Web 服务组合动态演化的实例可迁移性

宋 巍 马晓星 吕 建
(南京大学计算机软件新技术国家重点实验室 南京 210093)
(南京大学计算机软件研究所 南京 210093)

摘 要 组合网络上既有 Web 服务以构造新的增值服务正逐渐成为一种主流软件形态.而 Web 服务组合常需进行修改演化以优化服务组合的内部业务流程,适应开放的工作环境.在服务组合演化时,为让尽可能多的执行中的服务组合实例享受到新业务流程带来的好处,应尽可能地将其动态地迁移到新流程下继续执行.同时为避免实例迁移到新流程后引发死锁等动态演化错误,需对这些迁移实例加以约束.文中提出了一个服务组合动态演化过程框架,在此框架下形式化地定义了一种新的实例可迁移性标准,并给出了相应的判定算法.与已有的可迁移性标准相比,该标准在确保不会产生动态演化错误的同时,可允许更多的实例迁移.最后,通过一个旅行代理的服务组合案例,说明了文中工作的有效性和可行性.

关键词 服务组合;动态演化;实例级迁移;可迁移性标准;Petri 网
中图法分类号 TP311 **DOI 号**: 10.3724/SP.J.1016.2009.01816

Instance Migration in Dynamic Evolution of Web Service Compositions

SONG Wei MA Xiao-Xing LU Jian
(*State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing* 210093)
(*Institute of Computer Software, Nanjing University, Nanjing* 210093)

Abstract Constructing new, value-added Web services by composing existing ones is stepping into the mainstream of software development. To adapt to the open environments and optimize the inner business processes, Web service compositions often need to evolve dynamically. To maximize the benefit of the evolvement, as many running instances as possible should be migrated to the new version of the service composition specification. However not all running instances can migrate safely because some dynamic change bugs (e.g., deadlocks) could occur after these instances have migrated to the new version. This paper presents a framework to support the dynamic evolution of Web service compositions under which a new formal migratability criterion is proposed. It also proposes the algorithm to check whether an instance of a service composition can migrate to the new version and- if so- the algorithm can also obtain the corresponding target state under the new version where the suspended instance can resume its execution after migration. Compared with the existing migratability criteria in the literature, the authors' criterion allows migrations of more running instances without introducing potential dynamic change bugs. Lastly, a case study based on a travel agency application is also included to demonstrate the effectiveness and feasibility of the proposed approach.

Keywords Web service composition; dynamic evolution; instance-level migration; migratability criterion; Petri nets

收稿日期:2009-04-20;最终修改稿收到日期:2009-07-23.本课题得到国家"九七三"重点基础研究发展规划项目基金(2009CB320702)、国家"八六三"高技术研究发展计划项目基金(2007AA01Z178,2009AA01Z117)和国家自然科学基金(60736015,60721002)资助.
宋 巍,男,1981 年生,博士研究生,主要研究方向为服务计算、软件工程与软件方法学.E-mail: songwei@ics.nju.edu.cn.**马晓星**(通信作者),男,1975 年生,博士,教授,主要研究领域为 Internet 软件技术、软件体系结构和软件构件.E-mail: xxm@nju.edu.cn.**吕 建**,男,1960 年生,博士,教授,博士生导师,主要研究领域为软件自动化、并行程序形式化方法、面向对象语言与环境和网构软件.

1 引　言

Web 服务作为一种自主而开放的应用实体,具有松散耦合、平台无关、互操作性强等特点,特别适合在 Internet 环境中发布和使用[1].为保证 Web 服务的可复用性,单个服务的粒度不宜过大,因此,当需要构造复杂的分布式应用时,须将现有的一些 Web 服务有效地整合在一起,以满足用户的需求.这种通过整合已有服务(称为组件服务或伙伴服务)来向最终用户提供增值服务的技术称为 Web 服务组合①.

工业界描述服务组合主要有两种方式[2]:服务编制(orchestration)和服务编排(choreography).服务编制从单个企业或组织的视角定义了一个工作流过程以何种顺序与各个伙伴服务进行交互,从而得到一个新的增值服务.服务编排则从全局的视角描述了参与协作的服务间的对等消息交互(可以看作是服务间的通信协议).服务编排同服务编制的区别在于服务编排没有一个中心控制者,参与协作的各个服务以对等的方式进行交互.在实际应用中,服务编制和服务编排互为补充,服务编排可用在服务组合的设计阶段,用以明确各个参与者在协作中所扮演的角色,并以此指导各组织内部的服务编制.BPEL②、WS-CDL③ 分别是服务编制和服务编排事实上的标准.

在开放、动态、多变的网络环境下,服务组合需要经常调整、演化以应对不断变化的内部策略和外部环境[3-6].服务组合的演化也可发生在两个层面上:(1)编制层面的服务组合演化(各个服务参与者内部的演化);(2)编排层面的服务组合演化(服务间协同结构和交互协议的演化).其中第 1 类演化是基础,第 2 类演化通常需要多个参与服务各自同时实施第 1 类演化来共同实现.本文集中考虑编制层面的服务组合演化问题,而将编排层面的服务组合演化留为进一步的工作.服务编制需要演化的原因主要有组织内业务过程的再工程(纠错、优化等)以及业务过程本身出于商业策略的改变或者为应对新的法律法规而做的变动.注意:局部的服务编制演化不能影响到服务编排中其它的伙伴服务,即需要保证其演化对伙伴服务透明.

在不影响最终用户以及伙伴服务的前提下,Web 服务组合可以自主地演化.但需要确保 BPEL 流程的可替换性,即要求变动之后的 BPEL 流程与变动之前的 BPEL 流程在行为上是相容的[7],从而 BPEL 的变动可以对最终用户和伙伴服务透明.我们称这种服务组合 BPEL 代码版本的变化为服务组合的静态演化,该问题已得到学术界和工业界的关注[5,7-9].然而,在同一服务组合流程下,可能有多个服务组合实例在同时运行着.在 BPEL 流程演化时,对于尚未开始执行的服务组合实例,可以按照静态演化后的 BPEL 流程进行执行.而对于正在执行中的服务组合实例,有以下 3 种不同的处理方法:(1)按照新的 BPEL 流程重新执行,该处理方式放弃了已执行部分的结果,这对最终用户以及伙伴服务来说往往是不可接受的;(2)按照原来的流程继续执行,这种方式不能使当前的服务组合实例享受到新流程带来的好处;(3)尽量将当前的服务组合实例动态地迁移到新 BPEL 流程下继续执行.本文采用第 3 种处理方式,我们称这种处理方式为服务组合的动态演化.

我们的目标是让尽可能多的服务组合实例迁移到新流程下的一个有效的目标状态下恢复执行,然而,并不是所有的服务组合实例都可以迁移.例如,某些服务组合实例迁移到新流程下会引发死锁等动态演化错误,这部分实例是不允许被迁移的.因此,服务组合动态演化的难点在于判定服务组合实例的可迁移性以及如何确定可迁移实例的目标状态.我们可以利用当前服务组合实例的执行历史信息(即已执行活动序列)来判断该组合实例的可迁移性[10].然而,由于 BPEL 只是实现层面的服务组合编程语言,直接在 BPEL 上为每个单独的服务组合实例进行手动的演化分析是非常耗时且易于出错的.因此,本文首先将 BPEL 代码转换为易于处理的形式化模型(本文采用 Petri 网),而后在此模型的基础上进行服务组合的动态演化分析.本文方法的特点在于不仅可以很好地描述 BPEL 中的控制流(即内部业务逻辑),还有助于 BPEL 流程中的数据流分析,从而可以为服务组合实例的可迁移性判定以及目标状态的确定提供必要的帮助.

在本文的方法中,根据当前服务组合实例的已执行活动序列信息,我们提出一个新的服务组合实

① 在本文中,Web 服务与 Web 服务组合可简称为服务与服务组合.

② Alves A et al. Web Services Business Process Execution Language Version 2.0. OASIS Standard, 2007. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf

③ Kavantzas N et al. Web services choreography description language version 1.0. Technique Reports, World Wide Web Consortium, 2005. http://www.w3.org/TR/ws-cdl-10/

例的可迁移性标准，并证明了通过该迁移性标准所获得的服务组合实例的目标状态是有效的(valid)，即不会引入一些动态演化错误(dynamic change bugs)[11]. 与现有工作[3-4,10-15]的可迁移性判定方法相比较，一方面，本文提出的服务组合实例的可迁移性标准综合考虑了服务组合的控制流和数据流，避免了由于将一些原本不可迁移的实例迁移到新流程下而引发的错误；另一方面，本文提出的服务组合实例的可迁移性标准具有更好的柔性和适用性，可以让更多的原本就可迁移的服务组合实例动态地迁移到新流程下继续执行.

本文第 2 节介绍相关工作以及与本文工作的比较；第 3 节对 Petri 网的基本概念进行了介绍；第 4 节提出一个基于模型(Petri 网)的服务组合动态演化过程框架，且在此框架下，提出一个柔性的服务组合实例可迁移性标准及其算法实现；第 5 节通过一个旅行代理的服务组合实例，阐明了本文方法可以方便而有效地支持服务组合的动态演化；最后，总结全文并指出下一步的研究工作.

2 相关工作

在本节中，我们将对服务组合的动态演化、工作流的动态演化以及自适应软件等方面的相关工作进行一下简要的回顾.

在服务组合的动态演化方面，Papazoglou 教授在文献[5]中较为全面地总结了几种常见的服务组合演化类型，并提出了一个面向演化的服务生命周期(lifecycle)方法学以应对演化带来的特殊要求. 然而，该工作并没有对服务组合的动态演化进行具体的讨论. 文献[16]引入了服务演化管理的理念，并对服务(包括组合服务)在演化过程中所产生的各个不同版本的一致性问题进行了研究. 该工作虽然涉及到服务组合的动态演化，但是并没有对服务组合实例的可迁移性进行深入的探讨.

文献[3]关注于服务编制层面的服务组合动态演化. 它提出的服务组合实例的可迁移性判定方法的核心在于判定当前服务组合实例是否与目标模型兼容. 该方法可以处理签证申请等一类组合服务的动态演化，这类组合服务的共同点在于该类服务仅接受消息，而不发送消息. 因此，该方法不能推广到一般的服务组合动态演化中. 而且，该方法在判定当前服务组合实例是否与目标模型兼容时，并没有考虑数据流因素. 此外，该方法只在模型层面考虑问题，并没有涉及到具体的 BPEL 流程.

