ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.21, No.6, June 2010, pp.1181−1195
doi: 10.3724/SP.J.1001.2010.03666

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

一种需求驱动的自主 Web 服务聚集方法*

叶荣华[1,2+], 金 芝[2,3], 王璞巍[4], 郑丽伟[2], 杨夏芬[1]

[1](浙江师范大学 计算机科学与技术研究所,浙江 金华 321004)

[2](中国科学院 数学与系统科学研究院,北京 100190)

[3](中国科学院 计算技术研究所,北京 100190)

[4](中国人民大学 数据工程与知识工程教育部重点实验室,北京 100872)

Approach for Autonomous Web Service Aggregation Driven by Requirement

YE Rong-Hua[1,2+], JIN Zhi[2,3], WANG Pu-Wei[4], ZHENG Li-Wei[2], YANG Xia-Fen[1]

[1](Institute of Computer Science and Technology, Zhejiang Normal University, Jinhua 321004, China)

[2](Academy of Mathematics and Systems Science, The Chinese Academy of Sciences, Beijing 100190, China)

[3](Institute of Computing Technology, The Chinese Academy of Sciences, Beijing 100190, China)

[4](Key Laboratory of Data Engineering and Knowledge Engineering of the Ministry of Education, Renmin University of China, Beijing 100872, China)

+ Corresponding author: E-mail: rhye@zjnu.edu.cn, http://www.zjnu.cn

Ye RH, Jin Z, Wang PW, Zheng LW, Yang XF. Approach for autonomous Web service aggregation driven by requirement. *Journal of Software*, 2010,21(6):1181−1195. http://www.jos.org.cn/1000-9825/3666.htm

Abstract: A new concept, Autonomous Web Service (AWS), to search requirement autonomously, is introduced in this paper. An intention-behavior-achievement mechanism based on environment ontology is proposed to specify the service request and the capability of AWS. A model of AWS aggregation driven by requirement has been figured out. The matching algorithm between the requirements and the AWS capability and the AWSs aggregating algorithm have been designed based on the intention-behavior-achievement mechanism. Finally, a case study is given to show the feasibility of this method.

Key words: AWS (autonomous Web service); requirement driven; aggregation; intention; behavior; service composition

摘 要: 引入具有主动搜索需求能力的自主 Web 服务概念,提出基于环境本体的意图-行为-实现机制,用于描述服务请求和自主 Web 服务能力,构型了一种需求驱动的自主 Web 服务聚集的模型.给出了基于意图-行为-实现机制的需求能力匹配算法和自主 Web 服务聚集算法.最后,通过对应用案例的研究来展示该方法的可行性.

关键词: 自主 Web 服务;需求驱动;聚集;意图;行为;服务组合

中图法分类号: TP311 文献标识码: A

* Supported by the National Natural Science Foundation of China under Grant Nos.90818026, 60625204 (国家自然科学基金); the Zhejiang Provincial Natural Science Foundation of China under Grant No.Y107625 (浙江省自然科学基金); the Renmin University of China Science Foundation under Grant No.22382075 (中国人民大学科学研究基金)

Received 2008-12-25; Accepted 2009-06-01; Published online 2009-11-20

大量存在的Web服务依托Internet及其上的相关协议形成了一个分布计算框架.在这个框架中,信息和服务能够在需要时以一种计算机可以处理的方式被提供[1],这种框架称为面向服务的体系结构(service-oriented architecture,简称SOA).SOA由3种基本的角色组成[2]:(i) 服务的提供者,是提供服务的主体;(ii) 服务目录,提供服务的查找功能;(iii) 服务的请求者,是为了完成某一目标,查找和调用服务的主体.在SOA中,Web服务由服务提供者提供,并注册在服务目录中,供服务请求者查找和调用的实体.为了实现这一机制,许多标准已被制订,如WSDL(Web services definition language)用于描述Web服务的接口,UDDI(universal discovery description and integration)提供了一种有效的方式来发现Web服务,而SOAP(simple object access protocol)则定义了Web服务的消息通信机制.

SOA使得Web从只能传输供人阅读的网页发展到可以实现各种异构信息源的有效集成,但是这种集成还主要停留在语法层面上.也就是说,目前对于Web服务的使用还是要借助于人工方式.为了实现Web服务发现、组合和调用的自动或半自动化,部分研究者提出了语义Web服务的概念,其中最重要的成果是OWL-S(ontology Web language for services)[3]和WSMO(Web service modeling ontology)[4],它们通过在现有体系中为Web服务添加语义注释静态地来描述其能力.

然而,无论是在传统的SOA还是语义Web服务中,服务都被视为是一个被动的实体,它被动地等待服务请求者来发现和调用.这种架构有一些明显的不足:(1) 随着Web服务数量的不断增加,会导致服务目录负载过大,从而使得服务查询的效率下降;(2) Web服务被使用的机率低,就像在网页搜索引擎中往往不会去点击排序在后面的搜索结果一样,可能会出现大量Web服务永远不会被真正使用的情况;(3) 不能很好地发挥服务提供者的主动性.如果Web服务不再是被动的计算实体,而是主动的服务实体,那么SOA将表现出怎样的特征呢?这是面向服务计算的一个新的研究方向.文献[5]曾经提出服务实体向服务需求聚合的磁石效应,并通过自动机制设计建立服务实体联盟.

围绕需求驱动的自主服务聚合中的研究问题,本文在环境本体[6]的基础上提出了一种意图-行为-实现机制,将服务建模为具有实现某些意图行为的自主计算实体.当发现服务需求后,这些自主计算实体主动申请实现(部分)需求;服务需求判断聚集的服务实体是否可以完全实现需求,从而形成可满足服务请求的服务实体组合.本文以此为基础建立了一个服务聚集模型,并给出了该服务聚集模型下的需求驱动的自主Web服务聚集算法.

本文第1节概述部分相关工作.第2节在定义面向需求与服务匹配的意图-行为-实现机制的基础上,给出需求和服务能力匹配方法和服务聚集模型.第3节详细论述基于服务聚集模型的服务聚集关键算法.第4节通过一个案例说明模型与算法的有效性.第5节是结论和进一步的研究目标.

1 相关工作

从效果上看,本文提出的需求驱动的自主Web服务聚集其实是一种Web服务的组合方法.由于现实中的应用一般都非常复杂,为了分散和简化应用逻辑、提高服务可重用性,单个Web服务都不可能做得非常复杂.因此,现实中复杂服务的应用需要组合多个简单的Web服务[7].目前,服务组合研究主要包括:(1) 基于工作流模型的方法.在此类方法中,工作流被用作分布活动的协调引擎或服务组合的建模定义的工具.其中,BPEL4WS(business process execution language for Web services)[8]是IBM、微软和EBA公司提出的一种基于工作流的Web服务组合语言,它通过一个流程将不同的Web服务组合起来,并且这个流程本身也可以作为一个Web服务发布.目前,通过BPEL4WS进行Web服务的组合基本上采用的是人工组合方式.eFlow[9]是HP实验室开发的一个用于规范、执行、管理服务组合的平台,它对服务流程的自适应性提供一定程度的动态支持;(2) 基于状态演算的方法.文献[10]是最早关注Petri网理论在服务组合中应用的工作之一,它使用Petri网描述服务以及服务的顺序、并行、选择、任意顺序、循环、鉴别器和引用等组合模式,提出了相应的代数系统描述递归的组合.文献[11]提出一种扩展的Petri网SRN(service/resource net)用于描述SIG(spatial information grid)中的组合服务;(3) 基于进程代数的方法.文献[12]通过建立Web服务描述与PI演算中进程描述的对应关系,实现Web服务组合的形式化描述;(4) 基于语义的方法.OWL-S[3]是一种描述Web服务的上层本体,其目的是通过对Web服

务进行语义描述实现机器对服务的无歧义理解,从而能够完成对 Web 服务的自动发现、调用和组合.

