2）给出了消息接口形式化定义和相容性检查方法；3）对服务交互的形式语义进行了刻画与分析，规范了服务消息交互行为。

本文第 2 节介绍服务计算和形式化方法的基础知识；第 3 节给出服务工作流相关的定义；第 4 节说明消息接口的相容性检查方法；第 5 节刻画基于消息接口的服务交互过程的形式语义；第 6 节进行了实例研究，以说明本文方法的有效性；最后是本文结论与下一步工作的展望。

2　基础知识

服务和规范往往是密不可分的。服务是现实业务的一种抽象，通常具有规范定义的接口。规范则意味着精确化和形式化，可以避免二义性。本节将分别介绍服务计算和形式化方法的相关知识，消息接口交互建模方法结合了这两方面知识。

2.1　服务计算

对于服务计算，不同学者从不同方面给出了定义。从软件系统设计与开发的角度出发，荷兰科学家 Mike 认为“服务计算是一种以服务为基本元素进行应用系统开发的方式”[7]；从学科的角度出发，张良杰认为“服务计算是一门跨计算机与信息技术、商业管理与咨询服务的基础学科，其目标在于利用服务科学和服务技术消除商业服务和信息技术服务之间的鸿沟”；从服务技术的应用角度出发，北卡罗来纳州立大学 Munindar 和南加州大学 Michael 认为“服务计算是集服务概念、服务体系架构、服务技术和服务基础设施于一体，指导如何使用服务的技术集合”[8]；从分布式计算的角度出发，Orlowska 等人认为“服务计算是从面向对象和面向构件的计算演化而来的一种分布式计算模式，它使得分布在企业内部或跨越企业边界的不同商业应用系统能实现快捷、灵活的无缝集成与相互协作”[9]。本文认为，服务计算（services computing）是一种以服务为基本组成单元来支持分布、异构环境的计算范型。

2.2　形式化方法

形式化方法（formal methods）基于数学的理论研究计算机的相关问题。形式化方法主要包括：基于模型的形式化方法、基于逻辑的形式化方法、基于进程代数的形式化方法、基于网络的形式化方法。其中，基于模型的形式化方法以自动机、Z 方法和 VDM 方法等为代表，利用已知的数学抽象（元组、集合、序列、映射等）为系统的结构特征、状态特征和行为特征构建模型；基于逻辑的形式化方法以计算树逻辑、时态逻辑、模态逻辑为基础，利用逻辑形式表示对系统进行描述、推理和验证；以 CSP，CCS 和 π 演算等为主的基于进程代数的形式化方法对并发过程具有较强的描述能力，能够较好地描述分布式系统的相关特性；基于网络的形式化方法中最具代表性的是 Petri 网方法，Petri 网具有严格的数学定义和可视化的图形表示，能够根据数据流显式地给出并发系统模型，另外还有谓词变换网等方法。形式化方法能够在早期发现服务系统中的不一致、歧义、不完全、错误等情况，提高系统的可靠性，被广泛应用于各种系统的建模和验证，如分布式软件系统[10]、工作流系统[11]等。

本文拟通过对上述研究对象与方法的分析，提出服务消息交互的元建模方法，即构建基于工作流的服务模型，检查消息接口的相容性，并利用逻辑规则和函数描述对服务交互过程的语义进行准确刻画。整体过程如图 1 所示。

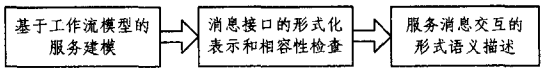

图 1　服务消息交互的元建模过程

3　服务工作流

服务是一种基于关系的交互活动，这种交互至少发生在一个服务提供方和一个服务消费方之间，以实现一个确定的商业目标或解决方案[12]。服务消费方可以是用户，也可以是另一个服务。当服务消费方仍为服务时，便产生了服务的交互问题。由于服务具有工作流的特性，本节基于工作流对服务进行建模。首先给出工作模块的定义：

定义 1（工作模块）　工作模块可以用语法规则进行定义：

$$B::=\perp \mid LB$$

$$L::=Mevent \mid par;id(MList) \mid process;id(\mu,\gamma)$$

$$MList::=Mevent \mid Mevent,MList$$

$$Mevent::=in(M) \mid out(M)$$

工作模块可以是空，也可以是由若干切片顺序连接成的块。$\perp$ 通常表示最小元，这里表示空，有时候将起始工作模块或终止工作模块记为空。一个切片 L 是简单消息操作 $Mevent$、并行消息操作 $par;id(Mevent,\cdots,Mevent)$ 或处理操作 $process;id(\mu,\gamma)$，其中 M 表示消息，μ 表示消息环境，γ 表示数据环境，简单消息操作包括输入操作 $in(M)$ 和输出操作 $out(M)$。实际上，$process;id(\mu,\gamma)$ 的粒度可大可小，一个子功能可以用一个处理操作表示，也可以用多个处理操作表示。

