ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.19, No.5, May 2008, pp.1083−1098
DOI: 10.3724/SP.J.1001.2008.01083

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

需求驱动的主动网构实体聚合*

郑丽伟[1,3], 金 芝[1,2+]

[1](中国科学院 数学与系统科学研究院,北京 100190)

[2](中国科学院 计算技术研究所,北京 100190)

[3](中国科学院 研究生院,北京 100049)

Requirement Driven Aggregation of Active Internetware Entities

ZHENG Li-Wei[1,3], JIN Zhi[1,2+]

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

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

[3](Graduate University, The Chinese Academy of Sciences, Beijing 100049, China)

+ Corresponding author: E-mail: zhijin@amss.ac.cn

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

Abstract: For Web services, a typical Internetware, a new Web service composition model is given: Requirement driven active Internetware entities aggregating from the aspects of active service, requirement driven and automated aggregating. In this model, service Agents discover requirement actively and aggregate around the requirement automatically. The Agent group aggregated for a given requirement becomes an effective multi-Agent system through a process of mechanism design, negotiation, and collaboration. For making Agents have the ability to discover and understand requirements, a function ontology has been created, which includes function descriptions and function decomposition modes in special domain. For the service Agents collaboration, requirement driven automated mechanism design is put forward, and a collaboration model named barycenter model based on justice principle is also constructed. At last a specification for the final multi-Agent system is given.

Key words: active entity; requirement driven; aggregation; Agent collaboration

摘 要: 从主动服务、需求驱动、自主聚合的角度,提出了需求驱动的主动网构实体聚合模型.在该模型中,主动网构实体形成服务 Agent,这些服务 Agent 主动发现需求并向需求聚集.针对同一需求聚集形成的服务 Agent 群,通过机制设计、协商、协作最终形成能够满足需求的多 Agent 系统.为了实现服务 Agent 对需求的发现和识别,构造了一个功能本体,包括特定领域的功能描述以及功能分解模式.为了实现服务 Agent 的协作,提出了需求驱动的自动机

* Supported by the National Natural Science Foundation of China under Grant Nos.60496324, 60625204 (国家自然科学基金); the National Basic Research Program of China under Grant No.2002CB312004 (国家重点基础研究发展计划(973)); the National High-Tech Research and Development Plan of China under Grant No.20060101Z1113 (国家高技术研究发展计划(863)); the Knowledge Innovation Program of the Chinese Academy of Sciences (中国科学院知识创新工程); the Key Laboratory of Management, Decision and Information Systems, the Chinese Academy of Sciences (中国科学院管理、决策与信息系统实验室)

Received 2007-06-13; Accepted 2007-10-15

制设计,并给出了一种基于公平原则的协作模型——重心模型.最后得出由聚合产生的多 Agent 系统规格表示.

关键词: 主动实体;需求驱动;聚合;Agent 协作
中图法分类号: TP393 文献标识码: A

网构软件是一种新型的软件开发形态[1].在这种新的软件开发形态中,软件实体以开放自主的软件服务形式存在于 Internet 的各个节点上,软件开发过程就成为这些软件实体间通过互联、互通、协作、联盟构成组合软件的过程.

面向服务的计算可以说是网构软件的一种典型的实现方式.服务以可共享与可集成的资源为基础来构建,通常表现为一个独立的业务功能.因此,它具有更大的粒度和更强的独立性,这使得服务间具有松耦合的特性,从而为形成灵活、动态的服务组合带来了方便.以服务组合形式出现的网构软件比对象、构件等早期形式更能适应开放、动态、多变的 Internet 使用环境.目前的 Web 服务平台涉及 3 个角色(即服务提供者、服务代理和服务请求者)和 3 个操作(即发布、发现和绑定[2]).服务代理管理和发现服务,服务提供者将可提供服务的信息在服务代理上注册,并管理外界对其服务的访问.服务请求者代表用户在服务代理上搜索合适的服务,并建立与服务的链接.代表性的工作可以分为 3 个层次,其中,底层是一组传输协议,也是被广泛使用的 Internet 标准,如 HTTP,FTP,UDP 等;中间层部分是目前 W3C 推荐的 Web 服务的相关协议标准,包括简单对象访问协议 SOAP[3] 和 Web 服务描述语言 WSDL[4]等;上层部分是统一描述、发现、集成协议 UDDI 和进行服务组合的 WSFL,BPEL4WS[5−7]等.在整个服务发现和组合的过程中,Web 服务这种网构实体通常以一种被动程序实体的形式存在,实体的发现和组合均由服务请求者或者服务代理通过直接的服务调用或者用服务组合语言编写程序等方式进行,其动态性和适应性受到很大的限制.

如果网构实体是一些主动的服务实体,情形会如何呢?实际上,Agent 理论和技术为服务实体由“被动”变“主动”提供了一条可行的路径.本文提出了一种需求驱动的主动网构实体聚合模型,在该模型中,服务实体被看成为主动的程序实体,即服务 Agent,服务 Agent 能够发现服务请求者提出的服务需求,并主动向服务需求聚集.聚集起来的服务 Agent 通过机制设计和协商协作过程形成能够满足服务需求的多 Agent 系统.为了实现 Agent 对需求的发现和识别,本文还构造了一个功能本体,它包括特定领域的功能描述以及功能分解模式.为了实现服务 Agent 的协作,提出了需求驱动的自动机制设计,并给出了一种基于公平原则的协作模型——重心模型.最后得出由聚合产生的多 Agent 系统规格表示.

本文第 1 节是方法概述.第 2 节给出功能本体的主要部分和基本概念.第 3 节讨论需求驱动的多 Agent 聚集.第 4 节介绍需求驱动的自动机制设计过程和重心模型,并给出一种对应于需求解决方案的 MAS 系统规格框架.第 5 节通过案例研究展示所提出方法的有效性.第 6 节和第 7 节是相关工作比较和方法总结.

1 方法概述

需求驱动的服务 Agent(或简称 Agent)聚合过程分为两个阶段:需求发现及服务 Agent 的聚集阶段和服务 Agent 协作满足需求的协作阶段.其中,聚集阶段又包含两步,即需求发现与识别以及需求驱动的服务 Agent 聚集.协作阶段也有两步,即需求驱动的自动机制设计以及针对特定需求目标的服务 Agent 协作.

服务 Agent 首先必须能够发现并识别需求.传统的服务发现使用一种公共的、标准化的服务描述语言(如 WSDL)来描述服务,以使服务搜索和发现程序可以识别该服务.我们也需要这样一种公共的语言或者词汇表来辅助 Agent 发现和理解需求.本文主要从功能角度研究服务 Agent 对需求的发现和满足,因此,我们建立了一个领域功能本体(function ontology)[8,9],这个功能本体在需求发现和 Agent 聚集阶段中起着关键作用,它用于支持需求的功能描述和 Agent 的能力描述,使得在功能层面上服务 Agent 可以理解需求,并且能对是否参与该需求的解决作出判断.

我们沿用了传统 Agent 理论中的自利 Agent 假设,即所有 Agent 都试图使自身收益最大化,把服务 Agent 也作为一种自利型 Agent.而需求的解决能够为参与的 Agent 带来收益.服务 Agent 为了获得尽可能多的收益,

主动向包含与自身能力匹配的功能的需求聚集,形成服务 Agent 群体向需求聚集的“磁石”现象,如图 1 所示.我们称这种现象形象为需求对 Agent 的“磁石效应”.

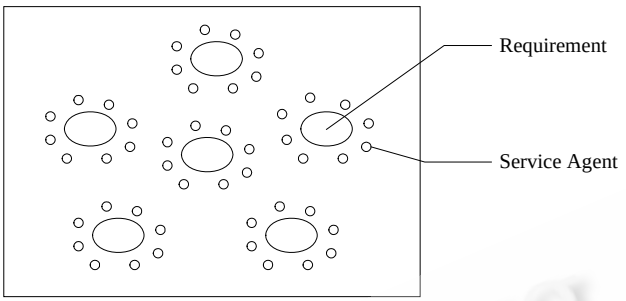

Fig.1 Magnet effect

图 1 磁石效应

为了能使聚集起来的 Agent 群形成有效协作,我们对 Agent 行为作出一些约束和导向,以确保协作有效.传统多 Agent 理论通过机制设计来产生使 Agent 行为满足约束的全局行为规则,这种规则被称为机制.例如,拍卖中的拍卖规则就是一种机制.本文借鉴 Contizer 和 Sandholm 提出的自动机制设计模型(automated mechanism design)[10],提出了需求驱动的自动机制设计模型来建立服务 Agent 协作中的机制.

