ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.21, No.2, February 2010, pp.261−276
doi: 10.3724/SP.J.1001.2010.03735

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

基于服务组合的可信软件动态演化机制*

曾 晋+, 孙海龙, 刘旭东, 邓 婷, 怀进鹏

(北京航空航天大学 计算机学院,北京 100191)

Dynamic Evolution Mechanism for Trustworthy Software Based on Service Composition

ZENG Jin+, SUN Hai-Long, LIU Xu-Dong, DENG Ting, HUAI Jin-Peng

(School of Computer Science and Engineering, BeiHang University, Beijing 100191, China)

+ Corresponding author: E-mail: zengjin@act.buaa.edu.cn

Zeng J, Sun HL, Liu XD, Deng T, Huai JP. Dynamic evolution mechanism for trustworthy software based on service composition. *Journal of Software*, 2010,21(2):261−276. http://www.jos.org.cn/1000-9825/3735.htm

Abstract: This paper is concerned with trustworthy software constructed through service composition and is especially emphasizes guaranteeing the trustworthiness of networked software through dynamic evolution of composite services. First of all, a set of evolution operations preserving soundness of composite services is proposed so as to avoid the complex verification process. Second, a composite service evolution method with availability guarantee is provided, the main idea of which is to create redundant execution path to improve the availability of a composite service. Third, to deal with runtime instances after dynamic composite service evolution, a live instance migration algorithm is designed to support the correct evolution enforcement. Finally, a composite service execution engine supporting dynamic evolution is developed and the effectiveness of the proposed method is showed through a set of experiments.

Key words: trustworthy software; dynamic evolution; composite service; Web service

摘 要: 以基于服务组合的可信软件为研究对象,重点研究通过组合服务的动态演化机制保障网络化软件的可信性.首先,提出了一个合理性保持的演化操作集,避免复杂的验证过程,使得演化后的组合服务保持结构合理性;在此基础上,通过构造冗余路径的方式给出了一个面向可用性保障的组合服务演化方法;然后,针对组合服务动态演化过程中运行实例的处理,设计了一个组合服务演化中运行实例在线迁移算法,为正确实施演化提供支持;最后,设计实现了一个支持动态演化的组合服务执行引擎,并通过实验验证了所提出方法的有效性.

关键词: 可信软件;动态演化;组合服务;Web 服务

中图法分类号: TP311 **文献标识码**: A

随着 Internet 的广泛应用和网络技术的快速发展,面向服务的软件体系结构 SOA(service oriented architecture)作为一种新型的网络化软件应用模式已经被工业界和学术界广为接受.特别是作为实现 SOA 的重要技术,Web 服务极大地推动了 SOA 在电子商务、金融、电信等领域的应用.在实际应用中,单个的 Web 服务

* Supported by the National High-Tech Research and Development Plan of China under Grant Nos.2007AA010301, 2006AA01A106, 2009AA01Z419 (国家高技术研究发展计划(863))

Received 2009-06-15; Accepted 2009-09-11

功能有限,往往难以满足复杂的业务需求.因此,服务组合成为网络化软件开发的主要方法,基于这种方法所开发的软件在技术形态上体现为组合服务.

由于软件本身和软件环境的复杂性日益增加,基于服务组合的网络化软件面临着很多挑战,特别是软件可信性保障成为一个重要的研究问题.通常,软件的可信性[1]是指软件的行为、结果和用户的预期是一致的,其外延包括软件的可用性、可靠性、完整性、可维护性和可生存性等.Internet的开放性、分布自治性和无中心控制,以及软件系统本身的异构性和动态性等特征进一步增加了保障软件可信性的难度.特别是受到应用需求、软件运行环境以及用户请求的动态变化的影响,网络化软件的功能和非功能属性是动态变化的,例如电子商务应用动态地增加新型的第三方支付服务、网络拥塞导致在线CRM系统执行效率降低等,仅仅依靠软件开发期模型分析、软件验证和测试等保障软件可信性的静态方法已经难以适应网络化软件的可信性保障需求,因此必须研究一种能够动态地保障网络化软件可信性的方法.基于以上分析,本文以基于服务组合的可信软件为研究对象,重点研究通过组合服务的动态演化机制保障网络化软件的可信性.结合网构软件的演化性定义[2],本文中的组合服务演化是指组合服务的业务流程结构可以根据应用需求和网络环境变化而进行动态调整,主要表现在其组件服务数目的可变性、结构关系的可调节性和结构形态的动态可配置性上.一般来说,流程结构调整和组件服务的选择是组合服务演化的两种主要形式.

在国家高技术研究发展计划(863)课题“可信的国家软件资源共享与协同生产环境”的支持下,我们承担了子课题“面向服务的软件生产线”的研发,旨在以服务计算技术解决Internet环境下软件的设计、生产和运行维护问题,其中的一个重要的问题就是基于服务组合的软件可信性保障.在课题研发中,通过研究组合服务的动态演化方法以保障网络化软件的可信性.已有的服务组合动态演化工作[3−8]主要涵盖了演化时间、演化操作分类以及演化带来的影响等方面,在此基础上,本文拟从以下 3 个方面研究基于服务组合的可信软件动态演化问题:首先,在组合服务演化中的一个基本问题是如何保证演化的正确性(correctness)[6].当演化操作实施不当时,可能会给已有的组合服务带来额外问题,如逻辑死锁、组件服务不可达等.特别地,对于面向流程的组合服务,合理性(soundness)[3,9]是其正确性的重要准则.但是,已有的验证方法复杂度过高[10],对于一些要求持续在线的组合服务动态演化,过长的验证时间是难以接受的.因此需要给出一种方法,能够最大限度地避免复杂的验证过程,并且保证演化后的组合服务业务流程仍然具有合理性.其次,在保证组合服务演化的正确性基础上,如何通过动态演化保障组合服务的可信性.如前所述,可信性作为软件的一个综合属性,具体表现为可用性、可靠性和响应时间等[11].传统的方法是在流程结构确定的情况下,通过规划方法进行组件服务的选择[12]以动态地保障组合服务的可信性,单纯依靠这种方法在某些情况下难以奏效.例如,在所有组件服务的可用性都很低的情况下,无法单一通过选择不同的组件服务来保障组合服务的整体可用性.在这种情况下,流程结构演化能够有效弥补其不足,本文重点研究通过流程结构的演化来动态保障组合服务的可信性.最后,在组合服务演化实施方面,当组合服务的流程结构动态演化后,如何处理正在运行的组合服务实例是必须要考虑的问题.例如,电子商务、金融应用等关键事务处理系统都需要 7×24 小时不间断执行,且同时运行着大量的组合服务实例.中断现有的业务进行事务回滚和补偿,会带来大量的额外花费和时间开销[3].因此,需要一种在组合服务的流程结构动态演化后在线处理运行实例的机制,以保证流程实例能够正确地在线迁移到新的流程结构.

针对以上的 3 个问题,本文的研究思路如图 1 所示.首先,为了保障演化操作的可信性,提出了一个业务流程合理性保持的演化操作集.该集合中定义了流程结构的基本演化操作,并且证明了使用这些操作对确保演化后流程结构的合理性,从而避免了传统方法在演化后的复杂验证过程.其次,针对演化结果的可信性问题,特别针对可用性这一可信属性,给出了一种面向可用性保障的组合服务演化方法.该方法通过分析组合服务业务流程结构,使用基本演化操作集构造冗余路径来保证业务流程中“关键区域”的可用性.再次,针对组合服务动态演化实施过程的可信性问题,设计了一种组合服务演化中运行实例在线迁移方法,能够判断运行实例是否能够运行时迁移,并且可以准确地计算出迁移后的实例状态.最后,基于面向方面编程AOP(aspect oriented programming)[13]的思想,设计实现了一个支持动态演化的组合服务执行引擎以及相关的实验,证明了可信的组合服务动态演化方法的合理性和有效性.

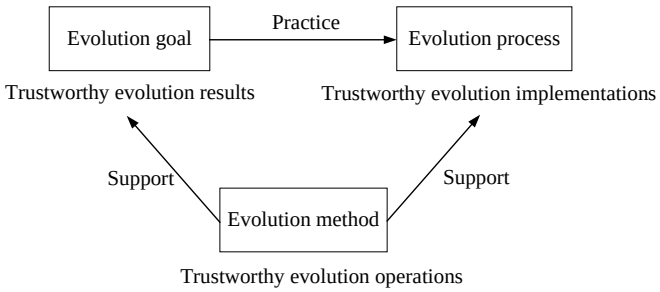


Fig.1 Research on dynamic evolution of trustworthy software based on service composition

图 1 基于服务组合的可信软件动态演化研究思路

本文第 1 节分析与本文相关的研究工作.第 2 节定义组合服务流程合理性保持的演化操作集.第 3 节给出一个面向可用性保障的组合服务演化方法.第 4 节中设计一种组合服务演化中运行实例在线迁移方法.第 5 节介绍一个支持动态演化的组合服务执行引擎以及相关的实验.最后对本文的工作进行总结.