定义 2（服务工作流）　服务工作流是一个六元组 $SWF=(B,S,F,R,\mu,\gamma)$，其中：

1）B 是工作模块的有穷集合；

2）$S\in B$ 是初始工作模块；

3）$F\in B$ 是终止工作模块；

4）$R:B\times C\rightarrow B$ 是模块连接关系，C 表示转移条件；

5）$\mu\subseteq M$ 是消息环境；

6）γ 是数据环境，存储了本地数据信息。

服务工作流是对服务的抽象。工作模块是服务工作流的基本组成单位，通常表示一个相对独立的子功能。初始工作模块标识了工作流的开始，而终止工作模块标识了工作流的结束。模块连接关系将各个工作模块连接起来，通过转移条件决定一个工作模块的后继模块。这个条件可能与消息环境有关，即根据返回的消息决定执行的模块，也可能是根据处理操作产生的数据结果发生转移，转移条件通常为消息或数据变量的集合。消息环境记录了当前的消息情况，改变消息环境的方式有 3 种：1)通过一个输入操作在消息环境中增加一个消息；2)通过一个输出操作在消息环境中减少一个消息；3)通过一个处理操作更新消息环境。数据环境存储了与消息交互无关的本地数据。

4 消息接口的相容性检查

不同服务间的交互依赖于消息的传递，而消息的传递是有时序约束的。对于两个消息来说，串行的消息必须在前一个消息传递完成后后一个消息才可以传递；并行的消息可以同时传递。消息的传递具有一定的模式，且该模式可以间接作为服务接口。基于消息的交互主要由消息操作触发，本文利用形式语言描述的消息操作模式如下：

$MEP::=seq(Itemseq)$

$Itemseq::=Item|Item,Itemseq$

$Item::=Mevent|par:id(MList)$

$MList::=Mevent|Mevent,MList$

$MEvent::=in(M)|out(M)$

MEP 表示消息操作模式，$Itemseq$ 表示操作项的序列。一个操作项 $Item$ 可以是简单消息操作 $MEvent$，也可以是并行消息操作 $par(MList)$。简单消息操作分别记为消息输入操作 $in(M)$ 和消息输出操作 $out(M)$。seq，par 是两个关键字，分别表示顺序和并行。$seq(e_1,e_2,e_3)$ 表示 e_1,e_2,e_3 这 3 个消息操作具有顺序发生关系，即先发生 e_1，再发生 e_2，最后发生 e_3。$par:id(e_1,e_2,e_3)$ 表示 e_1,e_2,e_3 这 3 个消息操作可以并行发生。例如，一个消息操作模式 $mp_x=seq(in(m_1),par:a1(out(m_2),out(m_3),out(m_4)),in(m_5),out(m_6))$。

定义 3(消息操作序列) 消息操作序列是一个由消息操作组成的序列 $MEQ=\langle e_1,e_2,\cdots,e_i,\cdots,e_n\rangle$，$e_i\in\{in(m_i),out(m_i)\}$ 且 $1\leqslant i\leqslant n$。

一个消息操作模式实际上对应了一个消息操作序列的集合，记 $\rho:MEP\rightarrow\{MEQ\}$ 为计算消息操作模式所对应的消息序列的算子。对于上述给定的 mp_x，则有 $\rho(mp_x)=\{\langle in(m_1),out(m_2),out(m_3),out(m_4),in(m_5),out(m_6)\rangle,\langle in(m_1),out(m_2),out(m_4),out(m_3),in(m_5),out(m_6)\rangle,\langle in(m_1),out(m_3),out(m_4),out(m_2),in(m_5),out(m_6)\rangle,\langle in(m_1),out(m_3),out(m_2),out(m_4),in(m_5),out(m_6)\rangle,\langle in(m_1),out(m_4),out(m_2),out(m_3),in(m_5),out(m_6)\rangle,\langle in(m_1),out(m_4),out(m_3),out(m_2),in(m_5),out(m_6)\rangle\}$。消息操作序列集即为消息接口的语义指称物。

