软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2012,23(11):2923−2936 [doi: 10.3724/SP.J.1001.2012.04298]
©中国科学院软件研究所版权所有.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

基于组织的面向 Agent 程序设计及其语言 Oragent*

胡翠云+, 毛新军, 陈 寅

(国防科学技术大学 计算机学院,湖南 长沙 410073)

Organization-Based Agent-Oriented Programming and Language Oragent

HU Cui-Yun+, MAO Xin-Jun, CNEN Yin

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

+ Corresponding author: E-mail: hcy56316@163.com

Hu CY, Mao XJ, Chen Y. Organization-Based agent-oriented programming and language Oragent. *Journal of Software*, 2012,23(11):2923−2936 (in Chinese). http://www.jos.org.cn/1000-9825/4298.htm

Abstract: In the construction of dynamic and open multi-agent systems, several issues in existing agent-oriented programming should be solved including a lack of high-level abstraction, a great gap between the implementation and design models, insufficient execution mechanism and programming constructs to support dynamics. To deal with these issues, this paper proposes an organization-based agent-oriented programming approach, which takes organizations, groups, roles, and agents as first-class entities to narrow the gap between implementation and design models. Moreover, this approach introduces serveral organization mechanisms, i.e. role enactment mechanism and role-based interactions, to support the dynamics such as the dynamic composition of the agents' behaviors and dynamic interactions among agents. Based on the above ideas, an organization-based agent-oriented programming language, Oragent, is designed by defining its abstract syntax and formal operational semantics. Finally, a case is studied to show how to construct dynamic and flexible multi-agent systems with the programming approach and Oragent language.
Key words: organization; role; agent-oriented programming; dynamics; role-based interaction; role enactmen

摘 要: 当前,面向 Agent 程序设计在支持动态开放多 Agent 系统开发方面存在一系列的不足,如缺乏高层抽象、底层实现模型与高层设计模型相脱节、在支持系统动态性方面缺乏有效的运行机制和语言设施等.针对这些问题,提出一种基于组织的面向 Agent 程序设计方法.该方法将组织、Group、角色和 Agent 等高层抽象作为一阶实体,缩小了多 Agent 系统的设计模型与实现模型之间的概念鸿沟;借助于组织学中的机制——角色扮演机制、基于角色的交互——支持系统动态性的规约和实现,如 Agent 行为的动态组合、动态的交互等,基于该程序设计思想,设计了基于组织的面向 Agent 程序设计语言——Oragent,定义了其抽象语法和形式语义,并通过案例分析说明了如何基于该程序设计思想和 Oragent 语言来构造和实现动态而灵活的多 Agent 系统.
关键词: 组织;角色;面向 Agent 程序设计;动态性;基于角色的交互;角色扮演
中图法分类号: TP18 **文献标识码**: A

---

* 基金项目: 国家自然科学基金(61070034, 90818028, 91024030); 教育部博士点基金(20094307110007); 新世纪优秀人才支持计划; 浙江师范大学计算机软件与理论省级重中之重学科重点基金(ZSDZZZZXK33)
收稿时间: 2012-06-08; 定稿时间: 2012-08-15

© 中国科学院软件研究所 http://www.jos.org.cn

随着计算机和网络技术的飞速发展和广泛应用,越来越多的软件以 Internet 为运行环境(如电子商务系统、信息管理系统、应急管理系统等).为了适应 Internet 的开放性、动态性以及不可控等特点,运行在 Internet 环境下的软件系统呈现出新的特点[1,2]:(1) 自主的软件实体.软件系统由来自不同组织的自主的软件实体构成,软件实体可以动态地加入或退出系统,并可以自主地决定提供服务与否.(2) 不确定的交互.由于 Internet 的开放性,软件实体交互的对象和方式无法在设计时获知,只有在运行时才能确定.(3) 灵活的系统结构.为了适应动态多变的运行时环境,软件系统的结构不是一成不变的,而是随着时间和环境的变化而不断调整的.如何构造运行在 Internet 上动态开放的软件系统,对传统的软件理论、技术、程序设计和运行平台提出了一系列挑战[3].

为了应对这种挑战,人们开始探索新的软件技术,其中,面向 Agent 的软件工程(agent-oriented software engineering,简称 AOSE)被认为是一种支持动态开放系统开发的新型软件范型[3,4].它将软件系统看成一个多 Agent 系统(multi-agent system,简称 MAS),将自主 Agent 作为基本的运行实体,提供了一系列有别于主流软件工程技术(如 OO、构件技术等)的高层抽象、自然建模、问题分解、系统组织和模块化等软件工程机制和手段[5].为了充分发挥 AOSE 在构造开放动态软件方面的优势和潜力,简单、有效的面向 Agent 程序设计(agent-oriented programming,简称 AOP)语言成为 AOSE 走向实践的迫切需求[6].

Shoham 在 20 世纪 90 年代初首次提出了面向 Agent 程序设计的思想,并设计了第一个面向 Agent 程序设计语言——Agent-0[7].随后,来自学术界和工业界的专家、学者和工程实践人员针对 AOP 进行了深入的研究和实践,在程序设计语言和运行平台方面取得了一系列的研究成果[8,9].根据依赖的理论和技术的不同,现有的 AOP 语言和运行平台可以分为如下两类:(1) 基于对象技术.这类方法将 Agent 视为一种具有自主行为的特殊对象,通过对现有面向对象程序设计语言扩展描述和定义 Agent 结构和行为的类库或语言设施来支持 MAS 的开发和运行,如基于 Java 开发的 MAS 运行平台 Jade[10]、扩展 Java 的 JAL 和 Jack[11].(2) 基于逻辑学.这类方法将 Agent 视为一个拥有自主决策逻辑的实体,基于各种逻辑系统(如一阶谓词逻辑、时序逻辑等)来表示和定义 Agent 的结构和行为.这类程序设计语言主要是对 Prolog 扩展信念、目标、规划等心智概念来支持 MAS 的开发和运行,如 GOAL[12],Jason[13],CONGOLOG[14],3APL[15]及 2APL[16](2APL 融合了 Java 和 Prolog 两种语言)等.这些语言和平台虽然采用不同的语法和语义,但它们大多是基于目标、信念、规划等心智概念构造和定义 Agent,采用消息传递实现 Agent 的交互,在构造和实现动态开放 MAS 方面存在如下不足:

- 基于目标、信念、规划等心智概念构造多 Agent 系统,主要是从(分布式)人工智能领域来构造 MAS,如侧重于 Agent 的自主决策能力;对软件工程原则考虑不足,如缺乏有效的模块化、重用、分解和组合机制.
- 对动态性的支持不够.类似于面向对象程序设计,目前的面向 Agent 程序设计语言主要基于 Agent 类或 Agent 模板来定义或描述 Agent 的状态和行为,Agent 的状态和行为在设计时定义,运行过程中难以变化.另外,基于消息的交互机制,Agent 需要知道与之交互的 Agent 地址,缺乏支持实现动态交互的语言设施和机制.
- 程序设计模型与分析和设计模型脱节.目前,面向 Agent 的分析和设计往往基于组织抽象的思想,将 MAS 视为一个组织或社会,利用组织学和社会学中的抽象概念(如组织、角色、协议、组织规则等)对现实世界进行高层抽象和自然建模,如 ARG,Gaia,Moise+,MaSE 等[17].因此,基于组织抽象的设计模型和基于心智概念的程序设计模型之间存在概念鸿沟,往往需要程序员将设计模型中的组织概念人为地转换为 Agent 的心智概念[18],容易出现设计信息的丢失和设计与实现的不一致性等问题,严重阻碍了 AOSE 的实践性以及其优势的发挥.

针对上述需求和问题,本文提出了一种基于组织的面向 Agent 程序设计方法,通过将组织抽象的思想引入面向 Agent 的程序设计,为程序员提供组织、角色等高层和自然直观的语言设施来构造 MAS,从而缩小了设计模型与实现模型之间的概念鸿沟;同时,借鉴组织学的动态性机制为 MAS 的动态性提供编程和运行机制.基于上述思想,本文定义了基于组织的面向 Agent 程序设计语言——Oragent,给出了其抽象语法和形式语义.

本文第 1 节介绍基于组织的面向 Agent 程序设计的核心思想.第 2 节定义 Oragent 语言的抽象语法.第 3 节

© 中国科学院软件研究所 http://www.jos.org.cn

基于迁移系统定义 Oragent 语言的语义.第 4 节基于在线拍卖系统演示如何基于 Oragent 语言构造动态开放的 MAS.第 5 节对相关工作进行对比分析.第 6 节总结全文并讨论下一步的工作.