1 相关工作

文献[6]讨论了演化的正确性问题,认为需要一个正确性准则来判断演化后的组合服务是否具有原有的性质并且不导致额外错误,如逻辑死锁、组件服务不可达等.但是对于面向流程的组合服务,一个最重要的正确性准则是保证演化后组合服务的业务流程结构的合理性[3,9],即保证组合服务正确地终结.对于一个复杂的业务流程难以直接地判断其是否满足合理性,一般通过判定流程结构的活性和有界性来验证其合理性,但其算法复杂度是EXPSPACE-hard的[10].特别是对于一些要求持续在线的组合服务的动态演化,过长的验证时间是难以接受的.因此,给出一个具有合理性保持功能的演化操作集是一个可行的解决方案.在保障组合服务可信性方面,一般的方法是先刻画组合服务业务流程的结构和组件服务的功能,然后在服务实际运行前,根据用户的可用性需求以及业务流程的结构采用规划方法来自动选择具体的组件服务[12].另外一些工作考虑到了运行期组件服务可用性的变化,提出了通过重规划方法[14]支持组合服务在运行期动态重新选择组件服务,以更好地满足可用性需求.但是在很多情况下(例如,没有足够的候选服务,或者候选服务的可用性都很低),仅仅通过动态选择组件服务难以保障所需要的可用性.在实施演化过程时,运行实例在线迁移是一个具有挑战性的问题,即使演化后的组合服务是合理的也不能保证任何状态的实例都能迁移,可能导致动态变更错误(dynamic change bug)[3,15,16],即在新的组合服务运行实例中找不到与旧组合服务业务流程结构约束下的运行实例状态相对应的状态.因此,提出了变更区域(change regions)的概念和求解方法,用于限制某些特定流程结构下运行实例的迁移.然而,变更区域的计算复杂度过高($O(n^4(n!)^2)$)[16].由此提出了流程继承[17]和松弛的流程投影[18]方法.但是,这些方法对业务流程结构性调整的支持尚不完善,例如,不支持流程结构调整;其次,针对演化后组合服务包含的组件服务以及结构关系与旧实例不一致的情况,当前也没有一种完善的解决方法来确定新的状态位置.

2 组合服务流程合理性保持的演化操作集

为了精确地描述基于服务组合的可信软件动态演化研究,本文采用工作流网(workflow net,简称WF-net)[9]形式化地描述组合服务.工作流网是一种特殊的Petri网,具有明确的形式化语义定义、图形化的表示方法以及成熟的分析技术.

定义 1(工作流网(WF-net))[9]. 一个Petri网$WFN=(P,T,F)$称为工作流网,当且仅当:

- 存在一个初始库所$i\in P$,使得${}^\bullet i=\varnothing$;
- 存在一个终结库所$o\in P$,使得$o^\bullet=\varnothing$;
- 每一个节点$x\in P\cup T$都位于从i到o的一条路径上.

工作流网非常适合于刻画组合服务业务流程的动态行为,在工作流网中,组件服务采用变迁(transition)来

表示,变迁被触发的条件采用库所(place)表示,而流关系(flow relationship)用来规范变迁间的逻辑结构关系.此外,一个工作流网的状态(state)被定义为其所有库所中标记(token)的分布情况,使用一个$|P|$维向量M来表示在工作流网中正在运行的一个实例的实时状态,M中的每一个元素意味着对应库所中标记的数量,其中,M_p用来表示库所p中标记数量.特别地,使用M_0(M_{end})表示初始状态(终结状态),其中,M_0(M_{end})中只有初始库所(终结库所)有标记.一个变迁$t\in T$在状态M下处于使能状态,当且仅当对于每一个$(p,t)\in F$使得$M_p>0$.并且,如果t是使能的,那么t能够使工作流网进入新的状态M',满足对于每一个$(p,t)\in F$和$(t,p')\in F$,使得 $M'_p=M_p-1$ 并且 $M'_{p'}=M_{p'}+1$,记作 $M[t\rangle M'$.

定义 2(合理的工作流网(sound WF-net))[9]. 一个工作流网$WFN=(P,T,F,i,o,M_0)$是合理的,当且仅当:

- 对于每一个从初始状态M_0的可达状态M,存在一个从状态M到终结状态M_{end}的变迁序列,形式化表示为

$$\forall M(M_0 \xrightarrow{*} M) \Rightarrow (M \xrightarrow{*} M_{end}).$$

- 终结状态M_{end}是从初始状态M_0可达的惟一终止状态,且结束时其中至少会有 1 个标记,形式化表示为

$$\forall M(M_0 \xrightarrow{*} M \wedge M \geq M_{end}) \Rightarrow (M = M_{end}).$$

- 在工作流网 WFN 中没有死变迁,形式化表示为

$$\forall t \in T\ \exists M,\ M' \text{ s.t. } M_0 \xrightarrow{*} M \xrightarrow{t} M'.$$

合理性是一个重要的业务流程结构正确性准则,它能够保证组合服务正确的终结.定义 2 中的第 1 个条件规定了从初始状态M_0开始,总可能达到终结状态M_{end};第 2 个条件指出了工作流网处于M_{end}状态时,即一个标记被放入库所o的时刻,所有其他库所应该是空的;最后一个条件规定了从初始状态M_0不存在死变迁.一般来说,对于一个复杂的工作流网难以直接判断其合理性.在本文中,不是在组合服务演化后再验证其合理性,而是通过定义一个基本演化操作集来保持原组合服务的合理性.另外,本文中所有的组合服务均采用无循环结构的工作流网进行建模,并且假设每个组件服务(变迁)在一个组合服务中最多出现 1 次,并且本文不考虑任何数据流和数据约束问题.

下面给出一个组合服务合理性保持的演化操作集 OP,包括替换、增加、删除和结构调整等操作,保证了合理工作流网采用 OP 中任意有限个操作进行演化后仍然是合理的.

定义 3(替换操作(replacement operation)). 设$WFN_1=(P_1,T_1,F_1,i_1,o_1,M_{01})$是一个合理工作流网$WFN=(P,T,F,i,o,M_0)$的合理子网,且$WFN_2=(P_2,T_2,F_2,i_2,o_2,M_{02})$是合理的工作流网,并且$P_1\cap P_2=\varnothing$,$T_1\cap T_2=\varnothing$,$F_1\cap F_2=\varnothing$,那么使用 WFN_2 替换 WFN_1 后获得 $WFN'=(P',T',F',i',o',M'_0)$ 是一个工作流网,其中,$P'=(P\backslash P_1)\cup P_2$,$T'=(T\backslash T_1)\cup T_2$,$F'=(F\backslash F_1)\cup F_2\cup F''$满足$F''=\{(x,i_2)\in P\times T_2|(x,i_1)\in F_1\}\cup\{(o_2,y)\in T_2\times P|(o_1,y)\in F_1\}$.初始状态 M'_0 是一个$|P'|$维度的向量.这里,$S\backslash S'$表示集合S与S'中不同的元素的集合,即该集合中的元素在S中但不在S'中(如图 2 所示).

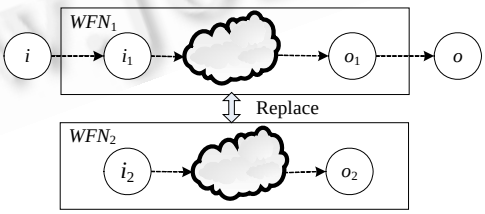


Fig.2 Replacement

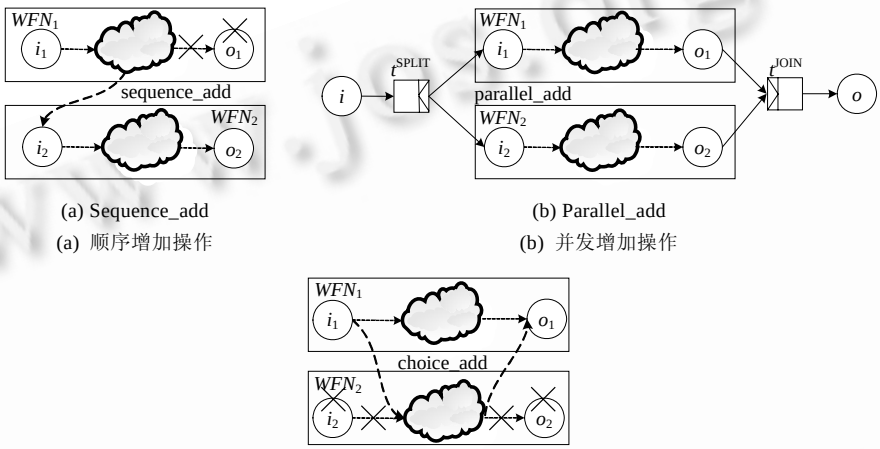
图 2 替换操作

显然,一个合理工作流网中的某个变迁被替换为另一个合理的工作流网后,得到的新工作流网仍然是合理的(详见文献[19]中的定理 3).而替换操作是上述结论的一个延伸,因为从外部来看,一个合理工作流网的行为与一个变迁相似.因此,我们有如下结论:

命题 1. 替换操作能够保持一个合理工作流网的合理性.