我们前期工作[4]初步涉及到服务编排层面的动态演化. 在用户需求改变的条件下，当前的服务组合实例必须进行迁移以满足用户新的需求. 此时问题的关键在于求出一个合适的目标状态(简称目标状态)从而可以充分利用已执行部分的结果. 在该工作中，我们提出了目标状态的判定标准，并给出了求解目标状态的算法. 本文工作关注于服务编制层面的服务组合动态演化，并且考虑了数据流对服务组合实例可迁移性的影响，因此本文工作和前期工作是互为补充的.

在工作流研究领域，工作流动态演化的问题得到了较为广泛的关注[10-15]. 为避免工作流实例迁移时所引入的动态演化错误，Aalst 等人在他们的工作[11]中限定演化前后的工作流模型必须满足继承关系(继承关系是借助于分支互模拟来定义的). 该方法虽然可以避免引入动态演化错误，但是不能处理演化前后的工作流模型不满足继承关系的情形，因此不具有一般性. 为此，Aalst 在文献[12]中提出了一种不限定工作流演化类型的工作流迁移方法. 该方法通过比较变化前后的工作流模型，找出工作流模型中所有发生了变化的区域. 只有当工作流实例不在变化区域里执行的时候，才允许工作流实例的迁移. 这种方法可以避免动态演化错误，然而在该方法中，某些工作流实例可能不能在工作流模型发生演化的第一时间进行迁移，从而使得这些工作流实例不能立即享受到流程演化所带来的好处.

借鉴数据库系统中的可串行化理论，Casati 等提出可以利用工作流实例的已执行活动序列信息来判断一个工作流实例是否可以迁移[10]. 德国乌尔姆大学 Rinderle 和 Dadam 等人将这一实例可迁移的判定方法引入到他们的 ADEPT 项目[13-15]中. 然而，他们的可迁移性判定方法还比较严格，因而可以导致一些原本可迁移的工作流实例得不到迁移. 例如他们没有考虑活动间的数据不相关性，从而要求只有当工作流实例的已执行活动序列严格按照既定顺序在目标模式下重现时，该工作流实例才可迁移[13,15]. 而这一要求在某些情形下过于严格. 本文的方法考虑了活动间的数据不相关性，不限定数据不相关的活动必须在目标模式下按照既定的执行顺序重现，从而具有更好的柔性，可以让更多的实例迁移到新流程下继续执行. 因此，本文的方法可以看作是 ADEPT 中可迁移判定方法的有力补充.

在自适应软件的相关研究中，文献[17]中的工

作与本文的工作具有一定的相似性.在该工作中,一个软件系统可以从一个模式迁移到另一个模式的前提条件是:在迁移前、迁移中和迁移后 3 个阶段,整个软件系统必须保持某些全局性质不变.这些全局性质是用时序逻辑表示的高层规约,在该工作中,这些高层规约所起的作用与本文中的"目标状态的有效性(参见定义 6)"所起的作用类似.

此外,还有一些工作关注于如何利用软件体系结构信息来帮助实现系统的动态演化以及自适应.具有代表性的工作主要有 UCI 的 Archstudio[18] 和 CMU 的 Rainbow[19].国内北京大学的 PKUAS 系统[20]以及南京大学的 ARTEMIS 系统[21]也是基于软件体系结构信息来实现软件系统的调整.这些工作更强调从系统全局层面来规约和实现系统的演化,可供我们未来实现服务编排层面的服务组合动态演化时参考.

3　背景知识

本文将要提出的服务组合动态演化方法是基于 Petri 网模型的,为便于讨论,本节首先介绍一些 Petri 网的基本概念.关于 Petri 网的详细介绍,读者可参考文献[22].

定义 1(带标签的 Petri 网).　带标签的 Petri 网(简称 Petri 网)是一个四元组 $N=(P,T,F,l)$,其中:

$P=\{p_1,p_2,\cdots,p_m\}$为库所集合;

$T=\{t_1,t_2,\cdots,t_n\}$为变迁集合,且$P\cap T=\varnothing$;

$F\subseteq(P\times T)\cup(T\times P)$为连接库所和变迁的有向弧集合;

$l:T\to L$ 是一个标签(label)函数,其中 L 为活动标签的集合.

在 Petri 网中,若 $a\in P\cup T$,记 ${}^{\cdot}a=\{b\mid(b,a)\in F\}$, $a^{\cdot}=\{b\mid(a,b)\in F\}$.我们称 ${}^{\cdot}a$ 和 $a^{\cdot}$ 分别为 a 的前集与后集.

定义 1 只描述了 Petri 网的结构部分,为表达 Petri 网的状态,Token 被引入到 Petri 网中.这些 Token 是一些标志,用小黑点表示.Token 只能包含在库所当中(在本文所使用的 Petri 网中,每个库所最多可容纳的 Token 数量限定为 1),Token 在 Petri 网库所中的分布即表示 Petri 网的状态,称为标识(Marking).M 常用来表示 Petri 网的标识,它是一个 m 维向量,其中分量 $M(p)$ 表示库所 p 中的 Token 数,一般用 M_0 表示初始标识.$M=[p_i,p_j]$表示在标识 M 下,库所 p_i 和 p_j 中各含有一个 Token,而其余库所中不含 Token.二元组(N,M)(即五元组(P,T,F,l,M))被称为一个标识网,N 称为该标识网的基网.

定义 2(变迁发生规则).　标识网 $\Sigma=(N,M)$ 的变迁发生规则为

(1)对于 Σ 中任意一个变迁 $t\in T$,若 $\forall p\in{}^{\cdot}t$ 满足 $M(p)=1$,则 t 在标识 M 下有发生权(enabled),记为$(N,M)[t\rangle$,若 t 在标识 M 下没有发生权,记为 $\neg(N,M)[t\rangle$;

(2)若$(N,M)[t\rangle$,则在标识 M 下,变迁 t 可以发生(fire),t 的发生得到一个新的标识 M',该过程记为$(N,M)[t\rangle(N,M')$.且对 $\forall p\in{}^{\cdot}t-t^{\cdot}$,$M'(p)=M(p)-1$;对 $\forall p\in t^{\cdot}-{}^{\cdot}t$,$M'(p)=M(p)+1$;对 $\forall p\notin t^{\cdot}\cup{}^{\cdot}t$ 或者 $p\in{}^{\cdot}t\cap t^{\cdot}$,$M'(p)=M(p)$.

从标识 M 可达的所有标识的集合记为$[N,M\rangle$.依据变迁的发生规则,我们可以进行 Petri 网的可达性分析[22],从而获得 Petri 网的所有标识.

在服务组合的相关研究中,开放工作流网(open Workflow Nets,简称 oWFNs)常用来描述 BPEL 流程[8,23-25].开放工作流网是一类特殊的标识 Petri 网,它在工作流网[11-12]的基础上增加了用于异步消息传输的接口库所.通过现有技术,可以将任一 BPEL 流程转化为开放工作流网的形式[25],BPEL 中的活动用开放工作流网的变迁来表示,活动的名称以及输入和输出可以作为变迁上的标签.例如活动 invoke 的输入变量集和输出变量集分别为 in 和 out,则开放工作流网中表示该活动的变迁的标签就是 invoke(in,out).开放工作流网的严格定义如下.

定义 3(开放工作流网).　一个开放工作流网是一个二元组 $\Sigma=(N,M_0)$,$N=(P,T,F,l)$为开放工作流网的基网,其中:

$P=P_M\cup P_I\cup P_O$, P_M,P_I,P_O 分别为内部库所、输入库所和输出库所的集合;

T 为变迁集合,且 $P\cap T=\varnothing$;

$F\subseteq(P\times T)\cup(T\times P)$为有向弧集合,且有 $F\cap(T\times P_I)=\varnothing$, $F\cap(P_O\times T)=\varnothing$;

$l:T\to L$ 是一个标签(label)函数,其中 L 为活动标签的集合.

M_0 为 N 的初始标识,且对 $\forall p\in P_I\cup P_O$,$M_0(p)=0$.另外,Ω 为 N 的终止标识集合:$\forall M_f\in\Omega$,$\forall t\in T$,$\neg(N,M_f)[t\rangle$,且对 $\forall p\in P_I\cup P_O$,$M_f(p)=0$.

开放工作流网的组合可以用于分析 BPEL 之间的交互,然而为了对 BPEL 流程进行单独的分析,

文献[8]指出可以仅考虑开放工作流网的内部(inner).开放工作流网的内部可以通过删除开放工作流网中的输入和输出库所以及与这些库所相连的有向弧而得到,与此同时,初始标识和终止标识也需做相应的调整[8].开放工作流网 Σ 的内部记为 $inner(\Sigma)$.下面引入开放工作流网内部弱终止性(weakly terminating)的概念.服务组合动态演化时需要保持弱终止性.

定义 4(弱终止性). 令 (N,M_0) 为开放工作流网的内部,若对于 $\forall M\in(N,M_0\rangle$ 有 $\exists M_f\in(N,M\rangle\cap\Omega$,则称 (N,M_0) 满足弱终止性.

若开放工作流网的内部满足弱终止性,则可保证相应的 BPEL 流程是无死锁和活锁的(终止标识是正常的结束状态,不是死锁或活锁)[8].因此,弱终止性可以被认为是服务组合能够正确执行的基本条件.在实际应用中,我们可以先将相应的 Web 服务转化为开放工作流网,然后通过 Petri 网的可达性分析技术[22]来检验弱终止性是否满足.若不满足,可以根据失败原因指导后续的修改,直到最终满足弱终止性.我们规定本文所考虑的开放工作流网的内部均满足弱终止性.

4 基于模型的服务组合动态演化

4.1 基于模型的服务组合动态演化过程框架

Web 服务组合的动态演化需判定相应的动态变化能否传播到正在执行的服务组合实例上,即判定服务组合实例是否符合动态演化的要求.Web 服务组合常用 BPEL 语言实现,而 BPEL 实质上是一种实现层面的编程语言,直接在 BPEL 上为每个单独的实例进行手动的演化分析是费时而易错的.