文献[6,13]提出了一种构建基于环境本体的 Web 服务能力规格说明的系统方法.在环境本体中,与应用相关的每一个环境实体(也称为环境资源)被建模为一个类树层次状态机(tree-like hierarchical state machines,简称 THSM),并且还为这些 THSM 建立依赖关系,这种依赖关系表明,两种环境实体可能存在要求某个 Web 服务作为桥梁的消息交换.最终,将 Web 服务的能力建模为对环境实体产生的影响.该方法本质上仍是一种基于本体的方法,为 Web 服务的自动发现提供了一种有效的解决方案.

从服务主动搜索需求的角度来研究 Web 服务的组合目前报道得还比较少,文献[5]提出的需求驱动的主动网构实体聚合是一种基于功能本体和自动机制设计 AMD(automated mechanism design)的服务 Agent 协作机制,其实质是利用功能本体实现服务 Agent 向需求聚合的“磁石效应”,并通过自动机制设计为需求的不同产出给出服务 Agent 类型组合方案,服务 Agent 在此基础上协作产生最终的需求解决方案.由于引入了服务 Agent,所以服务可以主动地搜索需求.但容易发现,该方法更像是一种针对特定目标的多 Agent 协作机制,但多 Agent 的协作过程是一种不可控过程,不能确保一定产生有效的服务组合.

不难发现,一般人类的社会活动既有需求者通过搜索发现自己需要的服务,并要求其提供服务,也有提供者主动查找自己能提供服务的需求,并主动推销服务.两者相互补充,构成了一个完整的服务与需求的发现机制.以此类推,在 SOA 中也不应该仅仅只有服务请求者通过服务目录查询服务这样一种服务发现方式,服务提供者也应该可以主动地发现他们可以满足的服务请求.因此,本文提出的自主 Web 服务就是以服务主动搜索需求的方式来最终实现服务的发现,是对现有 SOA 中服务发现机制不足的有益补充.其好处是:首先,由于大量需求由服务主动发现而最终实现,所以可以减轻现有服务目录的查询压力;其次,由于受服务利益的驱动,会尽可能地发现需求,从而可以提高服务本身的被利用率;最后,自主服务会在发现需求的过程中充分考虑自身的利益.例如,当发现自己能够为多个需求提供服务时可以选择对自己更为有利的一个,从而使自己的利益最大化.以上这些都是传统 SOA 所不具有的.

2 需求驱动的服务聚集架构

2.1 问题描述

设想一种环境,在这个环境中,Web 服务不再是被动地等待服务请求者发现和调用的计算实体,而是能够在环境中主动地搜索已发布需求的服务实体.当服务实体发现能够提供服务的需求时,就向该需求申请提供服务,一旦申请的服务实体已经能够满足需求,就可以协作完成需求.为此,我们构建了一个需求驱动的服务聚集平台 APDR4AWS(the aggregation platform driven by requirement for autonomous Web service),如图 1 所示.

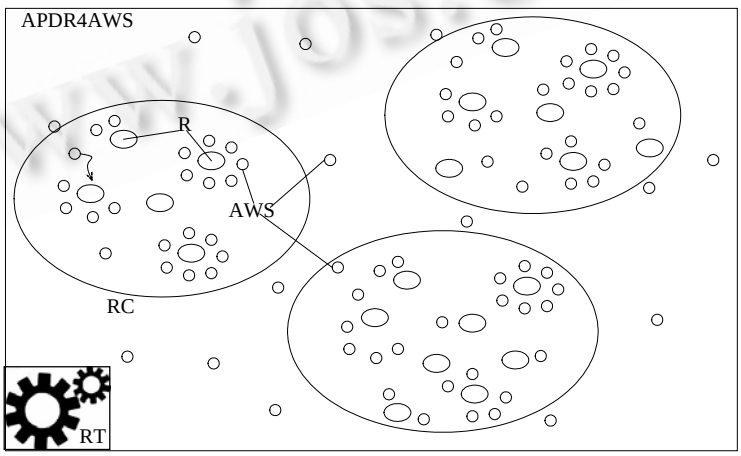


Fig.1 Basic framework of APDR4AWS

图 1 APDR4AWS 的基本框架

这个平台主要由 3 部分组成:(1) 一组用于发布需求的需求容器(记为 RC),一个需求容器可以发布多个需求(也称为服务请求),但所有这些服务请求在一种分类机制下是同类型的,不同的需求容器用于发布不同类型的需求;(2) 一群能够主动搜索需求的自主 Web 服务(记为 AWS),每个 AWS 都是自主的服务实体,在搜索分析对需求可满足性时,它可以表现为一个主体(agent)的特性,而在最终被调用执行时,就是一个普通的 Web 服务;(3) 一个分类机制(记为 RT),为了提高 AWS 搜索服务请求的效率,根据这种分类机制,需求被分类到各个需求容器中,AWS 也可以根据 RT 找到相应的需求容器,这样可以大大减少搜索需求的范围.

为了实现 AWS 向需求的聚集,必须要用一种统一的方法来描述需求和 AWS 能力.本文以环境本体作为统一的语义基础,并通过引入意图(intention)-行为(behavior)-实现(achievement)机制来分别描述需求和 AWS 能力以及能力是否能够满足需求等信息,目标是找到一种 AWS 能力与需求的匹配方法,并最终实现一种需求驱动的 AWS 聚集方法.

2.2 意图-行为-实现机制

面向需求与服务匹配的意图-行为-实现机制是建立在环境本体基础上的.在人工智能领域,最早给出本体(ontology)定义的是 Neches 等人,他们将本体定义为"给出构成相关领域词汇的基本术语和关系,以及利用这些术语和关系构成的规定这些词汇外延的规则的定义"[14].但目前大部分的领域本体都缺少对概念所对应实体的状态及状态间转变的描述,从而无法表达领域实体在时间序列上的变化情况.文献[6]提出了构建环境本体作为 Web 服务的语义指称,通过引进类树层次状态机来描述环境资源的生命周期,以描述领域实体的动态特性.

定义 **1**. 环境本体(environment ontology,简称 EO):定义为一个六元组$\{Rsc,\mathfrak{R},rel,HSM,Exch,res\}$,其中:

- Rsc 是环境资源的有限集;
- $\mathfrak{R}$是一个关系的集合;
- $rel:\mathfrak{R}\to Rsc\times Rsc$ 是环境资源间的一个关系函数,设 $c_1,c_2\in Rsc,r\in\mathfrak{R}$,则 $rel(r)=(c_1,c_2)$表示 c_1,c_2 具有关系 r;
- HSM 是类树层次状态机(记为 THSM)的有限集合;
- $Exch\subseteq HSM\times HSM$ 是 THSM 间的一种消息交换关系;
- $res:Rsc\leftrightarrow HSM$ 是环境资源与 THSM 之间的一一对应关系.

EO 是一个可以共享的本体,它本身并不依赖于某个具体的 Web 服务,是对某个应用领域相关资源操作语义的一种形式化表示.文献[6]在 EO 的基础上通过定义 Web 服务对资源的影响来规格化 Web 服务的能力描述.本文在上述环境本体的基础上,构建面向 AWS 和需求的意图-行为-实现模型.