定义 4(增加操作(addition operation)). 令$WFN_1=(P_1,T_1,F_1,i_1,o_1,M_{01})$和$WFN_2=(P_2,T_2,F_2,i_2,o_2,M_{02})$是两个合理的工作流网,且$P_1\cap P_2=\varnothing,T_1\cap T_2=\varnothing,F_1\cap F_2=\varnothing$:

- **顺序增加(sequence_add($WFN_1\rightarrow WFN_2$))**. 称$WFN=(P,T,F,i_1,o_2,M_0)$是把WFN_2顺序增加到WFN_1后获得的工作流网,其中,$P=(P_1\setminus\{o_1\})\cup P_2,T=T_1\cup T_2,F=\{(x,y)\in F_1|y\neq o_1\}\cup\{(x,i_2)\in T_2\times P_1|(x,o_1)\in F_1\}\cup F_2$(如图 3(a)所示).
- **并发增加(parallel_add($WFN_1\|WFN_2$))**. 称$WFN=(P,T,F,i_1,o_2,M_0)$是把WFN_2并发增加到WFN_1上获得的工作流网,其中,$P=P_1\cup P_2\cup\{i,o\},T=T_1\cup T_2\cup\{t^{\mathrm{SPLIT}},t^{\mathrm{JOIN}}\},F=F_1\cup F_2\cup\{\langle i,t^{\mathrm{SPLIT}}\rangle,\langle t^{\mathrm{SPLIT}},i_1\rangle,\langle t^{\mathrm{SPLIT}},i_2\rangle,\langle o_1,t^{\mathrm{JOIN}}\rangle,\langle o_2,t^{\mathrm{JOIN}}\rangle,\langle t^{\mathrm{JOIN}},o\rangle\}$(如图 3(b)所示).
- **选择增加(choice_add(WFN_1+WFN_2))**. 称$WFN=(P,T,F,i_1,o_2,M_0)$是把WFN_2选择增加到WFN_1上获得的工作流网,其中,$P=P_1\cup(P_2\setminus\{i_2,o_2\}),T=T_1\cup T_2,F=F_1\cup\{(x,y)\in F_2|x\neq i_2\wedge y\neq o_2\}\cup\{(i_1,y)\in P_1\times T_2|(i_2,y)\in F_2\}\cup\{(x,o_1)\in T_2\times P_1|(x,o_2)\in F_2\}$(如图 3(c)所示).



(a) Sequence_add
(a) 顺序增加操作

(b) Parallel_add
(b) 并发增加操作

(c) Choice_add
(c) 选择增加操作

Fig.3 Addition
图 3 增加操作

上述的增加操作是采用顺序、并发以及选择方式将一个合理的工作流网增加到另一个合理的工作流网上,显然获得的新工作流网是合理的.因此有如下的结论:

命题 2. 增加操作能够保持一个合理工作流网的合理性.

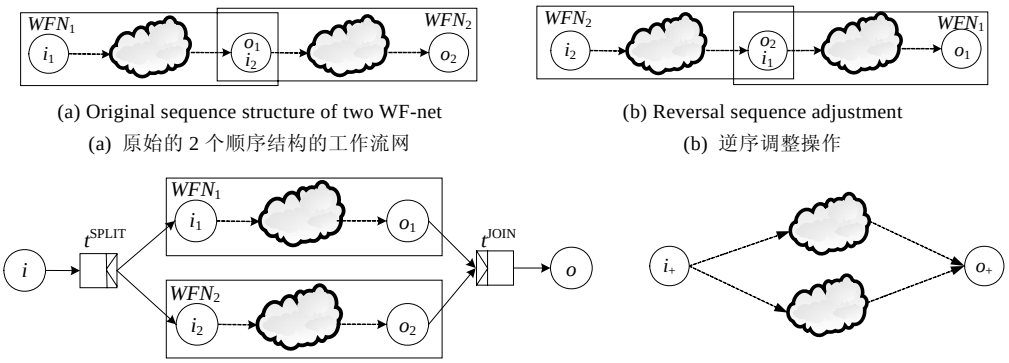
删除操作是上述增加操作的逆操作,同样存在顺序删除、并发删除和选择删除等具体操作,限于篇幅,这里不详细讨论其定义,但是可以很容易地证明删除操作也是合理性保持的.对于组合服务的演化,除了替换、增加和删除操作以外,一些应用场景还需要对原有的组合服务业务流程结构进行调整.本文定义了如下的结构调整操作,它们均是对一个合理工作流网的相关合理子网进行调整.

定义 5(顺序结构调整(sequence structure adjustments)). 令$WFN=(P,T,F,i,o,M_0)$是由$WFN_1=(P_1,T_1,F_1,i_1,o_1,M_{01})$和$WFN_2=(P_2,T_2,F_2,i_2,o_2,M_{02})$顺序组成的合理工作流网,其中,$P=P_1\cup P_2,T=T_1\cup T_2,F=F_1\cup F_2,i=i_1,o_1=i_2,o=o_2$(如图 4(a)所示):

- **逆序调整(reversal_sequence adjustment)**. $WFN'=(P,T,F,i',o',M_0)$是由WFN_1和WFN_2经过逆序调整后得到的工作流网,其中,$i'=i_2,o'=o_1,o_2=i_1$(如图 4(b)所示).
- **顺序并发调整(sequence-to-parallel adjustment)**. $WFN_{\|}=(P_{\|},T_{\|},F_{\|},i_{\|},o_{\|},M_{0\|})$是由$WFN_1$和$WFN_2$经过顺序并发调整后得到的工作流网,其中,$P_{\|}=P\cup\{i_{\|},o_{\|}\},T_{\|}=T\cup\{t^{\mathrm{SPLIT}},t^{\mathrm{JOIN}}\},F_{\|}=F\cup\{\langle i_{\|},t^{\mathrm{SPLIT}}\rangle,\langle t^{\mathrm{SPLIT}},i_1\rangle,\langle t^{\mathrm{SPLIT}},$

$i_2\rangle$,$\langle o_1,t^{JOIN}\rangle$,$\langle o_2,t^{JOIN}\rangle$,$\langle t^{JOIN},o_{\|}\rangle\}$(如图 4(c)所示).

- **顺序选择调整(sequence_to_choice adjustment)**. $WFN_+=(P_+,T_+,F_+,i_+,o_+,M_{0+})$是由$WFN_1$和$WFN_2$经过顺序选择调整后得到的工作流网,其中,$P_+=(P\setminus\{i_x,i_y,o_x,o_y\})\cup\{i_+,o_+\}$,$T_+=T$,$F_+=\{(p,q)\in F|p\neq i_1,i_2\wedge q\neq o_1,o_2\}\cup\{(i_+,q)|(i_1,q)\in F_1\vee(i_2,q)\in F_1\}\cup\{(p,o_+)|(p,o_1)\in F_1\vee(p,o_2)\in F_2\}$(如图 4(d)所示).



(a) Original sequence structure of two WF-net
(a) 原始的 2 个顺序结构的工作流网

(b) Reversal sequence adjustment
(b) 逆序调整操作

(c) Sequence_to_parallel adjustment
(c) 顺序并发调整操作

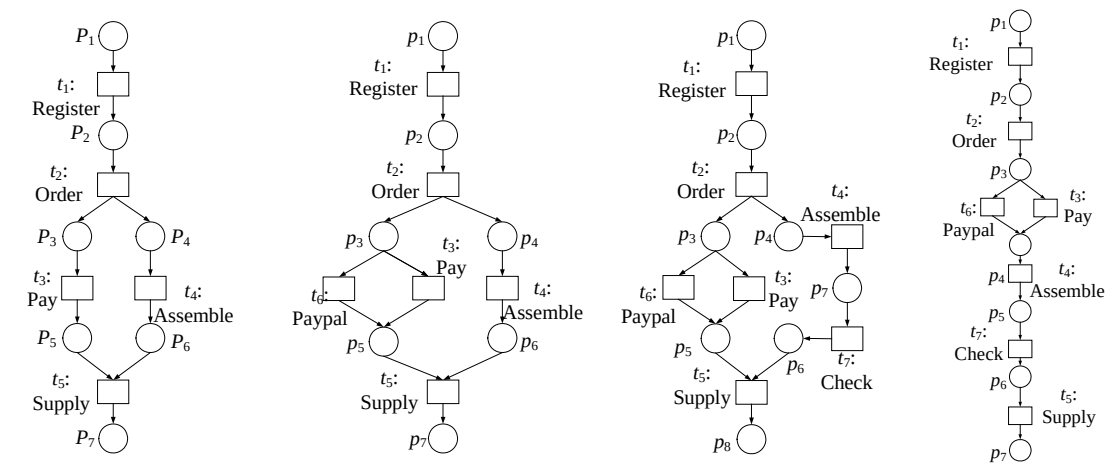
(d) Sequence_to_choice adjustment
(d) 顺序选择调整操作

Fig.4 Sequence structure adjustments
图 4 顺序结构调整操作

同样限于篇幅,这里省略并发顺序调整和选择顺序调整的定义,它们分别是顺序并发调整和顺序选择调整的逆操作.本文提出的结构调整操作是在两个合理的工作流子网间进行,可以从定义和图示中明确地看到调整后所获得的新工作流网是合理的,因此有如下的结论:

命题 3. 结构调整操作操作能够保持一个合理工作流网的合理性.

