ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.19, No.5, May 2008, pp.1113−1124
DOI: 10.3724/SP.J.1001.2008.01113

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

基于Agent的网构软件构件模型及其实现*

常志明+, 毛新军, 齐治昌

(国防科学技术大学 计算机学院,湖南 长沙 410073)

Component Model and Its Implementation of Internetware Based on Agent

CHANG Zhi-Ming+, MAO Xin-Jun, QI Zhi-Chang

(School of Computer Science, National University of Defense Technology, Changsha, China 410073)

+ Corresponding author: E-mail: zmchang@nudt.edu.cn, http://www.nudt.edu.cn

Chang ZM, Mao XJ, Qi ZC. Component model and its implementation of Internetware based on Agent. *Journal of Software*, 2008,19(5):1113−1124. http://www.jos.org.cn/1000-9825/19/1113.htm

Abstract: As a novel software paradigm evolved by the Internet, Internetware is still faced with many challenges, such as expressing explicit environments, autonomous software entities, self-adaptive run mechanisms, etc. From component perspective, this paper presents EBDI (electronic business document exchange) architecture to describe the components which can autonomously plan themselves at runtime to handle variable environments, and uses dynamic binding relationship to illuminate the self-adaptive and evolutionary components. Throughout formal role model, run status, autonomous run and dynamic evolving run mechanisms for components of Internetware are addressed. As the platform of Internetware, DAgent-Internetware prototype can provide support for Internetware based on DAgent to design, develop, deploy, run and evolve.

Key words: Internetware; component; DAgent; EBDI (electronic business document exchange) architecture; dynamic binding relationship

摘 要: 网构软件代表了 Internet 环境下的一种新型的软件形态,但仍然面临着外部环境显式化、软件实体主体化、运行机制自适应等问题.从构件的角度出发,提出了 EBDI(electronic business document exchange)结构以表示能够根据环境变化实施自主行为的构件,利用动态绑定关系解释了构件的自适应演化特征.根据形式化的 Role 模型,描述了构件的运行状态、自主运行及自适应演化运行机制.开发了 DAgent-Internetware 原型作为网构软件的支撑平台,支持以 DAgent 为构件的网构软件从设计到实现、部署、运行、演化等一系列流程.

关键词: 网构软件;构件;DAgent;EBDI 结构;动态绑定关系

中图法分类号: TP311 **文献标识码**: A

近年来,随着计算机网络尤其是Internet的日益普及和广泛应用,越来越多的软件系统运行和部署在网络环

* Supported by the National Natural Science Foundation of China under Grant No.60773018 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.2007AA01Z135 (国家高技术研究发展计划(863)); the Postgraduate Innovation Fund of National University of Defense Technology of China under Grant No.070604 (国防科学技术大学研究生创新基金)

Received 2007-11-14; Accepted 2008-03-11

境上.软件形态逐渐由确定性目标转向动态性目标,由基于实体的驱动向着基于协同的驱动发展.软件系统越来越强调根据需求的变动不断地进行演化,动态地调整软件的拓扑结构和行为.如何在开放、动态、难控的Internet环境下实现各类资源的共享和集成以及复杂软件系统的开发,已成为计算机软件技术面临的一项挑战性研究课题[1].当前软件系统的复杂性远远超出现有软件工程抽象(如面向对象、面向构件等)的处理能力,并将作为一种重要的驱动力,促使软件工程甚至计算机科学范型的根本性变革[2].网构软件正是在这样的背景下提出来的.它是对开放、动态和难控网络环境下的分布式软件系统的一种抽象,包括一组分布于Internet环境下各个节点且具有主体化特征的软件实体,并能为这些软件实体提供交互和协同[1].网构软件代表了一类具有主体性、协同性、反应性、自适应性、自演化性、多态性等特征的复杂系统[3].

目前,人们从多个方面对网构软件技术进行了深入的研究,取得了一系列的研究成果,包括以软件体系结构为中心的网构软件开发方法[4]、网构软件模型[5]、网构软件的信任度量及演化模型[6]等等.然而,这些工作大多是从宏观的角度对网构软件进行分析和建模,很少从构件的角度支持具有自适应和自演化特征的网构软件系统的开发、部署和运行,因此,网构软件技术的研究仍然面临着许多挑战性的课题:(1) 如何表示网构软件中的构件模型,这种模型应该具有显式的环境描述和实体的自主性,实体可以主动地观察环境,作出合理的规划,以实现系统的设计目标;(2) 如何描述网构软件中构件的自适应演化能力,在不同的情景下展现出不同的行为,以适应环境的变化;(3) 如何提供相应的软件开发包和运行平台来支持网构软件系统的开发,并能有效支持与主流技术以及遗留系统之间的集成.下面分别对这 3 个问题进行阐述.

网构软件的实体具有相对独立性、主动性和自适应性,具有感知外部系统运行和使用环境的能力.针对第 1 个问题,我们认为网构软件的自适应和动态演化技术必须提供主体化(即内容的自包含性、结构的独立性与实体的适应性)程度更高的概念、模型和技术来支持网构软件中实体的建模、分析、实现和运行.利用 Agent 技术,网构软件可以解决对驻留环境进行感知、表示和分析的问题.我们将网构软件的构件实体抽象和物化为具有 EBDI(electronic business document exchange)结构的软件 Agent,这主要是因为软件 Agent 及其技术能够有效满足网构软件中构件主体化和环境显式化的要求.这样,构件就能够感知环境并且自主地实施规划以完成目标,从而可以自然地描述复杂系统的分布控制.

Agent理论研究自 20 世纪 90 年代以来非常活跃,并取得了许多研究成果,包括反应式结构、BDI逻辑和模型、基于情境演算的结构等.目前,已经出现了 20 余种Agent开发环境,具有代表性的有JADE,JACK等,并在工程实践中出现了很多基于Agent的应用系统[7].然而,我们注意到,现有软件Agent技术或者是借助于传统的人工智能技术,难以实现与主流技术之间的集成,或者是借鉴面向对象技术的思想,利用Agent类(Agent class)或者Agent类型(Agent type)的概念来抽象描述Agent的行为、作为生成软件Agent的模板[8],但这样的Agent在运行期间的行为规约是固定不变的,难以充分发挥面向Agent计算的灵活性和技术潜力.鉴于这些缺陷与第 2 个问题的需求,我们利用动态绑定关系[9]及其思想自然而直观地解释、描述和分析网构软件中构件的自适应和动态演化特征,并通过一种严格且形式化的方法将这一机制及其运行机理表述清楚.利用动态绑定关系,Agent能够根据感知到的环境变化,在其生命周期中动态地加入或者退出某些行为规约,并可以将行为规约置于活跃或者不活跃状态,从而在变化的环境中展示出不同的行为.