为此,我们的方法用具有严格操作语义的 Petri 网模型①来描述 BPEL 流程,这不仅可以很好地描述 BPEL 中的控制流(即内部业务逻辑),还有助于 BPEL 流程的数据流分析,从而可以为服务组合实例的动态演化提供帮助.

具体地,本文提出的基于模型的服务组合动态演化方法可分如下 5 步进行:

(1)构建 Web 服务组合源模式.将演化前的 BPEL 流程(源流程)转化为开放工作流网的形式.该转化已经有相关的技术和工具予以支持[25],因此可直接利用已有的工具进行处理.在服务组合动态演化时,变化的只是服务组合的内部业务逻辑而伙伴服务保持不变,因此只需考虑开放工作流网的内部,这样做可以使问题得到简化.我们称该开放工作流网内部的基网为源模式,记为 N_s.通过该转化,源流程中的活动用 N_s 中的变迁来表示,而活动的名称以及输入变量集和输出变量集可以作为变迁上的标签.源模式 N_s 刻画了源流程中的控制流,为第(4)步中求解服务组合实例的已执行活动在源流程下所使用的变量集合提供了准备.

(2)构建 Web 服务组合目标模式.从源流程开始,根据流程所需的改动,借助于文献[7]中提出的规则,可以获得一个与原来 BPEL 流程相容的(compatible)新 BPEL 流程(目标流程).可直接利用已有的工具[25]将目标流程转化为开放工作流网的形式.我们称该开放工作流网内部的基网为目标模式,记为 N_t.通过该转化,目标流程中的活动用 N_t 中的变迁来表示,而活动的名称以及输入变量集和输出变量集可以作为变迁上的标签.目标模式 N_t 刻画了目标流程中的控制流,为第(4)步中求解服务组合实例的已执行活动在目标流程下所使用的变量集合提供了准备.

(3)暂停当前正在执行的服务组合实例,并获取各个服务组合实例的历史执行信息(即已执行活动序列).当服务组合需要动态演化时,需暂停当前 BPEL 流程下所有正在执行的服务组合实例的执行.同时现有的工作流技术支持:每当执行一个活动的时候,可以将该活动记录在该实例的已执行活动序列中[3,15].因而,当服务组合需要动态演化时,可以方便地从日志文件中获取当前正在执行的各个服务组合实例的已执行活动序列.

(4)判定服务组合实例的可迁移性以及确定目标状态.根据服务组合实例的已执行活动序列,判断在源模式下正在执行的服务组合实例是否可以迁移到目标模式.在判定服务组合实例的可迁移性时,不仅要判断其已执行活动是否可以在目标模式下重现,而且要判断相应活动所使用的变量是否也能够在目标模式下重现.若该实例可迁移,还需进一步确定迁移的目标状态(该目标状态必须是有效的).目标状态是有效的当且仅当它不会引入一些动态演化的错误.

(5)恢复已暂停服务组合实例的执行.若服务组合实例可迁移到目标模式,则可根据已执行的活动信息以及目标状态下能够发生的活动,来确定在新的服务组合流程下从哪一个活动开始恢复执行.

① 除 Petri 网外,其它能够描述并发的形式化模型也可以使用.

若服务组合实例不可迁移,则在源流程下恢复该实例的执行.

图 1 对上述的服务组合动态演化过程框架进行了总结.由于第(1)～(3)步的工作可以利用现有的技术予以解决,本文的工作主要侧重于第(4)步,最后也对第(5)步进行了简要的介绍.

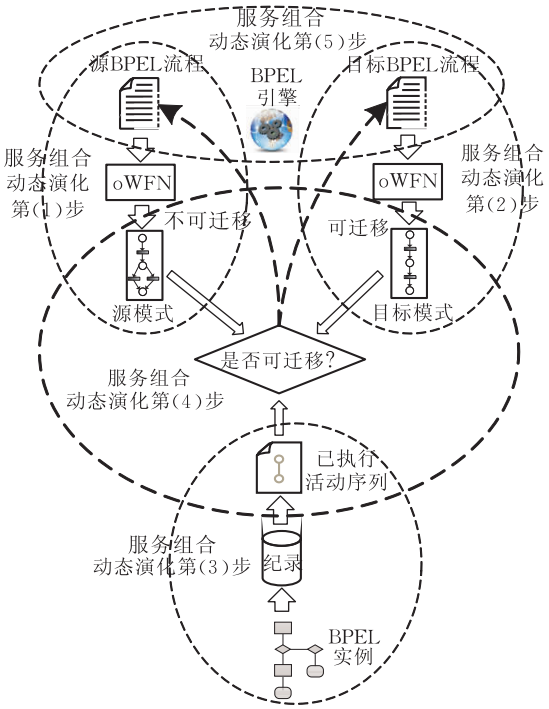

图 1　基于模型的服务组合动态演化过程框架

4.2　服务组合实例可迁移性标准

本节讨论服务组合动态演化的关键环节:服务组合实例的可迁移性判定以及可迁移实例目标状态的确定.服务组合实例的可迁移性要求正在源模式下执行的服务组合实例必须符合目标模式,即要求服务组合实例迁移前所执行的活动序列能够在目标模式下重现.若服务组合实例可迁移,还需确定迁移的目标状态,以便服务组合实例迁移到目标模式后能够继续执行.可迁移性标准不能过于严格,只要服务组合实例迁移后不会发生死锁等动态演化错误,则该实例即可进行迁移,从而能够让尽可能多的实例迁移到目标模式.

下面首先介绍一种最简单的服务组合实例迁移情形.假设在源模式 N_s 下某一正在执行的服务组合实例的已执行活动序列为 σ,如果 σ 中的活动可以在目标模式下按照原有的执行顺序重现,则该实例可以迁移到目标模式.在判定 σ 是否可重现时,不仅要求 σ 中的每个活动可以在目标模式 N_t 下重现,而且要求每个活动所使用的变量也要在目标模式下重现.不失一般性,我们假设 BPEL 中的变量不存在重名问题.在服务组合模式(源模式和目标模式)中活动的标签包含三部分内容,即活动的名称以及活动的输入变量集和输出变量集.

注.服务编排层面的演化(例如减少或增加通信活动)势必会影响到服务之间的交互.本文只考虑服务编制层面的演化,而将服务编排方面的演化留为进一步的工作.服务编制层面的演化其变化范围局限在一个组织内部,不会波及到伙伴服务.这可以通过 BPEL 目标程序与源程序的相容性[7] 予以保证.相容性的判定问题读者可参考文献[7,23-24].在这样的前提下,那些在目标模式中重现的活动,其输入变量集和输出变量集是保持不变的,唯一可能变化的是该活动在源模式和目标模式下所使用的变量可能是在不同处定义的.为检验每个活动所使用的变量是否可以在目标模式下重现,我们先引入一些基本概念.

定义 5(活动所定义的变量).　若服务组合模式 N(源模式或目标模式)中活动 t 的某一输出变量为 v,则称 v 为活动 t 所定义的变量,记为 v_t.活动 t 所定义的所有变量形成的集合记为 $Def(t)$.

注.本文下面所提到的"变量"均指这种带有定义位置的变量.为便于服务组合实例的可迁移性判定,下面引入一种特殊的数据流分析技术:求解服务组合实例沿着组合模式 N 中的某一活动序列 ρ 执行时,到达 ρ 中某一活动 t 的所有变量定义,记为 $reachingDefs(N,\rho,t)$.在活动序列 ρ 中先后出现的两个活动 a 和 b,活动 a 所定义的变量 v_a 可能被活动 b 所定义的变量 v_b 覆盖.因此,ρ 中某一活动 a 所定义的变量 v_a 能够沿活动序列 ρ 到达活动 t 当且仅当 v_a 没有被 a 和 t 之间的活动定义的变量所覆盖.我们用 $Kill(N,\rho,t)$(简记为 $Kill(t)$)来表示当组合模式 N 的某一实例沿活动序列 ρ 执行时,所有到达活动 t 而被活动 t 定义的变量所覆盖的变量定义构成的集合.

算法 1 给出了 $reachingDefs(N,\rho,t)$ 的具体求解过程.通过算法 1 我们可以发现:$reachingDefs(N,\rho,t)$ 中的变量定义并不包含 t 自身所定义的变量 $Def(t)$,这点是需要特别注意的.算法 1 中集合 $Defs$ 的元素个数在循环的过程中不断变化,为了对算法 1 的时间复杂度进行分析,我们假定 $Defs$ 在循环过程中包含的元素个数最大值为 l;同样地,假设活动序列 ρ 中 t 之前的所有的活动的最大输出变量个数为 m;n 为活动 t 处于活动序列 ρ 中的次序.

该算时间复杂度的分析的难点在于分析第 9～12 行中以及第 13 行中的集合运算的时间开销.我们发现若 $Defs$ 中的元素(即变量定义)按照字母顺序排列,则将一个新产生的变量定义加入到 $Defs$ 时,可采用折半法找到相应的插入位置.因此算法 1 的时间复杂度为 $O(n\times m\times\log_2 l)$.

算法 1. $reachingDefs(\text{in } N, \text{ in } \rho, \text{ in } t)$.

```
1.  begin
2.  Defs←∅
3.  for j←1 to |ρ|
4.    if t=ρ[j]
5.      return Defs
6.    endif
7.    Def(ρ[j])←∅
8.    Kill(ρ[j])←∅
9.    for every v∈OutPut(ρ[j])
10.     Def(ρ[j])←Def(ρ[j])∪{v_ρ[j]}
11.     Kill(ρ[j])←Kill(ρ[j])∪{v_a | v_a∈Defs}
12.   endfor
13.   Defs←(Defs−Kill(ρ[j]))∪Def(ρ[j])
14. endfor
15. return Defs
16. end
```

若 ρ 为某一待迁移实例的已执行活动序列,ρ 中某一活动 t 所使用的变量集合记为 $Use(N,\rho,t)$.基于算法 1,我们可求出 $Use(N,\rho,t)$,具体过程见算法 2.采用与算法 1 同样的分析方法,算法 2 的时间复杂度为 $O(n\times m\times\log_2 l+k\times\log_2 l)$,其中 k 为活动 t 的输入变量个数,l,m,n 的含义与算法 1 一致.