定义 **2**. 意图:定义为一个六元组(r,s_0,I,S_m,s_t,O),其中:

- $r\in Rsc$ 是一个环境资源,Rsc 是环境本体中的资源集;
- s_0 是 r 的初始状态;
- $I=I_{bas}\cup I_{wait}$ 是输入信息的有限集,表示为了实现意图可以提供的信息.其中:I_{bas} 为基本输入集,其每个输入值均直接给出;I_{wait} 为等待输入集,其每个输入值则要通过后续的消息交换得到;
- S_m 为 r 的中间状态集,表示意图实现过程中资源允许经过的状态;
- s_t 为 r 的目标状态,表示意图实现后资源所处的状态;
- O 为一个输出集,表示意图实现中必须得到的输出信息.

Intention 表示期望在给定输入集的情况下,环境资源能够通过某种状态转换关系从初始状态经中间状态集达到目标状态,并得到一个输出集.例如有这样一个意图(记为 I_{cc}):期望能够通过一张有效的信用卡购买机票.图 2 表示了环境本体中对资源 creditcard(信用卡)的描述,则意图 I_{cc} 可以表示为表 1.

为了便于服务实体的成功聚集,我们还假设 Intention 应满足以下两个条件:

(1) Intention 是原子的,即它不能被分解为更小的几个 Intention;

(2) 一个环境资源上可以作用有一个或多个 Intention.

定义 **3**. 行为:定义为一个七元组(r,s_0,I,S_m,s_t,O,T),其中:

- $r\in Rsc$ 是一个环境资源,Rsc 是环境本体中的资源集;

- s_0 是 r 的初始状态;
- I 输入信息的有限集;
- S_m 为行为作用下 r 的中间状态集;
- s_t 为行为作用下 r 的目标状态;
- O 为行为作用下 r 的输出集;
- $T\subseteq\Omega\times\Omega(\Omega=\{s_0\}\cup\{s_t\}\cup S_m)$为行为作用下 r 的状态转换关系.

Behavior 表示在环境本体的语义下作用于环境资源,并使其按某种目标变化的能力.例如,我们可以将“用信用卡支付”定义为一个行为(记为 B_{cc}),表 2 是 B_{cc} 内容的具体描述,图 3 是 B_{cc} 的类树层次状态机.这里,⇒表示超级状态向子状态集中的默认状态的转换,而→是普通的状态转换.由此我们不难看出,Behavior 是环境本体中资源 THSM 的一个子集.

Tabel 1 Content of I_{cc}

表 **1** I_{cc} 的内容

Item		Content
r		*creditcard*
s_0		*valid*
I	I_{bas}	ϕ
	I_{wait}	{*paidInfo*}
S_m		{*non-charged*}
s_t		*charged*
O		{*chargedInfo*}

Tabel 2 Content of B_{cc}

表 **2** B_{cc} 的内容

Item	Content
r	*creditcard*
s_0	*valid*
I	{*paidInfo*}
S_m	{*non-charged*}
s_t	*charged*
O	{*validInfo*,*non-chargedInfo*,*chargedInfo*}
T	{(*valid*⇒*non-charged*),(*non-charged*→*charged*)}

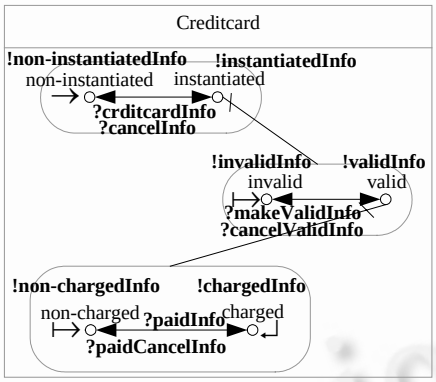


Fig.2 Domain THSM of resource creaditcart

图 2 环境资源 creaditcard 的领域 THSM

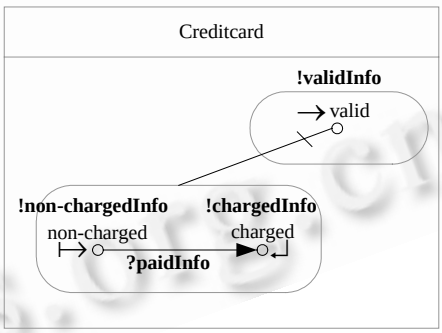


Fig.3 THSM of B_{cc}

图 3 B_{cc} 的 THSM 表示

如果某个 AWS 能够实施一个行为,我们就可以认为它具有了这样一种能力,由此我们可以很自然地定义出 AWS 的能力.与 Intention 类似,我们可以在一个环境资源上定义多个 Behavior.

定义 **4**. 实现(achievement):如果一个意图(记为 *Int*)和一个行为(记为 *Beh*)满足下列 3 个条件,则称行为 *Beh* 实现了意图 *Int*,记为 *Achieve*(*Beh*,*Int*):

- $r^{Int}=r^{Beh}$(上标 *Int* 和 *Beh* 分别表示该符号代表 Intention 和 Behavior 中的相应标记,下同);
- $s_0^{Int}=s_0^{Beh}, s_t^{Int}=s_t^{beh}, S_m^{Int}\supseteq S_m^{Beh}$;
- $I^{Int}\supseteq I^{Beh}, O^{Int}\subseteq O^{Beh}$.

容易看出,Achieve 表达了这样一种情形:某个 Behavior 是否能够完成一个 Intention 所定义的要求.当它们的对象是同一个环境资源,有相同的目标(将环境资源从某个初始状态通过一系列状态转换达到目标状态),并且中间状态集和输入输出集具有某种包含关系时,这种 Achieve 即可自然达到.显然,上面例子中的行为 B_{cc} 能够

实现意图 I_{cc},即 $Achieve(B_{cc},I_{cc})$.

2.3 服务需求的发现

服务实体首先要能够发现服务请求,为此,要先给出服务请求的描述.同时,每个服务实体还需要了解自己的能力.这样,服务实体才能根据自己的能力去寻找适合自己参与的服务请求.

定义 5. 需求(*Req*):定义为一个三元组(*T*,*S*,*W*),其中:

- $T=INTS\cup AOSJ$ 是一个转换的有限集,这里的转换分为两类:
  (1) $INTS=\{Int_i|0<i\leq n\}$是一组 Intention 的有限集;
  (2) $AOSJ=\{And\text{-}Split,And\text{-}Join,Or\text{-}Split,Or\text{-}Join\}$是控制流程分裂与合并的特殊转换;
- S 是一个表示位置的有限集;
- $W\subseteq(T\times S)\cup(S\times T)$表示 T 指向 S 或者 S 指向 T 的有向弧.

由定义可知,这里将需求建模为一组作用在领域资源上的意图,以及这些意图之间的控制逻辑关系.简单而言,需求就是期望按照一定的控制逻辑使一些环境资源从一些状态转换到另一些状态.当然,这个过程会给出必要的输入,也会得到一些输出,还有可能其中的一些输入要来自于另外一些行为的输出.图 4(a)~图 4(c)分别表示了需求中两个意图 I_1 与 I_2 之间的顺序、并发与选择关系.