1 基于组织的面向 Agent 程序设计方法

本节主要介绍基于组织的面向 Agent 程序设计(organization-based agent-oriented programming,简称 OrgAOP)的核心思想,OrgAOP 编程模型和计算模型的详细介绍见文献[19].OrgAOP 将一个 MAS 看成一个 Group 集,每个 Group 由一组相互交互的 Agent 构成,Group 是动态、开放的,即 Agent 之间的交互动态创建,Agent 可以动态地加入或退出 Group.一方面,Group 为 Agent 提供了交互的上下文,即只有处于同一个 Group 中的 Agent 才能进行交互;另一方面,Group 作为一个整体向外部提供服务,即在其成员动态变化的情况下,仍能提供稳定而有效的服务.OrgAOP 将组织和角色作为一阶抽象实体对 Group 和 Agent 的结构和行为进行抽象和描述.组织是对一组具有相同结构和管理行为的 Group 的抽象,Group 是组织的具体运行实例.Group 的管理行为是指 Group 基于运行环境(如用户、网络带宽等信息)对其结构进行动态调整,主要通过调整其成员的数量来实现,即 Group 基于运行环境动态创建新的 Agent 或移除空闲或出错的 Agent.角色是对 Agent 在特定组织中状态和行为的描述,即角色可以看作是在某个组织中具有相同属性和行为的一组 Agent 的抽象.不同于组织与 Group 之间的实例化关系(类似于面向对象中的类与对象的关系),Agent 与角色之间是扮演关系,Agent 可以同时扮演多个角色并在运行过程中动态地改变其角色,因此,Agent 可以看作是一个开放的角色集.OrgAOP 将 Agent 扮演的角色区分为活跃状态和非活跃状态,以避免多个角色之间的行为冲突.OrgAOP 认为,只有处于活跃状态的角色的行为才能被 Agent 执行,因此,如果两个角色的行为存在冲突,只需规定两个角色不能同时处于活跃状态即可.

图 1 给出了 OrgAOP 的计算和编程模型,下面从系统(程序)的描述、基本模块、组合机制和交互机制这 4 个方面对 OrgAOP 进行描述.

- 系统(程序).在设计时,OrgAOP 将 MAS 看成是一个具有层次结构的组织,组织可以嵌套,即一个组织分解为一组子组织;在运行时,OrgAOP 允许 Agent 在多个组织中扮演多个角色,动态参与不同的交互,呈现出复杂网络的结构.
- 模块.模块是指构成系统的基本要素.OrgAOP 以组织和角色作为基本的编程实体来构造层次的组织结构,组织和角色都是可重用的模块;以 Group 和 Agent 为基本的运行实体来描述和实现 Agent 社会的复杂网络结构.另外,角色作为 Agent 在不同组织中的行为抽象,实现了 Agent 的模块化开发.Agent 可以看作是一个动态的角色组合,Agent 在角色变换的过程中实现了 Agent 状态的重用.因此,在 OrgAOP 中,组织、角色和 Agent 都是可重用的实体.
- 组合机制.组合机制描述了模块如何组装成系统.在 OrgAOP 中,组织可以看作是一个相对稳定的角色集;同时,组织还可以包括子组织,因此在设计时,角色与组织和组织之间通过静态的包含关系来构造程序.在运行时,OrgAOP 提供了动态组合机制来实现动态开放的 MAS:(1) 角色扮演机制实现 Agent 行为的动态组合.Agent 行为不是设计时定义好的,而是在运行过程中根据环境的变化通过动态扮演不同的角色动态获取的.(2) Group 是一个动态聚合体,其内部成员是不断变化的,即 Agent 可以动态地加入或退出 Group.Group 可以根据运行时环境动态的增加或删除其内部的 Agent.
- 交互机制.交互机制描述了系统运行实体间如何进行通信以实现合作和协同.传统的面向 Agent 程序设计语言中,Agent 之间主要是基于消息进行交互,其假设设计时消息的发送者必须知道接收者的地址,无法满足开放系统的要求.OrgAOP 提出了基于角色的交互机制,在开发阶段,程序员定义 Agent 的交互行为时,无须知道具体的 Agent 地址,只需知道与之交互的角色即可,具体交互的 Agent 在运行过程中动态地获取.Agent 所扮演的角色可视为 Agent 可提供的服务,因此在基于角色的交互中,Agent 只关心其需要的服务而不关心服务的具体提供者,符合动态开放系统的要求,实现了 Agent 间的动态交互和协同;同时,基于角色的交互还可以实现多播的消息传递,即 Agent 可以发送消息给某个角色当前所有的扮演者.

© 中国科学院软件研究所 http://www.jos.org.cn

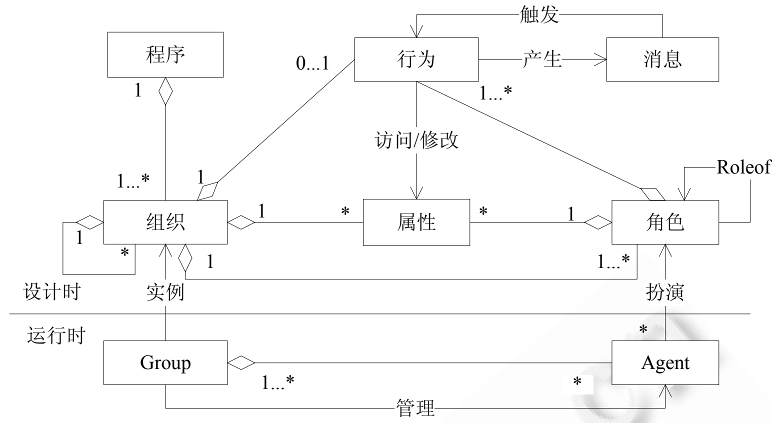

Fig.1 Computational and programming model of OrgAOP

图 1 OrgAOP 的计算和编程模型

2 基于组织的面向 Agent 程序设计语言——Oragent 的抽象语法

Oragent 语言为程序员基于 OrgAOP 基本思想来构造和实现 MAS 提供了核心语言设施和编程机制.基本语言设施主要包括组织、角色、Agent、行为、动作和消息等.文献[20]定义了 Oragent 语言的具体语法.具体语法是告诉程序员具体写什么,怎么才能构造有效的程序;而抽象语法对于决定程序的意义是非常重要的,它是定义语义和实现运行时平台的基础.图 2 给出了 Oragent 语言的抽象语法.第 3 节将基于该抽象语法给出 Oragent 语言的形式化操作语义.

$$
\begin{aligned}
exp &::= \varepsilon \mid var \mid \mathbf{self} \mid \mathbf{nil} \mid \mathbf{new}^r \mid \mathbf{new}^o \mid f(exp,\ldots,exp)\\
act_{transfer} &::= \mathbf{enact}\ g.r \mid \mathbf{deact}\ g.r \mid \mathbf{activate}\ g.r \mid \mathbf{deactivate}\ g.r\\
act_{regulate} &::= \mathbf{new}\ r \mid \mathbf{kill}\ r \mid \mathbf{kill}\ r^* \mid \mathbf{fire}\ r \mid \mathbf{fire}\ r^*\\
act_{send} &::= r.m(exp,\ldots,exp) \mid r^*.m(exp,\ldots,exp) \mid _.m(exp,\ldots,exp) \mid a.m(exp,\ldots,exp)\\
invk &::= act_{transfer} \mid act_{regulate} \mid act_{send} \mid act(exp,\ldots,exp)\\
mt &::= r!m(exp,\ldots,exp) \mid !m(exp,\ldots,exp)\\
stm &::= ; \mid var = exp; \mid t\ var; \mid invk; \mid \mathbf{recevie}\ \{stm_{handle}\} \mid \mathbf{on}(mt)\ block \mid \mathbf{if}\ exp\ block\ \mathbf{else}\ block \mid \mathbf{while}(exp)\ block\\
stm_{handle} &::= mt : stm\ldots stm\\
block &::= \{stm\ldots stm\}\\
beh &::= \mathbf{when}\ block \mid \mathbf{loop\ when}\ block\\
modifier &::= \mathbf{interanl} \mid \mathbf{external}\\
role &::= modifier\ r(t\ var,\ldots,t\ var)\ \mathbf{roleof}\ r,\ldots,r\{beh\ldots beh\}\\
org &::= o\{role\ldots role\ beh\}
\end{aligned}
$$

Fig.2 Abstract syntax of Oragent language

图 2 Oragent 语言的抽象语法

图 2 在描述 Oragent 语言的抽象语法时,对特定符号做如下约定:ε表示空表达式,*var* 表示变量,*f* 表示可以作用在变量上的操作算子,*t* 表示变量 *var* 的类型名,*r*,*o*,*a*,*g*,*act* 和 *m* 分别表示角色名、组织名、Agent 标识符、Group 标识符、动作名和消息名,**self** 是 Agent 或 Group 内指向自身标识符的常量,$\mathbf{new}^r$ 和 $\mathbf{new}^o$ 分别表示基于角色 *r* 创建的新 Agent 的标识符和基于组织 *o* 创建的新 Group 的标识符.