对于消息 m_i，称 $in(m_i)$ 和 $out(m_i)$ 操作契合。用一个函数判断操作契合，记为 $Agree:MEvent\times MEvent\rightarrow BOOL$。操作契合可以分为以下 4 种情况：1)同一消息的传入、传出操作相契合，即 $Agree(in(m_i),out(m_i))=$True；2)同一消息的两个传入操作不契合，即 $Agree(in(m_i),in(m_i))=$False；3)同一消息的两个传出操作不契合，即 $Agree(out(m_i),out(m_i))=$False；4)不同消息的操作之间均不契合，如 $Agree(in(m_i),out(m_j))=$False。

定义 4(消息操作序列契合) 已知消息操作序列 $meq_1=\langle e_1,e_2,\cdots,e_i,\cdots,e_m\rangle$ 和 $meq_2=\langle e_1',e_2',\cdots,e_i',\cdots,e_n'\rangle$，若 $m=n$ 且对于 $1\leqslant i\leqslant n$，$Agree(e_i,e_i')=$True，则称两个消息操作序列契合。

消息操作序列契合说明它们之间的消息动作是完全“一致”的。为了方便说明，用一个函数判断两个消息操作序列契合，记为 $Accord:MEQ\times MEQ\rightarrow BOOL$。例如已知两个消息操作序列 $meq_x=\langle in(m_2),out(m_4),in(m_5)\rangle$ 和 $meq_y=\langle out(m_2),in(m_4),out(m_5)\rangle$，则 $Accord(meq_x,meq_y)=$True。

定义 5(消息接口) 已知一个服务工作流，称该服务工作流从初始工作模块到终止工作模块的最短路径对应的消息操作模式为消息接口。

由服务工作流可以得到消息接口。消息接口的生成算法如下。

算法 1 消息接口生成算法

输入：服务工作流 SWF=(B,S,F,R,μ,γ)

输出：消息接口 INF

1. 初始化序列 Q 为空
2. 令 P 为初始到终止工作模块的最短路径上的工作模块序列
3. WHILE P 不为空
4. B ← P 的第一个工作模块
5. WHILE B ≠⊥
6. L ← B 的第一个切片
7. IF L 是简单消息操作或并行消息操作
8. 将 L 加入到 Q 的 L 序列尾
9. 删掉 B 的第一个切片
10. 删掉 P 的第一个模块
11. 返回 seq(Q)

定义 6(消息接口相容性) 已知两个消息接口 inf_1 和 inf_2，若存在消息操作序列 $meq_i\in inf_1$ 和 $meq_j\in inf_2$，使得 $Accord(meq_i,meq_j)=$True，则称接口 inf_1 和接口 inf_2 是相容的。

接口相容意味着两个接口之间存在着一个契合的消息操作序列，说明具有这两个接口的服务可以绑定，进而交互。接口相容性检查的形式化表示为：$\exists meq_i\in inf_1\wedge meq_j\in inf_2.Accord(meq_i,meq_j)=$True。例如，假设 $mp_y=seq(out(m_1),in(m_2),in(m_3),in(m_4),out(m_5),in(m_6))$，则由于存在 $\langle out(m_1),in(m_2),in(m_3),in(m_4),out(m_5),int(m_6)\rangle\in\rho(mp_y)$ 与 $\langle in(m_1),out(m_2),out(m_3),out(m_4),in(m_5),out(m_6)\rangle\in\rho(mp_x)$ 契合且 $\rho(mp_y)\neq\rho(mp_x)$，因此消息操作模式(或接口)mp_y 与 mp_x 相容。

5 基于消息传递的服务交互语义

本文采用一种半结构化的语义描述方法，利用规则和函数对服务的交互过程进行描述。为了简化，将两个服务间的交互抽象为服务与中间环境的交互，中间环境仅对这两个服

务可见。服务交互的过程是消息有序传递的过程，而消息的有序传递则依赖于操作启动和模块执行。操作启动主要是指开始一个操作。一个操作开始后，可能会因为条件无法满足而不能执行，如输入消息事件在中间环境中没有指定接收的消息时将被阻塞。模块执行是当满足转移条件时，转到新的模块执行，本文中模块转换的条件为当前消息环境或数据环境。在描述服务交互语义前，给出一些环境变量[1]的定义：

1）当前工作模块 $CB\in B$，是服务工作流正在执行的唯一模块。

2）消息环境 $\mu=\{m|m\in M\}$，是当前服务的消息缓冲区，服务成功读取一个消息时将其存入消息缓冲区。

3）数据环境 γ 是本地存储的非消息的信息集合。

4）中间环境 $EW=\{m|m\in M\}$，是服务直接交互的外部对象。服务将消息发送到中间环境，并从中间环境接收所需的消息。当两个服务通过相容性检查时，建立两者专用共享的中间环境。