算法 2. $Use(\text{in } N, \text{ in } \rho, \text{ in } t)$.

```
1.  begin
2.  Use(t)←∅
3.  Defs←reachingDefs(N,ρ,t)
4.  for every v∈InPut(t)
5.    if ∃v_a∈Defs
6.      Use(t)←Use(t)∪{v_a}
7.    endif
8.  endfor
9.  return Use(t)
10. end
```

令 $N_s=(P_s,T_s,F_s,l_s)$ 和 $N_t=(P_t,T_t,F_t,l_t)$ 分别表示源模式与目标模式.若 $Use(N_s,\rho,t)=Use(N_t,\rho,t)$,则表明活动 t 在源模式 N_s 下所使用的变量可以在目标模式 N_t 下重现.在给出服务组合实例的可迁移性标准之前,我们须对 N_t 的初始标识 M_0 进行定义.对 $\forall p\in P_t$,若 $\neg\exists t\in T_t$ 使得 $(t,p)\in F_t$,$M_0(p)=1$;否则 $M_0(p)=0$.

可迁移性标准 1. 服务组合的源模式和目标模式分别为 N_s,N_t,在源模式 N_s 下某一正在执行的服务组合实例的已执行活动序列为 σ,若 σ 满足以下两个条件:(1) $(N_t,M_0)[\sigma\rangle(N_t,M_t)$;(2) $\forall t\in\sigma$,$Use(N_s,\sigma,t)=Use(N_t,\sigma,t)$,则该服务组合实例可迁移到 N_t,且目标状态为 (N_t,M_t).

定义 6(目标状态的有效性). 若某一服务组合实例迁移到目标模式的目标状态下恢复执行后,不会引入一些动态演化的错误(包括死锁或活锁、跳过活动、重复执行活动),则称该目标状态是有效的(valid).

注. 在本文中,我们规定目标状态的有效性是判定服务组合实例是否可以迁移的"源标准". 而可迁移性标准 1(包括下面提出的可迁移性标准 2～4)可以看作是满足该"源标准"的充分条件,它们的提出可以使得服务组合实例可迁移性的判定具有切实可操作性.

可以证明根据可迁移性标准 1 所获得的目标状态是有效的.然而,上述可迁移性标准仅能处理 σ 中的活动可以在目标模式下按照原有的执行顺序重现的情形.由于目标模式可能对服务组合实例已执行过的部分做出修改(删除活动、添加活动、改变活动的执行顺序等),这往往使得服务组合实例已执行过的活动序列不能在目标模式下按照既定的执行顺序完全重现.在这种情形下,不能简单地认为服务组合实例不可迁移.为便于理解,我们通过举例的方式,逐步放松服务组合实例的可迁移性标准,从而使得尽可能多的服务组合实例能够迁移到目标模式下.

下面分析图 2 所示的例子,其中图 2(a)和(b)分别代表服务组合的源模式和目标模式.在目标模式下,活动 t_2 被删除.假设某一待迁移的服务组合实例的已执行活动序列为 t_1t_2,由于 $\neg(N_t,M_0)[t_1t_2\rangle$,根据可迁移性标准 1,该实例不能迁移到目标模式下.然而在目标模式下,本来就不需要执行活动 t_2,因此要求 t_2 在目标模式下重现并没有意义.实际上,在判定服务组合实例的已执行活动序列是否可以重现时,我们无须考虑目标模式下不存在的活动.因此,我们可以将可迁移性标准 1 进行放松,得到可迁移性标准 2.

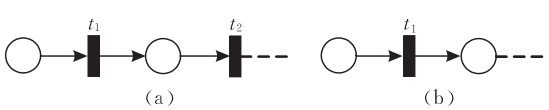

图 2 服务组合动态演化实例举例 1

可迁移性标准 2.　服务组合的源模式和目标模式分别为 N_s,N_t,在源模式 N_s 下某一正在执行的服务组合实例的已执行活动序列为 σ,令 $\delta=\sigma\uparrow T_t$($\sigma\uparrow T_t$ 表示 σ 到目标模式活动集合 T_t 的投影,即去掉 σ 中不属于集合 T_t 中的活动,而 σ 中剩余活动的相对顺序保持不变).若 δ 满足以下两个条件:(1)$(N_t,M_0)[\delta\rangle(N_t,M_t)$;(2)$\forall t\in\delta$,$Use(N_s,\sigma,t)=Use(N_t,\delta,t)$,则该服务组合实例可迁移到 N_t,且目标状态为 (N_t,M_t).

在可迁移性标准 2 的条件下,图 2 例子中的服务组合实例可以迁移,并且不难证明根据可迁移性标准 2 所获得的目标状态是有效的.然而,可迁移性标准 2 仍可进一步放松.为定义新的可迁移性标准,我们首先引入一些基本概念.

定义 7(直接数据依赖).　在服务组合模式 N 下,对于某一服务组合实例的已执行活动序列 σ 中的任意两个活动 $\sigma[i]$ 和 $\sigma[j]$,其中 $i<j$,我们称活动 $\sigma[j]$ 直接数据依赖于活动 $\sigma[i]$,当且仅当 $Use(N,\sigma,\sigma[j])\cap Def(\sigma[i])\neq\varnothing$.

定义 8(间接数据依赖).　对于某一服务组合实例的已执行活动序列 σ 中的 3 个活动 $\sigma[i]$,$\sigma[k]$ 和 $\sigma[j]$,其中 $i<k<j$,若活动 $\sigma[j]$ 直接或间接数据依赖于活动 $\sigma[k]$,活动 $\sigma[k]$ 直接或间接数据依赖于活动 $\sigma[i]$,则活动 $\sigma[j]$ 间接数据依赖于活动 $\sigma[i]$.

通过上述定义,不难发现直接数据依赖不具有传递性,而间接数据依赖具有传递性.下面将直接数据依赖和间接数据依赖统称为数据依赖.

定义 9(数据不相关).　对于某一服务组合实例的已执行活动序列 σ 中的任意两个活动 $\sigma[i]$ 和 $\sigma[j]$,若活动 $\sigma[j]$ 不数据依赖于活动 $\sigma[i]$,同时 $\sigma[i]$ 也不数据依赖于活动 $\sigma[j]$,则称活动 $\sigma[j]$ 与活动 $\sigma[i]$ 数据不相关.

σ 中与活动 $\sigma[i]$ 数据不相关的所有活动形成一个集合,记作 $dataUnrelevant(N,\sigma,\sigma[i])$.算法 3 给出了相应的求解方法.

算法 3.　$dataUnrelevant(\text{in } N,\text{in } \sigma,\text{in } \sigma[i])$.

```
1.  begin
2.  boolean visited[|σ|]={false}
3.  Set s←σ−σ[i]
4.  s←s−dataUse(N,σ,σ[i], *visited)
5.  s←s−dataUsed(N,σ,σ[i], *visited)
6.  return s
7.  end
dataUse(in N, in σ, in σ[i], in *visited)
8.  begin
9.  Set s1←∅
10. Use(σ[i])←Use(N,σ,σ[i])
11. for every v_a∈Use(σ[i]), and a∈s
12.    if visited[a]=false
13.       visited[a]=true
14.       s1←s1∪{a}
15.       s1←s1∪dataUse(N,σ,a, *visited)
16.    endif
17. endfor
18. return s1
19. end
dataUsed(in N, in σ, in σ[i], in *visited)
20. begin
21. Set s2←∅
22. for j←i+1 to |σ|
23.    if σ[j]∈s2
24.       continue
25.    endif
26.    if visited[σ[j]]=false &
          Use(N,σ,σ[j])∩Def(σ[i])≠∅
27.       visited[σ[j]]=true
28.       s2←s2∪{σ[j]}
29.       s2←s2∪dataUsed(N,σ,σ[j], *visited)
30.    endif
31. endfor
32. return s2
33. end
```

我们简要介绍一下算法 3,它调用了两个递归算法,其中 $dataUse(\text{in } N,\text{in } \sigma,\text{in } \sigma[i],\text{in } *visited)$ 返回的是活动 $\sigma[i]$ 所数据依赖的所有活动构成的集合,而 $dataUsed(\text{in } N,\text{in } \sigma,\text{in } \sigma[i],\text{in } *visited)$ 返回的是数据依赖于活动 $\sigma[i]$ 的所有活动构成的集合.由于算法 3 中涉及许多集合运算,我们不再具体分析它的时间复杂度.读者可以利用类似于算法 1 的分析方法得出它的时间复杂度是多项式级的.

可迁移性标准 2 的要求还过于严格,我们通过图 3 所示的例子进行说明,其中图 3(a)和(b)分别代表服务组合的源模式和目标模式.在目标模式下,活动 t_1 和 t_2 的执行顺序进行了交换.假设某一待迁移实例的已执行活动序列为 t_1t_2,由于 $\neg(N_t,M_0)[t_1t_2\rangle$,根据可迁移性标准 2,该实例不能迁移到目标模式下.然而如果活动 t_1 和 t_2 数据不相关(参见定义 9),则

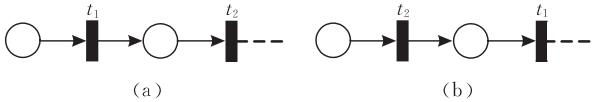

图 3　服务组合动态演化实例举例 2

交换 t_1 和 t_2 的执行顺序并不会引起动态演化错误.因此在这种条件下,应该允许该服务组合实例的迁移.

在一个已执行活动序列 σ 中,我们可以找到所有数据不相关的活动对.由于数据不相关的两个活动之间的执行次序可以是任意的,如果变换它们在 σ 中的顺序,可以得到一系列活动序列,这些活动序列的集合记为 $DU(\sigma)$.在引入 $DU(\sigma)$ 的基础上,我们给出如下可迁移性标准.

可迁移性标准 3. 服务组合的源模式和目标模式分别为 N_s,N_t,在源模式 N_s 下某一正在执行的服务组合实例的已执行活动序列为 σ,令 $\delta=\sigma\uparrow T_t$.若 $\exists\lambda\in DU(\delta)$,满足以下两个条件:(1) $(N_t,M_0)[\lambda\rangle(N_t,M_t)$;(2) $\forall t\in\lambda$, $Use(N_s,\sigma,t)=Use(N_t,\lambda,t)$,则该服务组合实例可迁移到 N_t,且目标状态为 (N_t,M_t).