这里给出一个供应链管理中的订单处理组合服务的应用场景(如图 5 所示).



(a) Original composite service
(a) 原始的组合服务

(b) Add a service Paypal
(b) 增加付款服务 Paypal

(c) Add a service Check
(c) 增加检查服务

(d) Structure adjustment
(d) 结构调整

Fig.5 Evolution process of composite service order-processing
图 5 订单处理组合服务的演化过程

图 5(a)描述了一个订单处理组合服务,其中包括用户注册(register)、提交订单(order)、银行转账(pay)、货物装配(assemble)以及运输服务(supply)这 5 个组件服务(变迁).其中,转账服务与装配服务因没有直接的因果关

系所以并发执行.为了支持多种支付方法,该组合服务在传统的银行转账付款功能基础上,需要增加对 Paypal(贝宝,一种第三方的在线转账服务)付款的支持,这时,可以应用选择增加操作加入 Paypal 服务(如图 5(b)所示).考虑到货物装配时可能会发生错误,导致用户得到的货物与订单不一致,因此采用顺序增加操作在装配服务后增加了一个检查服务 Check(如图 5(c)所示).此外,考虑到有些恶意客户虽然订购货物但是并不付款,导致货物被无效地装配,因此对业务流程结构应用并发顺序调整操作,使得用户先付款再装配货物(如图 5(d)所示).

3 面向可用性保障的组合服务演化方法

可信性是软件的一种综合度量属性,而可用性是其中的一个重要度量指标.本节以软件可用性保障为目标,给出如何基于第 2 节提出的合理性保持的演化操作集,通过组合服务的演化达到这一目标.本节提出一种基于动态修改组合服务业务流程的结构保障可用性的方法,其基本思路就是在原有流程结构中添加合适的冗余路径.但是,对一条完整的流程路径进行冗余会产生很大的开销,因为其可能包含数十个组件服务,而简单从中选取一段可用性差的子路径进行冗余,很可能破坏原始流程结构的合理性或者改变原有组合服务的功能.基于以上考虑,本文提出了一种可用性保证的组合服务演化方法——AvailEvo.该方法旨在根据已有组合服务的执行记录和逻辑结构生成一个合适的组件服务冗余路径.该冗余路径的加入不影响原有组合服务的合理性和功能,并且能够有效提高组合服务的可用性.

AvailEvo 方法可以看作是对传统基于规划的服务组合方法的延伸和优化,其基本思想是:首先通过对组合服务历史执行记录的分析,找到影响可用性指标下降的组件服务集合以及执行频率最高的变迁路径,由此可以得到一个“基本变迁序列”,即该变迁序列中的某些组件服务显著影响了组合服务的整体可用性;然后,基于该变迁序列生成一个冗余路径,并对冗余路径和原来组合服务的一个合理子流程进行选择增加操作,得到新的组合服务业务流程结构;最后,利用经典的规划方法重新选择组件服务.由此可知,该方法的核心问题可以归结为求解合适的冗余变迁序列以及其对应的原来工作流网中的一个合理子网.为了进一步说明 AvailEvo 方法,这里给出变迁路径和变迁间关系的定义:

定义 6(变迁路径(transactions path)). 对于一个合理的工作流网$WFN=(P,T,F,i,o,M_0)$,从初始状态M_0到终结状态M_{end}的变迁序列称作该工作流网上的一个变迁路径,记作tp,显然$tp\in T^*$;另外,将WFN上所有变迁路径的集合称为该工作流网的日志(log),记作L.

定义 7(变迁间的关系(transitions relations)). 设L是工作流网$WFN=(P,T,F,i,o,M_0)$的日志,并且$a,b\in T$是该工作流网中的两个变迁.

- 对于变迁路径$tp=t_1t_2\ldots t_n\in L$,若$t_i=a,t_j=b$且$j\geq i+1$,则称在变迁路径tp上a是b的前驱(或b是a的后继),记作$a>_{tp}b$.
- 对任意的变迁路径$tp\in L$,若tp同时包含变迁a,b时都有$a>_{tp}b$,则称在日志L上a是b的前驱(或b是a的后继),记作$a\rightarrow_L b$.
- 对于变迁路径$tp,tp'\in L$使得$a>_{tp}b$并且$b>_{tp'}a$,则称在日志L上a与b是并发关系,记作$a\|_L b$.
- 对任意的变迁路径$tp\in L$,若tp同时包含变迁a,b时都有$a\not>_{tp}b$并且$b\not>_{tp}a$,则称在日志L上a与b是选择关系,记作$a\#_L b$.

AvailEvo算法给出了可用性保证的组合服务演化方法的处理过程.该算法的输入是描述一个组合服务业务流程结构的合理工作流网WFN^O、经过对执行记录分析后得到的执行频率最高的路径tp和未达到预期可用性指标的组件服务集合S,输出是经过演化后增加了冗余路径的新组合服务WFN^N.首先,根据S和tp可以很容易地得到一个基本变迁序列ts,ts是由tp中包含的变迁集合与S的交集生成,但ts并不只包含S中的变迁.作为一个完整的变迁序列,ts还需要包含必要的tp中的变迁(第 1 行、第 2 行),在此不考虑ts是空或者只有 1 个变迁的特殊情况,即$|ts|\geq 2$.至此,可以认为ts中的部分组件可用性未达到预期值,导致的整体组合服务可用性降低.但是,由ts中变迁组成子工作流网未必是合理的,如果以ts为冗余路径进行演化,则将破坏原有组合服务的执行语义.因此,该算法的重点是以ts为基础求解出一个最小的变迁序列ts',使得ts'中包含的变迁在工作流网WFN中能够组成一个合理

工作流子网WFN'.事实上,求解ts'的过程就是一个对ts进行扩展的过程,通过比较ts中相邻两个变迁的关系来扩展ts.由于ts是一个有向的序列,首先是进行正向扩展,当ts中和t_i存在并发关系的变迁集合Set_1与和t_{i+1}存在并发关系的变迁集合Set_2不相同时,说明它们属于两个不同的并发结构,需要将这两个并发结构中的所有变迁加入ts中(这里需要加入并发结构开始变迁t^{SPLIT}和汇合变迁t^{JOIN});当ts中和t_i存在选择关系的变迁集合与和t_{i+1}存在选择关系的变迁集合不相同时,说明它们属于不同的选择结构,需要将与其处于同一选择结构的变迁加入ts中(第3行~14行),同理进行类似的逆向扩展(第15行~23行).最后,将正向扩展求得的变迁序列$ts_{FORWARD}$与逆向扩展求得的变迁序列$ts_{BACKWARD}$进行合并即得到ts',并且可以求得在WFN^O中由ts'中的变迁组成的合理工作流子网WFN'——关键区域,使用ts'对其进行冗余处理即完成算法(第24行~27行).

AvailEvo Algorithm (Evolve an old process definition to the new one with high availability).

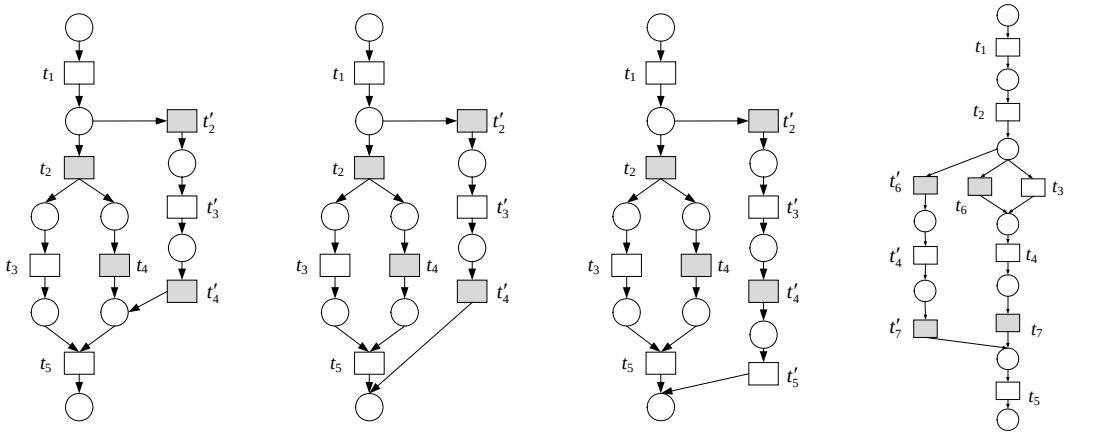
Input: $WFN^O=(P^O,T^O,F^O,i^o,o^o,M_0^O),tp,S$.