将交互环境记为 $\tau=(CB,EW,\mu,\gamma)$。对于服务当前工作模块的监控，引入环境主规则：

$CB=B_i\wedge Judge(\mu,\gamma,C)\wedge(B_i\times C\rightarrow B_j)\rightarrow Cal:\underline{\perp,B_j}\uparrow【CB=B_j】$

$CB=B_i\wedge B_i=F\rightarrow\perp$

当满足转换规则的条件，即当前状态是某个模块转换条件的前件模块，同时满足消息环境或数据环境所要求的条件时，发生模块转换，进入当前模块的执行处理阶段。*Cal* 是执行算子；↑是一个分隔符，【 】括号里的内容是对环境变量的操作，即更改环境变量的值，且一个【 】整体操作是原子的。需要特别说明的是，环境主规则是规则，而规则意味着满足前件条件，即可发生后件。与函数不同的是，规则不需要主动调用，而是在条件满足时自然发生。

Cal 给出了工作执行过程，其实质是一个递归函数。该函数的第一个元素是工作切片，第二个元素是调用模块。事实上，这个函数的出口是由环境变量中的当前工作模块确定的。一旦当前模块与调用模块不符，则说明当前模块不需要继续执行，发生执行转移，当前的所有工作切片应该迅速收敛（这用⊥表示停止）。$Judge:\mu\times\gamma\times C\rightarrow BOOL$ 是一个条件判断函数，用于判断当前的消息环境、数据环境和转移条件是里否满足一种关系，这种关系与具体实现有关。当满足条件时，发生模块执行转移，改变当前工作模块，同时开始执行该工作模块，即 $Cal:\underline{\perp,B_j}$。第二个规则为终止规则，当前工作模块为终止模块时，终止。

$Cal:\underline{\perp,B}=(CB=B\rightarrow Cal:\underline{First(B),B},\perp)$

$Cal:\underline{in(m_i),B}=(Waitfor:m_i\in EW\wedge CB=B\rightarrow Cal:\underline{Next(in(m_i)),B}\uparrow【EW\backslash m_i,\mu\cup\{m_i\}】,\perp)$

$Cal:\underline{out(m_i),B}=(CB=B\rightarrow\underline{Cal:Next(out(m_i)),B}\uparrow【EW\cup\{m_i\},\mu\backslash m_i】,\perp\uparrow【EW\cup\{m_i\},\mu\backslash m_i】)$

$Cal:\underline{par:id(e_1,e_2,\cdots,e_k),B}=(CB=B\rightarrow Cal:par:id(Cal:\underline{e_1,B},Cal:\underline{e_2,B},\cdots,Cal:\underline{e_k,B}),\perp)$

$Cal:\underline{par:id(\perp,\perp,\cdots,\perp),B}=(CB=B\rightarrow Cal:\underline{Next(par:id(\perp,\perp,\cdots,\perp)),B},\perp)$

$Cal:\underline{process:id(\mu,\gamma),B}=(CB=B\rightarrow Cal:Next(\underline{process:id(\mu,\gamma)),B}\uparrow【\tau[process:id(\mu,\gamma)/(\mu,\gamma)]】,\perp)$

其中，函数 $First:B\rightarrow L$ 表示取模块的第一个执行切片，函数 $Next:L\rightarrow L$ 表示取切片的下一个切片。这里描述了 6 种情况，分别为切片触发、消息输入、消息输出、消息并行发生、消息并行结束、数据处理。切片触发是指刚开始某个模块所需进行的切片触发，若调用模块与当前模块一致，则获取模块的第一个切片并继续执行该切片。消息输入和消息输出分别是单个消息的接收和发送过程。其中，消息输入可能会发生阻塞，即 $Waitfor:m_i\in EW$。阻塞函数 *Waitfor* 的实现如下：$Waitfor:m_i\in EW=(m_i\in EW\rightarrow\text{True},Waitfor:m_i\in EW)$。当消息到位、阻塞解除时，继续执行下一个切片，同时从中间环境中删去消息，在消息环境中加入消息。消息输出没有阻塞动作，作用为在中间环境中加入消息，从消息环境中删除消息。并行性是消息传递的一个重要特点，消息并行发生是同时发生多个消息操作的情况。例如一个消息操作为 $par:x1(in(m_1),in(m_3),out(m_4),in(m_5))$，其中 $in(m_1),in(m_3),out(m_4),in(m_5)$ 均没有后继切片。若调用模块为 B_1，且处理前 $\mu_i=\{m_4,m_6\}$，$EW=\{m_1\}$ 在处理执行期间的当前状态 $CB=B_1$ 一直未发生改变，则其一种情景的语义[2]为：