在可迁移性标准 3 的条件下,图 3 例子中的服务组合实例可以迁移,并且不难证明根据可迁移性标准 3 所获得的目标状态是有效的.然而,可迁移性标准 3 还是过于严格,我们通过分析图 4 所示的例子进行说明.图 4(a)和(b)分别代表服务组合的源模式和目标模式.在目标模式下,活动 t_1 和 t_2 之间加入了一个新的活动 t_3.假设某一待迁移实例的已执行活动序列为 t_1t_2,因为 $\neg(N_t,M_0)[t_1t_2\rangle$,根据可迁移性标准 3,该实例不能迁移到目标模式下.若在目标模式下活动 t_3 的加入并没有影响到活动 t_2 的变量使用情况(但可能影响到其它后续活动的变量使用),则此时活动 t_1 和 t_2 的执行结果可以在目标模式下重用.具体操作如下:若 $(N_t,M_0)[t_1\rangle(N_t,M_1)$,我们可以将 (N_t,M_1) 作为该服务组合实例的目标状态.因此该实例在目标模式下恢复执行,首先需要执行活动 t_3,随后执行活动 t_2.然而由于活动 t_2 已经在源模式下执行过,并且其变量的使用情况与在目标模式的变量使用情况一致,因此可以重用活动 t_2 的执行结果(无需重新执行),只需继续执行 t_2 的后续活动.在这种情况下,活动序列 t_1t_2 也被认为可以在目标模式下重现(称为分段重现).通过这个例子的启示,我们可以进一步放松可迁移性标准.

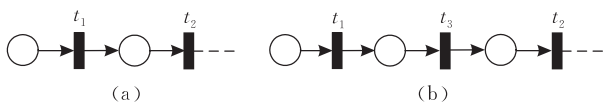

图 4 服务组合动态演化实例举例 3

可迁移性标准 4. 服务组合的源模式和目标模式分别为 N_s,N_t,在源模式 N_s 下某一正在执行的服务组合实例的已执行活动序列为 σ,令 $\delta=\sigma\uparrow T_t$.若 $\exists\lambda=\lambda_1\lambda_2\cdots\lambda_n\in DU(\delta)$,$\exists\rho=\theta_1\lambda_1\theta_2\lambda_2\cdots\theta_n\lambda_n$(其中 θ_j 或者为空活动序列,或者为目标模式下可接连发生的活动序列且 $\theta_j\cap\lambda=\varnothing$,满足以下两个条件:(1) $(N_t,M_0)[\rho\rangle$;(2) $\forall t\in\lambda$, $Use(N_s,\lambda,t)=Use(N_t,\rho,t)$,则该服务组合实例可迁移到 N_t.如果 $\theta_1,\theta_2,\cdots,\theta_j$ 均为空活动序列而 θ_{j+1} 为非空活动序列,且 $(N_t,M_0)[\lambda_1\lambda_2\cdots\lambda_j\rangle(N_t,M_t)$,则 (N_t,M_t) 即为目标状态.

注.在可迁移性标准 4 中,我们确定实例迁移的目标状态时,需要在目标模式下找到第一个属于 ρ 而不属于 σ 的活动 a($a\in\theta_{j+1}$).由于活动 a 没有在源模式下执行过,因此目标状态定为 (N_t,M_t) 以保证实例迁移后不会跳过活动 a.至于 a 之后的活动,我们将在算法 5 中给出处理办法.

定理 1. 根据可迁移性标准 4 所获得的目标状态是有效的.

证明. 根据定义 6,需判定服务组合实例迁移到目标模式的目标状态下恢复执行后,不会引入死锁或活锁、跳过活动、重复执行活动等错误.

(1) 不会引入死锁或活锁错误.由于 $(N_t,M_0)[\lambda_1\lambda_2\cdots\lambda_j\rangle(N_t,M_t)$,即 $(N_t,M_t)\in(N_t,M_0\rangle$,则根据目标模式的弱终止性,$\exists M_f\in\Omega$,$M_f\in(N_t,M_t\rangle$.所以,服务组合实例迁移之后不会发生死锁或活锁错误.

(2) 不会引入跳过活动的错误.由于 $(N_t,M_0)[\lambda_1\lambda_2\cdots\lambda_j\rangle(N_t,M_t)$,并且活动序列 $\lambda_1\lambda_2\cdots\lambda_j$ 中的所有活动在源模式下已经执行过,所以 $\lambda_1\lambda_2\cdots\lambda_j$ 中的活动都没有跳过.令 $\rho'=\rho-\lambda_1\lambda_2\cdots\lambda_j$,若 $(N_t,M_0)[\rho\rangle(N_t,M'_t)$,则 $(N_t,M_t)[\rho'\rangle(N_t,M'_t)$,由于服务组合实例从目标状态 (N_t,M_t) 下恢复执行,因此 ρ' 中尚未执行过的活动也不会跳过.而根据目标模式的弱终止性,$\exists M_f\in(N_t,M_t\rangle\cap\Omega$,即 $\exists\rho''$ 满足 $(N_t,M'_t)[\rho''\rangle(N_t,M_f)$,因此 ρ'' 中的活动不会被跳过.综上,对于满足 $(N_t,M_0)[\rho+\rho''\rangle(N_t,M_f)$ 的活动序列 $\rho+\rho''$,其中没有跳过的活动.

(3) 不会引入重复执行活动的错误.服务组合实例从目标状态 (N_t,M_t) 下恢复执行,所以只需考虑 $\rho'=\rho-\lambda_1\lambda_2\cdots\lambda_j$ 中的已执行活动是否必须重新执行即可.令 $\lambda'=\lambda-\lambda_1\lambda_2\cdots\lambda_j$,对于 $\forall t\in(\rho'\cap\lambda')\subseteq(\rho\cap\lambda)$,由于 $Use(N_s,\lambda,t)=Use(N_t,\rho,t)$,所以 ρ' 中在源模式下已经执行过的活动无需再次执行.因此不存在必须重新执行的活动. 证毕.

根据可迁移性标准 4,可以判定图 4 例子中的服务组合实例能够迁移到目标模式下.通过上述分

析我们发现,可迁移性标准 4 的要求更为松弛,适用性更广,因此它能让更多的服务组合实例迁移到目标模式下.算法 4 给出了可迁移性标准 4 的实现,该算法不仅可以判定服务组合实例的可迁移性,并且当实例可迁移的时候可以求出相应的目标状态.

算法 4. $targetState(\text{in } N_s, \text{ in } N_t, \text{ in } \sigma)$.

```
1.  begin
2.  δ←σ↾T_t
3.  ρ←∅
4.  ts←∅
5.  currentM_t←M_0
6.  handledSet←∅
7.  reachingDefs←reachingDefs(N_s,σ,σ[|σ|])∪
                 Def(σ[|σ|])
8.  i←1
9.  k←∞
10. j←1
11. while j≤|δ|
12.   if δ[j]∈handledSet
13.     j←j+1
14.     continue
15.   endif
16.   if(N_t,currentM_t)[δ[j]⟩(N_t,M_t)
17.     if Use(N_s,λ,δ[j])⊆reachingDefs
18.       ρ←ρ+δ[j]
19.       i←i+1
20.       currentM_t←M_t
21.       j←j+1
22.       if j=|δ| & ts=∅
23.         ts←currentM_t
24.       endif
25.     else
26.       ts←∅
27.       break
28.     endif
29.   else if ∃t∈dataUnrelevant(N,σ,σ[j]) and
              (N_t,currentM_t)[t⟩(N_t,M_t)
30.     if Use(N_s,λ,t)⊆reachingDefs
31.       ρ←ρ+t
32.       i←i+1
33.       currentM_t←M_t
34.       handledSet←handledSet∪{t}
35.     endif
36.   else if ∃t∉δ, satisfying(N_t,currentM_t)[t⟩(N_t,M_t)
37.     if reachingDefs satisfying inPut(t)
38.       ρ←ρ+t
39.       i←i+1
40.       currentM_t←M_t
41.       Def(t)←∅
42.       Kill(t)←∅
43.       for every v∈outPut(t)
44.         Def(t)←Def(t)∪{v_t}
45.         Kill(t)←Kill(t)∪{v_a|v_a∈reachingDefs}
46.       endfor
47.       reachingDefs←(reachingDefs−Kill(t))∪
                       Def(t)
48.       if ts=∅
49.         ts←currentM_t
50.         k←i
51.       endif
52.     endif
53.   else
54.     ts←∅
55.     break
56.   endif
57. endwhile
58. return(ts,ρ,k)
59. end
```

算法 4 的输入为源模式 N_s 和目标模式 N_t 以及 N_s 下某一服务组合实例的已执行活动序列 σ.算法 4 的输出为三元组 (ts,ρ,k).若输出 ts 为 $\varnothing$,则表示 σ 不能在 N_t 下重现,因而相应的服务组合实例不能迁移到目标模式下;否则,ts 中返回的即为目标状态,此时 ρ 返回的为目标模式下满足可迁移性标准 4 中条件(1)$(N_t,M_0)[\rho\rangle$ 和(2)$\forall t\in\lambda$, $Use(N_s,\lambda,t)=Use(N_t,\rho,t)$ 的一个活动序列.当返回值 $k=\infty$ 时,实例迁移到目标状态 ts 后可以执行 ts 下任意一个可以执行的活动;否则,实例迁移到目标状态 ts 后,需要沿活动序列 ρ 的方向继续执行,且执行的第 1 个活动为 $\rho[k]$.

下面简要介绍一下算法 4.算法 4 中第 2～10 行是相关变量初始化,其中第 7 行中调用了算法 1.第 11～57 行为算法 4 的核心,其大致思路是:首先判断 $\delta(\delta\leftarrow\sigma\upharpoonright T_t)$ 中的某一活动 $\delta[j]$ 是否可以在目标模式的当前状态 $currentM_t$ 下重现(第 12～28 行),若可以重现则继续判断 $\delta[j+1]$ 是否可以重现,直到 δ 中的活动全部重现为止;若 $\delta[j]$ 不能重现,则可以判断 δ 中与 $\delta[j]$ 数据不相关的活动是否可以在 $currentM_t$ 下重现(第 29～35 行);若与 $\delta[j]$ 数据不相关的活动也无法重现,则可以判断目标模式下不属于 δ 中的活动是否可以在 $currentM_t$ 下发生,若找到这样的活动,则执行该活动并在新得到的 $currentM_t$ 下继续上述判定(第 36～52 行);若找不到这样的活动,则该实例无法在目标模式下重现(第

53～55 行).