当前主流的对象技术在支持网构软件的构件开发方面存在一些不足.由于在面向对象技术中实体的被动性和依赖性,连接方式的单一性以及系统结构与目标的确定性,使得构件在自主行为和自适应演化方面存在诸多的局限性.而现有的软件中间件平台主要着重于开放环境的互连与互操作,缺乏对构件自适应演化的支持.针对第 3 个问题,我们在面向 Agent 开发平台的基础上作进一步的扩展和包装,基于 EBDI 结构和动态绑定关系,为构件提供了自主和自适应行为,从而支持网构软件的开发、部署和运行.

本文第 1 节提供网构软件中的构件模型,分别描述构件的 EBDI 结构、动态绑定关系和自适应演化行为.第 2 节描述构件的运行机制,分别定义构件的运行状态以及自主行为和动态演化行为的运行机制.在此基础上,第 3 节介绍支持构件开发和运行的平台原型,并通过网上交易系统的案例验证构件的自主性、动态性和自适应演化的特性.第 4 节比较相关工作.最后对本文进行总结,讨论进一步的研究方向.

1 基于 Agent 的构件模型

针对网构软件面临的问题,我们提出了一个基于 Agent 网构软件的构件模型,将 DAgent 作为网构软件中的构件形式,利用 EBDI 结构描述构件的自主行为,通过动态绑定关系体现构件的自适应演化行为.本节将从 EBDI 结构、动态绑定关系和自适应演化行为 3 个核心方面对该构件模型进行详细介绍.

DAgent(dynamic Agent)是一类特殊的 Agent,除了具有一般 Agent 的自主性以外,它还可以通过动态调整其内部的行为规约,在变化的环境中展现出自适应行为.因此,DAgent 是实施自主行为和自适应行为的基本单元,我们将 DAgent 作为网构软件中的构件形式.Role 内部采用 EBDI 结构,可以视为系统中对自主行为规约和实现的模块单元.EBDI 结构是对 BDI 结构的扩展,增加了环境的定义.Dagent 含有一个自适应策略 Strategy,表示 DAgent 在其生命周期内可以采取的一系列自适应行为规则,每个规则是对 DAgent 自适应演化的描述.这样,DAgent 实例通过感知外部的环境,并根据当前的运行状态,利用对 Role 的动态绑定操作调整自身的 EBDI 结构.该模型的特点及三者之间的关系如图 1 所示,其中 Strategy 与 Role 之间具有依赖关系,每条规则可能会使用若干 Role.DAgent 与 Role 具有动态绑定关系,这种关系不同于传统面向对象中的关联关系,DAgent 既可以视为 Role 的管理者,也可以视为 Role 的合成物.这种构件形式与传统的构件相比,具有环境显式化、自主性和自适应性的特点,其具体涵义如下:

- 环境显式化:Role 中的 EBDI 结构将环境进行显式定义.软件实体能够感知外部环境的变化,并根据自身的规划推理和自适应行为规则对环境变化作出响应.
- 自主性:DAgent 是对传统 BDI 结构 Agent 的扩展,因此能够自主地根据目标进行推理,产生规划并实施具体行为.
- 自适应性:由于 DAgent 与 Role 之间具有动态绑定关系,DAgent 实例在运行期间可以根据环境的变化按照自适应策略动态地绑定不同的 Role,从而改变自身的 EBDI 结构并展现不同的行为.

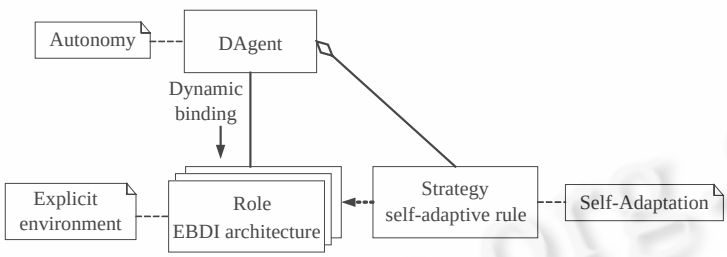


Fig.1 Component model based on Agent for Internetware

图 1 基于 Agent 的网构软件中的构件模型

1.1 EBDI结构

BDI结构在学术界和工业界都有很大的影响[10].BDI结构中包括信念、期望和意图这 3 个认知部件.信念定义了Agent对自身的理解和认知,是Agent进行动作决策的基础;期望反映了Agent可能试图实现的任务和目标,体现了Agent的某种意向;意图是对未来动作的合理选择,它将影响和约束Agent的行为决策和实施,是Agent动作的起因.

BDI 结构的行为决策方式和思想有助于理解、分析和描述 Agent 的自主行为.目前,已经出现了一些基于 BDI 结构的 Agent 开发环境,如 JADEX,JACK 等.BDI 结构通常将环境视为系统消息,将其封装在信念集合里,但在这种方式中环境的内容受到限制,而且没有将环境描述显式化,不利于表达 Agent 对环境的感知和影响,难以表示 Agent 根据环境变化进行自身演化.因此,我们对 BDI 结构进行扩展,提出了 EBDI 结构作为网构软件中构件的基础,其中 E 表示构件的环境集合,包括系统中其他运行实体(DAgent 实例)和资源等(为了统一表达方式,我们将资源封装成 DAgent,使运行实体之间的交互一致).环境 E 能够通过感知外部事件获取其他实体执行动作

和变化信息.环境是外部可见的,并影响实体的内部状态;同样,实体动作的执行也会对环境产生影响,并通过发出事件改变环境.Role 中的 EBDI 结构及各部分之间的关系如图 2 所示,其中 Deliberation 是决策部件(本文采用反应式推理,并将规划嵌入到意图中).