$$\xrightarrow[\mu_i=\{m_4,m_6\},\gamma_i,EW=\{m_1\}]{CB_i=B_1}Cal:\underline{par:x1(in(m_1),in(m_3),out(m_4),in(m_5)),B_1}\tag{1}$$

$$Cal:\underline{par:x1(Cal:in(m_1),B_1,Cal:in(m_3),B_1,Cal:out(m_4),B_1,Cal:in(m_5),B_1),B_1}\tag{2}$$

$$\xrightarrow[\mu_i=\{m_1,m_6\},\gamma_i,EW=\{m_4\}]{CB_i=B_1}Cal:\underline{par:x1(\perp,Cal:in(m_3),B_1,\perp,Cal:in(m_5),B_1),B_1}\tag{3}$$

$$\xrightarrow[\mu_i=\{m_1,m_6\},\gamma_i,EW=\{m_4,m_6\}]{CB_i=B_1}Cal:\underline{par:x1(\perp,Cal:in(m_3),B_1,\perp,Cal:in(m_5),B_1),B_1}\tag{4}$$

$$\xrightarrow[\mu_i=\{m_1,m_6,m_6\},\gamma_i,EW=\{m_3\}]{CB_i=B_1}Cal:\underline{par_1(\perp,Cal:in(m_3),B_1,\perp,\perp),B_1}\tag{5}$$

$$\xrightarrow[\mu_i=\{m_1,m_3,m_5,m_6\},\gamma_i,EW=\{\}]{CB_i=B_1}Cal:\underline{par_1(\perp,\perp,\perp,\perp),B_1}\tag{6}$$

$$\xrightarrow[\mu_i=\{m_1,m_3,m_5,m_6\},\gamma_i,EW=\{\}]{CB_i=B_1}\perp\tag{7}$$

---

[1] 这里环境的含义是指当前服务可读写。

[2] 为了说明并行过程，仅抽取了一个服务的执行动作。

对于该情况，消息并行执行过程中未发生模块转换。在步骤(2)和步骤(3)，并行发生了消息操作 $in(m_1)$ 和 $out(m_4)$。在步骤(2)和步骤(3)的执行过程中，由于中间环境中没有消息 m_5，$Cal:\underline{in(m_5)},B_1$ 发生接收消息阻塞。直到步骤(4)，环境中间件中出现了 m_5，例如在步骤(4)前另一个服务的 $out(m_5)$ 操作完成，两服务间的中间环境中出现了 m_5，$Cal:\underline{in(m_5)},S_1$ 可以继续进行。同理，$Cal:\underline{in(m_3)},B_1$ 也发生了阻塞情况。在步骤(4)期间，中间环境中的 m_4 被另一个服务作为输入取走，所以到步骤(5)时中间环境中不存在 m_4。整个消息操作过程的实质是环境的不断改变，以满足内部实现细节的需要。

当转换条件满足且存在对应的模块转换关系时，模块发生转换，前一模块(即调用模块)相关的消息操作应终止。对于上例，在执行完步骤(2)且 $CB=B_1\wedge(\{m_1\}\subseteq SE)\wedge(B_1\times\{m_1\}\rightarrow B_2)$ 成立时，发生模块转换，原来的消息操作迅速收敛并停止，启动一个新的执行。实际上，这种情况可以模拟由接收消息确定的模块转换。

$$\xrightarrow[\mu_i=\{m_1,m_6\},\gamma_i,EW=\{m_4\}]{CB_i=B_1} Cal:\underline{par:x1(in(m_1),in(m_3),out(m_4),in(m_5))},B_1 \quad (1)$$

$$Cal:\underline{par:x1(Cal:in(m_1),B_1,Cal:in(m_3),B_1,Cal:out(m_4),B_1,Cal:in(m_5),B_1)},B_1 \quad (2)$$

$$\xrightarrow[\mu_i=\{m_1,m_6\},\gamma_i,EW=\{m_4\}]{CB_i=B_2} Cal:\underline{par:x1(\perp,\perp,\perp,\perp)},B_1 \quad (3)$$

$$\xrightarrow[\mu_i=\{m_1,m_6\},\gamma_i,EW=\{m_4\}]{CB_i=B_2} \perp \quad (4)$$

$$Cal:\underline{\perp},B_2 \quad (1)$$

$$\xrightarrow[\mu_i=\{m_j,\cdots,m_k\},\gamma_i,EW=\{m_v,\cdots,m_w\}]{CB_i=B_2} \cdots\cdots \quad (2)$$

6 实例分析

实际生活中有许多服务交互的实例，如共享高性能计算机资源。本节举一个实例来说明本文方法的有效性。在该例中，假定将一台具有较大存储容量的计算机作为一个服务对外提供，数据提交服务需要对其进行访问。两个服务间的交互如图 2 所示。