算法 4 的主要过程在于寻找满足可迁移性标准 4 中条件(1)和(2)的活动序列 ρ. 但要注意:在判断已执行活动所使用的变量是否可以在目标模式重现时,第 17 行使用了条件 $Use(N_s,\lambda,\delta[j])\subseteq reachingDefs$ 来代替可迁移性标准 4 中的条件(2) $Use(N_s,\lambda,\delta[j])=Use(N_t,\rho,\delta[j])$,其效果是相同的. 第 30 行也采用了同样的处理方式.

采用与算法 1 类似的分析方法,可以分析出算法 4 的时间复杂度也是多项式级的.

最后,我们讨论如何恢复已暂停服务组合实例的执行,这对应 4.1 节中所提出的服务组合动态演化框架的第(5)步. 对于可迁移的服务组合实例,通过算法 4 我们可以获得迁移的目标状态 ts. 为使该实例在源模式下已执行的活动序列能够在目标模式下重用,该实例在目标状态下必须沿活动序列 ρ 的方向进行执行,遇到在源模式下已经执行过的活动则跳过,否则执行目标 BPEL 流程中的相应活动. 具体过程如算法 5 所示. 若服务组合实例不可迁移,则在源流程下继续该实例的执行.

算法 5. *resumeExecution*(in σ, in N_t, in ts, in ρ, in k).

```
1.  begin
2.  currentM_t ← ts
3.  if k != ∞
4.    for j ← k to |ρ|
5.      if(N_t, currentM_t)[ρ[j]⟩(N_t, M_t)
6.        currentM_t ← M_t
7.      endif
8.      if ρ[j] ∉ σ
9.        do activity(ρ[j])
10.     endif
11.   endfor
12. endif
13. while currentM_t ∉ Ω
14.   if ∃t∈N_t, satisfying (N_t, currentM_t)[t⟩(N_t, M_t)
15.     currentM_t ← M_t
16.     do activity(t)
17.   endif
18. endwhile
19. end
```

令 $\rho'=\rho[k]\rho[k+1]\cdots\rho[|\rho|]$,若 $(N_t,ts)[\rho'\rangle(N_t,M_t)$ 且存在活动序列 ρ'' 和标识 $M_f\in\Omega$ 使得 $(N_t,M_t)[\rho''\rangle(N_t,M_f)$,则算法 5 的时间复杂度为 $O(m+n)$,其中 m 和 n 分别代表 ρ' 和 ρ'' 中活动的个数.

5 案例分析

本节通过一个旅行代理的 Web 服务组合实例来说明本文方法的有效性和可行性. 该旅行代理服务可以根据用户所提交的查询请求,查询相应的 airline 服务和 hotel 服务,并将得到的服务价格等具体信息返回给客户. 若客户对查询结果不满意,则返回继续查询;若客户对查询结果满意,则告知旅行代理服务确认此次预定. 随后,旅行代理服务调用相应的 airline 服务和 hotel 服务,并将预定信息返回给客户. 该旅行代理服务的 BPEL 流程如图 5 所示.

```
〈process name="travel_agency"〉
 〈partnerlinks〉
  〈partnerlink name="client"…/〉
  〈partnerlink name="airline"…/〉
  〈partnerlink name="hotel"…/〉
 〈/partnerlinks〉
 〈sequence〉
  〈repeatUntil〉
   〈receive partnerlink="client" operation="travel_query"
    variable="queryInput"〉
   〈assign〉
    〈copy〉〈from〉$queryInput. airline〈/from〉
          〈to〉$airlineQuery〈/to〉〈/copy〉
    〈copy〉〈from〉$queryInput. hotel〈/from〉
          〈to〉$hotelQuery〈/to〉〈/copy〉
   〈/assign〉
   〈invoke partnerlink="airline" operation="airline_query"
    inputVariable="airlineQuery" outputVariable="airlineResult"〉
   〈invoke partnerlink="hotel" operation="hotel_query"
    inputVariable=" hotelQuery" outputVariable="hotelResult"〉
   〈assign〉
    〈copy〉〈from〉$airlineResult〈/from〉
          〈to〉$queryOutput. flight〈/to〉〈/copy〉
    〈copy〉〈from〉$hotelResult〈/from〉
          〈to〉$ueryOutput.hotel〈/to〉〈/copy〉
   〈/assign〉
   〈invoke partnerlink="client"
    operation="query_responce" inputVariable="queryOutput"〉
   〈receive partnerlink="client"
    operation="acknowledgement" variable="acceptance"〉
  〈/repeatUntil condition $acceptance="true"〉
  〈assign〉
   〈copy〉〈from〉$queryOut.airline〈/from〉
         〈to〉$airlineBooking〈/to〉〈/copy〉
   〈copy〉〈from〉$queryInput.hotel〈/from〉
         〈to〉$hotelBooking〈/to〉〈/copy〉
  〈/assign〉
  〈flow〉
   〈invoke partnerlink="airline" operation="airline_request"
    inputVariable="airlineBooking"
    outputVariable="airlineOffer"〉
   〈invoke partnerlink="hotel" operation="hotel_request"
    inputVariable="hotelBooking" outvariable="hotelOffer"〉
  〈/flow〉
  〈assign〉
   〈copy〉〈from〉$airlineOffer〈/from〉
         〈to〉$travelOffer. airlineoffer〈/to〉〈/copy〉
   〈copy〉〈from〉$hotelOffer〈/from〉
         〈to〉$travelOffer. hoteloffer〈/to〉〈/copy〉
  〈/assign〉
  〈invoke partnerlink="client"
   operation="travel_offer" inputVariable="travelOffer"〉
 〈/sequence〉
〈/process〉
```

图 5 旅行代理服务组合的源流程

然而,随着使用的增多,该旅行代理服务对原有流程进行了一些优化:(1)为提高流程的执行效率,airline_query 和 hotel_query 两个 invoke 活动改为并行执行;(2)由于 airline 服务和 hotel 服务结成服务联盟,凡已订购相应 airline 服务的客户,再订购 hotel 服务时可以享受一定的优惠,为此原本并行执行的两个活动 airline_request 和 hotel_request 改为串行执行;(3)由于统计的需要,在不影响通信活动的前提下,增加了一个内部活动 record,用以记录客户所订购的服务.更新后的 BPEL 流程如图 6 所示.

```
〈process name="travel_agency"〉
 〈partnerlinks〉
  〈partnerlink name="client"…/〉
  〈partnerlink name="airline"…/〉
  〈partnerlink name="hotel"…/〉
 〈/partnerlinks〉
 〈sequence〉
  〈repeatUntil〉
   〈receive partnerlink="client" operation="travel_query"
    variable="queryInput"〉
   〈assign〉
    〈copy〉〈from〉$queryInput.airline〈/from〉
          〈to〉$airlineQuery〈/to〉〈/copy〉
    〈copy〉〈from〉$queryInput.hotel〈/from〉
          〈to〉$hotelQuery〈/to〉〈/copy〉
   〈/assign〉
   〈flow〉
    〈invoke partnerlink="airline" operation="airline_query"
     inputVariable="queryInput.flight"
     outputVariable="airlineResult"〉
    〈invoke partnerlink="hotel" operation="hotel_query"
     inputVariable="queryInput.hotel"
     outputVariable="hotelResult"〉
   〈/flow〉
   〈assign〉
    〈copy〉〈from〉$airlineResult 〈/from〉
          〈to〉$queryOutput.flight〈/to〉〈/copy〉
    〈copy〉〈from〉$hotelResult〈/from〉
          〈to〉$queryOutput.hotel〈/to〉〈/copy〉
   〈/assign〉
   〈invoke partnerlink="client"
    operation="query_responce" inputVariable="queryOutput"〉
   〈receive partnerlink="client"
    operation="acknowledgement" variable="acceptance"〉
   〈/repeatUntil condition $acceptance="true"〉
   〈assign〉
    〈copy〉〈from〉$queryOut.airline 〈/from〉
          〈to〉$airlineBooking〈/to〉〈/copy〉
    〈copy〉〈from〉$queryInput.hotel〈/from〉
          〈to〉$hotelBooking〈/to〉〈/copy〉
   〈/assign〉
   〈invoke partnerlink="airline" operation="airline_request"
    inputVariable="airlineBooking" outputVariable="airlineOffer"〉
   〈invoke partnerlink="hotel" operation="hotel_request"
    inputVariable="hotelBooking" outvariable="hotelOffer"〉
   〈assign〉
    〈copy〉〈from〉$airlineOffer〈/from〉
          〈to〉$travelOffer.airlineoffer〈/to〉〈/copy〉
    〈copy〉〈from〉$hotelOffer〈/from〉
          〈to〉$travelOffer.hoteloffer〈/to〉〈/copy〉
   〈/assign〉
   〈invoke partnerlink="client"
    operation="travel_offer" inputVariable="travelOffer"〉
   〈extensionActivity〉
    〈b4p:peopleActivity name="record"
          inputVariable="travelOffer"〉
    〈/b4p:peopleActivity〉
   〈/extensionActivity〉
 〈/sequence〉
〈/process〉
```

图 6　旅行代理服务组合的目标流程

为了表达的方便,我们分别称图 5 和图 6 所示的 BPEL 流程为源流程和目标流程.对于新到达的客户请求,旅行代理服务可以按照目标流程给予相应的处理.为了尽可能地让客户享受到目标流程所带来的便利以及出于维护方便等方面的原因,对于尚未处理完的客户请求,要求旅行代理服务能动态地将可迁移的服务组合实例迁移到目标流程下继续执行.