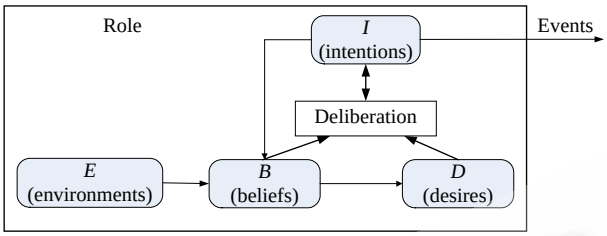


Fig.2 EBDI architecture of Role

图 2 Role 中的 EBDI 结构

在具体的实现中,我们利用Java语言中的事件机制表示环境E.在Agent开发平台中,我们用信念(belief)、目标(goal)和规划(plan)分别表示信念B、期望D和意图I,并利用对象表示信念和期望,用具体的函数表示意图.为了说明DAgent的自主运行和自适应演化,我们对Role中的EBDI结构进行抽象并提供了形式化的描述.对于Role,我们可以用四元组表示:Role $r=\langle E_r,B_r,D_r,I_r\rangle$,其中$E_r$表示环境集合、$B_r$信念集、$D_r$期望集、$I_r$意图集.我们用$L$表示一阶逻辑语言,$A$表示动作集合,具体表达如下:

- E_r表示该Role能够感知的环境集合,集合包含一组环境表达式,其定义为$\gamma::=E\exists agent\in Role.doAction(agent,\alpha)|E\forall agent\in Role.doAction(agent,\alpha)|\gamma\wedge\gamma'$,其中$\alpha\in A$,表示该Role所能观察到的系统中扮演Role的某个运行实体执行动作α、扮演Role的所有运行实体都执行动作α、多个事件发生.
- B_r表示该Role的信念集合,该集合包括一组信念表达式,其定义为$\beta::=B\phi|\neg\beta|\beta\wedge\beta'$,其中$\phi\in L$,$B\phi$表示有信念$\phi$.
- D_r表示该Role的期望集合,该集合包括一组期望表达式,其定义为$\kappa::=D\phi|\neg\kappa|\kappa\wedge\kappa'$,其中$\phi\in L$,$D\phi$表示有期望$\phi$.
- I_r表示该Role的意图集合,该集合包括一组实施意图表达式,其定义为$\pi::=\alpha|\beta?|\pi+\pi'|\pi;\pi'|\pi^*$,其中$\alpha\in$A,$\beta\in B_r$,$\beta?$表示验证实体是否具有信念$\beta$,$\pi+\pi'$表示不定向选择,$\pi;\pi'$表示顺序执行,$\pi^*$是循环执行.

目前,我们对EBDI结构中的环境描述比较简单,但在此基础上可以做一系列扩展,比如增加事件的组合模式等.具有EBDI结构的运行实体能够感知外部环境,环境变化可能会改变信念,信念的改变可能会导致新期望的产生.实体根据当前的信念、期望选择适当的意图加以执行,而在意图的实施过程中又可能会引起信念的变化,从而导致期望和意图再次发生变化.因此,在自主运行过程中各部分之间关系的形式化描述如下(其中$\gamma\in E_r$;$\beta,\beta'\in B_r;\kappa,\kappa'\in D_r;\pi,\pi'\in I_r$):

- $\gamma\wedge\beta\rightarrow\beta'$,环境改变信念;
- $\beta\wedge\kappa\rightarrow\kappa'$,信念改变目标;
- $\beta\wedge\kappa\wedge\pi\rightarrow\pi'$,根据信念、期望和意图产生新的意图;
- $\pi\wedge\beta\rightarrow\beta'$,意图的实施改变信念.

1.2 动态绑定关系

动态绑定关系在日常生活中非常普遍,类似于人们根据需要经常变换自身的角色.比如在一个网上交易系统中,Alex 需要购买书籍 Software Engineering,该书价格 25 元;Alex 有书 Artificial Intelligence,价格 10 元,而且 Alex 有现金 20 元.系统启动后,Alex 首先加入 Buyer,当他准备购买 Software Engineering 时发现自己的钱不够,Alex 钝化当前的 Buyer 角色,暂时停止购买.为了获得足够的现金,Alex 加入 Seller,打算卖掉自己的书 Artificial Intelligence,等待卖出并获得足够的现金后,再激活 Buyer 角色,重新购买 *Software Engineering*.在这个

场景中,Alex 从加入 Buyer 角色,到钝化 Buyer 并加入 Seller,再到激活 Buyer 的这些过程就体现了实体的自适应行为.

在本文中,动态绑定关系是指 DAgent 实例在其生命周期内可以动态地绑定或者释放某个 Role,所绑定的 Role 可以处于激活(active)状态也可以处于非激活(inactive)状态,一个 DAgent 实例在某一时刻可以同时绑定多个 Role.因此,通过动态绑定操作,DAgent 实例可以改变自身的 EBDI 结构.

图 3 描述了 DAgent 与 Role 之间的动态绑定关系,其中圆圈表示 DAgent 所绑定的 Role 的状态,有向箭头表示 Role 的操作.在运行过程中,DAgent 所绑定的 Role 具有以下两种不同状态:

(1) active 状态:DAgent 实例绑定某个 Role 期间的一种状态,DAgent 实例可以响应该 Role 的外部事件,有其信念和期望,根据 Role 的信念和期望选择合理的意图.

(2) inactive 状态:DAgent 实例绑定某个 Role 期间的一种状态,DAgent 实例无法响应该 Role 的外部事件,其信念和期望不会影响 DAgent 实例对意图中动作的选取.

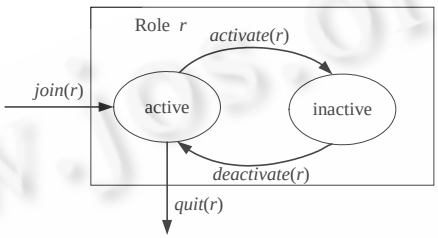


Fig.3 Dynamic binding relationship for DAgent and Role

图 3 DAgent 与 Role 的动态绑定关系

为了支持 DAgent 实例动态绑定 Role 以及 Role 状态的变迁,我们引入了 4 个基本操作:

- join:DAgent 实例对某个 Role 进行实例化,同时有该 Role 具体的外部事件、信念、期望和意图,并使得该 Role 处于 active 状态的操作.
- quit:DAgent 实例在某个 Role 处于 active 状态时,释放该 Role 的外部事件、信念、期望和意图的操作.
- activate:DAgent 实例使得某个 Role 由 inactive 状态变成 active 状态的操作.
- deactivate:DAgent 实例使得某个 Role 由 active 状态变成 inactive 状态的操作.

从图 3 中可以看出,一个 DAgent 实例也可以同时绑定多个 Role,而且一个 Role 可以同时被多个 DAgent 实例所绑定.由于被绑定的 Role 可能处于两种状态,若 DAgent 实例对某个 Role 同时绑定多次,则难以判断该 Role 所处的状态.这样,我们限定一个 DAgent 实例绑定某一个 Role 的同时,不能再次绑定该 Role.动态绑定关系给 DAgent 带来了更强的动态性,这可能会引起不一致.比如,在网上交易系统中,当 Seller 卖出的物品和 Buyer 买入的物品相同时,这两个 Role 的期望可能产生不一致,若某个 DAgent 实例同时绑定了 Seller 和 Buyer,就可能出现这样的矛盾.对于这种不一致性的定义,我们将在第 2.3 节中进行讨论.