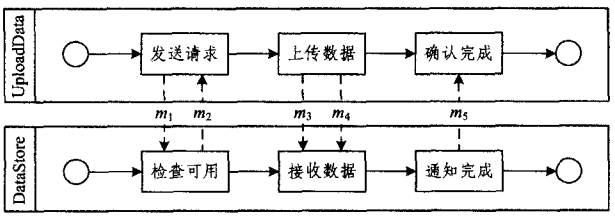

图 2 数据存储服务交互的例子

(1)服务建模

消息集 $M=\{m_1,m_2,m_3,m_4,m_5,m_6\}$。

对 UploadData 服务进行建模。服务 UploadData 有 5 个工作模块，分别将其记为，b_0：初始模块，b_1：发送请求，b_2：上传数据，b_3：确认完成，b_4：终止模块。记 $S_{UD}=(B_1,S_1,F_1,R_1,\mu_1,\gamma_1)$，其中：$B_1=\{b_0,b_1,b_2,b_3,b_4\}$；$S_1=b_0$；$F_1=b_4$；$R_1=\{b_0\times\{\}\rightarrow b_1,b_1\times\{m_2\}\rightarrow b_2,b_2\times\{m_3,m_4\}\rightarrow b_3,b_3\times\{m_5\}\rightarrow b_4\}$；$\mu_1=\{m_1,m_3,m_4\}$；$\gamma_1=\{\}$。

对 DataStore 服务进行建模。服务 DataStore 也有 5 个工作模块，分别记为，b_0'：起始模块，b_1'：检查可用，b_2'：接收数据，b_3'：通知完成，b_4'：终止模块。记 $S2=(B_2,S_2,F_2,R_2,\mu_2,\gamma_2)$，则：$B_2=\{b_0',b_1',b_2',b_3',b_4'\}$；$S_2=b_0'$；$F_2=b_4'$；$R_2=\{b_0'\times\{\}\rightarrow b_1',b_1'\times\{m_2\}\rightarrow b_2',b_2'\times\{m_3,m_4\}\rightarrow b_3',b_3'\times\{m_5\}\rightarrow b_4'\}$；$\mu_2=\{m_2,m_5\}$；$\gamma_2=\{\}$。

(2)接口相容性检查

分析可知，$INF_{UD}=seq(out(m_1),in(m_2),out(m_3),out(m_4),in(m_5))$，$INF_{DS}=seq(in(m_1),out(m_2),par:x1(in(m_3),in(m_4)),out(m_5))$。令 $s_1=\langle out(m_1),in(m_2),out(m_3),out(m_4),in(m_5)\rangle\in\rho(INF_{sc})$，$s_2=\langle in(m_1),out(m_2),in(m_3),in(m_4),out(m_5)\rangle\in\rho(INF_{ups})$，则 $Accord(s_1,s_2)=$True。服务 1 和服务 2 相容。

(3)交互过程

服务 UploadData 的执行语义：

$CB_1=b_0\wedge\{\}\subseteq\mu_1\wedge(b_0\times\{\}\rightarrow b_1)\rightarrow Cal:\underline{\perp},b_1\uparrow[CB_1=b_1]\rightarrow Cal:\underline{First(b_1)},b_1\rightarrow Cal:\underline{out(m_1)},b_1\rightarrow\underline{Cal:Next(out(m_1))},b_1\uparrow[EW\cup\{m_1\},\mu_1\backslash m_1]\rightarrow\underline{Cal:in(m_2)},b_1\rightarrow Waitfor:m_2\in EW\rightarrow\underline{Cal:Next(in(m_2))},b_1\uparrow[EW\backslash m_2,\mu_1\cup\{m_2\}]\rightarrow CB_1=b_1\wedge\{m_2\}\subseteq\mu_1\wedge(b_1\times\{m_2\}\rightarrow b_2)\rightarrow Cal:\underline{\perp},b_2\uparrow[CB_1=b_2]\rightarrow Cal:\underline{First(b_2)},b_2\rightarrow Cal:\underline{out(m_3)},b_2\rightarrow\underline{Cal:Next(out(m_3))},b_2\uparrow[EW\cup\{m_3\},\mu_1\backslash m_3]\rightarrow Cal:\underline{out(m_4)},b_2\rightarrow\underline{Cal:Next(out(m_4))},b_2\uparrow[EW\cup\{m_4\},\mu_1\backslash m_4]\rightarrow CB_1=b_2\wedge(\mu_1\cap\{m_3,m_4\}=\emptyset)\wedge(b_2\times\{m_3,m_4\}\rightarrow b_3)\rightarrow Cal:\underline{First(b_3)},b_3\rightarrow Cal:\underline{in(m_5)},b_3\rightarrow Waitfor:m_5\in EW\rightarrow\underline{Cal:Next(in(m_5))},b_3\uparrow[W\backslash m_5,\mu_1\cup\{m_5\}]\rightarrow CB_1=b_3\wedge\{m_5\}\subseteq\mu_1\wedge(b_3\times\{m_5\}\rightarrow b_4)\rightarrow Cal:\perp,b_4\uparrow[CB_1=b_4]\rightarrow CB_1=b_4\wedge b_4=F_1\rightarrow\perp$。