我们可以通过服务组合实例在源流程下的已执行活动序列信息来判断该组合实例的可迁移性.然而,当需要迁移的服务组合实例的数目很多时,为每个单独的实例进行手动的迁移,工作量很大,且容易出错.因此,我们可以采用本文所提出的基于模型的方法分如下 5 步进行处理:

(1)将源流程转换为开放工作流网的形式(如图 7(a)所示),为分析的方便可将该开放工作流网的内部作为源模式 N_s(即删除图 7(a)中用虚线表示的接口库所以及与这些库所相连的有向弧).其中变迁所代表的活动及其输入变量和输出变量汇总在表 1 中.

表 1　图 7 中变迁所代表的活动及其输入输出变量

变迁	相应活动	活动输入	活动输出
t_1	travel_query	None	queryInput
t_2	assign	queryInput	airlineQuery hotelQuery
t_3	airline_query	airlineQuery	airlineResult
t_4	hotel_query	hotelQuery	hotelResult
t_5	assign	airlineResult hotelResult	queryOutput
t_6	query_responce	queryOutput	None
t_7	acknowledgement	None	acceptance
t_8	back to query	None	None
t_9	assign	queryOutput	airlineBooking hotelBooking
t_{10}	airline_request	airlineBooking	airlineOffer
t_{11}	hotel_request	hotelBooking	hotelOffer
t_{12}	assign	airlineOffer hotelOffer	travelOffer
t_{13}	travel_offer	travelOffer	None
t_{14}	log	travelOffer	None

(2)将目标流程转换为开放工作流网的形式(如图 7(b)所示),为分析的方便可将该开放工作流网的内部作为目标模式 N_t(即删除图 7(b)中用虚线表示的接口库所以及与这些库所相连的有向弧).其中变迁所代表的活动及其输入变量和输出变量汇总在表 1 中.

(3)暂停当前正在执行的服务组合实例,并从 BPEL 日志中获得各个服务组合实例的历史执行信息(即已执行活动序列).

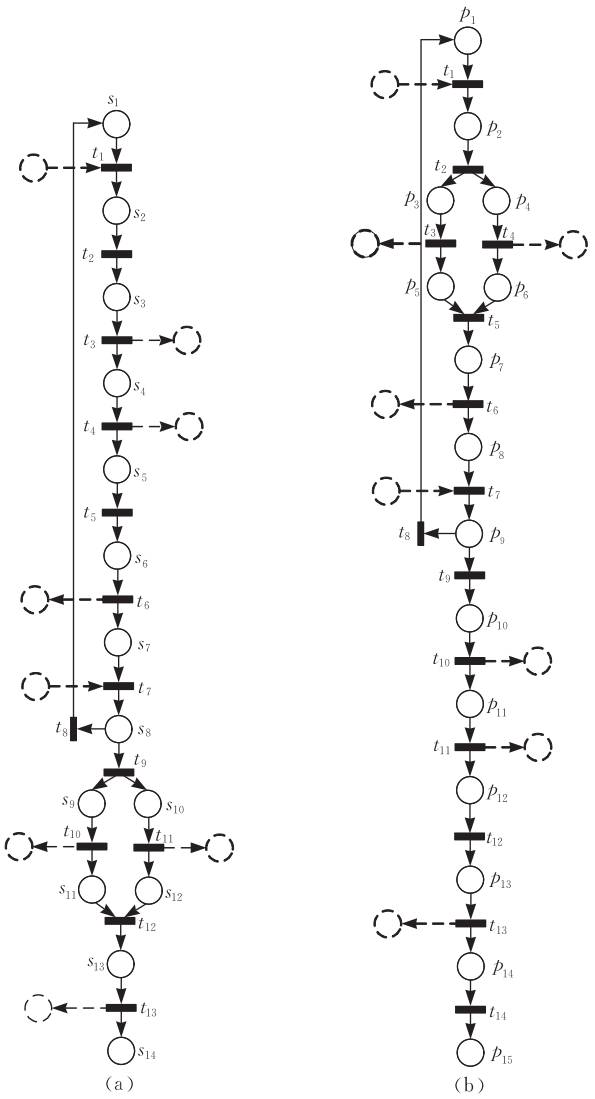

图 7 旅行代理服务组合的源模式与目标模式

(4)根据服务组合实例在源流程下的已执行活动序列,利用算法 4 来判断该实例是否可迁移.若可迁移,还需求出目标状态.

(5)若可迁移,则根据算法 5,将该实例迁移到目标流程的目标状态下继续执行;若不可迁移,则该实例继续按源流程进行执行.

在判定服务组合实例的可迁移性之前,我们须对目标模式 N_t 的初始标识 M_0 进行定义:由于 $\neg\exists t\in T_t$ 使得 $(t,p_1)\in F_t$,所以 $M_0(p_1)=1$;对于 $\forall p_i\neq p_1, M_0(p_i)=0$.

下面具体说明本文方法的可行性.某一服务组合实例(实例 1)在源流程下的已执行活动序列为 $\sigma_1=(t_1t_2t_3t_4t_5t_6t_7t_8)^2t_1t_2t_3$,其中 $(t_1t_2t_3t_4t_5t_6t_7t_8)^2$ 表示活动序列 $(t_1t_2t_3t_4t_5t_6t_7t_8)$ 执行了两次.现依据可迁移性标准 4 的要求对实例 1 的可迁移性进行判定.利用算法 4,可以获得三元组 (ts,ρ,k):其中:目标状态 $ts=(N_t,[p_4,p_5])$,这表明实例 1 可迁移;同时,存在目标模式下一个活动序列 $\rho=\sigma_1$,满足条件(1)$(N_t,M_0)[\rho\rangle$;(2)$\forall t\in\sigma_1, Use(N_s,\sigma_1,t)=Use(N_t,\rho,t)$(见表 2 第 2~9 行);而 $k=\infty$ 表明实例迁移到目标状态 $(N_t,[p_4,p_5])$ 后可以执行 $(N_t,[p_4,p_5])$ 下任意可执行的活动.具体地,BPEL 引擎可以依照算法 5 在目标流程下恢复该实例的执行.

表 2 已执行活动在源模式和目标模式下的变量使用

t	$Use(N_s,\sigma,t)$	$Use(N_t,\rho,t)$
t_1	None	None
t_2	$(\text{queryInput})_{t_1}$	$(\text{queryInput})_{t_1}$
t_3	$(\text{airlineQuery})_{t_2}$	$(\text{airlineQuery})_{t_2}$
t_4	$(\text{hotelQuery})_{t_2}$	$(\text{hotelQuery})_{t2}$
t_5	$(\text{airlineResult})_{t_3}(\text{hotelResult})_{t_4}$	$(\text{airlineResult})_{t_3}(\text{hotelResult})_{t_4}$
t_6	$(\text{queryOutput})_{t_5}$	$(\text{queryOutput})_{t_5}$
t_7	None	None
t_8	None	None
t_9	$(\text{queryOutput})_{t_5}$	$(\text{queryOutput})_{t_5}$
t_{10}	$(\text{airlineBooking})_{t_9}$	$(\text{airlineBooking})_{t_9}$
t_{11}	$(\text{hotelBooking})_{t_9}$	$(\text{hotelBooking})_{t_9}$

考虑另一待迁移服务组合实例(实例 2),若它在源流程下的已执行活动序列为

$$\sigma_2=t_1t_2t_3t_4t_5t_6t_7t_8t_9t_{11}t_{10}.$$

利用算法 4 可以得到三元组 (ts,ρ,k):其中目标状态 $ts=(N_t,[p_{12}])$,表明该实例可迁移;同时,存在目标模式下一个活动序列 $\rho=t_1t_2t_3t_4t_5t_6t_7t_8t_9t_{10}t_{11}\in DU(\sigma_2)$,满足条件(1)$(N_t,M_0)[\rho\rangle$;(2)$\forall t\in\sigma_2, Use(N_s,\sigma_2,t)=Use(N_t,\rho,t)$(见表 2 第 2~12 行);而 $k=\infty$ 表明实例 2 迁移到目标状态 $(N_t,[p_{12}])$ 后可以执行 $(N_t,[p_{12}])$ 下任意可执行的活动.当实例 2 迁移到目标状态 $(N_t,[p_{12}])$ 后,BPEL 引擎可以依照算法 5 在目标流程下恢复该实例的执行.

下面我们将利用一些代表性的工作中所提出的实例可迁移判定方法,对实例 1 和实例 2 的可迁移性进行分析,从而可以将这些方法与本文的方法进行比较.

(1)利用文献[3]中的方法对上述实例的可迁移性进行分析.在该方法中,服务组合实例可迁移需满足向前相容性和向后相容性.向后相容性需保证该实例的已执行活动序列可以在目标模式下按照既定顺序重现.假设服务组合实例在源模式下执行时暂停在状态 (N_s,M),向前相容性需保证该实例在源模式的状态 (N_s,M) 下的每一条后续活动执行序列都可以在目标模式下重现.由于 $\sigma_1=(t_1t_2t_3t_4t_5t_6t_7t_8)^2t_1t_2t_3$ 可以在目标模式下重现,因此实例 1 满足向后相容性;然而,由于可以在状态 $(N_s,[s_4])$ 下找

到一个后继活动序列 $\sigma_1'=t_4t_5t_6t_7t_9t_{11}t_{10}t_{12}t_{13}$ 不能在目标模式下重现,说明实例 1 不满足向前兼容性.因此在该方法中,实例 1 不能迁移到目标模式下.同理,可以验证实例 2 不满足向后兼容性,因此实例 2 也不可迁移.

(2)利用文献[11]中的方法对上述实例的可迁移性进行分析.该方法能够适用的前提是:服务组合源模式与目标模式之间满足继承关系.这种继承关系不允许将以前串行执行的活动改为并行执行或者将并行执行的活动改为串行执行.不难发现,图 7(a)和(b)中的模式之间不满足继承关系,因此在该方法中,实例 1 和实例 2 均不可迁移.

(3)利用文献[12]中的方法对上述实例的可迁移性进行分析.该方法能够适用的前提是:待迁移的服务组合实例当前不能运行在变化区域内.通过该方法,可发现实例 1 执行在变化区域内,而实例 2 已经离开变化区域,因此在该方法中,实例 1 不可迁移,而实例 2 可迁移.

(4)利用文献[13-15]中的方法对上述实例的可迁移性进行分析.该方法的可迁移性标准与本文的可迁移性标准 2 类似.利用该方法,容易判定实例 1 的已执行活动序列 σ_1 可以在目标模式下重现,因此实例 1 是可迁移的.而实例 2 的已执行活动序列 σ_2 无法在目标模式下按照既定的顺序重现,因此在此标准下,实例 2 是不可迁移的.