动作是构成行为的基本单元,动作的调用描述为 $act(exp_1,\ldots,exp_n)$,由动作名 *act* 和实参 $exp_1,\ldots,exp_n$ 构成.除了用户定义的动作外,Oragent 提供了一些基本动作用来支持角色的动态变迁、Group 结构的动态调整和基于角色的交互.角色扮演、角色去演、角色激活和角色钝化是 Oragent 为 Agent 提供的 4 个基本角色变迁动作,

动作名分别为 **enact**,**deact**,**activate** 和 **deactivate**,它们的实参为 g 和 r.其中,r 表示变迁的角色名,g 表示 r 所在的 Group 标识符.为了增强可读性,Group g 中的角色 r 表示为 $g.r$.创建 Agent、杀死 Agent 和解雇 Agent 是 Oragent 为 Group 提供的 3 个基本成员调整动作,动作 **new** r 表示基于角色 r 创建一个新的 Agent,**new**^{r} 表示新创建 Agent 的标识符,**kill** r 表示杀死角色 r 的一个扮演者,**kill** r^*表示杀死角色 r 的所有扮演者,**fire** r 表示解雇角色 r 的一个扮演者,**fire** r^*表示解雇角色 r 的所有扮演者.其中,杀死 Agent 与解雇 Agent 动作都是 Group 用来删除其成员的操作,其区别在于,Group 只能杀死本地 Agent(自己创建的 Agent),对于外来的 Agent 进行解雇操作.在 Oragent 中,Agent 的基本通信方式仍然是通过消息传递实现,消息描述为 $m(exp_1,\dots,exp_n)$,由消息名 m 和消息参数 $exp_1,\dots,exp_n$ 组成.不过,Oragent 基于角色为 Agent 之间的消息发送提供了多种描述方式:$r.m(exp_1,\dots,exp_n)$表示发送消息 m 给角色 r 的一个扮演者,$r^*.m(exp_1,\dots,exp_n)$表示发送消息 m 给角色 r 的所有扮演者,$_.m(exp_1,\dots, exp_n)$表示广播消息 m,其他的 Agent 可以用 on(mt)来订阅广播的消息.其中,mt 为消息模板,由发送者信息、消息名和消息参数组成,描述为 $r!m(exp_1,\dots,exp_n)$或$!m(exp_1,\dots,exp_n)$.与前者相匹配的消息需满足消息的发送者是角色 r 的扮演者,消息名为 m,消息的参数数目 n 且每个消息的类型相一致;后者则不关心消息的发送者.

在 Oragent 中,语句除了基本的赋值语句 $var{=}exp$ 和动作调用语句以外,还包括复合语句:条件语句、循环语句和消息处理语句.其中,条件语句和循环语句类似于 Java 中的定义.下面主要介绍消息处理语句.Oragent 提供两种消息处理语句:**receive** 语句表示 Agent 从其消息队列中取一消息,与其后定义的消息模板依次进行匹配,如果匹配成功则执行紧跟该消息模板的语句;如果取得的消息与 **receive** 中所有的消息模板都不匹配,则取下一条消息进行匹配;如果消息队列中所有的消息都不能与 **receive** 中任一消息模板匹配则阻塞当前行为;**on** 语句首先执行消息订阅动作,然后执行过程与 **receive** 语句一样.行为的定义是一个以关键字 **when** 开头的语句块;**loop** 表示该行为是循环运行的.

角色的定义描述为 *modifier* $r(t\ var,\dots,t\ var)$ **roleof** $r_1,\dots,r_n\{beh_1,\dots,beh_n\}$,由角色名、构造参数和一组行为组成,修饰符 *modifier* 的取值可以是 **internal** 或 **external**.关键字 **internal** 表示角色 r 为 Group 内部角色对外不可见,其扮演者都是本地 Agent(由角色所在的 Group 所创建的 Agent);**external** 表示角色 r 为外部角色,由外来 Agent 扮演.关键字 **roleof** 声明了角色之间的变迁关系,称为角色的角色.假设角色 r_1 是角色 r_2 的角色,即“r_1 **roleof** r_2”,则称 r_1 为 r_2 的基角色,r_2 为 r_1 的扩展角色.在 Oragent 中,基角色定义了其扩展角色对扮演者的要求,只有扮演了基角色的 Agent 才能扮演其扩展角色,即扩展角色的扮演者一定也是其基角色的扮演者.事实上,“**roleof**”关系也是一种“is-a”关系,我们可以认为扩展角色是基角色的角色.另一方面,扩展角色可以看作是基角色行为的扩展,即基角色的扮演者可以通过扮演其扩展角色来动态的扩展自身的行为.OrgAOP 认为,相同基角色的扩展角色之间存在变迁关系,Agent 在运行过程中可以在这些角色之间动态地切换,同时保留其基角色定义的状态和行为.组织的定义描述为 $o\{role,\dots,role\ beh\}$,由组织名、一组角色和组织结构调整行为组成.

3 Oragent 语言的形式语义

本节基于迁移系统定义 Oragent 语言中支持动态性的基本动作的操作语义.Oragent 语言支持动态性的基本动作可以分为 3 类:角色变迁动作、Group 结构调整动作和基于角色的消息发送动作.下面首先定义 Oragent 语言的静态语义,即 Oragent 程序中主要运行实体的状态和格局,然后定义这些格局之间的状态迁移规则,从而描述动作的操作语义.

3.1 静态语义

Agent、角色和 Group 是 Oragent 程序的 3 个基本运行实体,其中,Agent 可以执行除了 Group 结构调整动作以外其他所有类型的动作;Group 主要负责其成员的管理,主要执行 Group 结构调整动作;角色主要负责消息的路由以实现基于角色的动态交互,因此,角色需要保存其所有扮演者的标识符,并在运行过程中为其扮演者动态地分发消息.此外,角色还需要与 Group 进行交互,以实现 Group 结构的调整.定义 1~定义 3 分别给出了 Group、角色和 Agent 的格局定义.这里为了简化 3 类基本动作的操作语义的理解,我们省略了一些与这些动作无关的要素.例如,角色的行为集在运行过程中并不会发生变化,故在角色格局的定义中暂不考虑.

© 中国科学院软件研究所 http://www.jos.org.cn

定义 1. Group 格局定义为一个元组$\langle g,R,f\rangle$,其中:g 是 Group 的唯一标识符;R 为 Group g 中定义的角色集,包括了内部角色和外部角色;函数 f 表示 Group g 中每个角色的当前扮演者数目,定义为 $f{:}R^m\rightarrow N^m$,其中,N 为整数集,m 为 Group 中角色的数目.另外,$f[r/n]$表示用 n 替换 $f(r)$值后得到的新函数,即定义为

$$f[r/n](r')=\begin{cases}n, & r=r'\\ f(r'), & r\neq r'\end{cases}.$$

定义 2. 角色格局定义为一个四元组$\langle r,g,c,P\rangle$,其中,

- r 为角色名,在一个组织中是唯一的;
- g 为角色所在的 Group;
- c 是角色的可见性,$c\in$\{**internal**,**external**\};
- P 是角色当前扮演者组成的集合,其元素 p 是一个二元组$\langle a,s\rangle$,a 为扮演者的标识符,$s\in$\{**active**, **inactive**\}表示角色在扮演者所处的状态.

角色与角色之间的"**roleof**"关系用$\prec$表示,如果角色 r_1 是 r_2 的扩展角色或 r_2 是 r_1 的基角色,则表示为 $r_1\prec r_2$;如果角色 r_1 是 r_2 的直接扩展角色或 r_2 是 r_1 直接基角色,则表示为 $r_1\prec_d r_2$.