服务 DataStore 的执行流程：

$CB_2=b_0'\wedge\{\}\subseteq\mu\wedge(b_0'\times\{\}\rightarrow b_1')\rightarrow Cal:\underline{\perp},b_1'\uparrow[CB_2=b_1']\rightarrow Cal:\underline{\perp},b_1'\rightarrow Cal:\underline{First(b_1')},b_1'\rightarrow Cal:\underline{in(m_1)},b_1'\rightarrow Waitfor:m_1\in EW\rightarrow\underline{Cal:Next(in(m_1))},b_1'\uparrow[EW\backslash m_1,\mu_2\cup\{m_1\}]\rightarrow Cal:\underline{out(m_2)},b_1'\rightarrow\underline{Cal:Next(out(m_2))},b_1'\uparrow[EW\cup\{m_2\},\mu_2\backslash m_2]\rightarrow CB_2=b_1'\wedge(\mu_2\cap\{m_2\}=\emptyset)\wedge(b_1'\times\{m_2\}\rightarrow b_2')\rightarrow Cal:\underline{First(b_2')},b_2'\rightarrow Cal:\underline{par:x1(in(m_3),in(m_4))},b_2'\rightarrow Cal:\underline{par:x1(Cal:in(m_3),b_2',Cal:in(m_4),b_2')},b_2'\rightarrow Cal:\underline{par:x1(Waitfor:m_3\in EW,Waitfor:m_4\in EW)},b_2'\rightarrow Cal:\underline{par:x1(Cal:Next(in(m_3)),b_2'}\uparrow[EW\backslash m_3,\mu_2\cup\{m_3\}],Cal:\underline{Next(in(m_4))},b_2'\uparrow[EW\backslash m_4,\mu_2\cup\{m_4\}]),b_2'\rightarrow Cal:\underline{par:x1(\perp,\perp)},b_2'\rightarrow CB_2=b_2'\wedge(\{m_3,m_4\}\subseteq\mu_2)\wedge(b_2'\times\{m_3,m_4\}\rightarrow b_3')\rightarrow Cal:\underline{First(b_3')},b_3'\rightarrow Cal:\underline{out(m_5)},b_3'\rightarrow\underline{Cal:Next(out(m_5))},b_3'\uparrow[EW\cup\{m_5\},\mu_2\backslash m_5]\rightarrow CB_1=b_4'\wedge$

$\mu_2 \cap \{m_5\} = \emptyset \wedge (b_3' \times \{m_5\} \rightarrow b_4') \rightarrow Cal: \underline{\perp, b_4} \uparrow [CB_2 = b_4] \rightarrow CB_2 = b_4 \wedge b_4' = F_2 \rightarrow \perp$。

结束语　本文从消息传递过程的角度研究了服务消息交互行为的元建模方法。首先基于工作流模型对服务进行建模；其次给出了接口相容性的检查方法；再结合推理规则和递归函数，描述与分析了服务交互行为的语义，并探讨了消息并行的情况；最后利用数据存储的实例对本文方法的有效性进行了说明。与此同时，我们在研究中还发现，结合规则和函数的形式化描述方法有助于刻画有监控需求的现实问题。下一步将会继续规范复杂服务交互过程，尝试把多消息接口检查问题转换到经典方法上对问题进行求解，并对多个服务间的交互验证做进一步的研究。

参 考 文 献

[1] TSAI W T, BAI X Y, HUANG Y. Software-as-a-service (SaaS): perspectives and challenges[J]. Science China Information Sciences, 2014, 57(5): 1-15.

[2] BROY M, KRÜGER I H, MEISINGER M. A formal model of services[J]. ACM Transactions on Software Engineering and Methodology, 2007, 16(1): 5.

[3] RICCOBENE E, SCANDURRA P. A formal framework for service modeling and prototyping[J]. Formal Aspects of Computing, 2014, 26(6): 1077-1113.