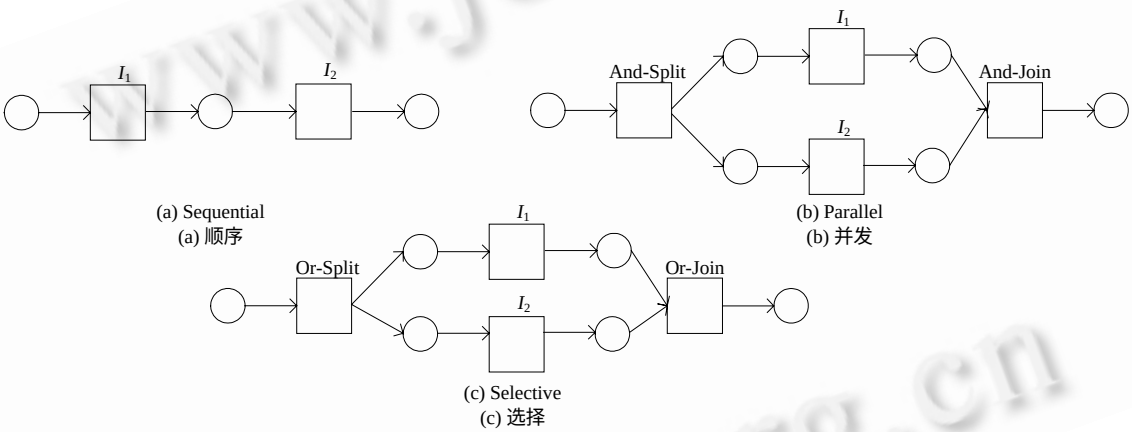


Fig.4 Control logic for intention in requirement

图 4 需求中意图的控制逻辑

定义 6. 后继(Succ):在一个需求中,对于 $I_1\in Req.INTS,I_2\in Req.INTS$,如果∃子路径 p,使得 $I_1\rightarrow p\rightarrow I_2$,则称 I_2 是 I_1 的后继,记为 $I_2=Succ(I_1)$.

后继表示实现两个意图的行为必须满足的先后关系.当要这两个行为间有消息交换,即一个行为的输出作为另一个行为的输入时,输入行为必须是输出行为的后继.因为只有先执行的行为才有确定的输出,才能为后执行的行为作为输入.

定义 7. AWS 能力(AWSC):定义为一个二元组(*BEHS*,*EXS*),其中:

- $BEHS=\{Beh_i|0<i\leq n\}$是一组 Behavior 的有限集合;
- $EXS\subseteq BEHS\times BEHS$ 是服务实体内两个行为之间的消息交换关系的有限集.

由定义可知,这里将 AWS 能力建模为一组行为的集合和一组行为间的消息交换关系.行为集 BEHS 保证了 AWS 有能够实施某些行为的能力,而有时为了完成这组行为,可能要求服务实体内部各行为之间进行必要的消息交换,消息交换关系集 *EXS* 记录了这些信息.

定义 8. AWS 能力与需求匹配:给定 AWS(其能力为 AWSC)和需求 *Req*,如果 $\forall Beh\in AWSC.BEHS$,$\exists Int\in Req.INTS$ 使得 $Achieve(Beh,Int)$,则称 AWS 能力与需求匹配.

这种匹配的本质是,AWS 有能力为需求提供服务,它可以分为两种情况:

(1) 对于$\forall Int \in Req.INTS$,都$\exists$一个 $Beh \in AWSC.BEHS$ 使得 $Achieve(Beh,Int)$,则称 AWS 的能力能够完全满足需求 Req.

(2) 如果至少$\exists$一个 $Int \in Req.INTS$,不$\exists Beh \in AWSC.BEHS$ 使得 $Achieve(Beh,Int)$,则称 AWS 的能力能够部分满足需求 Req.

由定义可以看出,一个 AWS 或者能够实现需求的全部意图,或者只能完成需求的部分意图.前者表示一个服务就可以完成需求,而后者说明需要更多的服务才能实现需求.由于前者是后者的一种特殊情况(即一个 AWS 向需求申请提供服务就完成了聚集),所以我们讨论的服务聚集一般指的是后者.

2.4 服务聚集框架

在上述概念的基础上,我们建立了一个需求驱动的 AWS 聚集框架,如图 5 所示.首先,在环境本体之上,用意图及其控制逻辑建模需求,用行为及其消息交换建模 AWS 的能力;其次,如果存在一个或多个 AWS 的能力能够完全满足需求(这里是指所有 AWS 的全部行为均能实现需求的所有意图),通过 AWS 行为间消息交换完成 AWS 向需求的聚集,形成能够满足需求的服务聚合体.

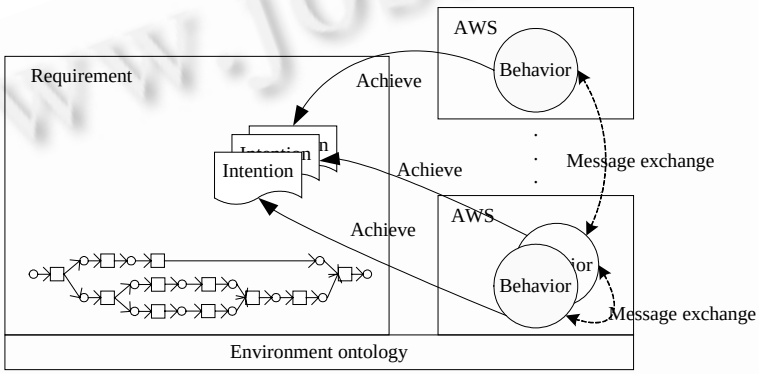


Fig.5 A framework of AWS aggregation driven by requirement

图 5 需求驱动的 AWS 聚集框架

不难发现,一次需求驱动的 AWS 聚集过程的成功结束,必须满足以下 3 个条件:

(1) 生成一个能够协作实现需求的 AWS 的有限集 AWSS;

(2) 对于$\forall Int \in Req.INTS$,$\exists$一个 $AWS \in AWSS$,$\exists Beh \in AWSC.BEHS$,使得 $Achieve(Beh,Int)$;

(3) 生成一个消息交换关系的有限集 $MEXS$,$MEXS=\Pi_1 \cup \Pi_2$ 分成两部分:

- $\Pi_1 = \bigcup_{AWS \in AWSS} AWSC.EXS$;
- $\Pi_2=\{(Beh',Beh)|$ 如果 $\exists Int \in Req.INTS, Int' \in Req.INTS$, 满足 $Int=Succ(Int'),Achieve(Beh,Int),Achieve(Beh',Int')$,且$\exists im \in Int.I_{wait},\exists om \in Beh'.O$,使得 $im=om$,同时$(Beh',Beh) \notin \Pi_1\}$.

上述Π_1表示所有 AWS 内部行为之间的消息交换关系,Π_2表示跨 AWS 的行为间消息交换关系.当一个意图 Int 有一个等待输入与一个行为 Beh'的输出相同时,可以为 Beh'与实现意图 Int 的行为 Beh 建立消息交换(Beh',Beh).

3 服务聚集关键算法

由上述需求驱动服务聚集框架可知,AWS 需要分析需求的可满足性,而需求需要了解当前聚集的服务是否已经可以通过协作完成自己的全部意图.所以,服务的聚集过程必须由 AWS 和需求协作完成.下面分别给出为完成聚集的服务端和需求端的关键算法.

3.1 AWS端算法

根据对需求和 AWS 能力的定义以及对于聚集过程的描述,我们可以给出如下 AWS 能力与需求匹配算法.

算法 **1**. 需求能力匹配 RCMatch.

输入:需求 *Req*,AWS 能力 *AWSC*=(*BEHS*,*EXS*);

输出:TRUE(匹配),FALSE(不匹配).

```
1. AllBehIsMatched=TRUE
2. Foreach 行为 Beh in AWSC.BEHS
3. {
4.     ThisBehIsMatch=FALSE
5.     Foreach 意图 Int in Req.INTS
6.         If Achieve(Beh,Int) Then
7.         {
8.             ThisBehIsMatch=TRUE
9.             Exit Foreach
10.        }
11.    AllBehIsMatched=AllBehIsMatched and ThisBehIsMatch
12. }
13. 返回 AllBehIsMatched
14. 结束
```

算法 1 描述了 AWS 能力与需求匹配的情况,当 AWS 能力中的所有行为都能实现需求中的一个意图时,算法返回 TRUE,这表示 AWS 能力 AWSC 能够满足全部或部分需求 *Req*;否则表示 AWS 不能为需求提供服务.为简单起见,这里不考虑 AWS 的部分能力与 *Req* 匹配的情况.