全局的机制仅能对 Agent 的行为产生约束和导向作用,却不能帮助 Agent 完成一次具体的协作.所以,我们给出了一种根据需求选择最终解决方案的服务 Agent 协作模型:重心模型.首先,各参与 Agent 针对给定需求目标,参考功能本体提出一种最有利于自身的目标解决方案.然后,依据公平性原则协商,在所有提出的方案中选出一个最终解决方案,并依此形成针对该需求目标的多 Agent 系统(MAS).

2 功能本体

功能本体(function ontology,简称 FO)是特定领域中可能的软件功能构成的概念组成的词汇表,它是 Agent 能力描述和需求功能描述的基础,为 Agent 发现和理解需求提供了保障.功能本体包含 3 组基本概念:资源、功能、功能分解模式.

资源. 在特定领域中,任意可以被 Agent 识别和操作的事物称为资源(resource),例如信息(值、数据等)、物理实体(硬件、人等)、软件系统等等.所有资源均可用一组特定的属性来表示.每个属性用于表征资源一个方面的特性.资源的属性可以分成两类,静态属性和动态属性.静态属性是指在资源的整个生命周期中,其值不随时间发生变化的属性,例如函数的名称、构件的 GUID 等等;而动态属性的值将随外界条件的改变而发生变化,例如显示设备的显示数据量,随着显示内容的不同,该属性会不断发生变化.

定义 1(资源). 资源可以用一个六元组来表示,

$$Res:=\langle SAttr,DAttr,SARan,DARan,ValFuncofSAttr,ValFuncofDAttr\rangle,$$

其中:

- $SAttr=\{sa_1,\ldots,sa_n\}$是静态属性的有限集合;
- $DAttr=\{da_1,\ldots,da_n\}$是动态属性的有限集合;
- $SARan=\{sv_1,\ldots,sv_m\}$是静态属性的值域;
- $DARan=\{dv_1,\ldots,dv_m\}$是动态属性的值域;
- $ValFuncofSAttr:SAttr\rightarrow SARan$ 是静态属性的取值函数;
- $ValFuncofDAttr:DAttr\rightarrow P(DARan)$是动态属性的取值函数,$P(DARan)$是 $DARan$ 的幂集.

为了方便表述,我们规定如下几个符号:

- @表示从属关系;

- *sa@res* 表示资源 *res* 的一个静态属性 *sa*;
- *da@res* 表示资源 *res* 的一个动态属性 *da*.

一个资源的所有动态属性,在给定顺序下都可以构成一个向量,当属性顺序确定时该向量唯一.我们称此向量为资源的动态属性向量.资源动态属性向量的任意一个有效取值为该资源的一个状态(state).由于资源的动态属性是可变的,所以状态也是可变的.资源状态的变迁表现了资源受外界作用而发生的属性改变.

定义 2(状态). 设资源 *res* 共有 *n* 个动态属性 $da_1,\ldots,da_n$,则 *res* 的一个状态为 $sa@res=(val_1,\ldots,val_n)$,其中,$val_i=ValFuncofDAttr(da_i),i=1,\ldots,n$.

因为外界作用导致资源的状态发生变迁,所以,我们用资源的状态变迁来表征这种外部作用,并称其为作用在资源上的一个行为(behavior).因此,一个行为表示一种作用在资源上的操作.

定义 3(行为). 行为可以表示为一个三元组,$Beh:=\langle res,s_0,s_1\rangle$,其中,*res* 表示行为 *Beh* 所作用的资源,s_0 表示行为作用前 *res* 的状态,s_1 表示行为作用后 *res* 的状态.

定义 4(行为的可加性). 若存在行为 $beh_1:=\langle res,s\_0,s\_1\rangle$,$beh_2:=\langle res,s\_1,s\_2\rangle$,则 beh_1 与 beh_2 可加.称 beh_1 为 beh_2 的一个直接前驱,beh_2 为 beh_1 的直接后继,并且规定 $beh_3:=\langle r,s\_0,s\_2\rangle$ 为 beh_1 和 beh_2 的复合行为,记为

$$beh_3=beh_1+beh_2.$$

功能是用户希望得到的某种服务或者希望计算机软件能够完成的某种任务.从资源的角度看,功能反映了用户希望的资源的某种状态变迁.例如,用户希望计算机软件将一些数据以报表的形式输出,即表示用户希望该软件的一个功能是将数据这种资源从无序状态转化成有序状态.而在功能本体中,资源的状态变迁是通过行为这个概念来刻画的.一个功能由一些行为组成,这些行为可以并行、顺序执行或者在给定条件下选择分支执行.

一个功能的描述包含资源、行为、行为条件和收益分配比例 4 个部分.资源部分描述该功能所有的可操作资源,用资源集表示.行为部分描述用户希望的资源状态变迁,用行为集表示.行为条件部分用于给出行为的执行条件.任意两个行为运行时可能是不相关的,如不同资源的状态变迁或者相同资源、但互相不可加的两个状态变迁.此时,两个行为是并发的.任意两个行为运行时也可能是顺序的,如两个可加行为可以顺序执行.还可能有分支,如一个行为有两个直接后继,那么就可以根据执行条件来选择哪个后继作为下一步行为.我们为功能中每个行为都给出了一个执行条件,这个条件可以为空即无条件执行,也可以为一个资源状态的逻辑表达式,例如 $st@res_1=Vec_1$ AND $st@res_2=Vec_2$,其中,Vec_1 和 Vec_2 分别是资源 res_1 和 res_2 的动态属性向量.收益分配比例是给出完成一个功能后可能获得的收益中各个行为所占的百分比,用一个不大于 1 的正数集合来表示.

定义 5(功能). 功能可以表示为一个六元组,

$$Fun:=\langle Beh,Res,Cond,CondFuncofBeh,Prop,PropFuncofBeh\rangle,$$

其中:

- *Res* 是功能的所有可操作资源构成的集合;
- *Beh* 是功能所含的行为集合;
- *Cond* 是一个资源状态的逻辑表达式集合;
- $CondFuncofBeh:Beh\rightarrow P(Cond)$是一个条件函数,用于给出 *Beh* 中每个行为的执行条件,其中,$P(Cond)$是条件集 *Cond* 的幂集;
- *Prop* 是各行为的收益分配比例的集合,并且有 $\sum_{i=1}^{|Beh|} p_i=1$,其中,$p_i\in Prop,0<p_i\leq 1,|Beh|$为集合 *Beh* 的基数,用于表示各行为在功能完成时可能获得的收益比例;
- $PropFuncofBeh:Beh\rightarrow Prop$ 是行为的收益比例函数.

在功能本体中,我们将特定领域中已有的功能分解方案进行总结、抽象,形成一种具有领域内通用性质的功能分解模式,作为领域内的公共知识为服务 Agent 的协作提供支持.由于行为的可加性,使得一个功能可以由多个行为构成.同样,多个功能根据一定条件、按照一定顺序也能构成更高层的功能.因此,对于大粒度的功能而言,一般总能由一些小粒度的子功能通过顺序、分支等方式组合而成.这些子功能及其组合方式就是该大粒度

功能的一个分解.

一个功能分解模式的描述包含子功能、功能条件、收益分配比例 3 部分.子功能是由所有构成该功能的子功能组成的集合.功能条件部分用于给出子功能的执行条件,用资源状态的逻辑表达式集合来描述.

定义 6(功能分解模式). 功能 *func* 的一个分解模式可以表示为一个五元组,

$$FuncDecMod(func):=\langle SubFuncs,Cond,CondFuncofSubFuncs,Prop,PropFuncofSubFuncs\rangle,$$

其中:

- $SubFuncs=\{func_1,\ldots,func_n\}$是一组功能的集合,其中,$func_i(1\le i\le n)$是 *func* 的一个子功能;
- *Cond* 是一个资源状态的逻辑表达式集合,其中,资源可以是功能 *func* 的资源集中任意的资源;
- $CondFuncofSubFuncs:SubFuncs\rightarrow P(Cond)$是一个条件函数,用于给出 *Subfuncs* 中每个子功能的执行条件,其中,$P(Cond)$是条件集 *Cond* 的幂集;
- *Prop* 是各子功能收益分配比例的集合,并且有 $\sum_{i=1}^{n} p_i=1$,其中,$p_i\in Prop,0<p_i\le 1$,用于表示各子功能在功能 *func* 完成时可能获得的收益比例;
- $PropFuncofSubFuncs:SubFuncs\rightarrow Prop$ 是子功能的收益比例函数.

表 1 列出了顶层功能本体的概念.