Output: $WFN^N=(P^N,T^N,F^N,i^N,o^N,M_0^N)$.

1 **begin**
2 $ts=SubPath(S,tp)$
3 $k=|ts|$
4 $ts_{FORWARD}=ts$
5 $ts_{BACKWARD}=ts$
6 **for** (i=1; $i\le k$; i++)
7 { $t_{i+1}=NEXT(t_i)$
8 **if** $t_{i+1}\neq$NULL
9 { **if** ($Set_1=\{x\in T|t_i\|_L x\})\neq(Set_2=\{x\in T|t_{i+1}\|_L x\})$
10 { $ts_{FORWARD}=ADD\_FORWARD(Set_1\cup\{x\in T|\{y\in T|x\|_L y\}==Set_1\})$
11 $ts_{FORWARD}=ADD\_FORWARD(Set_2\cup\{x\in T|\{y\in T|x\|_L y\}==Set_2\})$ }
12 **if** ($Set_3=\{x\in T|t_i\#_L x\})\neq(Set_4=\{x\in T|t_{i+1}\#_L x\})$
13 { $ts_{FORWARD}=ADD\_FORWARD(\{x\in T|\{y\in T|x\#_L y\}==Set_3\})$
14 $ts_{FORWARD}=ADD\_FORWARD(\{x\in T|\{y\in T|x\#_L y\}==Set_4\})$ } } }
15 **for** ($i=k$; $i\le k$; i−−)
16 { $t_{i-1}=PREVIOUS(t_i)$
17 **if** $t_{i-1}\neq$NULL
18 { **if** ($Set_1=\{x\in T|t_i\|_L x\})\neq(Set_2=\{x\in T|t_{i-1}\|_L x\})$
19 { $ts_{BACKWARD}=ADD\_BACKWARD(Set_1\cup\{x\in T|\{y\in T|x\|_L y\}==Set_1\})$
20 $ts_{BACKWARD}=ADD\_BACKWARD(Set_2\cup\{x\in T|\{y\in T|x\|_L y\}==Set_2\})$ }
21 **if** ($Set_3=\{x\in T|t_i\#_L x\})\neq(Set_4=\{x\in T|t_{i+1}\#_L x\})$
22 { $ts_{BACKWARD}=ADD\_BACKWARD(\{x\in T|\{y\in T|z\#_L y\}==Set_3\})$
23 $ts_{BACKWARD}=ADD\_BACKWARD(\{x\in T|\{y\in T|x\#_L y\}==Set_4\})$ } } }
24 $ts'=SYNCRETIZE(ts_{FORWARD},ts_{BACKWARD})$
25 $WFN'=GENERATION(ts')$
26 $WFN^N=Choice\_add(ts'+WFN')$
27 **end**

AvailEvo 算法的核心在于根据变迁之间的关系来求解冗余路径,而明确变迁间关系的前提必须是已知该工作流网的日志,计算工作流网日志的方法在下一节中具体介绍.这里要说明的是,该方法的最坏时间复杂度是指数的,所以 AvailEvo 算法的最坏时间复杂度也是指数的.另外,AvailEvo 算法的主体是对一个有限长度的基本变迁序列 ts 进行正向和逆向的循环操作,显然是可终结的;并且,该算法在计算过程中实质上是完整地包含了 ts 涉及的顺序、并发和选择结构,按照文献[9]第4.3.3节的结论,冗余路径 ts' 对应的工作流网子网 WFN' 一定是合理的.

图 6 说明了AvailEvo算法在构造冗余路径时对并发结构和选择结构的处理情况.假设上节中图 5(a)所示的组合服务在执行频率最高的变迁路径是$t_1t_2t_3t_4t_5$,其中,t_2和t_4的可用性较差,则基本变迁序列ts为$t_2t_3t_4$.显然,ts不是冗余路径,因为ts中包含的变迁在工作流网中无法对应一个合理子网.如果以ts为冗余路径则会导致演化后的工作流网不合理,如图 6(a)所示,如果执行冗余路径$t'_2t'_3t'_4$则无法正常终结,因为演化后的工作流网不是合理的;而如果如图 6(b)所示,仍然以$t'_2t'_3t'_4$为冗余路径,并使其与一个合理的子网进行选择增加操作,在这种情况下,执行冗余路径会忽略变迁t_5,这显然会破坏原有工作流网的功能.而按照AvailEvo算法对并发结构的处理,需要加入并发结构的汇聚变迁t_5,最终的冗余路径为$t_2t_3t_4t_5$,显然可以对应一个合理的子网,并且在执行冗余路径时的功能与演化前一致(图 6(c)所示).选择结构的处理相对简单,假设图 5(d)所示的组合服务执行频率最高的变迁路径是$t_1t_2t_6t_4t_7t_5$,其中,t_6和t_7的可用性较差,则基本变迁序列ts=$t_6t_4t_7$即冗余路径.因为在本例中没有其他变迁与t_6的选择关系一致(即与t_3是选择关系),如果有,则需要加入到冗余路径中(如图 6(d)所示).



(a) Unsound process structure (a) 不合理的流程结构

(b) Process structure breaching the original function (b) 破坏原有功能的流程结构

(c) Deal with parallel structure in AvailEvo algorithm (c) AvailEvo 算法对并发结构的处理

(d) Deal with choice structure in AvailEvo algorithm (d) AvailEvo 算法对选择结构的处理

Fig.6 Example of constructing redundant path

图 6 构造冗余路径的示例

4 组合服务演化中运行实例在线迁移方法

在实施组合服务动态演化的过程中,如何处理当前组合服务的运行实例是一个挑战性问题.特别是一些金融、电信和电子商务等关键系统需要 7×24 小时长时间不间断执行,难以采用传统的事务回滚或补偿的方法,而将旧的运行实例在线迁移到新组合服务的实例上是一个可行的解决方案.本质上来说,组合服务运行实例在线迁移问题可以归结为根据演化前的实例状态和演化操作来求解演化后在新组合服务业务流程结构下的实例状态问题.本节提出了一种组合服务演化中运行实例在线迁移方法——LiveMig.该方法首先能够判断当前时刻是否允许某运行实例在线迁移;其次,在允许迁移的情况下能够精确地计算出在新组合服务业务流程结构下对应的状态.本文的工作与其他工作的一个不同之处在于,不是直接地通过工作流网来判断是否能够进行实例迁移以及确定新实例中状态的位置,而是首先将演化前后的组合服务业务流程结构转换为一个以状态为顶点、变迁为边的可达图,然后通过对该可达图的分析,通过变迁序列来判断当前状态是否可以迁移,并且能够准确计算出演化后新实例中的状态位置.为了保证迁移的正确性,这里首先给出一个有效迁移的定义:

定义 8(有效迁移(valid migration)). 设$WFN^O=(P^O,T^O,F^O,i^O,o^O,M_0^O)$与$WFN^N=(P^N,T^N,F^N,i^N,o^N,M_0^N)$是两个合理工作流网,且$M^O$是$WFN^O$中的一个实例经过变迁序列$seq^O$到达的可达状态,则把$WFN^O$从状态$M^O$到$WFN^N$下状态$M^N$的迁移是有效迁移当且仅当:

- M^N是WFN^N的一个可达状态.
- 存在一个从WFN^N中初始状态M_0^N到M^N的变迁序列seq^N,使得$[seq^O]\backslash(T^O\backslash T^N)=[seq^N]\backslash(T^N\backslash T^O)$.这里,$[seq]$表示变迁序列$seq$中所有变迁的集合,$T^O\backslash T^N$和$T^N\backslash T^O$分别表示演化过程中删除的和增加的变迁.
- 对于每一个WFN^N中从M^N到终结状态M_{end}^N的变迁序列seq^N,seq^N不包含seq^O中的任意变迁,即迁移后不会重复执行已经执行过的变迁.

定义 8 中的第 1 个条件保证了新实例中的对应状态是可达的和可终结的;第 2 个条件确保了旧实例中任何必需的变迁在迁移后都已经被执行;最后一个条件指出旧实例中已经被执行过的变迁在迁移后不会被重复执行.在实际的处理过程中,首先转换一个合理的工作流网到可达图*graph* $G=(V,E)$,图中的每一个顶点表示状态而边上的权重表示变迁[20].利用可达图可以准确地确定组合服务实例的状态,并且能够很容易地计算出一个运行实例已经执行过的变迁序列以及将可能执行的变迁序列.