一般地,AWS 先根据一种分类机制发现一个需求容器 *RC*,然后在 *RC* 中搜索与自己的能力匹配的需求 *Req*,若找到,则向需求 *Req* 请求服务登记,并等待提供服务;否则,继续搜索其他容器.下面是 AWS 在给定 *RC* 中发现需求并请求提供服务的算法.

算法 **2**. 请求提供服务 AskToProvideService.

输入:AWS 能力 *AWSC*=(*BEHS*,*EXS*),需求容器 *RC*;

输出:TRUE(成功),FALSE(失败).

```
1. Foreach Req in RC
2.     If RCMatch(Req,AWSC) Then
3.     {
4.         向 Req 请求服务登记,并等待确认.
5.         If 在规定时间成功确认 Then 等待解散消息
6.         Else Continue
7.         If 在规定时间没有收到解散消息 Then 返回 TRUE,结束
8.         Else 等待一随机时间,转第 3 步
9.     }
10. 返回 FALSE,结束
```

当一个 AWS 在一个需求容器 *RC* 中找到了一个匹配的需求 *Req* 时,AWS 向 *Req* 请求登记,若 *Req* 确认其登记,并且没有在一个合理的时间内要求其解散,则单个 AWS 向需求的聚集成功.为了防止 AWS 资源死锁,需求如果在一个合理的时间内得不到所有的服务,则应释放已经聚集的 AWS 资源,即通知 AWS 解散.

3.2 需求端算法

为了能够实现部分匹配 AWS 的正确聚集,除了 AWS 要能发现需求并提交提供服务的请求以外,还要保证如下 3 点:

(1) AWS 中的一个行为 *Beh* 如果能够实现 *Req* 中的一个意图 *Int*,则 *Beh* 的输出必须能为 *Int* 后继的输入提供足够的信息;

(2) AWS 中内部具有消息交换关系的两个行为 Beh_1 和 Beh_2,如果实现需求中的两个意图 Int_1 和 Int_2,则必须保证 $Int_2=Succ(Int_1)$;

(3) *Req* 中已经可以被一个 AWS 的 Behavior 实现的 Intention 不能再被另一个 AWS 的 Behavior 实现.

为此,当需求 *Req* 收到服务登记请求时要作相应处理,算法 3 可以实现这一思想.

算法 3. 处理服务登记 DealWithReg.

输入:需求 *Req*,AWS 能力 *AWSC*=(*BEHS*,*EXS*),意图集Λ,类消息交换关系集*K*;

输出:Λ(意图集),*K*(类消息交换关系集),TRUE/FALSE(登记成功/失败).

前提:*RCMatch*(*Req*,*AWSC*)返回值是 TRUE.

/*Λ为 *R* 中所有 Intention 的集合,I是Λ中所有 Intention 的 I_{wait} 并集*/

/*下面两行用于排除需求中某些意图被重复实现的情况,以及不能为后继意图提供足够输入信息的 AWS*/

```
1. Foreach 行为 Beh in AWSC.BEHS
2. {
3.    Flag=FALSE
4.    Foreach 意图 Int in Λ
5.        If Achieve(Beh,Int) Then Flag=TRUE
6.    If Not Flag Then 返回 FALSE,结束
7. }
8. Foreach 消息交换关系 ex in AWSC.EXS
9. {
10.    记 ex=(Beh1,Beh2),Int1,Int2∈Req.INTS 满足 Achieve(Beh1,Int1),Achieve(Beh2,Int2)
11.    If Int2≠Succ(Int1) Then 返回 FALSE,结束
12. }
/*下面一步用于继承 AWS 内部行为间的消息交换关系*/
13. K=K∪AWSC.EXS
14. Foreach 行为 Beh in AWSC.BEHS
15. {
16.    记 Int∈Req.INTS 且 Achieve(Int,Beh)
17.    Foreach 类消息交换关系 k in K
          /*用 Beh 替换K的所有类消息交换关系中被它实现的 Intention.*/
18.       If k=f(Int) Then k=f(Beh)
19.    Λ=Λ−{Int}
20.    /*建立新的消息交换关系
21.    Foreach Int′ in {I|I=Succ(Int)}
22.    {
23.       If Int.O⊆Int′.Iwait Then
24.       {
```

25. **If** $Int' \in \Lambda$ and $Int.O \subseteq Int'.I_{wait}$ **Then**
26. 执行$K=K\cup\{\langle Beh,Int'\rangle\}$
27. **Else**
28. 执行$K=K\cup\{\langle Beh,Beh'\rangle|Achieve(Beh',Int)\}$
29. }
30. $\Gamma=\Gamma-Int.O$
31. }
32. }
33. 返回 TRUE,结束

算法 3 通过引入 Intention 集Λ,并当一个 Intention 被实现就从Λ中去掉它,保证了需求 *Req* 中的 Intention 不会被重复实现.K集记录了上述输入输出的单向类消息交换关系,之所以称为类消息交换关系,是因为在算法执行的中间阶段有可能出现行为与意图之间的消息交换,但当算法执行完成后,这些意图会被实现它的行为替换,从而使这种类消息交换关系成为真正的消息交换关系.在算法 3 基础上,下面给出需求驱动的 AWS 聚集算法.

算法 **4**. 需求驱动的 AWS 聚集 AWSAggregation.

输入:需求 *Req*;

输出:AWSS(一个能够完成需求 *Req* 的 AWS 集),K(类消息交换关系集).

1. 初始化$\Lambda=\{Int|Int\in Req\}$, $\Gamma=\bigcup_{Int\in\Lambda} Int.I_{wait}$,$K=\emptyset$,$AWSS=\emptyset$
2. 等待 AWS 服务登记一个时间段 T_a
3. **If** 在 T_a 时间内没有服务登记 **Then** 向 AWSS 中的每一个 AWS 发送解散消息,转到第 1 步
4. **Else** /*有一个 AWS 请求服务登记*/
5. {
6. **If** Not $DealWithReg(Req,AWSC,\Lambda,\Gamma,K)$ **Then** 转到第 2 步
7. $AWSS=AWSS\cup\{AWS\}$
8. 向 AWS 发送确认登记消息
9. **If** $\Lambda=\emptyset,\Gamma=\emptyset,AWSS\neq\emptyset$ **Then** 转第 12 步
10. **Else If** $\Lambda=\emptyset,\Gamma\neq\emptyset,AWSS\neq\emptyset$ **Then** 向 AWSS 中的每一个 AWS 发送解散消息,并转到第 1 步
11. **Else** 转到第 2 步

}

12. 成功聚集,返回 AWSS 和K,结束

算法 4 表明,只有当需求中所有的意图被实现,并且意图中所有等待输入都有相应行为的输出为之提供信息时,聚集过程才成功完成.另外,在一个需求容器 *RC* 中,如果需求 R_1 和 R_2 都需要 AWS_1 和 AWS_2 同时服务才能完成聚集,但某一时刻 AWS_1 已经登记到 R_1 中,而 AWS_2 已经登记到 R_2 中,这样,R_1 和 R_2 都无法完成聚集,这就是死锁.算法 4 通过判断在一个合理的时间段 T_a 内是否有新登记的 AWS 来保证不会出现死锁.

下面我们来证明以上算法能够满足第 2.4 节的服务聚集成功完成所必要的 3 个条件.