Table 1 Concept classes of top-level function ontology

表 1 顶层功能本体的概念列表

Concept class	Description	Super class
owl:thing	The root class.	without
Resource	The concept class of resource, including all the resource instances.	owl:thing
Attribute	The concept class of attribute.	owl:thing
Static attribute	The static attribute concept of resource.	Attribute
Dynamic attribute	The dynamic attribute concept of resource.	Attribute
State	A valid value of the dynamic attribute vector.	owl:thing
Behavior	The concept for describing the state transition of resource.	owl:thing
Composite behavior	The composition behavior concept of two additive behaviors.	Behavior
Function	The function concept, including all the function instances.	owl:thing
Sub-Function	The sub-function concept.	Function
Function decomposition mode	The concept of function decomposition mode.	owl:thing
Execution condition	A group of logic expressions defined in resource states.	owl:thing
Execution condition of behavior	The execution condition of behaviors in functions.	Execution condition
Execution condition of sub-function	The execution condition of sub-functions in function decomposition modes.	Execution condition
Payoff distribution proportion	The concept of payoff distribution proportion.	owl:thing
Payoff distribution proportion of behavior	The Payoff distribution proportion concept of behaviors in functions.	Payoff distribution proportion
Payoff distribution proportion of sub-function	The Payoff distribution proportion concept of sub-functions in function decomposition modes.	Payoff distribution proportion

3 需求驱动的Agent聚集

本节讨论需求驱动的服务 Agent 协作模型的第 1 阶段——聚集阶段.我们首先给出了基于功能本体的网构软件需求和 Agent 形式化表示,然后提出了一种依据功能本体构造功能分解树的方法,并在此基础上讨论聚集的形成.

3.1 网构软件需求的表示

网构软件需求是指用户提出的需要由网构软件来完成的功能.功能本体给出了特定领域中有关功能的知

识描述,以此为知识背景,可以给出网构软件需求的功能描述.

一个网构软件需求的描述包含两个部分:功能部分和代价部分.功能部分用于表示该需求所要求的功能,我们用一个功能的集合来表示;代价部分是指用户为此需求可以支付的代价,本文中主要指为完成需求的各个功能所支付的代价,可以用一个正实数集合表示.

定义 7(网构软件需求). 网构软件需求可以表示为一个三元组,

$$Req:=\langle Funcs,Cost,CostFuncofFuncs\rangle,$$

其中:

- *Funcs* 是需求所要求的功能构成的集合;
- $Cost=\{p|p\in\mathbf{R}\}$是一组对应于 *Funcs* 中功能,需求方所能付出的代价;
- $CostFuncofFuncs:Funcs\rightarrow Cost$ 是功能的代价函数.

3.2 服务Agent的表示

服务 Agent 为了能够识别网构软件需求,必须能够理解需求的功能表示,并且自身的能力描述和需求的功能描述必须可比较.所以,服务 Agent 的能力描述和网构软件需求的功能描述必须采用相同的术语.因此,同样采用功能本体作为知识背景,可以给出服务 Agent 的能力描述.

一个服务 Agent 的能力描述包含两个部分,即行为部分和行为的最小预期收益部分.行为部分用于描述一个服务 Agent 具有的所有行为,即它所能造成的各种资源状态变迁,是 Agent 能力的体现,可以用一个行为的集合表示;行为的最小预期收益是指服务 Agent 在参与需求的解决过程中对该行为所期望得到的最小收益.最小期望收益是 Agent 用于判断是否参与某需求解决的标准.下面从功能角度给出服务 Agent 的定义.

定义 8(服务 Agent). 一个服务 Agent 可以表示为一个三元组,

$$Agent:=\langle Beh,\mathrm{Min}Pay,\mathrm{Min}PFuncofBeh\rangle,$$

其中:

- *Beh* 是 Agent 所拥有的行为构成的集合;
- Min*Pay* 是 Agent 每个行为的最小期望收益构成的集合;
- $\mathrm{Min}PFuncofBeh:Beh\rightarrow\mathrm{Min}Pay$ 是行为的最小期望收益函数.

3.3 网构软件需求的分解

对于网构软件需求中任意一个功能,如果功能本体中存在它的 1 个或多个分解模式,就可以根据下面步骤为其构造一棵功能分解树.

设 *func* 为任意功能,且 *func* 有 *n* 个分解模式 $funcdm_1,funcdm_2,\ldots,funcdm_n$.按以下步骤为 *func* 构造一棵与或树:

- 将 *func* 作为树的根节点;
- 将 $funcdm_1,funcdm_2,\ldots,funcdm_n$ 作为根的或子节点;
- 将 $subfunc_{i1},\ldots,subfunc_{in},subfunc_{ik}\in SubFuncs@funcdm_i,(k=1,\ldots,n)$作为 $funcdm_i$ 的与子节点;
- 以每个与子节点为根,按照上述步骤分别构造它的与或子树,直至与子节点不存在分解方案.

我们称这样一棵与或树为功能 *func* 的功能分解树,它以 *func* 为根节点,以分解模式为或节点,以子功能为与节点,叶子节点是不可分解的子功能.功能分解树的作用是为服务 Agent 提供更多的可选择功能,也为网构软件需求提供更多被解决的机会.

3.4 服务Agent的聚集

对于一个服务 Agent 而言,选择并确定一个可参与的网构软件需求可以分为两步:第一,判定是否能够参与一个需求;第二,在所有能够参与的需求中确定最终要参与的需求.

首先,Agent 对需求的判别主要从功能角度进行,一个 Agent 如果拥有能够参与需求功能的行为,那么,该 Agent 就可以参与该需求的解决.我们把一个 Agent 可匹配的功能称为该 Agent 的一个合适功能.

定义 9(Agent 的合适功能). 设有 *Agent agt*:=⟨*AgtBeh*,Min*Pay*,Min*PFuncofBeh*⟩和功能 *func*:=⟨*FuncBeh*,*Res*,*Cond*,*CondFuncofBeh*,*Prop*,*PropFuncofBeh*⟩,若满足下列条件,则称 *func* 是 *agt* 的一个合适功能:

- *agt* 至少有 1 个行为 *beh*∈*AgtBeh*,使得满足 *beh*∈*FuncBeh*;
- 设 *funccost* 是功能 *func* 在完成时可获得的收益,则不等式 *funccost*×*PropFuncofBeh*(*beh*)≥Min*PFuncofBeh*(*beh*)成立.

Agent 的所有合适功能对它来说都是能够参与的.如果一个需求的功能集中恰好包含 1 个或者多个 Agent 的合适功能,则该需求对此 Agent 来说是能够参与的.如果在需求的一个功能的某个分解模式的子功能集中存在 Agent 的合适功能,那么 Agent 一样可以参与该功能的实现,同时也参与了需求的解决.那么,为了判定一个需求是否可参与,Agent 必须在需求的各个功能的功能分解树中搜索合适功能.Agent 只要在树中所有与子节点中找到一个合适的功能,就表示 *func* 对该 Agent 是可参与的.那么,Agent 对于需求的可参与性判定规则可以总结为:

- 对需求 *Req*,至少存在 1 个 $func_i@Req$ 是 Agent 能够参与的,其中,$func_i@Req$ 表示需求 *Req* 的一个功能 $func_i$.
- Agent 因参与该需求所获收益不少于自己的最小预期收益.

据此,Agent 可以在环境中找到能够参与的需求.Agent 成为这些需求的候选 Agent.对一个需求而言,所有的候选 Agent 都是通过上述方式聚集在它周围,这就是需求驱动的服务 Agent 聚集现象.

4 需求驱动的自动机制设计及重心模型

本节将讨论需求驱动的服务 Agent 协作模型的第 2 阶段——协作阶段.在本阶段中,服务 Agent 首先在自动机制设计的基础上形成有利于需求解决的联盟,之后,服务 Agent 联盟针对特定需求目标进行协作.我们给出了一种侧重公平性原则的多 Agent 协作模型——重心模型.

4.1 对特定需求的自动机制设计

在传统多 Agent 理论中,机制设计(mechanism desgin)用来产生使 Agent 行为满足约束的全局行为规则,这种规则被称为机制.例如,拍卖中的拍卖规则就是一种机制,所有参与者都必须遵循这种机制.在机制的约束下,Agent 的行为能够满足我们预先给出的约束,比如不能选择不利于需求解决的行为,或者不能提供虚假的服务信息等等.在自动机制设计中,在给定参与 Agent 类型集和产出集的条件下,将机制定义为 Agent 类型集合到产出的映射.其中,Agent 类型是通过将所有参与 Agent 按照某种特性进行分类得到的,产出(outcome)是来源于经济学中的一个概念,是指参与者群体行为所造成的结果.自动机制设计的过程就是在给定一组约束的情况下,计算得出满足约束的机制.遵循机制的 Agent 将形成针对特定产出的联盟,并且该联盟满足预先给定的约束.因此,我们采用自动机制设计来保障针对特定产出的服务 Agent 联盟的形成.