定义 9(作流网的可达图(reachability graph)). 工作流网$WFN=(P,T,F,i,o,M_0)$的可达图是$G=(V,E)$,其中:

- 顶点集合 V 是 WFN 中所有状态的集合.
- 从顶点 M 到 M'的边上有权重 t,当且仅当 $M[t\rangle M'$.

下面基于合理工作流网的可达图给出 LiveMig 算法.该算法包含两步,首先确定当前状态的迁移是否有效,当迁移有效时能够计算出迁移后的准确状态,否则不允许迁移,继续按照旧实例执行一个变迁后再重新执行本算法(见 LiveMig 算法).

LiveMig Algorithm (Migrate an instance state from old process definition to the new one).

Input: $WFN^O=(P^O,T^O,F^O,i^O,o^O,M_0^O)$, M^O, $ts^{\text{completed}}$;

$WFN^N=(P^N,T^N,F^N,i^N,o^N,M_0^N)$.

Output: M^N.

```
1   begin
2   G^N:=WFN2Graph(WFN^N)
3   L:=TransitionPath(G^N)
4   L:=DELETE(L,choice_add)
5   L′:=∅
6   for each ts∈L
7   {   ts′:=ts−T^N\T^O
8       L′:=L′∪ts′}
9   ts^cmp:=ts^cmp−T^O\T^N
10  k:=|ts^cmp|
11  if ts^cmp∈k·L′
12  {   t:=LAST(ts^cmp)
13      SubSet′:=CONTAIN(L′,ts^cmp)
14      SubSet:=EXTEND(SubSet′)
15      ts^N:=SELECT(SubSet,t)
16      M^N:=POST(G^N,ts^N,t)
17      end     }
18  else
19  {   while M≠M_end do
20      {   SubSet′:=COMPATIBLE(k·L′,ts^cmp)
21          if SubSet′≠∅
22          {   SubSet:=EXTEND(SubSet′)
23              (ts^N,t):=SELECT(SubSet,ts^cmp)
24              M^N:=POST(G^N,ts^N,ts_k)
```

```
25                  end     }
26              (M,t^NEW):=NEXT(M)
27              ts^cmp:=ts^cmp+t^NEW
28              k:=k+1      }     }
29      end
```

LiveMig算法展示了具体运行实例在线迁移过程.该算法的输入是演化前的旧工作流网WFN^O、已经执行完成的变迁序列ts^{cmp}、当前的状态M^O以及演化后的新工作流网WFN^N;输出为WFN^N下的演化后的对应有效迁移状态M^N.该算法首先计算新工作流网WFN^N的可达图G^N,基于可达图的广度优先方法能够获得新工作流网的变迁路径集,即日志L(第 2 行、第 3 行).对L进行预处理,删除通过选择增加操作(choice_add)而新增的所有新变迁序列(第 4 行),这是由于选择增加操作不会影响状态迁移,而且该操作和结构调整一起使用将可能导致潜在的错误.然后,将L中在演化过程中新加入的变迁删除,得到更新后的变迁序列集L'(第 5 行~8 行);同时,将删除ts^{cmp}中在演化过程中实际被删除的变迁,得到新的已完成变迁序列及其长度k(第 9 行、第 10 行).当ts^{cmp}属于L'中每个变迁序列的前k个变迁组成的新变迁序列集合时,状态迁移是允许的,这说明演化过程没有影响流程实例状态迁移(第 11 行).然后,求出ts^{cmp}的最后一个变迁t以及L'中包含所有包含ts^{cmp}的变迁路径子集$SubSet'$,并且把演化过程中的新增变迁再加回到$SubSet'$中,得到新工作流网中的实际变迁路径子集$SubSet$.最后,比较$SubSet$中的所有变迁路径,其中,t的位置在最左面的一个变迁路径ts^N被求出,根据可达图G^N变迁序列ts^N中权重为t的边的直接后继状态即为迁移后的对应状态M^N,此时算法可以终结(第 12 行~17 行).另一种情况是ts^{cmp}不属于L'中每个变迁序列的前k个变迁组成的新变迁序列集合,这意味着演化过程中使用了流程结构调整操作,当前的状态M可能无法被有效地迁移(第 18 行).这里首先判断M是否为终结状态M_{end},如果不是,则进入一个循环体(第 19 行).在循环体中,L'中的变迁路径子集$SubSet'$被求出,$SubSet'$中每个变迁序列的前k个变迁的顺序经过一定调整,恰好可以等于ts^{cmp},称$SubSet'$以k为长度兼容ts^{cmp}(第 20 行).这时,如果$SubSet'$不为空,则说明可以进行有效迁移,这时把演化过程中的新增变迁再加回到$SubSet'$中,得到新工作流网中的实际变迁路径子集$SubSet$.因此,可以求出所有能够以数量k兼容ts^{cmp}的最短的变迁序列ts^N以及ts^N的最后一个变迁t,根据可达图G^N变迁序列ts^N中权重为t的边的直接后继状态即为迁移后的对应状态M^N,此时算法可以终结(第 21 行~25 行).如果$SubSet'$不为空,则表示M^O状态不能被有效迁移,此时应该按照旧实例继续执行一个变迁t^{NEW}后得到新的状态M^O,然后该循环被再次执行(第 26 行~28 行).另外,计算工作流网可达图算法的最坏时间复杂度是指数的,所以LiveMig算法的最坏时间复杂度也是指数的.

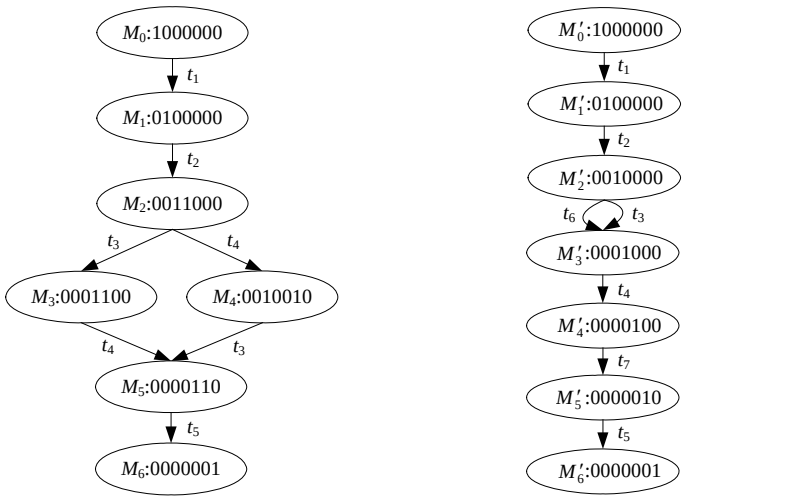
定理. LiveMig 算法是正确的,即它能够返回一个有效的迁移.

证明:限于篇幅,这里仅给出证明思路.首先,LiveMig算法显然是可终结的.假设WFN^O中已经执行了变迁序列seq^O到达旧状态M^O,并迁移到WFN^N中的新状态M^N.因为LiveMig算法基于工作流网的可达图,所以M^N必然是WFN^N中的一个可达状态.此外,LiveMig算法首先选择了一个变迁序列seq^N使得$[seq^O]\backslash(T^O\backslash T^N)=[seq^N]\backslash(T^N\backslash T^O)$,而且$M^N$是$WFN^N$中经过变迁序列$seq^N$的可达状态,因此定义 8 中的第 2 个条件满足.最后,由于本文中讨论的工作流网是无环结构,而且一个变迁在工作流网中最多出现 1 次,故每一个WFN^N中从M^N到终结状态M^N_{end}的变迁序列seq^N均不包含seq^O中的任何变迁. □

在此,使用LiveMig算法分析第 1 节中给出的供应链处理组合服务演化过程中的实例迁移问题,以说明该方法的可行性和正确性.可以看出,从原始的订单处理组合服务(图 5(a))经过选择增加(增加了一个新的付款服务t_6)、并发增加(增加了一个对货物的检查服务t_7)以及调整了并发结构(将原先的付款与配货的并发结构改为先付款再装配货物),最终得到了演化后的订单处理组合服务(图 5(d)).首先,得到原始的与演化后的两个组合服务的可达图(图 7(a)和图 7(b)),此处给出旧组合服务的可达图完全是为了方便说明问题,在实际操作中仅需要新的组合服务的可达图即可.