1.3 自适应演化行为

自适应演化策略带有一组关于DAgent演化行为的规则,每个规则说明了DAgent在什么样的条件下应该执行哪些动态绑定操作.因此,具体语法定义为[**When** environment expressions] [**Where** belief expressions] [**If** desire expressions] **Then** *op**,其中op是 4 个动态绑定操作之一,其形式化描述为$\gamma\wedge\beta\wedge\kappa\ |\!\rightarrow\{join(r),quit(r),activate(r),deactivate(r)|r\in Role\};$*,其中$\gamma\in E_r,\beta\in B_r,\kappa\in D_r$.为了理解动态绑定关系和演化规则,我们通过网上交易系统加以说明.该系统包含两个Role:Buyer和Seller.DAgent实例在演化的过程中,可以根据自身需要对Role进行动态绑定操作.

- 加入和退出某个角色:Alex 当前是 Buyer 用户,他在买到某物品之后,又加入了 Seller 想将该物品卖掉,即 $B(isActive(Buyer))\wedge D(wantSell(goods))|\!\rightarrow quit(Buyer);join(Seller)$.
- 激活或钝化某个角色:Alex 需要购买某种物品,但钱数不够,打算卖掉自己的一些物品,等到赚够足够的

现金后,再继续购买该物品,即 $B(\neg isCanBuy(goods))\wedge B(hasOtherGoods(goods))|\rightarrow deactivate(Buyer)$; $join(Seller)$;$B(isCanBuy(goods))\wedge B(isActive(Seller))|\rightarrow quit(Seller)$;$activate(Buyer)$.

- 同时扮演多个角色:当 Alex 是 Buyer 时,若其他 Buyer 想购买 Alex 的物品,他还可以加入 Seller 卖掉该物品,即 $E(\exists agent\in Buyer.doAction(agent,CallForProposal(goods)))\wedge B(\neg bindRole(Seller))|\rightarrow join(Seller)$.

对于自适应演化规则,我们利用 Java 中的基本语句加以实现.我们为环境建立了一个 EventList 对外部事件进行保存,并利用 EventList 判断是否出现特定环境.对于信念和期望,我们将它们表示为布尔表达式,通过 If 语句加以判断.动态绑定操作则利用 Java 反射机制定位并实例化特定的 Role,并作为 DAgent 对外部提供的基本接口,其运行机制见第 2.3 节.

2 构件的运行机制

DAgent 的动态行为可分为两类:自主行为和自适应演化行为.本节首先定义了 DAgent 实例的运行状态,在此基础上说明了其自主运行机制,最后通过定义动态绑定关系的 4 个操作描述了自适应演化运行机制.

2.1 DAgent运行模型

定义 1. 对于任意DAgent a,令$Role(a)=Role^A(a)\cup Role^I(a)$,表示DAgent a绑定的Role集合,其中$Role^A(a)$表示DAgent a所绑定的并处于active状态下的所有Role,$Role^I(a)$表示DAgent a所绑定的并处于inactive状态下的所有Role.

由于我们规定DAgent实例在绑定某个Role的同时不能再次绑定该Role,则有$Role^A(a)\cap Role^I(a)=\varnothing$.

定义 2. 对于任意DAgent a,令$s_a=\langle \Gamma_a,\Omega_a\rangle$,表示DAgent a的状态,其中:

- $\Gamma_a=\langle E_a^A,B_a^A,D_a^A,I_a^A\rangle$表示DAgent a所绑定的并处于active状态下的所有Role的状态,其中$E_a^A=\cup\{E_r|r\in Role^A(a)\}$,$B_a^A=\cup\{B_r|r\in Role^A(a)\}$,$D_a^A=\cup\{D_r|r\in Role^A(a)\}$,$I_a^A=\cup\{I_r|r\in Role^A(a)\}$分别表示DAgent a所绑定的并处于 active 状态下的所有 Role 的环境、信念、期望和意图集合.
- $\Omega_a=\langle E_a^I,B_a^I,D_a^I,I_a^I\rangle$表示DAgent a所绑定的并处于inactive状态下的所有Role的状态,其中$E_a^I=\cup\{E_r|r\in Role^I(a)\}$,$B_a^I=\cup\{B_r|r\in Role^I(a)\}$,$D_a^I=\cup\{D_r|r\in Role^I(a)\}$,$I_a^I=\cup\{I_r|r\in Role^I(a)\}$分别表示DAgent a所绑定的并处于 inactive 状态下的所有 Role 的环境、信念、期望和意图集合.

令$S_a=\cup\{s_a\}\cup\varnothing$,表示DAgent a的所有状态集合.

2.2 自主运行机制

目前,已有一些工作对 BDI 结构的自主规划提出了一些算法和实现,我们结合 EBDI 结构、动态绑定关系对该算法作进一步扩展.

该算法中涉及一些函数,这些函数与第 1.1 节中的 Role 模型中的关系相对应,具体包括:信念修改函数 Brf 根据实体当前的信念、事件或者动作更改实体的信念,定义为 $Brf:\wp(B)\times(Event\uplus A)\rightarrow\wp(B)$,其中 $Event$ 和 A 分别表示事件集合和动作集合;期望修改函数 Dgf 根据实体当前信念和期望产生新期望,定义为 $Dgf:\wp(B)\times\wp(D)\rightarrow\wp(D)$;意图产生函数 Igf 根据实体当前信念、期望和意图产生新的意图,定义为 $Igf:\wp(B)\times\wp(D)\times\wp(I)\rightarrow\wp(I)$;动作选择函数 Choose 根据实体当前意图生成具体动作序列,定义为 $Choose:\wp(I)\times\wp(A)\rightarrow A^*$.该算法的基本流程如图 4 所示,其中 DAgent a 运行涉及到的环境、信念、期望和意图都是处于 active 状态,具体过程为:

第 3 行:算法首先进入一个永久循环;

第 4 行:在该循环内部,DAgent 实例感知外部环境事件;

第 5 行:根据外部事件,利用 Brf 函数修改其信念;

第 6 行:根据信念,利用 Dgf 函数修改其期望;

第 7 行:通过当前信念、期望和意图,利用 Igf 函数为实体选择合适的意图;

第 8 行:根据当前意图,利用 Choose 函数生成具体实施的动作序列 II;

第 9 行:算法进入执行动作序列的循环,该循环的终止条件是:或者 DAgent 实例已经执行完动作序列,或者

该意图已经成功实现,或者该意图已经不可能实现;