首先,我们针对特定需求将 Agent 类型这个概念扩展到领域功能,将服务 Agent 的类型定义如下:

定义 10(服务 Agent 类型). 设 *req* 是一个需求,*agt* 是 *req* 候选 Agent 集中的一个服务 Agent,*func* 是 *agt* 在 *req* 中找到的一个合适功能,则 *func* 同时也是 *agt* 在 *req* 候选 Agent 集中的一个类型.

上述定义实质上是按合适功能对 *req* 的所有候选 Agent 进行了分类.因为一个服务 Agent 对同一需求可能有多个合适功能,所以,它也可能有多个类型.

同样,我们也把产出概念扩展为服务需求得到满足的结果.因为我们这里的群体行为是一个需求解决过程,那么,一个产出就是这个过程的一个可能结果.需求解决过程的结果由各种可能的需求解决情况决定,本文中即指需求功能集中功能的完成情况.例如,一个需求所有的功能都完成是该需求的一个产出.我们将需求的产出定义如下:

定义 11(需求的产出). 设 *req*:=⟨*Funcs*,*Cost*,*CostFunc*⟩是一个需求.其中,功能集 *Funcs*={$func_1$,…,$func_n$}.构造

向量 $\boldsymbol{o}=(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_n), \boldsymbol{x}_i=\begin{cases}0, & func_i\text{实现}\\ 1, & func_i\text{未实现}\end{cases}, i=1,\ldots,n$,则向量 $\boldsymbol{o}$ 是 req 的一个产出.

对于任意一个产出 $\boldsymbol{o}$,我们把 $\boldsymbol{o}$ 中所有可实现功能的代价和称为需求在产出为 $\boldsymbol{o}$ 时的代价,记为

$$OutcomeCost(\boldsymbol{o}).$$

若一个需求包含 n 个功能,则根据需求中各个功能不同的解决情况,一个需求可能的产出有 2^n 个.但对用户而言,这 2^n 个产出不都是有意义的,其中只有一部分是用户希望得到的产出,这些产出才是服务 Agent 联盟的目标,我们把用户希望得到的产出称为该需求的有效产出.自动机制设计将针对有效产出生成机制.

下面,我们将新的 Agent 类型和产出定义引入到自动机制设计中,给出我们需要的机制定义和约束定义.首先作如下假设:

- 需求的有效产出集为 O;
- 需求的所有候选 Agent 集合为 $AgentSet=\{agt_1,\ldots,agt_N\}$.

对于任意 $agt_i\in AgentSet$,为其构造一个有限类型集 Θ_i,一个效用函数 $u_i:\Theta_i\times O\to\mathbf{R}$.当需求的总代价给定时,任意一个 Agent 类型对应的分解树中的节点根据分配比例可以计算得到该节点功能完成时的收益.这样,对于 Agent agt_i 的任意一个类型 $\boldsymbol{\theta}_i$、任意产出 $\boldsymbol{o}_i$,如果当 $\boldsymbol{o}_i$ 发生时,$\boldsymbol{\theta}_i$ 对应的节点可以被完成,那么,效用函数的值就是该节点的收益;如果当 $\boldsymbol{o}_i$ 发生时,$\boldsymbol{\theta}_i$ 对应的节点不能被完成,则效用函数的值为 0.

定义机制之前,必须说明利润和收益之间的关系.这里的利润是指完成用户需求后,需求提出者会从产生的收益中取一部分作为自己的利润,而用剩余部分支付给 Agent 作为报酬.而收益就是指 Agent 由于完成了需求所创造的全部收益.显然,在需求提供者不要求获得利润的情况下,Agent 能够获得其全部创造的收益.但是,如果需求提出者也是 Agent,那么它如何获得收益呢?所以,应该允许利润的存在,当然利润是收益的一部分.

需求都是要求有确定产出的,也就是由需求给出的有效产出集都是确定的,并且是必需的.我们必须定义一种确定型机制.

定义 12(确定机制). 一个无利润确定机制由一个产出选择函数构成,$\boldsymbol{o}:\Theta_1\times\ldots\times\Theta_n\to O$.所谓无利润是指在此机制下,Agent 可以获得自己创造的全部收益.一个有利润的确定机制由一个产出选择函数 $\boldsymbol{o}:\Theta_1\times\ldots\times\Theta_n\to O$ 和对于每个 Agent 的一个利润选择函数构成,该利润选择函数为 $\pi_i:\Theta_1\times\ldots\times\Theta_n\to\mathbf{R}$,其中,$\pi_i(\boldsymbol{\theta}_1,\ldots,\boldsymbol{\theta}_n)$给出 Agt_i 在类型序列为$(\boldsymbol{\theta}_1,\ldots,\boldsymbol{\theta}_n)$的情况下所创造的利润.

为了保证服务 Agent 能够正常参与协作以及服务 Agent 联盟的有效,参考自动机制设计理论,我们定义了 3 类约束:个体理性约束 IR(individual rationality constraints)、激励合作约束 IC(incentive compatiblility)和目标约束 OC(objective constraints).IR 约束保证每一个参与的 Agent 都能获得一定的收益;IC 约束主要的目标是确保 Agent 在参与协作时不提供虚假信息,在我们的模型中是指虚假的类型报告;OC 约束保证产生的机制中,Agent 类型集对应的产出是可以由该类型集对应的服务 Agent 联盟实现的.

我们定义两类 IR 约束:一类是当 Agent 只知道自己的类型的情况下,要约束 Agent 参与协作,记为 ex interim IR;另一类是当 Agent 不仅知道自己的情况,还知道其他所有参与者的类型的情况下,要约束 Agent 参与协作,记为 ex post IR.

定义 13(IR 约束). 设 $\delta\in\mathbf{R}$ 为任意 Agent agt_i 的收益下限,如果收益达不到该下限,则 Agent 退出需求域.

一个确定机制满足 ex interim IR,当且仅当对于任意 Agent i 和任意类型 $\theta_i\in\Theta_i$,有

$$E_{(\boldsymbol{\theta}_1,\ldots,\boldsymbol{\theta}_{i-1},\boldsymbol{\theta}_{i+1},\ldots,\boldsymbol{\theta}_n)|\boldsymbol{\theta}_i}[u_i(\boldsymbol{\theta}_i,\boldsymbol{o}(\boldsymbol{\theta}_1,\ldots,\boldsymbol{\theta}_n))-\pi_i(\boldsymbol{\theta}_1,\ldots,\boldsymbol{\theta}_n)-\delta]\geq 0.$$

一个确定机制满足 ex post IR,当且仅当对于任意 Agent agt_i 和任意类型向量$(\boldsymbol{\theta}_1,\ldots,\boldsymbol{\theta}_n)\in\Theta_1\times\ldots\times\Theta_n$,有

$$u_i(\theta_i,\boldsymbol{o}(\boldsymbol{\theta}_1,\ldots,\boldsymbol{\theta}_n))-\pi_i(\boldsymbol{\theta}_1,\ldots,\boldsymbol{\theta}_n)-\delta\geq 0.$$

IC 用于约束 Agent,使之永远不可能被激励去给出一个虚假类型.构成此种约束的方法通常有两种,一种是基于优势策略,一种是基于贝耶斯-纳什均衡.

优势策略是指 Agent 给出自己真实类型信息比提供虚假信息能获得更多收益,所以,称提供真实信息为优势策略.贝耶斯-纳什均衡是指在 Agent 仅知道自己的类型而对其他参与者的类型一无所知的情况下采取某种策略,使得所有参与者都无法通过改变现行策略而获得更多收益,在这里是指无法通过提供虚假类型报告来获

得更多收益.