假设当前旧的组合服务实例已经执行了用户注册、提交订单和银行转账服务,即已完成的变迁序列为$t_1t_2t_3$,当前的状态为M_3.按照LiveMig算法对演化后的组合服务变迁路径集$\{t_1t_2t_3t_4t_7t_5, t_1t_2t_6t_4t_7t_5\}$进行预处理,删除新

增的变迁t_7以及包含选择增加操作引入的t_6所在的变迁序列$t_1t_2t_6t_4t_7t_5$,得到的变迁序列集为$\{t_1t_2t_3t_4t_5\}$.显然,$t_1t_2t_3$包含于$t_1t_2t_3t_4t_5$.因此,新的状态为t_3的直接后继状态 M_3',完全满足本文对合理迁移的定义.考虑另外一种情况,当完成的变迁序列为$t_1t_2t_4$,当前的状态为M_4时,$t_1t_2t_4$不包含或兼容于$t_1t_2t_3t_4t_5$,只能让旧实例继续执行一个有意义的变迁,得到一个能被$t_1t_2t_3t_4t_5$兼容的新变迁序列$t_1t_2t_4t_3$.因此,新的状态为t_4的直接后继状态 M_4',同样是一次合理迁移.这里进一步解释了LiveMig算法中关于删除包含选择增加操作引入变迁的变迁序列(见第 4 行)的实际意义,假设当完成的变迁序列为$t_1t_2t_4$,即状态为M_4时,如果不删除包含选择增加操作引入变迁的变迁序列,则变迁序列集为$\{t_1t_2t_3t_4t_5,t_1t_2t_4t_5\}$.容易看出,$t_1t_2t_4$包含于$t_1t_2t_4t_5$.因此,迁移后的新状态为 M_4',这会导致不能执行付款服务t_3或t_6,不符合合理迁移的定义.



(a) Reachability graph of original composite service order-processing
(a) 原始的订单处理组合服务可达图

(b) Reachability graph of evolved composite service order-processing
(b) 演化后的订单处理组合服务可达图

Fig.7 Reachability graph of composite service
图 7 组合服务的可达图

5 支持动态演化的组合服务引擎及实验分析

5.1 系统设计

基于上述研究成果,我们设计实现了一个支持动态演化的组合服务执行引擎 SOAREngine.该执行引擎是面向服务的软件生产线的重要组成部分.SOAREngine 主要与服务总线和监控管理工具进行交互,其体系结构如图 8 所示.

SOAREngine 完整地实现了上文提到的研究工作,合理性保持的演化操作集作为演化策略的基本元素,以配置文件的形式存储.SOAREngine 支持基于 HTN(hierarchical task network)规划的组件服务自动选择,当绑定了具体组件服务的组合服务经过流程解析进入实例运行池后即可被实际调用,运行时日志处理模块将记录每个组合服务实例执行的信息,其中,可信属性评估模块将定期地将组合服务的实际可信属性与系统要求的可信性约束进行比较,如果偏差超过一定的阈值,则通知演化策略生成模块来产生一个可行的演化策略.目前实现了面向可用性保障的组合服务演化方法——AvailEvo 算法,支持自动地计算出冗余路径;而实例迁移模块实现了 LiveMig 算法,负责处理当组合服务业务流程结构演化后运行实例在线迁移的问题.

在SOAREngine中,组合服务业务流程采用BPMN(business process modeling notation)进行刻画.在先前的工作中,设计了一个BPMN可执行模型以及实现了一个基于该模型的执行引擎,这是SOAREngine系统实现工作的

基础.针对组合服务动态演化的需求,SOAREngine采用AOP关注点分离的思想设计和实现演化机制,将演化策略封装在方面(aspect)中,其中包括实施演化的切入点(pointcut,即演化的触发点)和实际的演化策略(advice,即一个基本演化操作的序列).在系统运行时,若切入点被触发则实施演化策略.类比AOP编程方法,演化动作分为Before,After和Around这3种类型,分别表示在切入点之前、之后和切入点本身进行演化操作,图9(a)显示了After类型的演化策略在一个组件服务后面插入了一个新的组件服务.另外,与传统的采用AOP思想进行业务流程演化的研究[13]不同,在Around模式下定义的切入点不局限于一个业务流程中的活动,可以是一个活动片段(即子流程),因此可以更好地支持业务流程调整操作(如图9(b)所示).

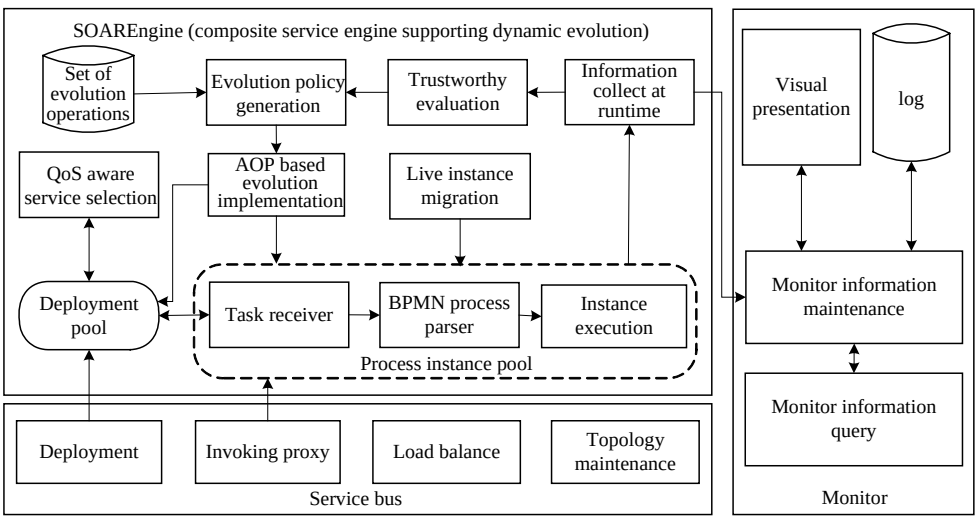
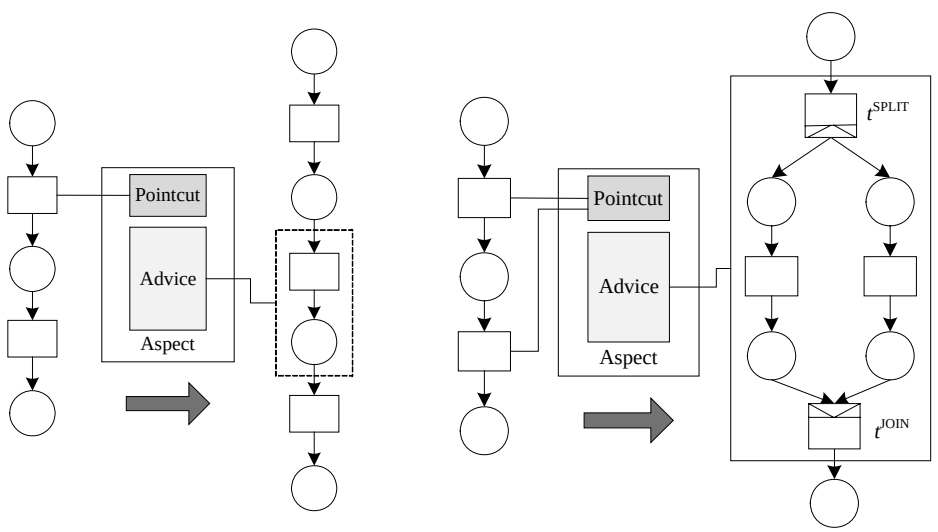


Fig.8 SOAREngine architecture

图 8 SOAREngine 的体系结构



(a) After evolution policy

(a) After 类型的演化策略

(b) Around evolution policy

(b) Around 类型的演化策略

Fig.9 AOP based evolution policy

图 9 基于 AOP 演化策略

5.2 实验分析

为了进一步评估本文提出的 AvailEvo 和 LiveMig 算法可行性和实际执行能力,我们进行了仿真实验.仿真程序使用 JDK1.5 和 Eclipse3.3 开发,运行在一台 CPU 为 Intel core 2,主频 1.86G,1G 内存的 PC 机上.实验 1 和实验 2 反映了 AvailEvo 算法的执行情况.实验 1 对比了在 QoS 感知服务组合研究中常用的 HTN 规划算法和本文提出的 AvailEvo 算法的执行性能,其中随机生成规模为 5~40 的组合服务业务流程,每个组件服务包含 1 000 个候选服务,候选服务的使用花费在 5~10 的区间上随机取值,可用性在 90%~98%的区间上随机取值,并且每组做 100 次取平均值.由图 10 可以看出,在不同的组合服务规模下,AvailEvo 算法的执行不会显著增加 HTN 算法的执行时间.实验 2 说明了应用 AvailEvo 算法后提高组合服务可用性的程度.表 1 给出了实验结果数据,从中可以看出,在不同的组合服务规模下,AvailEvo 算法均能有效提高其可用性,并且由于可用性的计算是乘积关系,因此随着组合服务规模的增加,其可用度呈衰减关系,而且规模越大,提高可用性花费的成本越高.