证明:根据算法 4 第 1 步的初始化$\Lambda=\{Int|Int\in Req\}$可知,意图集$\Lambda$最初由需求的所有意图组成,所以必然为非空.根据算法 4 第 6 步、第 7 步,如果只要有一个 AWS 的行为能够实现部分还没有被实现的意图,则这个 AWS 就会被加到 AWSC 中.又由算法 4 第 9 步、第 10 步可知,算法成功结束必须是$\Lambda\neq\emptyset$成立,所以返回的 AWSS 必然非空,即满足条件(1).

根据算法 4 第 1 步的初始化$\Lambda=\{Int|Int\in Req\}$可知,意图集$\Lambda$最初由需求的所有意图组成,再根据算法 3 的第 19 步,若Λ中的一个意图 *Int* 被一个行为 *Beh* 实现,则$\Lambda=\Lambda-\{Int\}$,由算法 4 的第 9~12 步可知,算法成功结束条件$\Lambda=\emptyset$成立,即对于$\forall Int\in Req.INTS$,$\exists$一个 $AWS\in AWSS$,$\exists Beh\in AWSC.BEHS$,使得 $Achieve(Beh,Int)$.由此可见,算法满足条件(2).

算法 3 的第 13 步$K=K\cup AWSC.EXS$ 表明,AWS 内部行为间的消息交换关系依然被继承到聚集服务的消息交换关系集 *MEXS* 中,即算法能够确保其中包含Π_1;另外,算法 3 的第 21~31 步保证了当两个行为实现的意图具有后继关系,并且当前意图的输出集包含于后一个意图的输入集时,为这两个行为建立消息交换关系或为一个行为与意图建立类消息交换关系.而算法 3 的第 8~12 步则保证了该类消息交换关系最终能够替换成真正的消息交换关系,即形成 AWS 间的消息交换关系Π_2.

综上,算法满足条件.证毕. □

4 应用案例研究

本节用一个应用案例来说明基于意图-行为-实现机制的 AWS 聚集的思想和最终效果.本案例扩充了文献[15]中的“旅行安排”的例子,并沿用该文献定义的环境本体,对一个确定的需求展示 AWS 聚集及算法的执行过程.最后给出聚集后的 AWS 组成的带消息通信的 THSM 集合,即服务实体的聚合体,它表示 AWS 之间如何协作完成需求的细节.

4.1 需求的表示

有一个旅行安排需求(记为 *Req_Travel_Arrange*),要求能为客户提供旅行中的机票(或火车票)代购及旅馆预订服务,并能够使用信用卡进行支付.根据相应的环境本体及需求的定义,我们提取了 4 个环境资源(见表 3).

Table 3 Environment resource of requirement *Req_Travel_Arrange*

表 **3** 需求 *Req_Travel_Arrange* 的环境资源

Environment resource	Description
planeticket	Planeticket
trainticket	Trainticket
hotelroom	Hotelroom
creditcard	Creditcard

表 4 给出了需求 *Req_Travel_Arrange* 中的 5 个意图(I_1~I_5).I_1 表示需求期望机票(或火车票)代购及房间预订可以用一张有效的信用卡完成支付功能,状态由 *valid* 通过{*non-charged*}转换为 *charged*,3 个等待输入 *hotelroom-paidInfo*(房间预订支付信息)、*planeticket-paidInfo*(机票代购支付信息)和 *trainticket-paidInfo*(火车票代购支付信息)必须由第三方提供,并有输出{*validInfo*,*non-chargedInfo*,*chargedInfo*};I_2 表示需求期望能够完成对空房间的预订功能;I_3,I_5 分别表示需求期望能够完成机票和火车票的代购功能;I_4 表示需求期望能够完成对已经卖出机票(或火车票)的投递功能.

Table 4 Requirement *Req_Travel_Arrange*

表 **4** 需求 *Req_Travel_Arrange* 的表示

Item	Intention				
	I_1	I_2	I_3	I_4	I_5
r	*creditcard*	*hotelroom*	*planeticket*	*ticket*	*trainticket*
s_0	*valid*	*vacancy*	*available*	*sold*	*available*
s_t	*charged*	*paid*	*sold*	*delivered*	*sold*
S_m	{*non-charged*}	{*cancelled*, *ordered*}	{*cancelled*, *ordered*}	{*non-delivered*}	{*cancelled*, *ordered*}
I_{bas}	∅	{*reVacancyInfo*, *orderInfo*, *orderedInfo*, *orderCancelInfo*}	{*reAvailableInfo*, *orderInfo*, *orderedInfo*, *orderCancelInfo*}	{*deliveryInfo*}	{*reAvailableInfo*, *orderInfo*, *orderedInfo*, *orderCancelInfo*}
I_{wait}	{*hotelroom-paidInfo*, *planeticket-paidInfo*, *trainticket-paidInfo*}	∅	∅	∅	∅
O	{*validInfo*, *non-chargedInfo*, *chargedInfo*}	{*cncelledInfo*, *vacancyInfo*, *paidInfo*}	{*cncelledInfo*, *availableInfo*, *paidInfo*}	{*non-deliveredInfo*, *deliveredInfo*}	{*cncelledInfo*, *availableInfo*, *orderedInfo*, *paidInfo*}

图 6 是需求中 5 个意图间的控制逻辑图,它表示预订房间意图(I_2)和机票或火车票代购意图(I_3,I_5)可以并发执行,信用卡支付意图(I_1)必须在预订房间与代购机票或火车票之后,机票和火车票的代购意图可以选择执行,而投递意图(I_4)必须在票代购之后.

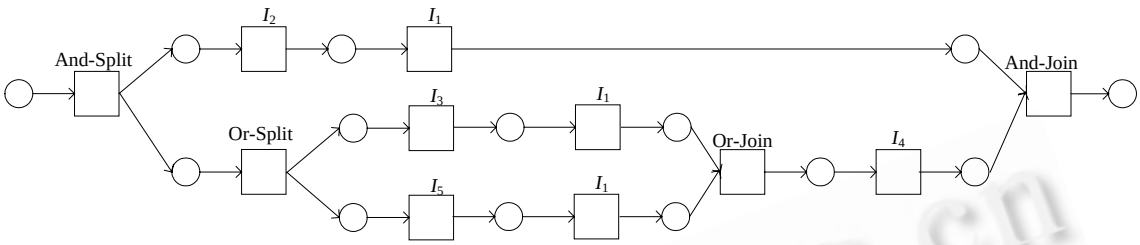
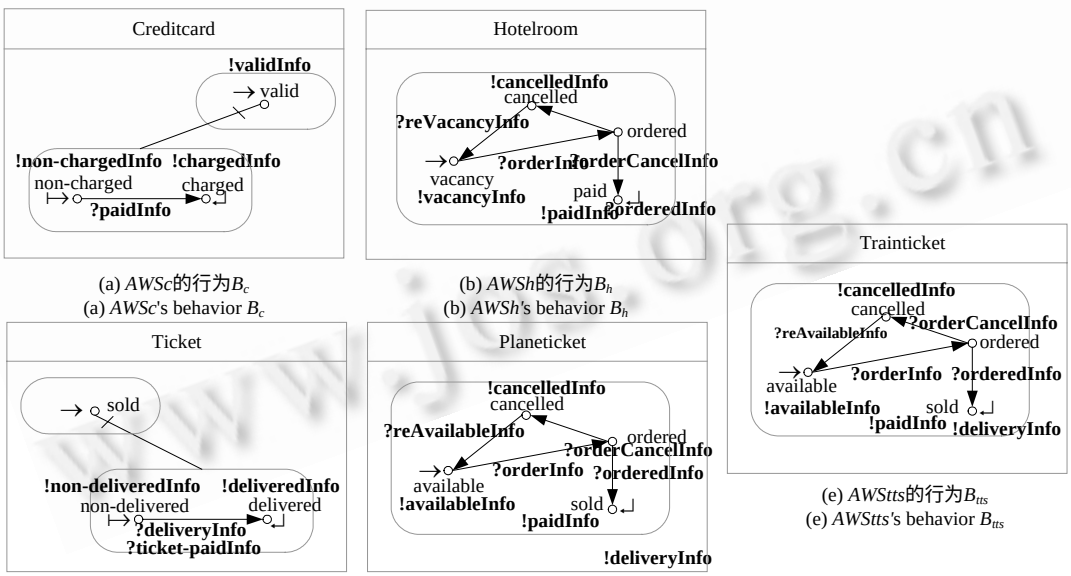