定义 14(IC 约束). 基于优势策略的 IC 在确定机制中定义如下:

对于任意 Agent i 和任意类型向量$(\boldsymbol{\theta}_1,\ldots,\boldsymbol{\theta}_i,\ldots,\boldsymbol{\theta}_n)\in\Theta_1\times\ldots\times\Theta_i\times\ldots\times\Theta_n$,对任意可能的虚假类型报告 $\hat{\boldsymbol{\theta}}_i\in\Theta_i$, 有

$$u_i(\boldsymbol{\theta}_i,\boldsymbol{o}(\boldsymbol{\theta}_1,...,\boldsymbol{\theta}_i,...,\boldsymbol{\theta}_n))-\pi_i(\boldsymbol{\theta}_1,...,\boldsymbol{\theta}_i,...,\boldsymbol{\theta}_n)\geq u_i(\boldsymbol{\theta}_i,\boldsymbol{o}(\boldsymbol{\theta}_1,...,\hat{\boldsymbol{\theta}}_i,...,\boldsymbol{\theta}_n))-\pi_i(\boldsymbol{\theta}_1,...,\hat{\boldsymbol{\theta}}_i,...,\boldsymbol{\theta}_n).$$

基于贝叶斯-纳什均衡的 IC 在确定机制中定义如下:

对于任意 Agent agt_i、任意类型$\boldsymbol{\theta}_i\in\Theta_i$和任意可能提供的虚假类型 $\hat{\boldsymbol{\theta}}_i\in\Theta_i$,有

$$E_{(\boldsymbol{\theta}_1,...,\boldsymbol{\theta}_{i-1},\boldsymbol{\theta}_{i+1},...,\boldsymbol{\theta}_n)|\boldsymbol{\theta}_i}[u_i(\boldsymbol{\theta}_i,\boldsymbol{o}(\boldsymbol{\theta}_1,...,\boldsymbol{\theta}_i,...,\boldsymbol{\theta}_n))-\pi_i(\boldsymbol{\theta}_1,...,\boldsymbol{\theta}_i,...,\boldsymbol{\theta}_n)]\geq$$

$$E_{(\boldsymbol{\theta}_1,...,\boldsymbol{\theta}_{i-1},\boldsymbol{\theta}_{i+1},...,\boldsymbol{\theta}_n)|\boldsymbol{\theta}_i}[u_i(\boldsymbol{\theta}_i,\boldsymbol{o}(\boldsymbol{\theta}_1,...,\hat{\boldsymbol{\theta}}_i,...,\boldsymbol{\theta}_n))-\pi_i(\boldsymbol{\theta}_1,...,\hat{\boldsymbol{\theta}}_i,...,\boldsymbol{\theta}_n)].$$

定义 15(OC 约束). 一个确定机制满足 OC 约束,当且仅当对于任意 Agent agt_i 和任意类型向量$(\boldsymbol{\theta}_1,\ldots,\boldsymbol{\theta}_n)\in\Theta_1\times\ldots\times\Theta_n$,有 $\sum_{i=1}^{N}\hat{u}_i(\boldsymbol{o}(\boldsymbol{\theta}_1,...,\boldsymbol{\theta}_n))=OutcomeCost(\boldsymbol{o}(\boldsymbol{\theta}_1,...,\boldsymbol{\theta}_n))$, 其中, $\hat{u}_i(\boldsymbol{o}(\boldsymbol{\theta}_1,...,\boldsymbol{\theta}_n))$ 为在类型组合为$(\boldsymbol{\theta}_1,\ldots,\boldsymbol{\theta}_n)$且目标产出为 $\boldsymbol{o}(\boldsymbol{\theta}_1,\ldots,\boldsymbol{\theta}_n)$时 agt_i 所能获得的收益.OC 约束能够在参与服务 Agent 不退出(IR 约束满足)、不虚报类型(IC 约束满足)的情况下,保证目标产出能够实现.

在给定机制和约束定义以后,可以采用优化算法计算符合约束的机制.此类计算的复杂性分析可参见文献[11].确定和随机的自动机制设计均可在多项式时间内完成.一般情况下,我们也不需要为此专门开发优化算法,只需采用已有的一些效果较好的优化算法包即可,例如 CPLEX[12].

在满足上述 3 类约束的机制约束下,Agent 将针对机制中规定的有效产出形成联盟,下一步服务 Agent 将为达到目标产出而进行协作.

4.2 重心模型——从产出到可执行集

机制设计完成后,Agent 在机制约束下,不同的服务 Agent 联盟将对应于不同的产出.为了得到收益,Agent 联盟必须选择一个能够给它们带来较好收益的产出.这里我们先研究一个产出和它对应的 Agent 联盟.要实现产出,参与者就必须提出方案.当然,每一个参与者都有权提出方案.而由于 Agent 是收益驱动的,所以,自己提出的方案总是能使自己收益最大化的方案.通常,我们会希望能在各方妥协的条件下提出一个大家都有较好收益但不是最好收益的方案,但是无论妥协多少次,总存在收益多少的问题,而每个参与者都希望在最终的方案中能得到较多的收益.因为如果妥协的结果是让某些 Agent 放弃对自己收益的追求,那么最终将导致独裁.所以,我们必须给出一种公平的方案选择机制,在这种选择机制中,最终结果的选取对每个参与者而言都是公平的,并且总的收益尽可能地大.为此,我们借鉴了组织理论中的重心方法[13]提出了重心协作模型,它是一种公平的且最终收益对总体而言尽可能大的方案选择机制.

前面得出了服务 Agent 联盟的集合,集合中的每个元素是由 n 个服务 Agent 组成的一个联盟,如果称其中每个 Agent 是该联盟的一个参与者,则可以假设共有 n 个参与者.由于在机制约束下产出是可实现的,所以,每个参与者必能根据功能本体提出一种产出的解决方案,每一种方案都会对应一个收益分配向量.我们把这样的分配向量记为$(\boldsymbol{x}_1,\boldsymbol{x}_2,\ldots,\boldsymbol{x}_n)$.对于 n 个参与者,设它们各自的理想分配方案对应的收益分配向量为 $\boldsymbol{Pay}_i=(\boldsymbol{a}_{i1},\boldsymbol{a}_{i2},\ldots,\boldsymbol{a}_{in})$,$i=1,\ldots,n$,理想最终收益分配向量为 $\boldsymbol{Pay}_s=(\boldsymbol{x}_1,\boldsymbol{x}_2,\ldots,\boldsymbol{x}_n)$,那么,每个参与者的损失可以用距离函数表示:

$$dist_i(\boldsymbol{Pay}_s)=\sqrt{(\boldsymbol{x}_1-\boldsymbol{a}_{i1})^2+(\boldsymbol{x}_2-\boldsymbol{a}_{i2})^2+\ldots+(\boldsymbol{x}_n-\boldsymbol{a}_{in})^2} \tag{1}$$

由函数(1)定义全体损失函数:

$$f(p)=d_1^2(\boldsymbol{Pay}_s)+d_2^2(\boldsymbol{Pay}_s)+\ldots+d_n^2(\boldsymbol{Pay}_s) \tag{2}$$

这个函数衡量了以 $\boldsymbol{Pay}_s$ 为收益分配方案时全体的不满意程度.在这个函数中,每个参与者所占的权重是相等的,所以它是公平的.于是,$f(p)$就是所有参与者作为一个整体对 $\boldsymbol{Pay}_s$ 体现.那么,我们就可以合理地假设能使 $f(p)$达到最小的分配方案就是 $\boldsymbol{Pay}_s$.计算 f 关于变量 x_i 的偏导数,并令它们等于 0.由于 f 是可微的并且非负,所以,f 一定能取得满足下列方程的点集上的极小值.

$$\frac{\partial f}{\partial x_i}=2(\boldsymbol{x}_i-\boldsymbol{a}_{1i})+2(\boldsymbol{x}_i-\boldsymbol{a}_{2i})+...+2(\boldsymbol{x}_i-\boldsymbol{a}_{ni})=0 \tag{3}$$

由此可得:

$$nx_i=a_{1i}+a_{2i}+\ldots+a_{ni}$$

$$x_i=\frac{1}{n}\sum_{j=1}^{n}a_{ji} \tag{4}$$

故

$$\boldsymbol{Pay}_s=(\boldsymbol{x}_1,\boldsymbol{x}_2,...,\boldsymbol{x}_n)=\frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{a}_{i1},\boldsymbol{a}_{i2},...,\boldsymbol{a}_{in})=\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{Pay}_i \tag{5}$$

f取得的极小值点就是集合{$\boldsymbol{Pay}_1$,…,$\boldsymbol{Pay}_n$}的重心.所以,这个模型称为重心模型.

这样得到的能使整体损失最小的分配方案 $\boldsymbol{Pay}_s$ 只是一个理想的收益分配方案,可能不存在可以与它对应的解决方案.所以,我们从 $\boldsymbol{Pay}_i$ 中选择一个与 $\boldsymbol{Pay}_s$ 距离最近的,即 $dist_i(\boldsymbol{Pay}_s)$最小的解决方案为最终方案.这样既保证了公平性,又可以使整体的收益尽可能地大.