第 10 行:从 *II* 中选择第 1 个动作;

第 11 行:去掉第 1 个动作后得到的动作序列;

第 12 行:执行 *II* 中的当前动作,动作的执行可能会向外部环境发出相应的事件;

第 13 行:动作的执行可能会改变实体的信念,并通过 Brf 函数加以表示;

第 14~16 行:与第 6~8 行类似,如果需要,则修改后的信念再次对期望、意图产生影响,进而改变其动作序列.

```
1   Function AutonomousRun
2   Begin
3       while true do
4           E:=Sensor(E_a^A)
5           B_a^A:=Brf(B_a^A,E)
6           D_a^A:=Dgf(B_a^A,D_a^A)
7           I:=Igf(B_a^A,D_a^A,I_a^A)
8           II:=Choose(I,A)
9           while not (Empty(II) or Succeed(I) or Impossible(I))
10              action←head(II)
11              II←tail(II)
12              execute(action)
13              B_a^A:=Brf(B_a^A,action)
14              D_a^A:=Dgf(B_a^A,D_a^A)
15              I:Igf(B_a^A,D_a^A,I_a^A)
16              II:=Choose(I,A)
17          end-while
18      end-while
19  End-Function AutonomousRun
```

Fig.4 The algorithm of autonomous run of DAgent

图 4 DAgent 的自主运行算法

该算法充分利用已实现的 BDI 算法,其中增加了事件服务以表示环境,将信念、期望和意图分别映射为信念、目标和规划,并将它们分为 active 状态和 inactive 状态.该自主运行算法仅针对 active 状态下的事件、信念、目标和规划进行操作.在该算法的第 17 行后面,可以加入自适应策略的执行函数,其动态演化运行机制见第 2.3 节.

2.3 动态演化运行机制

本节将给出 Role 一致性、协调性的定义,并在此基础上形式化定义 DAgent 动态绑定的 4 个操作.

定义 3. 对于Role $r=\langle E_r,B_r,D_r,I_r\rangle$,如果满足下列条件,则称$r$是一致的,否则称为不一致:

- $\wedge\beta\neq\bot$,其中$\beta\in B_r$; //一致的信念集
- $\wedge\kappa|\neq\bot$,其中$\kappa\in D_r$; //一致的期望集
- 对于$\forall\kappa\in D_r$,有$|\neq\kappa$,且$\wedge\beta|\neq\kappa$; //由信念到目标的过程中必须选择意图
- $\wedge\kappa'\cup_{\pi\wedge\kappa\rightarrow\kappa'}|\neq\bot$. //潜在的期望是一致的

定义 4. 对于Role $r=\langle E_r,B_r,D_r,I_r\rangle$和$r'=\langle E_{r'},B_{r'},D_{r'},I_{r'}\rangle$,如果$\langle E_r\cup E_{r'},B_r\cup B_{r'},D_r\cup D_{r'},I_r\cup I_{r'}\rangle$是一致的,则称$r$和$r'$是协调的.

对于集合 $R\subseteq Role$,如果 R 中的任意 r 和 r 都是协调的,则称 R 是协调的.

定义 5. 对于DAgent a和Role r,$s_a=\langle\Gamma_a,\Omega_a\rangle\in S_a$,如果$r\notin Role(a)$且$Role^A(a)\cup\{r\}$是协调的,则join操作对应于函数$F_{join}:S_a\times Role\rightarrow S_a$,即$F_{join}(\langle\Gamma_a,\Omega_a\rangle,r)=\langle\Gamma'_a,\Omega_a\rangle$,其中$\Gamma'_a=\langle E_a^A\cup E_r,B_a^A\cup B_r,D_a^A\cup D_r,I_a^A\cup I_r\rangle$.

F_{join}需要满足的前置条件是:DAgent a所绑定的并处于active状态的Role集合与$\{r\}$的并集必须是协调的.在*join*操作后,DAgent a状态中的Γ_a分别增加了r实例化后的环境、信念、期望和意图集合.根据定义 1 可以得

到:$Role(a)=Role(a)\cup\{r\}$,$Role^A(a)=Role^A(a)\cup\{r\}$.

定义 6. 对于DAgent $a,s_a=\langle\Gamma_a,\Omega_a\rangle\in S_a$,如果$r\in Role^A(a)$,则有quit操作对应于函数$F_{quit}:S_a\times Role\to S_a$,即$F_{quit}(\langle\Gamma_a,\Omega_a\rangle,r)=\langle\Gamma'_a,\Omega_a\rangle$,其中$\Gamma'_a=\langle E_a^A/E_r,B_a^A/B_r,D_a^A/D_r,I_a^A/I_r\rangle$.

F_{quit}需要满足的前置条件是:DAgent a待退出的Role r必须处于active状态.在quit操作后,DAgent a状态中的Γ_a分别去掉了r环境、信念、期望和意图集合,并有$Role(a)=Role(a)/\{r\}$,$Role^A(a)=Role^A(a)/\{r\}$.

定义 7. 对于DAgent $a,s_a=\langle\Gamma_a,\Omega_a\rangle\in S_a$,如果满足$r\in Role^I(a)$并且$Role^A(a)\cup\{r\}$是协调的,则有activate操作对应于函数$F_{activate}:S_a\times Role\to S_a$,即$F_{activate}(\langle\Gamma_a,\Omega_a\rangle,r)=\langle\Gamma'_a,\Omega'_a\rangle$,其中,

$$\Gamma'_a=\langle E_a^A\cup E_r,B_a^A\cup B_r,D_a^A\cup D_r,I_a^A\cup I_r\rangle,\ \Omega'_a=\langle E_a^I/E_r,B_a^I/B_r,D_a^I/D_r,I_a^I/I_r\rangle.$$

$F_{activate}$需要满足的前置条件是:待激活的Role r必须处于inactive状态,且a所绑定的并处于active状态的Role集合与$\{r\}$的并集必须是协调的.在activate操作后,$Role^A(a)=Role^A(a)\cup\{r\}$,$Role^I(a)=Role^I(a)/\{r\}$.