Fig.6 Control logic between intentions in requirement

图 6 需求中意图间的控制逻辑

4.2 AWS能力的表示

在这个案例中,我们定义了 5 个 AWS 的能力表示.其中:AWS_c 表示能够提供信用卡支付功能的服务,它有一个支付行为(记为 B_c);AWS_h 表示能够提供房间预订功能的服务,它有一个房间预订行为(记为 B_h);AWS_{td} 表示能够提供机票或火车票投递功能的服务,它有一个投递行为(记为 B_{td});AWS_{pts} 表示能够提供机票代购功能的服务,它有一个机票代购行为(记为 B_{ts});AWS_{tts} 表示能够提供火车票代购功能的服务,它有一个火车票代购行为(记为 B_{tts}).图 7 是以上 AWS 能力中 5 种行为的表示,其中,→表示初始状态,↵表示结束状态,↦为子状态集的默认状态,?表示输入,!表示输出,黑色箭头为状态转换,带小短线的直线为超级状态向子状态集的转换(这是一种非常状态转换,不需要输入触发).



(a) *AWSc*的行为B_c
(a) *AWSc*'s behavior B_c

(b) *AWSh*的行为B_h
(b) *AWSh*'s behavior B_h

(e) *AWStts*的行为B_{tts}
(e) *AWStts*'s behavior B_{tts}

(c) *AWStd*的行为B_{td}
(c) *AWStd*'s behavior B_{td}

(d) *AWSpts*的行为B_{pts}
(d) *AWSpts*'s behavior B_{pts}

Fig.7 Behavior of 5 *AWS*

图 7 5 个 *AWS* 行为的表示

根据算法 3,由于不具有实现需求中 4 个 Intention 的 Behavior 的 AWS 不会向需求进行服务登记,也就不能向需求聚集,所以这里只列出了 5 个 AWS,而将所有其他的 AWS 忽略掉.这里,每个 AWS 都只有一个行为,每个

行为正好实现需求 *Req_Travel_Arrange* 中的一个意图.

4.3 聚集过程模拟

根据算法 1,并结合以上关于 AWS 的能力与需求 *Req_Travel_Arrange* 的表示,我们容易得出:这里的每个 AWS 能力都与需求部分匹配.表 3 给出了算法 4 执行过程中 4 个输出结构中内容的动态变化情况.

Table 5 Simulation of the execution of algorithm 4

表 5 算法 4 执行过程模拟

Phase	4 structures			
	Λ	Γ	K	AWSS
Initialization	$\{I_1,I_2,I_3,I_4,I_5\}$	{*hotelroom-paidInfo*, *planeticket-paidInfo*, *trainticket-paidInfo*}	$\varnothing$	$\varnothing$
After AWS_c requesting to register service	$\{I_2,I_3,I_4,I_5\}$	{*hotelroom-paidInfo*, *planeticket-paidInfo*, *trainticket-paidInfo*}	$\varnothing$	$\{AWS_c\}$
After AWS_h requesting to register service	$\{I_3,I_4,I_5\}$	{*planeticket-paidInfo*, *trainticket-paidInfo*}	$\{\langle B_h,B_c\rangle\}$	$\{AWS_c,AWS_h\}$
After AWS_{pts} requesting to register service	$\{I_4,I_5\}$	{*trainticket-paidInfo*}	$\{\langle B_h,B_c\rangle,\langle B_{pts},B_c\rangle\}$	$\{AWS_c,AWS_h,AWS_{pts}\}$
After AWS_{td} requesting to register service	$\{I_5\}$	{*trainticket-paidInfo*}	$\{\langle B_h,B_c\rangle,\langle B_{pts},B_c\rangle\}$	$\{AWS_c,AWS_h,AWS_{pts},AWS_{td}\}$
After AWS_{tts} requesting to register service (End)	$\varnothing$	$\varnothing$	$\{\langle B_h,B_c\rangle,\langle B_{pts},B_c\rangle,\langle B_{tts},B_c\rangle\}$	$\{AWS_c,AWS_h,AWS_{pts},AWS_{td},AWS_{tts}\}$

可以证明,无论 5 个 AWS 以怎样的顺序聚集都不会影响最终的结果.图 8 给出了聚集结果 AWS 生成的 THSM 及相互之间的消息交换关系.不难发现,需求驱动的 AWS 聚集的结果是一个服务的组合,而这个组合的服务行为执行逻辑是由需求的意图控制逻辑决定的.

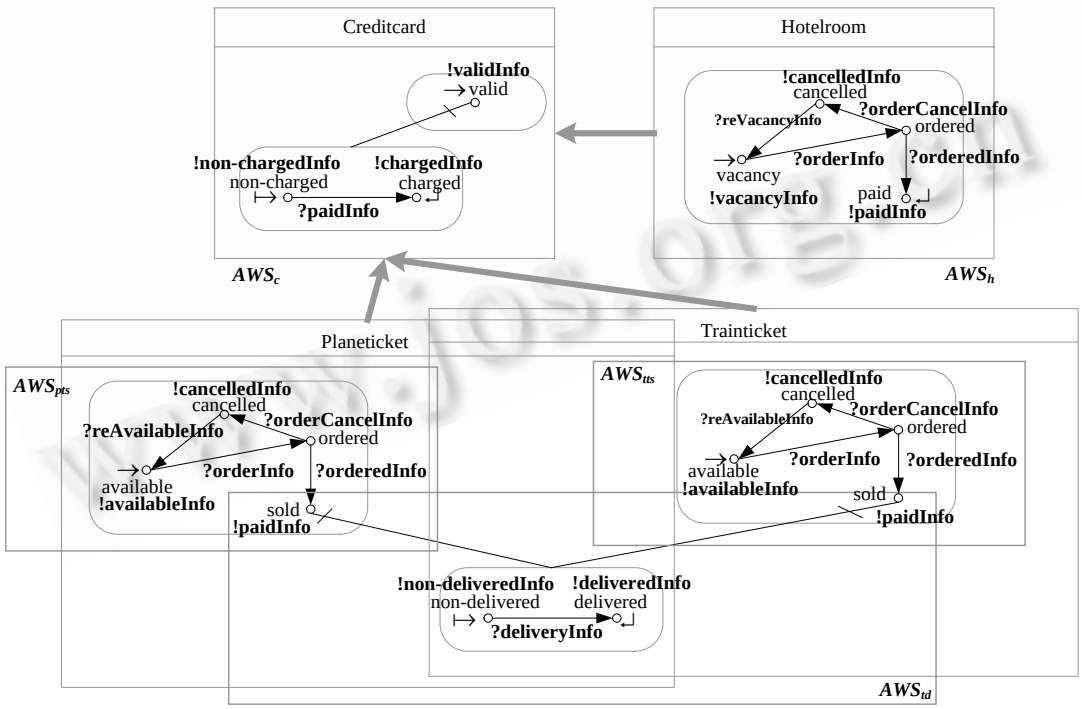


Fig.8 Result of AWS aggregation

图 8 AWS 聚集结果

4.4 案例讨论

文献[15]在环境本体的基础上提出了一种基于作用效果的 Web 服务组合方法,通过将 Web 服务的能力建模为服务对环境资源产生的影响,并将影响对应的 THSMs 与其上必要消息交换关系定义为一个语义模式(semantic schema),最后通过推理将若干个原子服务的语义模式组合成一个目标服务的语义模式,从而实现服务的组合.但这种服务组合方式没有实现服务的控制逻辑,也没有对子服务之间的消息交换作控制上的限制.本案例扩充了文献[15]采用的实例,使能包含顺序、并发和选择 3 种控制逻辑.通过比较我们发现,这两种方法的服务组合结果是相似的,AWS_c,AWS_h,AWS_{pts},AWS_{tts} 和 AWS_{td} 通过消息交换关系组成一个更大的 AWS_G,而这个 AWS 刚好能实现需求的全部意图.所以我们认为,这种基于意图-行为-实现机制的自主 Web 服务聚集模型实现了一种主动服务的组合过程.