当AMD所得到的机制不只1个时,Agent群可以选择产出.因为每一个产出都对应一个类型集,相当于该产出的一个由类型构成的方案,而每个方案必然能对应到一个Agent群的收益向量,于是,Agent群依然可以采用上面的重心模型选择出对大家都比较公平的一个产出.

```
MAS mas {
[Resource List]
R1: {//Resource R1
AType1 attr1;   //attribute attr1
AType2 attr2;
};
:
[Agent List]
A1: {   //Agent A1
b1i(r1i,V1i_1,V1i_2);   //behavior b1i
b1j(r1j,V1j_1,V1j_2);
:
};
:
[Executable Sequence]
Ai.bi_k1;   //behavior bi_k1 of Agent Ai
Aj.bj_k2;
:
Ah.bh_kn;
}
```

Fig.2 Specific description of a multi-Agent system
图2 一种MAS规格说明方式

由重心模型决策产生的最终解决方案可以表示为一个由Agent及其要采取的行为构成的序列:$AgentBehSeq=((agt_i,beh_k^{a_i}),(agt_j,beh_m^{a_j}),...),i,j,k,m\in\mathbf{N}$, 其中, agt_i 表示Agent $agt_i,beh_k^{a_i}$ 表示 agt_i 的标号为 k 的行为.联盟内的Agent按此序列选择执行各自的行为就构成了一个能够达成给定需求的某种有效产出的多Agent系统(MAS).

这种多Agent系统的一种规格说明方式如图2所示.其中,一个多Agent系统可以规格化为3个部分:[Resource List],[Agent List],[Executable Sequence].[Resource List]是资源列表,其中包含了该MAS系统所有可操作的资源;[Agent List]是Agent列表,包含了参与最终解决方案的所有服务Agent;[Executable Sequence]则按执行顺序给出了各Agent在该系统中所要采取的行为.

5 案例研究

本节以CELF[14]中的E-Learning远程教学作为研究的特定领域,首先介绍该领域的功能本体以及领域中的一些服务Agent.然后以在线考试作为一个需求案例,展现Agent聚集的过程和针对该需求的Agent协作过程.最后给出一个在线考试系统的MAS规格化表示.由于远程教学领域的功能本体是一个庞大的知识系统,所以,这里分别针对其中的资源、资源的状态空间、行为、功能、功能分解模式加以举例说明.

问题数据库是该领域的一个重要资源,它在远程教学中提供各类问题数据,帮助学生或者老师完成教学任务.其静态属性包括数据库的名称,动态属性包括一个用于描述数据库打开关闭以及读写等状态的属性变量.根据功能本体中资源的定义,问题数据库在功能本体中表示为

$$QuestionDatabase:=\langle\{databasename\},\{databasestate\},\{\text{“}QuestionData\text{”}\},$$
$$\{\text{“}open\text{”},\text{“}read\text{”},\text{“}write\text{”},\text{“}close\text{”}\},fun^{SA},fun^{DA}\rangle.$$

即资源 *QuestionDatabase* 名称为 *QuestionData*,其动态属性可取的值包括 *open*,*read*,*write*,*close*.我们用 *Open* 来

表示资源 *QuestionDatabase* 的一个状态(“*open*”),其余 3 个状态也类似表示,可以得到资源 *QuestionDatabase* 的状态空间:{*Open*,*Read*,*Write*,*Close*}.表 2 给出了领域内一些资源的名称和状态空间.

Table 2 Resources and their state space

表 2 资源及其状态空间

Resource	State space
QuestionDatabase	{*Open*,*Read*,*Write*,*Close*}
QestionBuffer	{*Empty*,*NotEmpty*}
AnswerBuffer	{*Empty*,*NotEmpty*}
StandardAnswerBuffer	{*Empty*,*NotEmpty*}
CompareResultBuffer	{*Empty*,*NotEmpty*}
TestPointBuffer	{*Empty*,*NotEmpty*}
TestPaper	{*Empty*,*NotEmpty*,*NotEvaluated*,*HasScore*,*Evaluated*}

下面以作用在资源 *QuestionDatabase* 上的一个行为 *OpenDatabase* 为例来说明功能本体中行为的定义.行为 *OpenDatabase* 的作用是打开一个数据库,即使资源 *QuestionDatabase* 的状态从 *Close* 变迁为 *Open*,根据功能本体中行为的定义,*OpenDatabase* 可以表示为 *OpenDatabase*:=〈*QuestionDatabase*,*Close*,*Open*〉.表 3 给出了领域内一些行为的名称、作用的资源以及始状态和终状态.

Table 3 Behaviors and the state transition

表 3 行为及其状态变迁

Behavior	Resource	Starting state	Ending state
OpenDatabase	*QuestionDatabase*	*Close*	*Open*
ReadFromDatabase	*QuestionDatabase*	*Open*	*Read*
GetQuestions	*QestionBuffer*	*Empty*	*NotEmpty*
CreateTestPaper	*TestPaper*	*Empty*	*NotEmpty*
DisplayTestPaper	*Monitor*	*DisplaybuffEmpty*	*DisplaybuffNotEmpty*
GetStandardAnswer	*StandardAnswerBuffer*	*Empty*	*NotEmpty*
GetAnswerFromUser	*KeyBoard*	*InputbuffEmpty*	*InputbuffNotEmpty*
RecordAnswer	*AnswerBuffer*	*Empty*	*NotEmpty*
CompareAnswer	*CompareResultBuffer*	*Empty*	*NotEmpty*
CalculatePoint	*TestPointBuffer*	*Empty*	*NotEmpty*
WritePointToPaper	*TestPaper*	*NotEvaluated*	*HasScore*
WriteCommentToPaper	*TestPaper*	*HasScore*	*Evaluated*
EvaluateTestPaper	*TestPaper*	*NotEvaluated*	*Evaluated*

下面我们以功能 *CreateTestPaper* 为例介绍功能本体中的功能.*CreateTestPaper* 从问题数据库中获得问题数据,并以试卷的形式输出这些问题数据.它包含两个行为:*GetQuestion* 和 *GreateTestPaper*,表 3 中介绍了这两个行为.对每个行为的收益分配比例是平均分配,即各 50%.行为 *GetQuestion* 的执行条件是资源 *QuestionDatabase* 的状态为 *Read*,行为 *CreateTestPaper* 的执行条件是资源 *QestionBuffer* 的状态为 *NotEmpty*.根据功能本体中功能的定义,*CreateTestPaper* 可以表示为

$$\begin{aligned}&\textit{CreateTestPaper}:=\langle\{\textit{GetQuestions},\textit{CreateTestPaper}\},\{\textit{QestionBuffer},\textit{TestPaper}\},\\&\quad\{\textit{Stateof}(\textit{QuestionDatabase})=\textit{Read},\textit{Stateof}(\textit{QestionBuffer})=\textit{NotEmpty}\},\\&\quad\textit{fun}^{C},\{50\%,50\%\},\textit{fun}^{P}\rangle.\end{aligned}$$

表 4 给出了功能 *CreateTestPaper*.

Table 4 Function *CreateTestPaper*

表 4 功能 *CreateTestPaper*

Behavior	Payoff proportion (%)	Condition
GetQuestions	50	*Stateof*(*QuestionDatabase*)=*Read*
CreateTestPaper	50	*Stateof*(*QestionBuffer*)=*NotEmpty*

下面我们以获取问题分值功能 *GetQuestionPoint* 为例介绍功能分解模式.

在该领域中,功能 *GetQuestionPoint* 从问题数据库中获取一份已有试卷中各个试题的分值.在功能本体中,它有一个功能分解模式,可以分解为子功能 *GetQuestionsFromPaper* 和 *GetDataFromDatabase*.由功能本体中功

能分解模式的定义,*GetQuestionPoint* 的分解模式可以表示为

$$FuncDecMod(GetQuestionPoint):=\langle\{GetQuestionsFromPaper,GetDataFromDatabase\},\{\},fun^{SC},\{40\%,60\%\},fun^{SF}\rangle.$$

表 5 给出了功能 *GetQuestionPoint* 的分解模式.

Table 5 A function decomposition mode of function *GetQuestionPoint*

表 5 功能 *GetQuestionPoint* 的一种功能分解模式

Subfunction	Payoff proportion (%)	Condition
GetQuestionsFromPaper	40	Null
GetDataFromDatabase	60	Null

以上我们通过举例介绍了远程教学领域中的功能本体.下面我们以 Agent *DatabaseSearcher* 为例,介绍 Agent 的表示以及需求 *OnlineTest* 的功能表示.