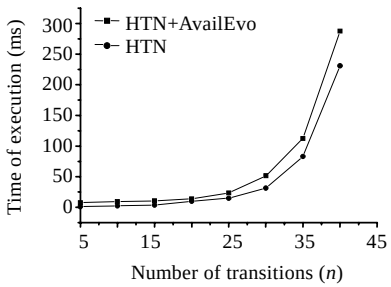


Fig.10 Comparison between the time costs of AvailEvo and HTN algorithm

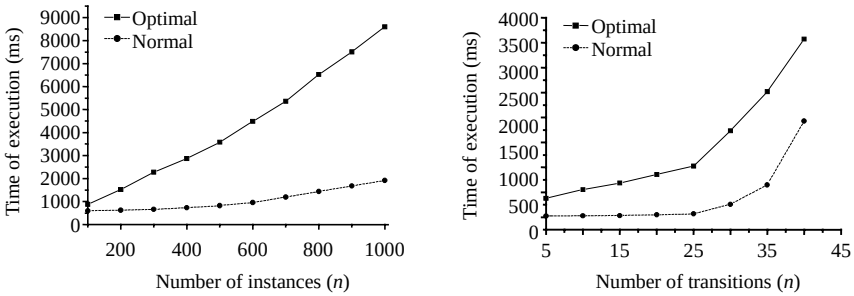
图 10 AvailEvo 算法与 HTN 算法的执行时间比较

Table 1 Effect of AvailEvo algorithm

表 1 AvailEvo 算法的效果

Services account (*n*)	HTN		HTN+AvailEvo	
	Cost (*n*)	Availability (%)	Cost (*n*)	Availability (%)
5	36.7	62.3	49.2	71.8
10	87	50.4	118	60.2
15	93	50.6	141.6	61.0
20	130.6	37.5	185.6	46.2
25	179.2	25.4	246.3	38.2
30	219.5	22.9	299.7	29.2
35	246.5	18.0	317.5	23.2
40	280.3	15.9	371.4	20.4

实验 3 和实验 4 验证了 LiveMig 算法的执行能力.其中,实验 3 说明了组合服务运行实例数量与迁移时间的关系(如图 11(a)所示).在该实验中,演化前的组合服务是一个随机生成的包括 20 个组件服务的合理的组合服务.基于本文提出的合理性保持的演化操作集,随机执行 10 个演化操作后得到演化后的组合服务流程结构,然后计算从 100 到 1 000 个运行实例状态迁移的时间,每组做 100 次,取平均值.实验 4 展示了组合服务规模与迁移时间的关系(如图 11(b)所示).在该实验中,演化前组合服务存在 1 000 个运行实例,同样随机执行 10 个演化操作后得到新的组合服务业务流程结构,然后计算演化前组合服务包含随机生成的 5~40 个组件服务时的全部实例迁移时间,每组做 100 次,取平均值.由于 LiveMig 算法是针对一个运行实例的迁移,当同时存在大量的实例进行迁移时,可以从两个方面进行适当的优化:一方面统一地求解新演化前组合服务业务流程的可达图;另一方面,可以对处于同一状态的多个实例只计算 1 次,优化后重新进行实验,得到图 11(a)和图 11(b)对应的虚线.可以看出,优化后迁移的性能有明显的提高.



(a) Comparison between time of execution and number of instances
(a) 执行时间与实例数量的对比

(b) Comparison between time of execution and number of transitions
(b) 执行时间与组合服务规模的对比

Fig.11 Performance of LiveMig algorithm
图 11 LiveMig 算法的性能

6 结 论

本文重点讨论了通过组合服务的动态演化机制保障网络化软件可信性的问题.首先,为了保障演化操作的可信性,提出了一个业务流程合理性保持的演化操作集.该集合中定义了流程结构的基本演化操作,并且证明了使用这些操作对确保演化后流程结构的合理性,从而避免了传统方法在演化后的复杂验证过程.其次,针对演化结果的可信性问题,特别是针对可用性这一可信属性,给出了一种面向可用性保障的组合服务演化方法.该方法通过分析组合服务业务流程结构,使用基本演化操作集构造冗余路径来保证业务流程中“关键区域”的可用性.再次,针对组合服务动态演化实施过程的可信性问题,设计了一种组合服务演化中运行实例在线迁移方法,能够判断运行实例是否能够运行时迁移,并且可以准确计算出迁移后的实例状态.最后,基于面向方面编程的思想,设计实现了一个支持动态演化的组合服务执行引擎以及相关的实验,证实了可信的组合服务动态演化方法的有效性.在下一步的工作中,将从组合服务自动化演化控制和演化后的流程模型管理等方面继续开展研究.

致谢 在此,我们向对全体参与“可信的国家软件资源共享与协同生产环境”课题研发工作的科研人员,尤其是北京航空航天大学课题组参与面向服务软件生产线子课题研发的师生表示感谢.

References:

[1] Chen HW, Wang J, Dong W. High confidence software engineering technologies. Chinese Journal of Electronics, 2003,31(12A): 1933−1938 (in Chinese with English abstract).

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

[3] von de Aalst WMP, Jablonski S. Dealing with workflow change: Identification of issues and solutions. Int’l Journal of Computer Systems Science & Engineering, 2000,15(5):267−276.

[4] Papazoglou MP. The challenges of service evolution. In: Bellahsene Z, Leonard M, eds. Proc. of the 20th Int’l Conf. on Advanced Information Systems Engineering. Berlin, Heidelberg: Springer-Verlag, 2008. 1−15.

[5] von de Aalst WMP, Basten T, Verbeek HMW, Verkoulen PAC, Voorhoeve M. Adaptive workflow—On the interplay between flexibility and support. In: Filipe J, Cordeiro J, eds. Proc. of the 1st Int’l Conf. on Enterprise Information Systems. 1999. 353−360.

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

[7] Andrikopoulos V, Benbernou S, Papazoglou MP. Managing the evolution of service specifications. In: Bellahsene Z, Leonard M, eds. Proc. of the 20th Int’l Conf. on Advanced Information Systems Engineering. Berlin, Heidelberg: Springer-Verlag, 2008. 359−374.

[8] Ryu SH, Casati F, Skogsrud H, Benatallah B, Saint-Paul R. Supporting the dynamic evolution of Web service protocols in service-oriented architectures. ACM Trans. on the Web, 2008,2(2):1−46.

[9] von de Aalst WMP, von Hee K. Workflow Management Models, Methods, and Systems. Cambridge: The MIT Press, 2002.

[10] Cheng A, Esparza J, Palsberg J. Complexity results for 1-safe nets. Theoretical Computer Science, 1995,147(1-2):117−136.

[11] Cardoso J, Sheth A, Miller J, Arnold J, Kochut K. Quality of service for workflows and Web service processes. Journal of Web Semantics, 2004,1(3):281−308.

[12] Zeng LZ, Benatallah B, Ngu AHH, Dumas M, Kalagnanam J, Chang H. QoS-Aware middleware for Web services composition. IEEE Trans. on Software Engineering, 2004,30(5):311−327.

[13] Charfi A, Mezini M. AO4BPEL: An aspect-oriented extension to BPEL. World Wide Web Journal, 2007,10(3):309−344.

[14] Canfora G, Penta MD, Esposito R, Villani ML. A framework for QoS-aware binding and re-binding of composite Web services. The Journal of Systems and Software, 2008,81(10):1754−1769.

[15] Ellis C, Keddara K, Rozenberg G. Dynamic change within workflow systems. In: Comstock N, Ellis C, eds. Proc. of the ACM Conf. on Organizational Computing Systems (ACM SIGOIS). New York: ACM Press, 1995. 10−21.

[16] von de Aalst WMP. Exterminating the dynamic change bug: A concrete approach to support workflow change. Information Systems Frontiers, 2001,3(3):297−317.

[17] von de Aalst WMP, Basten T. Inheritance of workflows: An approach to tackling problems related to change. Theoretical Computer Science, 2002,270(1-2):125−203.

[18] Sun P, Jiang CJ, Li XM. Workflow process analysis responding to structural changes. Journal of System Simulation, 2008,20(7): 1856−1863 (in Chinese with English abstract).

[19] von de Aalst WMP. Workflow verification: Finding control-flow errors using Petri-net-based techniques. In: van der Aalst W, Desel J, Oberweis A, eds. Proc. of the Business Process Management: Models, Techniques, and Empirical Studies. Berlin: Springer-Verlag, 2000. 161−183.

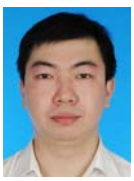
[20] Ye XM, Zhou JT, Song XY. On reachability graphs of Petri nets. Computers & Electrical Engineering, 2003,29(2):263−272.

附中文参考文献:

[1] 陈火旺,王戟,董威.高可信软件工程技术.电子学报,2003,31(12A):1933−1938.

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

[18] 孙萍,蒋昌俊,李湘梅.工作流过程的结构变化分析.系统仿真学报,2008,20(7):1856−1863.



曾晋(1981－),男,辽宁沈阳人,博士生,主要研究领域为服务计算,业务流程管理.

孙海龙(1979－),男,博士,讲师,CCF 会员,主要研究领域为服务计算,网络计算.

刘旭东(1965－),男,博士,教授,博士生导师,主要研究领域为可信网络计算技术,中间件技术.

邓婷(1977－),女,博士生,主要研究领域为服务计算,可信软件.

怀进鹏(1962－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为计算机软件与理论,网络计算技术,信息安全.