[4] BELKHIR W, CHEVALIER Y, RUSINOWITCH M. Parametrized automata simulation and application to service composition [J]. Journal of Symbolic Computation, 2015, 69: 40-60.

[5] CHEN S Z, FENG Z Y, WANG H. Service Relations and its Application in Services-oriented Computing[J]. Chinese Journal of Computers, 2010, 33(11): 2068-2083. (in Chinese)
陈世展，冯志勇，王辉. 服务关系及其在面向服务计算中的应用[J]. 计算机学报，2010，33(11)：2068-2083.

[6] WANG Z J, XU F, XU X F. Service network planning method for mass personalized functional requirements[J]. Journal of Software, 2014, 25(6): 1180-1195. (in Chinese)
王忠杰，徐飞，徐晓飞. 支持大规模个性化功能需求的服务网络构建[J]. 软件学报，2014，25(6)：1180-1195.

[7] PAPAZOGLOU M, GEORGAKOPOULOS D. Introduction to a special issue on service oriented computing[J]. Communication of ACM, 2003, 46(10): 25-28.

[8] SINGH M P, HUHNS M N. Service-oriented Computing: Semantics, Processes, Agents[M]. John Wiley & Sons, 2006.

[9] ORLOWSKA M E, WEERAWARNA S, PAPAZOGLOU M P, et al. Service-oriented Computing[C]// First International Conference Service-oriented Computing, Trento, Italy (ICSOC 2003). Preface, LNCS 2910, 2003.

[10] ZHANG C, DUAN Z H, TIAN C, et al. Modeling Verification and Test of Interactive Behaviors in Distributed Software Systems[J]. Journal of Computer Research and Development, 2015, 52(7): 1604-1619. (in Chinese)
张琛，段振华，田聪，等. 分布式软件系统交互行为建模，验证与测试[J]. 计算机研究与发展，2015，52(7)：1604-1619.

[11] SROKA J, HODDERS J, MISSIER P, et al. A formal semantics for the Taverna 2 workflow model[J]. Journal of Computer and System Sciences, 2010, 76(6): 490-508.

[12] ZHANG L J. EIC editorial; Introduction to the body of knowledge areas of services computing[J]. IEEE Transactions on Services Computing, 2008, 1(2): 62-74.

(上接第 20 页)

列给出了程序片段中这些程序在实际程序中出现的频率。CCodeExtractor 函数提取方法是通过指定的条件，在 AST 树上收集满足条件的代码片段，能够处理各种类别的代码。除了条件的指定外，变换过程不需要程序员的参与，实现了函数的自动化提取。

结束语　本文提出了一种自动化的代码提取的方法 CCodeExtractor，其将 C 程序中代码片段提取成单独的函数调用，一方面便于观察大规模程序局部片段的特征，另一方面能够方便程序的分析优化工作。另外，我们在 LLVM 上实现了一个循环提取的工具，实验表明该工具适用于不同规模的程序，并且能够进行正确的程序变换。

参 考 文 献

[1] FOWLER M. Refactoring: improving the design of existing code [M]. Pearson Education India, 1999.

[2] Coderefactoring[OL]. http://en.wikipedia.org/wiki/Code_refactoring.

[3] HAFIZ M, OVERBEY J. OpenRefactory/C: An infrastructure for developing program transformations for C programs[C]// Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity. ACM, 2012: 27-28.

[4] HAFIZ M, OVERBEY J, BEHRANG F, et al. OpenRefactory/C: An infrastructure for building correct and complex C transformations[C]// Proceedings of the 2013 ACM Workshop on Refactoring Tools. ACM, 2013: 1-4.

[5] Microsoft Visual Studio[OL]. http://www.wholetomato.com/features/ feature-refactoring.asp#extract.

[6] Extract method In Eclipse[OL]. https://sourcemaking.com/refactoring/ extract-method.

[7] JEFFREY L. Overbey, Farnaz Behrang, and Munawar Hafiz. A foundation for refactoring C with macros[C]// 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE). New York, NY, USA, 2014: 75-85.

[8] LATTNER C, ADVE V. LLVM: A compilation framework for lifelong program analysis & transformation[C]// International Symposium on Code Generation and Optimization, 2004 (CGO 2004). IEEE, 2004: 75-86.

[9] LATTNER C. LLVM and Clang: Next generation compiler technology[C]// The BSD Conference. 2008: 1-2.

[10] LLVM extractCode[OL]. http://llvm.org/docs/doxygen/html/CodeExtractor 8cpp _source.html.

[11] PROKAJ J. C code for SIFT feature point extraction[OL]. http://www.pudn.com/downloads93/sourcecode/graph/texture_mapping/detail365370.html.