Agent *DatabaseSearcher* 是一个拥有数据库搜索能力的 Agent,它有 *OpenDatabase* 和 *ReadFromDatabase* 两个行为,分别负责打开数据库和依据某关键字从数据库中搜索数据.根据前文中我们的 Agent 表示方式,*DatabaseSearcher* 可以表示为

$$DatabaseSearcher:=\langle\{OpenDatabase,ReadFromDatabase\},\{1,5\},fun^{BP}\rangle.$$

从中我们可以看到,行为 *OpenDatabase* 的最小期望收益为 1,而 *ReadFromDatabase* 的最小期望收益为 5.

下面我们以在线考试为需求来介绍需求的表示.需求 *OnlineTest* 包含 3 个功能:

CreateTestPaperWithDatabase,*Test*,*Evaluate*.

其中:

CreateTestPaperWithDatabase 根据问题数据库中的数据创建一份试卷;

Test 功能负责考试过程,从终端接收考试者的答案输入,并记录下来;

Evluate 功能负责对一份已提交的试卷进行评分和给出评语.

根据需求的定义,*OnlineTest* 可以表示为

$$OnlineTest:=\langle\{CreateTestPaperWithDatabase,Test,Evaluate\},\{40,30,30\},fun^{FC}\rangle.$$

领域中的 Agent 经过对需求的发现和判别,最后产生愿意参与需求 *OnlineTest* 解决的服务 Agent 共有 4 个,分别是负责数据库搜索的 *DatabaseSearcher*、负责试卷生成的 *TestPaperCreater*、负责考试过程与用户交互的 *InterfaceAgent*.Agent *DatabaseSearcher* 前面已经介绍过,这里不再赘述.表 6 给出了 Agent *TestPaperCreater* 和 Agent *InterfaceAgent* 的说明.

Table 6 Agent *TestPaperCreater* and *InterfaceAgent*

表 6 Agent *TestPaperCreater* 和 *InterfaceAgent*

Agent	Behaviors	Minimum*ExpectedPayoff*
TestPaperCreater	*GetQuestions*	10
	CreateTestPaper	10
InterfaceAgent	*DisplayTestPaper*	5
	GetAnswerFromUser	5
	RecordAnswer	5

由各个 Agent 的行为列表可以看出,它们都包含有需求所需的行为,那么,只要这些行为的收益能够大于 Agent 自身的预期收益,Agent 就会主动向需求聚集形成需求域.我们以 *DatabaseSearcher* 为例来计算其参与需求最终可得的收益.

DatabaseSearcher 的行为 *OpenDatabase* 可直接参与需求的功能 *CreateTestPaperWithDatabase*,该功能在需求中能够得到的收益为 40,*OpenDatabase* 在功能中收益分配比例为 10%,那么,该行为的最终收益为 4.而 *DatabaseSearcher* 对该行为的最小期望收益为 1,所以,*DatabaseSearcher* 必然愿意参与需求 *OnlineTest* 的解决.类似地,我们可以计算其他几个 Agent 的收益情况,可以得到相同的结论.

下面我们在需求域中进行自动机制设计,首先给出 Agent 类型和产出集.在需求 *OnlineTest* 中,各个 Agent

的合适功能如下:

- *DatabaseSearcher* 的合适功能有

  GetDataFromDatabase,*CreateTestPaperWithDatabase*,*GetQuestionPoint*.
- *TestPaperCreater* 的合适功能有

  CreateTestPaper,*GetQuestionsFromPaper*,*CreateTestPaperWithDatabase*.
- *InterfaceAgent* 的合适功能有 *Test*.
- *Evaluater* 的合适功能有 *Evaluate*,*JudgeWongOrRight*,*CalculateFinalPoint*,*GiveComment*.

所有的 Agent 类型包括

GetDataFromDatabase,*CreateTestPaperWithDatabase*,*GetQuestionPoint*,*CreateTestPaper*,
GetQuestionsFromPaper,*Test*,*Evaluate*,*JudgeWongOrRight*,*CalculateFinalPoint*,*GiveComment*,

共 10 种类型.将它们分别记为$\boldsymbol{\theta}_1,\ldots,\boldsymbol{\theta}_{10}$.

因为需求 *OnlineTest* 的 3 个功能对于该需求而言是缺一不可的,所以,可能的产出只有两个:$\boldsymbol{o}_1$,需求解决成功;$\boldsymbol{o}_2$,需求解决失败.

DatabaseSearcher 的可能类型有$\boldsymbol{\theta}_1,\boldsymbol{\theta}_2,\boldsymbol{\theta}_3$,其效用函数 u_1 见表 7.

Table 7 Utility function u_1 of *DatabaseSearcher*

表 7 *DatabaseSearcher* 的效用函数 u_1

u_1	o_1	o_2
θ_1	1.8	0
θ_2	40	0
θ_3	3	0

类似地,根据 Agent 的定义和功能本体,我们可以给出其他 3 个 Agent 的可选类型集和效用函数,这里不再罗列.

本案例中,需求提供者不需要利润,故而有了效用函数和类型集,我们只要选定了约束,就可以采用优化算法求出满足约束的机制.由于组合优化算法不是本文考虑的重点,所以,我们在这里直接给出本案例满足所有 IR 约束和 IC 约束的两条机制.

- $\{\boldsymbol{\theta}_2,\boldsymbol{\theta}_6,\boldsymbol{\theta}_7\}\rightarrow\boldsymbol{o}_1$;
- $\{\boldsymbol{\theta}_2,\boldsymbol{\theta}_3,\boldsymbol{\theta}_6,\boldsymbol{\theta}_8,\boldsymbol{\theta}_9\}\rightarrow\boldsymbol{o}_1$.

Agent 群根据上述机制针对产出 $\boldsymbol{o}_1$,可以提出两种方案:

- 方案 1:*DatabaseSearcher* 和 *TestPaperCreater* 合作完成功能 *CreateTestPaperWithDatabase*,

  InterfaceAgent 完成功能 *Test*;

  Evaluater 完成 *Evaluate*.
- 方案 2:*DatabaseSearcher* 和 *TestPaperCreater* 合作完成功能 *CreateTestPaperWithDatabase*,

  InterfaceAgen 完成功能 *Test*;

  DatabaseSearcher 和 *TestPaperCreater* 合作完成功能 *GetQuestionPoint*;

  Evaluater 完成 *JudgeWongOrRight*,*CalculateFinalPoint* 和 *GiveComment*.

两种方案对应的各 Agent 的收益向量为:

- 方案 1:(12,28,30,30);
- 方案 2:(13.5,29.5,30,27).

各个 Agent 对这两种方案的倾向是显然的,*DatabaseSearcher* 和 *TestPaperCreater* 倾向于方案 2,*Evaluater* 倾向于方案 1,*InterfaceAgent* 则两种均可.根据中心模型,可以计算出公平分配方案(12.75,28.75,30,28.5),方案 1 与公平方案的距离约为 1.84,方案 2 与公平方案的距离也约为 1.84,也就是说,两种方案无论选哪一种,在公平程度上都是一样的.所以,我们选择方案 1 作为最终方案,并将其表示为

{*DatabaseSearcher.OpenDatabase*,*DatabaseSearcher.ReadFromDatabase*,*TestPaperCreater.GetQuestions*,

TestPaperCreater.CreateTestPaper,InterfaceAgent.Test,Evaluater.Evaluate}.

下面根据上一节给出的 MAS 系统规格说明,将上述方案表示成一个 MAS 系统,如图 3 所示.

```
MAS OnlineTest{
[Resource List]
  QuestionDatabase:{...};
  QestionBuffer:{...};
  AnswerBuffer:{...};
  StandardAnswerBuffer:{...};
  CompareResultBuffer:{...};
  TestPointBuffer:{...};
  TestPaper:{...};
  KeyBoard:{...};
  Monitor:{...};
[Agent list]
  DatabaseSearcher:{
  OpenDatabase();
  ReadFromDatabase();
  };
  TestPaperCreater:{
  GetQuestions();
  CreateTestPaper();
  };
  InterfaceAgent:{
  DisplayTestPaper();
  GetAnswerFromUser();
  RecordAnswer();
  };
  Evaluater:{
  Evaluate();
  GetStandardAnswer();
  CompareAnswer();
  CalculatePoint();
  WritePointToPaper();
  WriteCommentToPaper();
  };
[Executable Sequence]
  DatabaseSearcher.OpenDatabase;
  DatabaseSearcher.ReadFromDatabase;
  TestPaperCreater.GetQuestions;
  TestPaperCreater.CreateTestPaper;
  InterfaceAgent.Test;
  Evaluater.Evaluate;
```

Fig.3 Specification of multi-Agent system *OnlineTest*

图 3 多 Agent 系统 *OnlineTest* 的规格说明

6 相关工作与比较

本文从主动服务、需求驱动、自主聚合的角度提出了一种主动网构软件实体的自动组合方法,其特点是:第一,实现需求驱动的网构实体动态组合;第二,借助功能本体更有效地产生需求解决方案.