定义 8. 对于DAgent $a,s_a=\langle\Gamma_a,\Omega_a\rangle\in S_a$,如果满足$r\in Role^A(a)$,则有deactivate操作对应于函数$F_{deactivate}:S_a\times Role\to S_a$,即$F_{deactivate}(\langle\Gamma_a,\Omega_a\rangle,r)=\langle\Gamma'_a,\Omega'_a\rangle$,其中,

$$\Gamma'_a=\langle E_a^A/E_r,B_a^A/B_r,D_a^A/D_r,I_a^A/I_r\rangle,\ \Omega'_a=\langle E_a^I\cup E_r,B_a^I\cup B_r,D_a^I\cup D_r,I_a^I\cup I_r\rangle.$$

$F_{deactivate}$需要满足的前置条件是:DAgent a待钝化的Role r必须处于active状态.在deactivate操作后,$Role^A(a)=Role^A(a)/\{r\}$,$Role^I(a)=Role^I(a)\cup\{r\}$.

3 实现平台原型及应用案例

3.1 支撑平台原型DAgent-Internetware

本文提出的构件模型DAgent已经在以Agent为主要计算单元的开发平台上进行了实现,能够与符合FIPA标准相容的多Agent系统相集成,实现了DAgent的自主行为和自适应演化行为.为了支持网构软件开发,我们开发了支撑平台原型DAgent-Internetware.DAgent-Internetware基于工业界和学术界广泛采用的JADE[11]和Spring平台,支持Dagent,Role和自适应策略Strategy的设计、开发和组装,并提供图形化界面、框架代码生成、DAgent的管理和监控等工具集以及设计运行时刻的接口API.该平台的设计框架如图 5 所示.

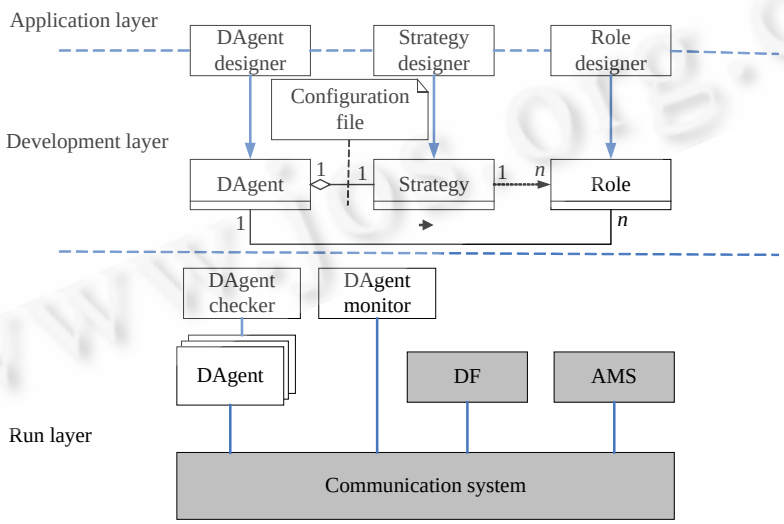


Fig.5 Framework of DAgent-Internetware

图 5 DAgent-Internetware 平台的框架

目前,DAgent-Internetware 平台原型总体分为两个层次:运行层和开发层.在运行层中,Communication System,DF 和 AMS 都是 JADE 平台重要的部件,分别负责 Agent 之间的通信、目录管理和 Agent 生命周期的管

理.在此基础上,我们开发的 DAgent 可以在平台上运行并与 FIPA 标准相容,可以与其他部件进行交互.此外,我们还开发了 DAgent Monitor,以监控和修改平台中所有 DAgent 实例的状态,DAgent Checker 用于检测 DAgent 实例在运行期间的信念和期望是否产生冲突.在开发层中,核心部分包括 DAgent,Strategy 和 Role. DAgent 实现了动态绑定关系的 4 个操作,并且通过 XML 配置文件指定了 DAgent 当前所使用的自适应演化策略.这样,在 DAgent 实例运行的过程中可以通过改变 XML 配置文件动态地更换自适应演化策略.平台为 Dagent,Strategy 和 Role 提供了基本的接口,用户可以根据应用需求进行扩展.我们还为这 3 个核心构件的设计提供了图形化的用户界面,用户可以通过这些图形界面设计具体的应用.

3.2 应用案例

为了展示 DAgent-Internetware 平台对自适应行为的支持,我们在该平台上开发了一个简单的网上交易系统的实例 TradingSys,其主要功能是模拟用户在网上交易的具体行为.我们首先开发了两个 Role:Buyer 和 Seller,用以实现基本的业务功能,具体的意图为:Buyer 向系统中所有的 Seller 发出购买请求,并等待回复,如果有回复,则与该 Seller 进行协商购买事宜;Seller 首先向 DF 进行注册,等待 Buyer 实例的购买请求,收到请求后,分析买卖的物品是否匹配,向 Buyer 发出货物的价格信息,并准备与 Buyer 进行协商.

为了说明实体运行过程中的动态演化,我们设计了 4 种不同的 Strategy:JoinSellerStrategy,JoinBuyerStrategy,TradingSellStrategy 和 TradingCreditStrategy,分别表示直接加入 Seller、直接加入 Buyer、在现金不足时卖掉一些货物、在现金不足时进行贷款.最后设计了 TradingDAgent,它只有一些基本的用于交易的属性和初始的演化策略.这样,TradingDAgent 的初始化实例仅仅是一个“空壳”(没有绑定任何 Role),需要根据其演化策略在运行的过程中对 Role 进行动态绑定操作,进而实施自主行为.我们设计了一个场景,实例化了 3 个 TradingDAgent 实例进行交易,其信息见表 1,其中×表示任意值.

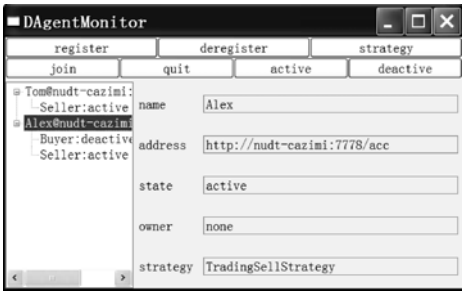
Table 1 TradingDAgent instances in TradingSys

表 1 TradingSys 案例中的 TradingDAgent 实例

Instance	Strategy	Goods owned	Goods to buy	Price	Number	Cash
Tom	JoinSellerStrategy	Software Engineering	×	25	2	5
Bill	JoinBuyerStrategy	×	Artificial Intelligence	×	×	50
Alex	TradingSellStrategy	Artificial Intelligence	Software Engineering	10	1	20

在这个场景中,Alex 首先绑定了 Buyer,准备(向 Tom)购买 Software Engineering.但交易过程中 Alex 发现现金不足,决定卖掉 Artificial Intelligence,因此钝化了 Buyer,绑定了 Seller,其 DAgent Monitor 如图 6(a)所示.当 Artificial Intelligence 出售之后(卖给 Bill),发现现金已经充足,则释放了 Seller,重新激活了 Buyer,其交易信息如图 6(b)所示.我们也可以在 Alex 运行的过程中更改其演化策略,比如在 Alex 发现自己现金不足时,将其策略更换为 TradingCreditStrategy,Alex 将采取贷款的方式重新进行交易,其交易信息如图 6(c)所示.