Oragent 将 Agent 扮演的角色区分为活跃角色和非活跃角色."**roleof**"的定义认为:扮演扩展角色的 Agent 一定扮演了基角色;同时,如果扩展角色处于活跃状态,则基角色也处于活跃状态.我们用活跃角色链ρ来表示 Agent 当前处于活跃状态的所有角色:$\rho=r_1r_2\ldots r_n$,其中,$r_1\prec_d r_2\prec_d\ldots\prec_d r_n$;$\rho(i)$表示$\rho$上第 i 个元素,即$\rho(i)=r_i,i\in\{1,2,\ldots,n\}$,并定义尾函数$\tau(\rho)=r_n$.函数 $sup{:}R\rightarrow 2^R$ 用来获取角色的所有基角色组成的集合,即 $sup(r)=\{r'|\forall r',r'\in R\wedge r\prec r'\}$.

定义 3. Agent 格局定义为一个四元组$\langle a,\rho,R_i,M\rangle$,其中,

- a 是 Agent 的唯一标识符;
- ρ是 Agent 的活跃角色链:$\rho=r_1r_2\ldots r_n$;
- R_i 是 Agent 扮演的所有处于非活跃的角色集,我们用 R^a 表示 Agent a 扮演的所有角色组成的集合:
$$R^a=\bigcup_{1\leqslant i\leqslant n}\rho(i)\cup R_i;$$
- M 为 Agent a 接收到的所有消息组成的集合.

3.2 动态语义

本节从单个 Agent 行为的执行、Group 对其成员调整行为的执行和 Agent 之间通信动作的执行这 3 个方面来研究 Oragent 语言动态语义,$\rightarrow_{agt},\rightarrow_{role},\rightarrow_{group}$ 分别表示发生在 Agent 格局、角色格局和 Group 格局的变迁.

3.2.1 角色变迁动作

Agent a 成功执行角色扮演动作"**enact** $g.r$"的前提:(1) r 的直接基角色处于活跃状态;(2) r 在 g 中有定义.设 r 的直接父角色为 r',则 a 执行"**enact** $g.r$"动作发生的格局变迁包括:Agent a 钝化 r'的所有子角色,并将 r 添加到活跃角色链尾部;角色 r 将 a 作为活跃扮演者;Group g 将角色 r 的扮演者数目加 1.假设 Agent a、角色 r 和 Group g 的格局分别为$\langle a,\rho,R_i,M\rangle$,$\langle r,g,c,P\rangle$和$\langle g,R,f\rangle$,则角色扮演动作的变迁规则定义如下:

$$\frac{r\in R,\exists i\in\{1,2,\ldots,m\}\ r\prec\rho(i)}{\langle a,\rho,R_i,M\rangle\rightarrow_{agt}\langle a,\rho',R_i',M\rangle\ \langle r,g,c,P\rangle\rightarrow_{role}\langle r,g,c,P\cup\{\langle a,\mathbf{active}\rangle\}\rangle\ \langle g,R,f\rangle\rightarrow_{group}\langle g,R,f[r/f(r)+1]\rangle}$$

其中,

- $\rho=r_1\ldots r_m$;
- $\rho'=r_1\ldots r_jr,r\prec_d r_j$;
- $R_i'=\begin{cases}R_i, & \text{if } r_j=r_m\\ R_i\cup\{\rho(i)\mid j<i\leqslant m\}, & \text{otherwise}\end{cases}.$

Agent a 成功执行角色扮演动作"**deact** $g.r$"的前提:(1) a 在 g 中扮演了角色 r;(2) r 处于活跃状态.a 执行"**deact** $g.r$"动作根据 r 是否是 a 中活跃角色链的尾角色发生不同的格局迁移:如果 r 是 a 中活跃角色链的尾角

© 中国科学院软件研究所 http://www.jos.org.cn

色,则 a 删除角色 r,r 删除其扮演者 a,同时,Group g 将角色 r 的扮演者数目减 1;如果 r 不是 a 中活跃角色链的尾角色,则 a 删除角色 r 及扮演的所有 r 的扩展角色,r 及 r 的所有扩展角色删除其扮演者 a(如果存在扮演者 a),同时,角色 r 及其扩展角色所在的 Group 将对应的角色扮演者数目减 1.假设 Agent a、角色 r 和 Group g 的格局分别为$\langle a,\rho,R_i,M\rangle$,$\langle r,g,c,P\rangle$和$\langle g,R,f\rangle$,则角色去演动作的变迁规则定义如下:

$$\frac{\exists i\in\{1,2,...,m\}\ r=\rho(i),\langle a,active\rangle\in A,\forall r'\in(\{r'\mid r'\in R^a\wedge r'\prec r\}\cup\{r\})}{\langle a,\rho,R_i,M\rangle\rightarrow_{agt}\langle a,\rho',R_i',M\rangle\ \langle r',g',c',P'\rangle\rightarrow_{role}\langle r',g',c',P'\setminus\{\langle a,\mathbf{active}\rangle\}\rangle\ \langle g,R,f\rangle\rightarrow_{group}\langle g,R,f'\rangle}.$$

其中,

- $\rho=r_1\ldots r_m$;
- $\rho'=\begin{cases}\bot, & r=r_1\\ r_1\ldots r_{j-1}, & r=r_j\end{cases}$,符号⊥表示活跃角色链为空;
- $R_i'=\begin{cases}R_i/\{r'\mid\forall r'\in R_i\ r'\prec r\}, & \text{if }\exists r'\in R_i\ r'\prec r\\ R_i, & \text{otherwise}\end{cases}$;
- $f'=\begin{cases}f[r/f(r)-1], & \text{if }r=\tau(\rho)\\ \forall r'\in(\{r'\mid r'\in R^a\wedge r'\prec r\}\cup\{r\})\ f[r'/f(r')-1], & \text{otherwise}\end{cases}$.

Agent a 成功执行角色激活动作"**activate** $g.r$"的前提:a 当前活跃角色链的尾角色是 r 的直接父角色.a 执行"**activate** $g.r$"动作发生的格局变迁包括:a 将 r 添加到活跃角色链的尾部并将 r 从其非活跃角色集中删除,同时,r 将 a 设置为活跃扮演者.假设 Agent a 和角色 r 的格局分别为$\langle a,\rho,R_i,M\rangle$和$\langle r,g,c,P\rangle$,其中,$\rho=r_1\ldots r_m$,则角色激活动作的迁移规则定义如下:

$$\frac{r\prec_d\tau(\rho)}{\langle a,\rho,R_i,M\rangle\rightarrow_{agt}\langle a,\rho',R_i\setminus\{r\},M\rangle\ \langle r,g,c,P\rangle\rightarrow_{role}\langle r,g,c,(P\setminus\{\langle a,\mathbf{inactive}\rangle\})\cup\{\langle a,\mathbf{active}\rangle\}\rangle},$$

其中,$\rho'=r_1\ldots r_m r$.

Agent a 成功执行角色钝化动作"**deactivate** $g.r$"的前提:(1) r 处于活跃状态;(2) r 存在基角色.a 执行"**deactivate** $g.r$"动作根据 r 是否是 a 中活跃角色链的尾角色发生不同的格局迁移:如果 r 是 a 中活跃角色链的尾角色,则 a 将 r 从活跃角色链中移除并添加到非活跃角色集中,同时,r 将其扮演者 a 从活跃状态变为非活跃状态;如果 r 不是 a 中活跃角色链的尾角色,则 a 将 r 及其处于活跃角色链的所有扩展角色从活跃角色链中移除并添加到非活跃角色集中,同时,r 及处于活跃角色链的所有扩展角色将 a 从活跃状态变为非活跃状态.假设 Agent a 和角色 r 的格局分别为$\langle a,\rho,R_i,M\rangle$和$\langle r,g,c,P\rangle$,其中,$\rho=r_1\ldots r_m$,则角色钝化动作的迁移规则定义如下:

$$\frac{\exists i\in\{2,...,m\}\ r=\rho(i),\forall r'\in(\{\rho(i)\mid\rho(i)\prec r\}\cup\{r\})}{\langle a,\rho,R_i,M\rangle\rightarrow_{agt}\langle a,\rho',R_i',M\rangle\ \langle r',g',c',P'\rangle\rightarrow_{role}\langle r',g',c',(P'\setminus\{\langle a,\mathbf{active}\rangle\})\cup\{\langle a,\mathbf{inactive}\rangle\}\rangle},$$

其中,

- $\rho'=r_1\ldots r_{j-1},r=r_j$;
- $R_i'=\begin{cases}R_i\cup\{r\}, & \text{if }r=r_m\\ R_i\cup\{r\}\cup\{\rho(i)\mid\rho(i)\prec r\}, & \text{otherwise}\end{cases}$.