目前,对网构软件实体自动组合的研究工作还不多,而 Web 服务组合方法主要有基于工作流和基于语义两个方面.基于工作流的服务组合,早期的有 IBM 的 WSFL[15]、BEA 的 WSCI[16],后来出现了 BPEL4WS,Eflow[17].其中最具代表性的是 BPEL4WS,它预先定义需求的业务过程,然后,根据过程中活动的 WSDL 描述选择参与的 Web 服务,最后按 BPEL4WS 定义的控制逻辑进行组合.但它预先定义的组合方式不能随着 Web 服务资源的更新变化而相应变化,所以只是一种静态的组合方式.与之相比,我们的方法不需要预先定义组合方式,最终的组合方案由参与的服务 Agent 自主协作产生,是一种由需求驱动的动态组合.如何使现有的基于 WSDL 描述的主动服务理解和应用功能本体并进行动态组合,还需进一步研究.基于语义的服务组合通常包括两部分:一部分为 Web 服务提供计算机可理解的语义信息,例如 OWL-S[18];另一部分在此基础上借助形式化方法实现服务组合.常见的用于 Web 服务组合的形式化方法有 AI 规则、Petri 网和进程代数.它们共同的特点是,首先用形式化方

法对目标建立组合模型,再根据此模型选择服务,是一种静态组合建模、动态绑定服务的模式,组合模式不能随着服务资源的变化而变化.而我们的方法虽然也需要对需求进行规格化表示,但是并不需要给出针对特定需求的组合模型.并且,随着参与的服务 Agent 的不同,可能产生的协作组合方案也是不同的,是一种真正的动态服务组合模型.

在基于 Agent 的网构软件模型[19]中,更多地将这种由构件或者服务实体组合而成的网构软件看成一种可人工设计的软件系统,在给定参与组合的软件实体后,通过第三方协同服务进行协作,并给出了面向体系结构的协同程序设计方法.显然,服务 Agent 间并没有真正的协商过程,Agent 协作过程完全由协同服务中的协同程序控制.这样做的好处是可以避免 Agent 自由协商中产生的不确定性,缺点是协同程序设计难度较大.而我们的模型则更多地将网构软件看成一个 MAS 系统.前述模型得到的软件如果从多 Agent 系统角度来看,则是一种具有中央控制的多 Agent 系统.我们的模型更强调 Agent 自主协商,不需要人工设计协同过程,而机制设计产生的机制也能够有效约束 Agent 行为避免出现对需求解决不利的情况.而功能本体更为需求解决提供了各种可能的解决方案,避免了人工设计,同时也不需要 Agent 具有问题求解的能力.

7 总结与展望

本文面向网构软件需要,着眼于广泛分布在 Internet 中的 Web 服务,基于多 Agent 理论和技术,提出了一种需求驱动的服务 Agent 协作模型.传统的服务发现和组合主要着眼于 Web 服务为被动实体的情况,本文从一个不同的角度来看面向服务的计算.我们假设 Web 服务为主动的服务 Agent,这些 Agent 可以主动寻找新发布的需求并参与到这些需求的解决过程中.当一个服务 Agent 无法满足整个需求时,多个服务 Agent 就必须协作来满足需求.这种协作可以被称为需求驱动的服务 Agent 协作.自主协商进行服务组合实现需求,可以大幅度降低服务组合的难度和复杂度,有利于推动面向服务的应用的发展.本文给出了一个初步的需求驱动的服务 Agent 协作模型,主要的创新点包括:

- 构建了一个功能本体.该本体提供了一种公共的术语用于需求的功能描述和服务 Agent 的能力描述.借此服务 Agent 可以理解需求描述,进而发现自己可参与的需求.
- 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.
- 给出了一个群体决策模型——重心模型,用于建立有效的服务 Agent 联盟以及形成有效的协作方案.
- 针对需求解决方案给出了一种多 Agent 系统规格化表示方法.

未来的工作主要包括两个方面:一方面是针对非功能性需求,给出合适的服务 Agent 组合;另一方面是解决非特定需求的服务 Agent 联盟形成问题.

致谢 在此,我们向参与本文内容讨论并提出宝贵建议的所有老师和同学表示衷心的感谢.

References:

[1] Yang FQ, Mei H, Lü J, Jin Z. Some discussion on the development of software technology. Acta Electronica Sinica, 2002,30(12A): 1901−1906 (in Chinese with English abstract).

[2] Kreger H. Web services conceptual architecture (WSCA 1.0). IBM Software Group Note. 2001. http://www-306.ibm.com/software/solutions/webservices-/pdf/WSCA.pdf

[3] SOAP version 1.2. 2004. http://www.w3.org/TR/soap/

[4] Christensen E, Curbera F, Meredith G, Weerawarana S. Web services description language (WSDL) 1.1. 2001. http://www.w3.org/TR/wsdl

[5] Weerawarana S. Business process with BPEL4WS: Understanding BPEL4WS. 2002. http://www-106.ibm.com/

[6] 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.

[7] Hou LS, Jin Z. FECT: A modelling framwork for automatically com-posing Web services. In: Advances in Web-Age Information Management. LNCS 3793, Heidelberg: Springer-Verlag, 2005. 320−332.

[8] Sycara K, Paolucci M. Ontologies in Agent architectures. In: Handbook on Ontologies in Information Systems. Berlin: Springer-Verlag, 2004. 343−364.

[9] Kitamura Y, Mizoguchi R. Functional ontology for functional understanding. In: Zhao F, Yip K, eds. Proc. of the 12th Int'l Workshop on Qualitative Reasoning (QR'98). Cape Cod: AAAI Press, 1998. 77−87.

[10] Sandholm T. Automated mechanism design: A new application area for search algorithms. In: Rossi F, ed. Proc. of the Int'l Conf. on Principles and Practice of Constraint Programming (CP). LNCS 2833, Heidelberg: Springer-Verlag, 2003. 19−36.

[11] Conitzer V, Sandholm T. Complexity of mechanism design. In: Darwiche A, Friedman N, eds. Proc. of the 18th Conf. on Uncertainty in Artificial Intelligence (UAI). San Fransisco: Morgan Kaufmann Publishers, 2002. 103−110.

[12] ILOG mathematical programming optimizers. 2007. http://www.ilog.com/products/cplex/

[13] Lucas WF, ed.; Wang GQ, *et al.*, Trans. Political and Related Models. New York: Springer-Verlag, 1983 (in Chinese).

[14] Computing Research Association. Cyberinfrastructure for education and learning for the future: A vision and research agenda. 2005. http://www.cra.org/reports/cyberinfrastructure.pdf

[15] Leymann F. Web services flow language. 2001. http://www-306.ibm.com/software/solutions/webservices/pdf/WSFL.pdf

[16] Web service choreography interface (WSCI) 1.0. 2002. http://www.w3.org/TR/wsci

[17] Casati F, Ilnicki S, Jin LJ, Krishnamoorthy V, Shan MC. Adaptive and dynamic service composition in eflow. In: Wangler B, Bergman L, eds. Proc. of the 12th Int'l Conf. on Advanced Information System Engineering. LNCS 1789, Heidelberg: Springer-Verlag, 2000. 13−31.

[18] OWL-S: Semantic markup for Web services. 2004. http://www.daml.org/services/owl-s/1.0/

[19] Lü J, Tao XP, Ma XX, Hu H, Xu F, Cao C. On Agent-based software model for internetware. Science in China (Series E), 2005, 35(12):1233−1253 (in Chinese with English abstract).

附中文参考文献:

[1] 杨芙清,梅宏,吕建,金芝.浅论软件技术发展.电子学报,2002,30(12A):1901−1906.

[13] Lucas WF,主编;王国秋,等,译.政治及有关模型.长沙:国防科技大学出版社,1996.

[19] 吕建,陶先平,马晓星,胡昊,徐锋,曹春.基于 Agent 的网构软件模型研究.中国科学(E 辑),2005,35(12):1233−1253.

郑丽伟(1979－),男,山西五台人,博士生,CCF 学生会员,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,CCF 高级会员,主要研究领域为软件需求工程,领域建模和基于知识的软件工程.