对于这个案例,我们在单机上启动多个容器进行模拟测试.其主要配置如下:CPU 为 Pentium 2.40GHz,内存为 512M,操作系统是 Windows XP Service Pack 2.运行结果为:Alex 在绑定 Buyer 并发出购买 Software Engineering 的请求的执行时间是 63ms,整个购买的交互时间为 80ms;Alex 在执行自适应演化规则(TradingSellStrategy 策略中的定义)中的最大时间为 15ms,其中包括了环境感知和动态绑定操作的时间,而动态绑定操作的时间几乎可以忽略.如果在分布式环境下测试,DAgent 实例之间的交互时间则会更长,因此我们可以看出，增加的动态绑定操作时间并不会给系统带来负担.



(a) When Alex has not enough money

(a) 当 Alex 的现金不足时

```
信息: Pre-instantiating singletons in factory [org.sp
Alex: join the role: Buyer
Alex: Found the following seller agents:
        Tom
Alex: select Tom
Alex: SoftwareEngineering sending cfp
Tom: SoftwareEngineering sending ready
Alex: I can't buy, I shall sell something ...
Alex: deactivate the role: Buyer
Alex: join the role: Seller
Bill: join the role: Buyer
Bill: Found the following seller agents:
        Alex
        Tom
Bill: select Alex
Bill: select Tom
Alex: ArtificialIntelligence sending ready
Bill: ArtificialIntelligence sending cfp
Bill: will buy
Alex: ArtificialIntelligence has been brought
Alex: quit the role: Seller
Alex: activate the role: Buyer
Alex: Found the following seller agents:
        Tom
Alex: select Tom
Alex: SoftwareEngineering sending cfp
Tom: SoftwareEngineering sending ready
Alex: will buy
Tom: SoftwareEngineering has been brought
```

(b) Trading Information by TradingSellStrategy

(b) 使用 TradingSellStrategy 策略的交易信息

```
Alex: SoftwareEngineering sending cfp
Tom: SoftwareEngineering sending ready
Alex: I can't buy, I shall sell something ...
Alex: deactivate the role: Buyer
Alex: join the role: Seller
2008-2-26 22:43:03 org.springframework.beans.factory.xml.XmlB
信息: Loading XML bean definitions from file [D:\DAgent\eclips
2008-2-26 22:43:03 org.springframework.context.support.Abstra
信息: Bean factory for application context [org.springframewor
2008-2-26 22:43:03 org.springframework.context.support.Abstra
信息: 1 beans defined in application context [org.springframew
2008-2-26 22:43:03 org.springframework.beans.factory.support.
信息: Pre-instantiating singletons in factory [org.springframe
Alex: I will change my strategy ...
Alex: join the role: Buyer
Alex: Found the following seller agents:
        Tom
Alex: select Tom
Alex: SoftwareEngineering sending cfp
Tom: SoftwareEngineering sending ready
Alex: I have not enough money, I shall creidt 10 dollors...
Alex: Found the following seller agents:
        Tom
Alex: select Tom
Alex: SoftwareEngineering sending cfp
Tom: SoftwareEngineering sending ready
Alex: will buy
Tom: SoftwareEngineering has been brought
```

(c) Trading Information by TradingCreditStrategy

(c) 使用 TradingCreditStrategy 策略的交易信息

Fig.6 The runtime information of TradingSys

图 6 TradingSys 的运行信息

4 相关工作

我们从网构软件的构件理论与实现、网构软件的开发方法、动态 Agent 理论及实现 3 个方面的相关工作与本文进行比较.

在网构软件的构件理论与实现方面:由于现有的构件技术难以表示实体的自主行为,文献[4]提出了自主构件的理论,自主构件保留了传统构件的属性和特征,并具有 Agent 的自主行为能力.自主构件具有构件主动化的特点,即自主构件的行为是目标驱动的,自主构件也向外提供服务,而且提供服务不再是被动的,它甚至可以自主地决定是否提供服务.此外,自主构件能够通过感知环境的变化,在行为上进行调整.总体上看,DAgent 模型与自主构件比较相似,但 DAgent 模型可以为构件提供主体化程度更高的支持,采用的动态绑定关系在构件的动态演化和自适应方面的粒度更大,可以更自然地对现实世界的问题加以描述.

在网构软件的开发方法方面:文献[5]基于移动 Agent 的原理、方法和技术,在面向对象方法与技术的基础上,从基础软件模型、关键技术支撑、主流技术应用 3 个层面提出了适合网构软件需求的开放协同模型.文献[4]以软件构件为基本实体,以软件体系结构为中心,以软件中间件为运行支撑的软件开发方法学 ABC 用以支持网构软件的开发.该方法反映了网构软件在问题空间实现自底向上、从“无序”到“有序”的构造过程.基于自适应软

件体系结构的分析与设计方法,支持自适应网构软件的开发.这些工作是从宏观的角度提出网构软件的开发方法,并从构件的角度对网构软件开发进行探讨.而本文则是从微观的角度解决了网构软件构件的环境显式化、实体主体化、运行机制自适应的问题,为网构软件的开发方法提供了一种新的解决方式.

在动态Agent理论与实现方面:目前许多方法学认为多Agent系统中的构件是封装了一些功能和行为的Agent Class,但是对构件的理解仍然局限于面向对象技术,Agent Class与Agent之间具有实例化关系,但这种实例化的关系在Agent运行期间不能更改自身的角色,难以满足复杂系统对动态性的要求,不利于开发出具有动态行为特征的多Agent系统[4].文献[12,13]为Agent Type提出了一些形式化的描述,提出了Agent对角色的操作:enact,deact,activate和deactivate这 4 个操作,但是它们限定了Agent在运行期间同时只能有 1 种角色处于激活(active)状态,这不能很好地解释现实世界中的很多问题(比如同时扮演多个角色),而且没有给出具体的实现.文献[14]基于Caste提出了形式化规约语言SLABS,可以为Agent在运行期间动态地join或者quit某个Caste进行建模和规约,同时在SLABSp程序设计语言中实现了这两个操作[15].但是该方法没有为join和quit操作定义良好的操作语义,其动态演化策略定义在Caste内部,因此Agent在运行期间难以动态地改变演化策略.文献[9,16]提出了Caste与Agent之间的动态绑定关系,为这种关系提出了4个基本操作:join,quit,activate和inactivate,并给出Agent运行模型,形式化定义了操作语义.与本文工作相比,Caste模型没有与主流的Agent技术和面向构件的技术相结合,没有将Agent的动态演化行为从规约中分离出来,也没有给出具体的实现.