3.2.2 结构调整动作

Group g 成功基于角色 r 创建 Agent 的前提:(1) r 在 g 中有定义;(2) r 没有基角色,Oragent 要求只能基于最顶层的基角色创建 Agent.Group g 执行"**new** r"动作的格局变迁包括:g 将 r 的扮演者数目加 1;r 增加 Agent $\mathbf{new}^r$ 为其活跃扮演者;Agent $\mathbf{new}^r$ 设置 r 为当前活跃角色.假设角色 r 和 Group g 的格局分别为$\langle r,g,c,P\rangle$和$\langle g,R,f\rangle$,则 Agent 创建动作的迁移规则定义如下:

$$\frac{r\in R,\neg\exists r'\in\Sigma_r\ r\prec r'}{\langle r,g,c,P\rangle\rightarrow_{role}\langle r,g,c,P\cup\{\langle\mathbf{new}^r,\mathbf{active}\rangle\}\rangle\ \langle g,R,f\rangle\rightarrow_{group}\langle g,R,f[r/f(r)+1]\rangle},$$

其中,Σ_r 为 MAS 中的所有角色组成的集合,新创建的 Agent 的格局为$\langle\mathbf{new}^r,r,\varnothing,\varnothing\rangle$.

Group g 成功杀死角色 r 的扮演者的前提:r 必须在 g 中有定义且为内部角色,r 没有基角色.杀死 Agent 的动

© 中国科学院软件研究所 http://www.jos.org.cn

作存在两种形式.动作“**kill** r”是杀死角色 r 的一个扮演者,具体杀死哪一个扮演者由角色 r 决定,执行动作“**kill** r”的格局变迁包括:g 将 r 的扮演者数目减 1;r 删除一个扮演者(优先删除非活跃状态的扮演者);不妨设角色 r 基于某种算法决定 Agent a 为预删除的扮演者,则 a 扮演的所有角色都将 a 从其扮演者删除,而且 a 扮演的每一个角色所在的 Group 都将该角色的扮演者数目减 1.假设角色 r 和 Group g 的格局分别为$\langle r,g,c,P\rangle$和$\langle g,R,f\rangle$,则动作“**kill** r”的迁移规则定义如下:

$$\frac{\sup(r)=\varnothing, c=\textbf{internal}, \exists\langle a,s\rangle\in P}{\forall r'\in(R^a\cup\{r\})\ (\langle r',g',c',P'\rangle\rightarrow_{role}\langle r',g',c',P'\setminus\{\langle a,s\rangle\}\rangle\ \langle g',R',f'\rangle\rightarrow_{group}\langle g',R',f'[r'/f(r')-1]\rangle)}.$$

动作“**kill** r^*”是杀死角色 r 的所有扮演者,执行动作“**kill** r^*”的格局变迁包括:g 将 r 的扮演者数目设置为 0;r 清空其扮演者;r 的每一个扮演者 a 所扮演的其他角色都将 a 从其扮演者删除,而且 a 扮演的每一个角色所在的 Group 都将该角色的扮演者数目减 1.假设角色 r 和 Group g 的格局分别为$\langle r,g,c,P\rangle$和$\langle g,R,f\rangle$,则动作“**kill** r^*”的迁移规则定义如下:

$$\frac{\sup(r)=\varnothing, c=\textbf{internal}}{\begin{array}{c}\langle r,g,c,P\rangle\rightarrow_{role}\langle r,g,c,\varnothing\rangle\ \langle g,R,f\rangle\rightarrow_{group}\langle g,R,f[r/0]\rangle\\ \forall\langle a,s\rangle\in P\ \forall r'\in R^a\ \ (\langle r',g',c',P'\rangle\rightarrow_{role}\langle r',g',c',P'\setminus\{\langle a,s\rangle\}\rangle\ \langle g',R',f'\rangle\rightarrow_{group}\langle g',R',f'[r'/f(r')-1]\rangle)\end{array}}.$$

Group g 成功解雇角色 r 的扮演者的前提:r 必须在 g 中有定义且为外部角色.解雇 Agent 的动作存在两种形式.动作“**fire** r”是解雇角色 r 的一个扮演者,具体解雇哪一个扮演者由角色 r 决定,执行动作“**fire** r”的格局变迁包括:g 将 r 的扮演者数目减 1;r 删除一个扮演者(优先删除非活跃状态的扮演者);不妨设角色 r 基于某种算法决定 Agent a 为解雇的扮演者,则 a 删除 r 及其所有的扩展角色(如果扮演了);如果 r 的扩展角色的扮演者有 a 则将其删除,同时,该角色所在的 Group 的扮演者数目减 1.假设 Agent a、角色 r 和 Group g 的格局分别为$\langle a,\rho,R_i,M\rangle$、$\langle r,g,c,P\rangle$和$\langle g,R,f\rangle$,则动作 **fire** r 的迁移规则定义如下:

$$\frac{r\in R, c=\textbf{external}, \exists\langle a,s\rangle\in P}{\begin{array}{c}\langle a,\rho,R_i,M\rangle\rightarrow_{agt}\langle a_{id},\rho',R_i',M\rangle\\ \forall r'\in(\{r'\mid r'\in R^a\wedge r'\prec r\}\cup\{r\})\ (\langle r',g',c',P'\rangle\rightarrow_{role}\langle r',g',c',P'\setminus\{\langle a,s\rangle\}\rangle\ \langle g',R',f'\rangle\rightarrow_{group}\langle g',R',f'[r'/f(r')-1]\rangle)\end{array}}.$$

其中,

- $\rho=r_1\ldots r_m$;
- $\rho'=\begin{cases}r_1\ldots r_{j-1}, & \text{if } \exists j\in\{1,2,\ldots,m\}\ r=r_j\\ \rho, & \text{otherwise}\end{cases}$;
- $R_i'=\begin{cases}R_i\setminus(\{r'\mid r'\in R_i\wedge r'\prec r\}\cup\{r\}), & \text{if } r\in R_i\\ R_i, & \text{otherwise}\end{cases}$.

动作“**fire** r^*”是解雇角色 r 的所有扮演者,执行动作“**fire** r^*”的格局变迁包括:g 将 r 的扮演者数目设置为 0,r 清空其扮演者;r 的每一个扮演者 a 删除 r 及其所有的扩展角色(如果扮演了);如果 r 的扩展角色的扮演者有 a 则将其删除,同时,该角色所在的 Group 的扮演者数目减 1.假设 Agent a、角色 r 和 Group g 的格局分别为$\langle a,\rho,R_i,M\rangle$、$\langle r,g,c,P\rangle$和$\langle g,R,f\rangle$,则动作“**fire** r^*”的迁移规则定义如下:

$$\frac{r\in R, c=\textbf{external}\ \forall\langle a,s\rangle\in P}{\begin{array}{c}\langle r,g,c,P\rangle\rightarrow_{role}\langle r,g,c,\varnothing\rangle\ \langle a,\rho,R_i,M\rangle\rightarrow_{agt}\langle a,\rho',R_i',M\rangle\ \langle g,R,f\rangle\rightarrow_{group}\langle g,R,f[r/0]\rangle\\ \forall r'\in\{r'\mid r'\in R^a\wedge r'\prec r\}\ \langle r',g',c',P'\rangle\rightarrow_{role}\langle r',g',c',P'\setminus\{\langle a,s\rangle\}\rangle\ \langle g',R',f'\rangle\rightarrow_{group}\langle g',R',f'[r'/f(r')-1]\rangle\end{array}}.$$

其中,

- $\rho=r_1\ldots r_m$;
- $\rho'=\begin{cases}r_1\ldots r_{j-1}, & \text{if } \exists j\in\{1,2,\ldots,m\}\ r=r_j\\ \rho, & \text{otherwise}\end{cases}$;
- $R_i'=\begin{cases}R_i\setminus(\{r'\mid r'\in R_i\wedge r'\prec r\}\cup\{r\}), & \text{if } r\in R_i\\ R_i, & \text{otherwise}\end{cases}$.