5 结论和下一步工作

通过案例讨论可以发现,本文提出的需求驱动的自主 Web 服务聚集模型实际上是一种主动服务组合的过程模型.在传统的方法中,Web 服务只是一个被动地供人发现和调用的对象,但越来越多的研究正逐步转向将 Web 服务视为一个自主的[16]、具有主动行为的计算实体.本文将 Web 服务看作是一个能够主动搜索和发现需求的主动对象,以环境本体为基础给出一种基于意图-行为-实现机制的需求和服务能力匹配方法,并设计了相应的算法实现了服务向需求的聚集.这种方法具有如下几个优点:

(i) 由于 Web 服务可以主动搜索需求,所以可以极大地提高服务被利用的机率;

(ii) 它是一种基于本体的方法,有利于需求和服务能力以一种无歧义和机器可理解的方式表示;

(iii) 它是一种基于状态机的、与实现无关的过程模型,适合在较高层次上进行推理;

(iv) 由于意图和行为都是原子的,所以建立在两者实现机制上的服务需求匹配相对简单;

(v) 用服务主动聚集的方式实现 Web 服务的组合,可以更好地体现服务方的利益.

目前,在该方法的研究中还有一些问题有待解决,包括:

(1) 在聚集过程中只考虑 AWS 到达的顺序,即先到先提供服务,而没有考虑按照服务质量进行优选;

(2) 构建的 APDR4AWS 平台还比较初步,还没有对内部的机制作更深入的研究;

(3) 在多需求多 AWS 环境下的聚集效果的测试也不够全面.这些都是我们下一步研究工作的重点.

致谢 在此,向对本文的完成提供过帮助的中国科学院数学与系统科学研究所和中国科学院计算技术研究所的老师和同学表示最衷心的感谢.

References:

[1] Sycara K, Paolucci M, Ankolekar A, Srinivasan N. Automated discovery, interaction and composition of semantic Web services. Journal of Web Semantics, 2003,1(1):27−46.

[2] Berardi D, Calvanese D, Giacomo GD, Lenzerini M, Mecella M. Automatic composition of e-services that export their behavior. In: Orlowska ME, Weerawarana S, Papazoglou MP, Yang J, eds. Proc. of the ICSOC 2003. LNCS 2910, Heidelberg: Springer-Verlag, 2003. 43−58.

[3] OWL-S Coalition. OWL-S 1.1 release. 2004. http://www.daml.org/services/owl-s/1.1/

[4] Roman D, Keller U, Lausen H, de Bruijn J, Lara R, Stollberg M, Polleres A, Feier C, Bussler2 C, Fensel D. Web service modeling ontology. Applied Ontology, 2005,1(1):77−106.

[5] Zheng LW, Jin Z. Requirement driven aggregation of active Internetware entities. Journal of Software, 2008,19(5):1083−1098 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/19/1083.htm

[6] Wang PW, Jin Z, Liu L, Cai GJ. Building toward capability specifications of web services based on an environment ontology. IEEE Trans. on Knowledge and Data Engineering. 2008,20(4):547−561.

[7] Yue K, Wang XL, Zhou AY. Underlying techniques for Web services: A survey. Journal of Software, 2004,15(3):428−442 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/15/428.htm

[8] Curbera F, Goland Y, Klein J, Leymann F, Roller D, Thatte S, Weerawarana S. Business process execution language for Web services version 1.1. IBM Document, 2002. http://www.ibm.com/developerworks/library/specification/ws-bpel/

[9] Casati F, Ilnicki S, Jin LJ, Krishnamoorthy V, Shan MC. Adaptive and dynamic service composition in eFlow. In: Benkt W, Lars B, eds. Proc. of the Int'l Conf. on Advanced Information Systems Engineering. LNCS 1789, Stockholm: Springer-Verlag, 2000. 13−31.

[10] Hamadi R, Benatallah B. A Petri net-based model for Web service composition. Proc. of the 14th Australasian Database Conf. on Database Technologies, 2003,143(17):191−200.

[11] Tang Y, Chen L, He K, Jing N. SRN: An extended Petri-net-based workflow model for Web service composition. In: Proc. of the IEEE Int'l Conf. on Web Servies (ICWS 2004). San Diego: IEEE Computer Society, 2004. 591−599. http://doi.ieeecomputersociety.org/10.1109/ICWS.2004.1314786

[12] Liao J, Tan H, Liu JD. Describing and verifying Web service using pi-calculus. Chinese Journal of Computers, 2005,28(4):635−643 (in Chinese with English abstract).

[13] Wang PW, Jin Z, Liu L. On constructing environment ontology for semantic Web services. In: Lang J, Lin FZ, Wang J, eds. Proc. of the 1st Int'l Conf. on Knowledge Science, Engineering and Management (KSEM 2006). LNAI 4092, Heidelberg: Springer-Verlag, 2006. 490−503.

[14] Neches R, FikesR, Finin T, Gruber T, Patil R, Senator T, Swartout WR. Enabling technology for knowledge sharing. AI Magazine, 1991,12(3):36−56.

[15] Wang PW, Jin Z. Web service composition: An approach using effect-based reasoning. In: Georgakopoulos D, Ritter N, Benatallah B, Zirpins C, Feuerlicht G, Schoenherr M, Motahari-Nezhad HR, eds. Proc. of the Service-Oriented Computing ICSOC 2006. Heidelberg: Springer-Verlag, 2007. 62−73.

[16] The Intelligent Software Agents Lab. Semantic Web services. Carnegie Mellon University. http://www.cs.cmu.edu/~softagents/web_services_into.html

附中文参考文献:

[5] 郑丽伟,金芝.需求驱动的主动网构实体聚合.软件学报,2008,19(5):1083−1098. http://www.jos.org.cn/1000-9825/19/1083.htm

[7] 岳昆,王晓玲,周傲英.Web 服务核心支撑技术:研究综述.软件学报,2004,15(3):428−442. http://www.jos.org.cn/1000-9825/15/428.htm

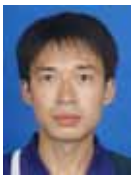
[12] 廖军,谭浩,刘锦德.基于π-演算的 Web 服务组合的描述和验证.计算机学报,2005,28(4):635−643.

叶荣华(1971－),男,浙江绍兴人,副教授,主要研究领域为智能网络计算,多 Agent 理论和技术,面向对象软件工程.

郑丽伟(1979－),男,博士,主要研究领域为知识工程,多 Agent 理论和技术.

金芝(1962－),女,博士,研究员,博士生导师,CFF 高级会员,主要研究领域为需求工程,领域建模,基于知识的软件工程.

杨夏芬(1987－),女,硕士,主要研究领域为智能网络计算.



王璞巍(1979－),男,博士,讲师,主要研究领域为 Web 服务,语义 Web,基于知识的软件工程方法.