5 总结和进一步工作

随着Internet的快速发展与普及,如何在开放、动态、难控的网络环境下实现各类资源的共享和集成已经成为计算机软件技术面临的重要挑战之一.为了应对挑战,网构软件的概念、方法和技术应运而生.但是,网构软件仍然面临着环境显式化、实体主体化、运行适应性等问题[1].针对这些问题,本文从构件的角度出发,将DAgent作为网构软件中的构件形式;提出了EBDI结构以表示能够根据环境变化实施自主行为的构件,利用动态绑定关系解释了构件的自适应和动态演化特征,并形式化定义了构件实体的动态运行机制.为了在实际的软件系统中应用上述技术,开发了DAgent-Internetware作为网构软件的支撑平台原型.该系统支持以DAgent为构件的网构软件从设计到实现、部署、运行、演化等一系列流程.

目前,在我们的工作中仍然有许多问题有待解决,其中包括:

- 在平台层面上,自适应策略描述语言的语法设计,解释器的实现目前还处于研究阶段.我们希望自适应策略描述语言能够提供丰富的表达能力,兼具较为简单的语法.由于现有 Agent 平台对主流构件技术的支持较少,我们仍然需要结合现有的 Agent 技术与构件技术来完善 DAgent-Internetware 支撑平台.
- 在方法层面上,需要提供较为完整的方法学以支持基于DAgent的网构软件开发;并结合软件体系结构技术,对网构软件系统的整体演化行为提供支持.具体的网构软件系统可能会有很多约束条件,需要保证系统在演化过程中的一致性和完整性.我们还将以 Bigraph 理论作为网构软件的全局演化提供支持.
- 在应用层面上,希望能够开发更多的应用案例,并在较大规模的网络环境下对以 DAgent 为构件的网构软件进行检验.

References:

[1] Lü J, Ma XX, Tao XP, Xu F, Hu H. Research and progress on Internetware. Science in China (Series E), 2006,36(10):1037−1080 (in Chinese with English abstract).

[2] Zambonelli F, van Dyke Parunak H. Towards a paradigm change in computer science and software engineering: A synthesis. The Knowledge Engineering Review, 2003,18(4):329−342.

[3] 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

[4] Mei H, Huang K, Zhao HY, Jiao WP. A software architecture centric engineering approach for internetware. Science in China (Series E), 2006,36(10):1100−1126 (in Chinese with English abstract).

[5] Lü J, Tao XP, Ma XY, 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).

[6] Wang Y, Lü J, Xu F, Zhang L. A trust measurement and evolution model for Internetware. Journal of Software, 2006,17(4): 682−290 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/17/682.htm

[7] Mao XJ, Chang ZM, Wang J, Wang HM. Agent oriented software engineering: State and challenge. Journal of Computer Research and Development, 2006,43(10):1782−1789 (in Chinese with English abstract).

[8] Dam KH, Winikoff M. Comparing Agent-oriented methodologies. In: Giorgini P, Henderson-Sellers B, Winikoff M, eds. Proc. of Agent-Oriented Information Systems. LNCS 3030, Berlin/Heidelberg: Springer-Verlag, 2004. 78−93.

[9] Mao XJ, Chang ZM, Shan LJ, Zhu H, Wang J. The dynamic casteship mechanism for modeling and designing adaptive Agents. In: Zhang K, Spanoudakis G, Visaggio G, eds. The 2nd Int'l Workshop on Agent-Oriented Software Development Methodologies (AOSDM 2006). 2006. 639−644.

[10] Cohen P, Levesque H. Intention is choice with commitment. Artificial Intelligence, 1990,42(2-3):213−261.

[11] Bellifemine F, Poggi A, Rimassa G. JADE-A FIPA-compliant Agent framework. In: Proc. of the 4th Int'l Conf. on the Practical Applications of Agents and Multi-Agent Systems (PAAM'99). London: The Practical Application Company, Ltd., 1999. 97−108.

[12] Dastani M, van Riemsdijk MB, Hulstijn J, Dignum F, Meyer JJC. Enacting and deacting roles in Agent programming. In: Odell J, Giordini P, Müller JP, eds. Proc. of the Agent-Oriented Software Engineering V. LNCS 3382, Berlin/Heidelberg: Springer-Verlag, 2005. 189−204.

[13] Dastani M, Dignum V, Dignum F. Role-Assignment in open Agent societies. In: Proc. of the 2nd Int'l Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2003). New York: ACM Press, 2003. 489−496.

[14] Zhu H. A formal specification language for Agent-oriented software engineering. In: Proc. of the 2nd Int'l Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2003). New York: ACM Press, 2003. 1174−1175.

[15] Wang J, Shen R, Zhu H. Towards an Agent oriented programming language with caste and scenario mechanisms. In: Dignum F, Dignum V, Koenig S, Kraus S, Singh MP, Wooldridge M, eds. Proc. of Int'l Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2005). New York: ACM Press, 2005. 1297−1298.

[16] Chang ZM, Mao XJ, Wang J, Qi ZC. Dynamic binding mechanism and its operational semantics of component in multi-Agent system. Journal of Computer Research and Development, 2007,44(5):806−814 (in Chinese with English abstract).

附中文参考文献:

[1] 吕建,马晓星,陶先平,徐锋,胡昊.网构软件的研究与进展.中国科学(E 辑),2006,36(10):1037−1080.

[3] 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

[4] 梅宏,黄罡,赵海燕,焦文品.一种以软件体系结构为中心的网构软件开发方法.中国科学(E 辑),2006,36(10):1100−1126.

[5] 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.

[6] 王远,吕建,徐锋,张林.一个适用于网构软件的信任度量及演化模型.软件学报,2006,17(4):682−290. http://www.jos.org.cn/1000-9825/17/682.htm

[7] 毛新军,常志明,王戟,王怀民.面向 Agent 的软件工程:现状和挑战.计算机研究与发展,2006,43(10):1782−1789.

[16] 常志明,毛新军,王戟,齐治昌.多 Agent 系统中软构件的动态绑定机制及其操作语义.计算机研究与发展,2007,43(10): 1782−1789.

常志明(1979－),男,辽宁沈阳人,博士生,主要研究领域为软件体系结构.

齐治昌(1942－),男,教授,博士生导师,主要研究领域为软件工程.

毛新军(1970－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为 Agent 理论和技术,软件工程.