© 中国科学院软件研究所 http://www.jos.org.cn

3.2.3 消息发送动作

本节考虑的消息发送动作局限于 Agent 与 Agent 之间的交互,因此,消息发送动作的执行只引起 Agent 格局的变化,而对角色和 Group 的格局没有影响.消息发送动作由消息发送者执行,引起消息接收者格局的变换:在消息接收者的消息队列中添加消息.在 Oragent 中,基于角色的交互是指具体接收消息的 Agent 是由角色来决定的,因此,消息被 Agent 成功接收的前提是角色中存在活跃的扮演者.我们用符号 $sender^m$ 表示消息 m 的发送者,假设角色 r 的格局为 $\langle r,g,c,P\rangle$,定义函数 $active(P)$ 表示角色 r 中活跃角色集,则消息发送动作"$r.m$"和"$r^*.m$"的迁移规则定义如下:

$$\frac{active(P)\neq\varnothing}{\exists a\in active(P)\ \langle a,\rho,R_i,M\rangle\to_{agt}\langle a,\rho,R_i,M\cup\{m\}\rangle},$$

$$\frac{active(P)\neq\varnothing}{\forall a\in active(P)\ \langle a,\rho,R_i,M\rangle\to_{agt}\langle a,\rho,R_i,M\cup\{m\}\rangle}.$$

Oragent 还提供了消息的广播操作"_.m",该操作中,消息的接收方是主动订阅消息,即消息会被发送给订阅了该消息的 Agent,我们用函数 $on(a)$表示 Agent a 订阅的消息集,于是,动作"_.m"的迁移规则定义如下:

$$\frac{\forall a\in\Sigma_a\wedge m\in on(a)}{\langle a,\rho,R_i,M\rangle\to_{agt}\langle a,\rho,R_i,M\cup\{m\}\rangle},$$

其中,Σ_a 表示系统中当前运行的所有 Agent 组成的集合.

4 案例分析

本节通过一个在线拍卖系统来说明如何基于 Oragent 语言构造更加动态灵活的 MAS.在线拍卖系统可以看作是一个拍卖组织(AuctionOrg),由买家(buyer)和卖家(seller)两个角色组成.拍卖组织又一步分解为两个子组织竞价组织(BidingOrg)和支付组织(PaymentOrg),分别负责竞价和在线支付的实现.竞价组织由拍卖商(auctioneer)、提供商(supplier)和竞价者(bidder)这 3 个角色构成,分别实现管理拍卖过程、提供竞拍商品和进行竞价等功能.支付组织由收款人(payee)、付款人(payer)和负责转账功能的代理者(broker)这 3 个角色组成.假设系统为每一次商品的拍卖创建一个竞价 Group(竞价组织的实例),同时,系统中只存在一个支付 Group,于是,在线拍卖系统的一次拍卖过程可以描述为:首先,卖家基于拍卖的商品创建拍卖 Group,并在其中扮演提供商角色;如果拍卖 Group 所拍卖的商品是买家要买的商品,则加入拍卖 Group 并扮演竞价者角色;如果拍卖成功,卖家和赢得竞价的买家同时加入支付 Group,分别扮演收款人角色和付款人角色.由此可以得出,竞价组织的竞价者和支付组织的付款人角色都是拍卖组织中买家角色的角色,而竞价组织中的提供商和支付组织中的收款人角色都是拍卖组织中卖家角色的角色.即,存在"Bidder **roleof** Buyer","Payer **roleof** Buyer","Supplier **roleof** Seller"和"Payee **roleof** Seller".由上面描述的拍卖过程可以得出在线拍卖系统具有如下动态性:

- 用户扮演的角色动态变化:例如,用户 Tom 进入系统后,扮演 Buyer 角色订阅购买的商品,当系统中竞拍商品中出现他要购买的商品时,就扮演 Bidder 角色进行竞价,如果他赢得了竞价则扮演 Payer 角色支付货款,否则退出 Bidder 角色.该角色变迁逻辑定义在 Buyer 角色中,如图 3(a)所示.
- Group 动态地调整结构:支付 Group 可以根据转账请求的多少动态地调整扮演代理者的 Agent 数量.例如,如果每个代理 Agent 处理的转账请求大于 5 个,则增加一个新的代理 Agent;否则,如果每个代理 Agent 的转账请求小于 5 且支付 Group 中的代理 Agent 多于 2 个,则删除一个代理 Agent,如图 3(c)所示.
- Agent 动态的获取交互对象:由于系统中 Agent 是动态变化的,Agent 的交互动态在设计时不可知,只能在运行时基于订阅/发布模式或基于角色来动态的获取交互的 Agent.

拍卖系统的 Oragent 代码如图 3 所示,其中,关键字"**within**"是对定义实体所属组织的声明,因此在 Oragent 语言中角色可以定义在其所属的组织内,也可以定义在独立的文件中.

© 中国科学院软件研究所 http://www.jos.org.cn

```
organization AuctionOrg{
 internal role Buyer(string good, int price) {
   ...
   when(){

    //subscribe the auction of good
    on(auction(good, BiddingOrg g)){
      enact g.Bidder(price).
    }

    //roles transfer behavior
    loop receive{
      Auctioneer !lower(BiddingOrg g):
        deact g.Bidder.
      Auctioneer!win(good, PaymentOrg g):
        enact g.Payer(price).
      Broker!payment(PaymentOrg g,"success"):
        deact g.Payer.
      }
   }
 }

 internal role Seller(string good, int price){
   when(){

     // create a bidding group for bidding
      biddingGroup=new BiddingOrg();
      enact biddingGroup.Provider(good,price);

      //role transfer behaviors
      loop receive{
        Auctioneer ! trading(PaymentOrg gid, int p):
          enact gid.Payee(p).
        Broker ! Payment("success"): deact Payee.
        Auctioneer ! Auction("success"):
          deact Provider.
      }
   }
 }
}
```

(a) 拍卖组织的部分代码

```
within AuctionOrg;
organization BiddingOrg{
   ...
 external role Bidder(int price) roleof Buyer{...}
 external role Provider(int price) roleof Seller{...}
  internal single role Auctioneer{
    when(Provider provider ! auction(
                 string good, int price)){
      setAuction(good, price);
      _.auction(mygroup, good)
      loop receive{
        Bidder b ! bid(int p):
            handleBidding(b, p).
      }after (2000)
      makeWinner();
      gid=findPaymentGroup();
      winner.win(good, gid);
      provider.trading(gid, good);
      receive{
        provider ! payment("failure"):
            handlePaymentFailure().
      }
    }
    //action
    handlePaymentFailure(){...}
  }
}
```

(b) 竞价组织的部分代码

```
within AuctionOrg;
organization PaymentOrg{
   ...
 internal role Broker{...}
 external role Payer(int money) roleof Buyer{...}
 external role Payee(int money) roleof Seller{...}
 // structure regulating  behavior
 loop when(){
 if(playersNum(Payer)/playersNum(Broker)>5){
   new Broker();
 }
 if(playersNum(Broker)<playersNum(Payer)){
   kill Broker;
 }
}
```

(c) 支付组织的部分代码

Fig.3 Code fragments of on-line auction system

图 3 拍卖系统的部分代码

5 相关工作

传统的 AOP 语言为单个 Agent 内部结构和行为的描述和构造提供了强大的语言设施,如信念、目标、规划等,大多数语言为 Agent 提供了基于消息的交互机制,此类语言的代表工作有 JadeX[21],3APL[15]和 GOAL[12]等.基于消息的交互机制,要求 Agent 必须知道与之交互的 Agent 地址,缺乏对动态交互的支持.JAL[11]是对 Java 扩展 agent,event 和 plan 等关键字来实现对 BDI 型 Agent 的描述和构造.它为 Agent 提供了基于事件的交互机制,支持动态交互.2APL[16]和 Jason[13]是新提出的针对多 Agent 系统的开发.它们不仅提供了构造单个 Agent 行为的语言设施,而且将环境作为一阶实体.它们将 MAS 看作一组 Agent 和一组环境构成.它们都将环境看作是一个 Java 对象,Agent 之间通过消息进行交互,而 Agent 与环境之间通过事件交互,另外,Agent 与 Agent 还可以通过环境进行间接交互,实现了开放系统中 Agent 之间的协同工作.

表 1 从提供的基本概念、交互机制、是否具有形式语义、是否具有运行平台以及是否对动态性的支持几个方面对当前主要的 AOP 语言进行了分析和对比,其中,M 表示基于消息交互、E 表示基于事件交互、√表示肯定、–表示否定.由表 1 可知,现有的程序设计语言主要基于心智概念来描述 Agent 的内部结构和行为,基于消息实现 Agent 的交互,对开放 MAS 动态性的支持不足.