表 3 对上述比较结果进行了汇总,其中符号√(×)表示该实例根据相关的可迁移性判定方法是可迁移的(不可迁移的).本文的定理 1 保证了服务组合实例 1 和实例 2 迁移后不会引发死锁等动态演化错误,因此应该将上述实例迁移到目标流程下执行.表 3 说明了相关工作中的实例可迁移判定方法还过于严格,不能使更多的服务组合实例动态地迁移到目标流程下执行.而本文的可迁移型标准具有更好的柔性.

表 3　上述实例在相关工作中的可迁移性比较

	文献[3]的工作	文献[11]的工作	文献[12]的工作	文献[11-15]的工作	本文的可迁移性标准 4
实例 1	×	×	×	√	√
实例 2	×	×	√	×	√

上述分析阐明了本文所提出的 Web 服务组合的动态演化方法可以方便地应用到旅行代理这一具体的服务组合案例中.与已有的方法相比,该标准在确保不会产生动态演化错误的同时,可允许更多的实例迁移.下一步,我们将探讨如何将本文的方法应用到更多的服务组合场景,以进一步验证本文方法的有效性和可行性.

6　结束语

本文提出了一个基于模型的 Web 服务组合动态演化过程框架,在此框架下,引入了相应的服务组合实例的可迁移性标准,并且证明了通过该可迁移性标准所获得的可迁移实例的目标状态是有效的,即迁移后不会引入一些动态演化的错误.相对于已有的服务组合可迁移性判定方法,本文的方法具有更好的柔性,它可以让更多的服务组合实例迁移到目标流程下执行.此外,本文还给出了可迁移性标准的算法实现,该算法不仅可以判定任一服务组合实例的可迁移性,而且当该实例可迁移时可以自动求出有效的目标状态.

我们下一步的工作包括:(1)探究服务组合实例可迁移的"源标准"(例如目标状态的有效性)与具有可操作性的可迁移性标准之间的关系,这有助于我们判断可迁移性标准的松弛程度;(2)研究当服务组合的待迁移实例数目较多时如何高效地判定待迁移实例的可迁移性;(3)研究服务编排(Choreography)层面的动态演化问题.

致　谢　感谢香港科技大学计算机科学及工程系张成志教授及相关课题组对我们的帮助,感谢审稿专家富有建设性的意见!

参　考　文　献

[1] Yu Jian, Han Yan-Bo. Service Oriented Computing—Principle and Application. Beijing: Tsinghua University Press, 2006(in Chinese)
(喻坚,韩燕波. 面向服务的计算——原理与应用. 北京:清华大学出版社, 2006)

[2] Peltz C. Web services orchestration and choreography. IEEE Computer, 2003, 36(10): 46-52

[3] Ryu S H, Casati F, Skogsrud H, Benatallah B, Saint-Paul R. Supporting the dynamic evolution of Web service protocols in service-oriented architectures. ACM Transactions on the Web, 2008, 2(2): 1-45

[4] Song W, Ma X X, Dou W C, Lü J. Toward a model-based approach to dynamic adaptation of composite services//Proceedings of the IEEE International Conference on Web Services. Beijing, China, 2008: 561-568

[5] Papazoglou M P. The challenges of service evolution//Proceedings of the International Conference on Advanced Infor-

mation Systems Engineering. Montpellier, France, 2008: 1-15

[6] Liu X M, Bouguettaya A. Managing top-down changes in service-oriented enterprises//Proceedings of the IEEE International Conference on Web Services. Utah, USA, 2007: 1072-1079

[7] König D, Lohmsnn N, Moser S, Stahl C, Wolf K. Extending the compatibility notion for abstract WS-BPEL processes//Proceedings of the International World Wide Web Conference. Beijing, China, 2008: 785-794

[8] Lohmann N, Massuthe P, Wolf K. Operating guidelines for finite-state services//Proceedings of the International Conference on Application and Theory of Petri Nets and Other Models of Concurrency. Siedlce, Poland, 2007: 321-341

[9] Decker G, Weske M. Behavioral consistency for B2B process integration//Proceedings of the International Conference on Advanced Information Systems Engineering. Trondheim, Norway, 2007: 81-95

[10] Casati F, Ceri S, Pernici B, Pozzi G. Workflow evolution. Data & Knowledge Engineering, 1998, 24(3): 211-238

[11] van der Aalst W M P, Basten T. Inheritance of workflows: An approach to tackling problems related to change. Theoretical Computer Science, 2002, 270(1-2): 125-203

[12] van der Aalst W M P. Exterminating the dynzamic changes bug: A concrete approach to support workflow change. Information Systems Frontiers, 2001, 3(3): 297-317

[13] Rinderle S, Reichert M, Dadam P. Flexible support of team processes by adaptive workflow systems. Distributed and Parallel Databases, 2004, 16(1): 91-116

[14] Rinderle S, Reichert M, Dadam P. Correctness criteria for dynamic changes in workflow systems — A survey. Data & Knowledge Engineering, 2004, 50(1): 9-34

[15] Rinderle S, Reichert M, Weber B. Relaxed compliance notions in adaptive process management systems//Proceedings of the International Conference on Conceptual Modelling. Catalonia, Spain, 2008: 232-247

[16] Andrikopoulos V, Benbernou S, Papazoglou1 M P. Managing the evolution of service specifications//Proceedings of the International Conference on Advanced Information Systems Engineering. Montpellier, France, 2008: 359-374

[17] Zhang J, Cheng B H C. Model-based development of dynamically adaptive software//Proceedings of the International Conference on Software Engineering. Shanghai, China, 2006: 371-380

[18] Oreizy P, Gorlick M M, Taylor R N et al. An architecture-based approach to self-adaptive software. IEEE Intelligent System, 1999, 14(3): 54-62

[19] Garlan D, Cheng S W, Huang A C, Scomerl B, Steenkiste P. Rainbow: Architecture-based self-adaptation with reusable infrastructure. IEEE Computer, 2004, 37(10): 46-54

[20] Huang Gang, Wang Qian-Xiang, Mei Hong, Yang Fu-Qing. Research on architecture-based reflective middleware. Journal of Software, 2003, 14(11): 1819-1826(in Chinese)
(黄罡, 王千祥, 梅宏, 杨芙清. 基于软件体系结构的反射式中间件研究. 软件学报, 2003, 14(11): 1819-1826)

[21] Ma Xiao-Xing, Yu Ping, Tao Xiao-Ping, Lu Jian. A service-oriented dynamic coordination architecture and its supporting system. Chinese Journal of Computers, 2005, 28(4): 467-477(in Chinese)
(马晓星, 余萍, 陶先平, 吕建. 一种面向服务的动态协同架构及其支撑平台. 计算机学报, 2005, 28(4): 467-477)

[22] Wu Zhe-Hui. Introduction to Petri Nets. Beijing: China Machine Press, 2006(in Chinese)
(吴哲辉. Petri 网导论. 北京: 机械工业出版社, 2006)

[23] Massuthe P, Reisig W, Schmidt K. An operating guideline approach to the SOA. Annals of Mathematics, Computing & Teleinformatics, 2005, 1(3): 35-43

[24] van der Aalst W M P, Lohmann N, Massuthe P, Stahl C, Wolf K. From public views to private views — Correctness-by-design for services//Proceedings of the International Workshop on Web Services and Formal Methods. Brisbane, Australia, 2007: 139-153

[25] Lohmann N, Massuthe P, Stahl C, Weinberg D. Analyzing interacting BPEL processes//Proceedings of the International Conference on Business Process Management. Vienna, Austria, 2006: 17-32

SONG Wei, born in 1981, Ph. D. candidate. His research interests include software engineering and methodology issues on services computing.

MA Xiao-Xing, born in 1975, Ph. D., professor. His current research interests include Internet-based software systems, software architectures and software components.

LU Jian, born in 1960, Ph. D., professor, Ph. D. supervisor. His research interests include software automation, formal methods for parallel programming, object-oriented languages and environments, and Internetware.

Background

This work is supported by the National Basic Research Program (973 Program) of China under grant No. 2009CB320702, the National High Technology Research and Development Program (863 Program) of China under grant No. 2007AA01Z178 and No. 2009AA01Z117, the National Natural Science Foundation of China under grant No. 60736015, and No. 60721002. The common theme of these projects is to investigate the software methodology and the enabling technology for Internet-based applications. This paper focuses on dynamic evolution, an import issue of Internet-based applications, especially, service-oriented applications.

Recently, service-oriented computing (SOC) has become a promising computing paradigm for developing distributed Internet applications. In this paradigm, value-added Internet applications can be constructed by outsourcing various services distributed in the network. Service composition is such an enabling technique to this end which has attracted much attention from both the academia and the industry.

Service compositions are subject to constant changes in open SOAs environments. One of the key challenges is how to evolve the composite services to adapt to the changing environments and evolving business rules. Recently, there are some researches concentrating on the static evolution of service compositions at both the orchestration level and the choreography level. However, few works have focused on the dynamic evolution of service compositions. This paper particularly focuses on dynamic evolution of service compositions at the orchestration level.

This paper proposes a novel framework for evolving Web service orchestrations. Under this framework, a new criterion for the migration of running service orchestration instances is devised, as well the corresponding checking algorithm. Compared with existing migratability criteria, the criterion allows the migration of more running instances without introducing any potential dynamic change bugs. To summarize, the proposed approach is more flexible than existing ones and thus more practical.

The research group has worked for several years on the topic of dynamic software adaptation and evolution in open, dynamic and autonomous network environments such as the Internet. Previous works focused on supporting dynamic evolution of software systems from the structural perspective. More specifically, authors investigated how to utilize runtime software architecture to support dynamic evolution of software systems. Authors have prototyped this technique in the authors' ARTEMIS middleware system. These results have been published in journals such as Science in China Series F: Information Sciences, Chinese Journal of Computers, Journal of Software, and some international conferences such as ICSE Doctoral Symposium, SEKE, ICWS, to name a few. To complement the previous research, this paper focuses on supporting dynamic evolution of service-oriented systems from the behavioral perspective.