© 中国科学院软件研究所 http://www.jos.org.cn

Table 1 Traditional AOP languages

表 1 传统 AOP 语言

	基本概念	通信机制	形式化语义	平台	动态性支持
JadeX	信念、目标、规划、能力和事件	M	–	Jade	√
JAL	规划、能力和事件	E	–	Jack	√
3APL	信念、目标、规划和规则	M	√	√	–
GOAL	知识、信念、目标和规则	M	√	√	–
2APL	信念、目标、规划、规则、Agent 和环境	M	√	√	–
Jason	信念、事件、规划、意图、Agent 和环境	M&E	–	Jason	–

基于组织抽象的思想来提高系统的动态性和灵活性,得到了软件工程领域(尤其是 AOSE)的广泛关注和重视,下面分别从面向对象程序设计和面向 Agent 软件工程两个方面介绍组织抽象在 AOSE 中的应用情况.

在面向对象程序设计中,角色被用来抽象对象的适应或动态行为,通过对象与角色之间的动态绑定关系实现对象行为的动态变化,从而构造更加动态的软件[22].powerJava[23]对 Java 扩展了组织类和角色类.它从组织的角度出发,利用组织定义对象的交互上下文,基于组织中的角色定义对象之间复杂的交互过程,因此,角色在面向对象的程序设计中的主要有两个作用:(1) 对象行为的动态组合,利用角色封装对象的可变行为,在运行过程中动态的绑定(或激活)角色;(2) 关注点分离,基于角色定义交互模式与具体的运行实体-对象相分离.

随着组织抽象思想在面向 Agent 分析和设计中的成功应用,将组织抽象思想引入面向 Agent 程序设计成为 AOSE 领域的新焦点.人们希望在底层为高层的组织概念和机制提高显式的支撑来缩小设计模型和实现之间的概念鸿沟,同时为 MAS 的动态性提供更好的支撑机制.目前,在实现阶段支持组织抽象的方法主要有两种:中间件(如 AMELI[24],JaCaMo[25],Janus[26],powerJade[27],MACODO[28]等)和程序设计语言(MetataM 的扩展[29]、2OPL[6]等).表 2 给出了目前主要组织中间件、运行平台和程序设计语言的基本概念和对动态性的支持情况.

Table 2 Organizational middleware, platforms and programming language

表 2 组织中间件、运行平台及程序设计语言

	基本概念	实现方法	角色变迁	结构调整	动态交互
AMELI	场景和协议	中间件	–	√	√
JaCaMo	组织、Group、角色和工件(artifact)	中间件	√	√	√
Janus	组织、角色和 Holon	开发包库	√	√	√
powerJade	组织和角色	开发包库	√	–	√
MACODO	组织和角色	中间件	–	√	√
MetataM[29]	上下文、内容和 Agent	语言	–	√	√
2OPL	组织、角色和 规范(norm)	语言	√	–	–
simpAL	组织、角色、工件和 Agent	语言	√	–	√
Oragent	组织、角色、Group 和 Agent	语言	√	√	√

开发组织中间件是目前将组织概念引入实现阶段的主要方法,通过组织中间件,使得基于传统 Agent 语言开发的 Agent 具有组织感知能力(organization-awareness),从而 Agent 根据组织状态、规则等信息来调整自身的行为并修改组织的状态,如 JaCaMo;另外,组织中间件还可以作为一种外部机制来对 Agent 的行为进行管理和约束,如 AMELI 和 MACODO.Janus 和 powerJade 分别是基于 Java 和 Jade 开发的支持组织、角色等组织概念的运行平台,它为开发人员提供一系列开发包和运行时支持.

由表 2 可知,中间件可以有效地集成高层的组织模型与底层的程序设计语言,并对动态性提供了有效的支持.然而,由于组织层和 Agent 层缺乏统一的概念抽象,往往是基于已有的程序设计语言提供一组软件库或软件开发包来支持高层抽象,一方面导致代码比较复杂和不自然,如在 JaCaMo 中,组织的动态性逻辑也定义在 Agent 内部,从而使得组织动态性与 Agent 的内部规划混合在一起,造成 Agent 代码比较复杂;另一方面,可能受现有语言的限制无法充分发挥组织抽象的优势,如 powerJade 在构造复杂 MAS 时可能引起 Agent(在 powerJade 中,角色也是一类特殊的 Agent)数量迅速膨胀和 Agent 交互急剧增加.

相反地,程序设计语言通过显式地定义高层的组织抽象概念,为程序员提供更加直观、自然的语言设

© 中国科学院软件研究所 http://www.jos.org.cn

施.Fisher 等人[29]对 MetataM 中的 Agent 结构扩展了上下文(context)和内容(content),用来描述 Group.通过增加基本操作元语实现 Group 结构的动态变化,基于 Group 提供了动态交互机制,但该语言缺乏对组织概念的显式定义.2OPL(organization oriented programming)是一个基于规则的程序设计语言,主要是用来描述基于规范(norm)的 Agent 组织,文献[6]给出了 2OPL 完整的语法和语义.2OPL 是一个描述 Agent 组织的程序设计语言,具有比较完整的理论基础,但没有考虑与具体的 Agent 语言相结合,限制了它的实践性.另外,2OPL 侧重于研究组织对 Agent 的行为管理和约束,从而实现全局目标与 Agent 自主性之间的平衡,目前对 Agent 组织的动态性支持不足. simpAL[30]类似于本文工作,都是试图从软件工程和主流的程序设计方法和技术出发,设计面向 Agent 的程序设计语言.simplAL 与 Oragent 的不同之处主要表现在两个方面:(1) simpAL 基于 artifact 实现 Agent 的模块化开发, Agent 在运行时通过动态的获取 artifact 来增加自己的能力;而 Oragent 基于角色实现 Agent 的模块化开发,Agent 在运行时通过动态的扮演角色来实现能力的增减.(2) simpAL 基于 artifact 实现 Agent 之间的动态协同,而 Oragent 基于角色和事件实现 Agent 的动态交互.

6 结束语

目前,主流的 BDI 型面向 Agent 程序设计在构造和描述动态开放 MAS 时的局限性表现在:缺乏高层抽象;与基于组织抽象的分析和设计模型脱节和缺乏对动态性的有效支持.本文提出了基于组织的面向 Agent 程序设计,一方面,该方法将组织和角色作为一阶的编程实体,为开发人员提供高层、自然直观的语言设施;另一方面,角色扮演机制和基于角色的交互机制为 MAS 的动态性提供了支持.本文还设计了支持基于组织的面向 Agent 程序设计思想的编程语言——Oragent 的抽象语法和操作语义.基于迁移系统定义的操作语义,为进一步验证 Oragent 语言的安全性、一致性等性质提供了理论基础,同时也简化了支持 Oragent 程序运行的基础设施的开发.

Oragent 语言遵守和支持封装、模块化、重用和关注点分离等程序设计原则,分别表现在:Agent 行为都封装在 Agent 内部,Agent 之间只能通过消息进行通信;基于角色实现了 Agent 的模块化开发;角色、组织甚至 Agent 都是可重用的实体;基于角色定义交互协议实现了交互协议与运行实体(即 agent)的分离以及组织管理行为(定义在组织中)与业务行为(定义在角色中)的分离.不同于现有的面向 Agent 程序设计语言,Oragent 具有如下的新特点:

- 基于组织对 MAS 进行分解和组合.将软件系统看成是一个组织,组织又可以分解成一系列的子组织,通过组织的嵌套实现从顶向下、逐步求精的开发过程.
- 基于角色实现 Agent 的模块化开发.角色描述了 Agent 在特定组织中的状态和行为,即角色可以看作是在某组织中具有相同属性与行为的一组 Agent 的抽象.角色与 Agent 之间是扮演关系,不同于面向对象中对象与类的实例化关系,一个 Agent 可以同时扮演多个角色,并且在运行过程中 Agent 可以动态地改变其扮演的角色,从而呈现出不同的行为.
- 基于角色扮演机制实现 Agent 行为的动态组合.Agent 是一个开放的角色集,即 Agent 所扮演的角色不是在设计时固定好的,而是在运行过程中根据上下文环境的变化而动态变化的,因此,Agent 可通过扮演不同的角色实现其行为的动态组合.
- 基于角色实现 Agent 之间的动态交互.基于角色的交互使得交互协议与运行实体相分离,Agent 通过动态地扮演角色来动态地获取交互对象和协议.
- Group 结构的动态调整.Group 作为一阶的运行实体具有主动性,即 Group 可以根据当前的环境或其内部状态的变化动态地调整其结构.

本文定义了 Oragent 语言的抽象语法和操作语义,下一步的工作包括:完善 Oragent 语言的具体语法和操作语义;基于形式化的操作语义验证 Oragent 语言的安全性、一致性等;开发支持 Oragent 程序运行的基础设施和进行一系列案例的分析.

© 中国科学院软件研究所 http://www.jos.org.cn

References:

[1] Mei H, Liu XZ. Internetware: An emerging software paradigm for Internet computing. Journal of Computer Science and Technology, 2011,26(4):588−599. [doi: 10.1007/s11390-011-1159-y]

[2] Wang J, Shen R, Wang HM. A programming language approach to Internet-based virtual computing environment. Journal of Computer Science and Technology, 2011,26(4):600−615. [doi: 10.1007/s11390-011-1160-5]

[3] Zambonelli F, Parunak HVD. Towards a paradigm change in computer science and software engineering: A synthesis. The Knowledge Engineering Review, 2003,18(4):329−342. [doi: 10.1017/S0269888904000104]

[4] Jennings NR. On agent-based software engineering. Artificial Intelligence, 2000,117(2):277−296. [doi: 10.1016/S0004-3702(99)00107-1]

[5] Mao XJ. Agent-Oriented Software Development. Beijing: Tsinghua University Press, 2005 (in Chinese).

[6] Tinnemeier NAM. Organizing agent organizations: Syntax and operational semantics of an organization-oriented programming language [Ph.D. Thesis]. Utrecht: Utrecht University, 2011.

[7] Shoham Y. Agent-Oriented programming. Artificial Intelligence, 1993,60(1):51−92. [doi: 10.1016/0004-3702(93)90034-9]

[8] Bordini R, Braubach L, Dastani M, Fallah-Seghrouchni AE, Gomez-Sanz J, Leite J, Ot'Hare G, Pokahr A, Ricci A. A survey of programming languages and platforms for multi-agent systems. Informatica, 2006,30(1):33−44.

[9] Dastani M. Programming multi-agent systems. In: Weyns D, Müller JP, eds. Proc. of the Agent-Oriented Software Engineering 2011. Heidelberg: Springer-Verlag, 2012. 23−52.

[10] Bellifemine F, Caire G, Greenwood D. Developing Multi-Agent Systems with JADE. New York: John Wiley & Sons, 2007.

[11] Evertsz R, Fletcher M, Jones R, Jarvis J, Brusey J, Dance S. Implementing industrial multi-agent systems using JACK^{TM}. In: Dastani M, Dix J, Fallah-Seghrouchni AE, eds. Proc. of the Programming Multi-Agent Systems (ProMAS 2003). LNAI 3067, Heidelberg: Springer-Verlag, 2004. 18−48.

[12] Hindriks K. Programming rational agents in GOAL. In: Carbonell JG, Siekmann J, eds. Proc. of the Multi-Agent Programming: Languages and Tools and Applications. Heidelberg: Springer-Verlag, 2009. 119−157.

[13] Bordini R, Wooldridge M, Hőbner J. Programming Multi-Agent Systems in Agentspeak Using Jason. New York: John Wiley & Sons, 2007.

[14] De Giacomo G, Lesperance Y, Levesque HJ. Congolog, a concurrent programming language based on the situation calculus. Artificial Intelligence, 2000,121(1-2):109−169. [doi: 10.1016/S0004-3702(00)00034-5]

[15] Hindriks K, de Boer F, van der Hoek W, Meyer JJC. Agent programming in 3APL. Int'l Journal of Autonomous Agents and Multi-Agent Systems, 1999,2(4):357−401. [doi: 10.1023/A:1010084620690]

[16] Dastani M. 2APL: A practical agent programming language. Int'l Journal of Autonomous Agents and Multi-Agent Systems, 2008, 16(3):214−248. [doi: 10.1007/s10458-008-9036-y]

[17] Isern D, Sánchez D, Moreno A. Organizational structures supported by agent-oriented methodologies. The Journal of Systems and Software, 2011,84(2):169−184. [doi: 10.1016/j.jss.2010.09.005]

[18] Bordini1 RH, Dastani M, Winikoff M. Current issues in multi-agent systems development. In: Carbonell JG, Siekmann J, eds. Proc. of the Int'l Wrokshop on Engineering Societies in the Agents' World. LNAI 4457, Heidelberg: Springer-Verlag, 2007. 38−61. [doi: 10.1007/978-3-540-75524-1_3]

[19] Hu CY, Mao XJ, Chen Y, Zhou HP. OrgMAP: An organization-based approach for multi-agent programming. In: Conitzer V, Winikoff M, Padgham L, eds. Proc. of the 11th Int'l Conf. on Autonomous Agents and Multiagent Systems. New York: ACM Press, 2012. 1437−1438.

[20] Hu CY, Mao XJ, Zhou HP. Programming dynamics of multi-agent systems. In: Kinny D, Hsu JY, Governatori G, Ghose A, eds. Proc. of the 14th Int'l Conf. on Agents in Principle, Agents in Practice (PRIMA). Heidelberg: Springer-Verlag, 2011. 287−298. [doi: 10.1007/978-3-642-25044-6_23]

[21] Pokahr A, Braubach L, Lamersdorf W. Jadex: A BDI reasoning engine. In: Bordini RH, Dastani M, Dix J, Fallah-Seghrouchni AE, eds. Proc. of the Multi-Agent Programming: Languages, Platforms and Applications. Heidelberg: Springer-Verlag, 2005. 149−174.

[22] Tamai T, Ubayashi N, Ichiyama R. An adaptive object model with dynamic role binding. In: Proc. of the Int'l Conf. of Software Engineering. New York: ACM Press, 2005. 166−175. [doi: 10.1145/1062455.1062498]

© 中国科学院软件研究所 http://www.jos.org.cn

[23] Baldoni M, Boella G, van der Torre L. Interaction between objects in powerjava. Journal of Object Technology, 2007,2(2):1−26.

[24] Esteva M, Rodriguez-Aguilar JA, Rosel B, Joseph L. AMELI: An agent-based middleware for electronic institutions. In: Proc. of the Int'l Conf. on Autonomous Agents and Multiagent Systems. Washington: IEEE Computer Society, 2004. 236−243.

[25] Boissier O, Bordini RH, Hübner JF, Ricci A, Santi A. Multi-Agent oriented programming with JaCaMo. Science of Computer Programming, 2011. [doi: 10.1016/j.scico.2011.10.004]

[26] Gaud N, Galland S, Hilaire V, Koukam A. An organisational platform for holonic and multiagent systems. In: Hindriks E, Pokahr A, Sardina S, eds. Proc. of the 6th Workshop on Programming Multi-Agent Systems (ProMAS). Heidelberg: Springer-Verlag, 2009. 104−119. [doi: 10.1007/978-3-642-03278-3_7]

[27] Baldoni M, Boella G, Genovese V, Grenna R, van der Torre L. How to program organizations and roles in the JADE framework. In: Goebel R, *et al*., eds. Proc. of the German Conf. on Multi-Agent system Technologies (MATES). Heidelberg: Springer-Verlag, 2008. 25−36. [doi: 10.1007/978-3-540-87805-6_4]

[28] Weyns D, Heasevoets R, Helleboogh A, Holvoet T, Joosen W. MACODO: Middleware architecture for context-driven dynamic agent organizations. ACM Trans. on Autonomous and Adaptive Systems, 2009,5(1):1−25.

[29] Ghidini C, Hirsh B, Fisher M. Programming group computations. In: Proc. of the 1st European Workshop on Multi-Agent System (EUMAS 2003). 2003.

[30] Ricci A, Santi A. Designing a general-purpose programming language based on agent-oriented abstractions: The simpAL project. In: Proc. of the 1st Int'l Workshop on Programming based on Actors, Agents and Decentralized Control (SPLASH 2011). New York: ACM Press, 2011. 159−170. [doi: 10.1145/2095050.2095078]

附中文参考文献:

[5] 毛新军.面向主体软件开发.北京:清华大学出版社,2005.

胡翠云(1985－),女,河南辉县人,博士生,主要研究领域为面向 Agent 的软件工程.

陈寅(1988－),男,硕士生,主要研究领域为面向 Agent 的软件工程.

毛新军(1970－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件工程,智能 Agent 理论和技术,自适应和自组织系统.

www.jos.org.cn

© 中国科学院软件研究所 http://www.jos.org.cn