软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2012,23(11):2885−2904 [doi: 10.3724/SP.J.1001.2012.04297]

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

面向 Agent 程序设计的研究*

毛新军+, 胡翠云, 孙跃坤, 王怀民

(国防科学技术大学 计算机学院,湖南 长沙 410073)

Research on Agent-Oriented Programming

MAO Xin-Jun+, HU Cui-Yun, SUN Yue-Kun, WANG Huai-Min

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

+ Corresponding author: E-mail: xjmao@nudt.edu.cn

Mao XJ, Hu CY, Sun YK, Wang HM. Research on agent-oriented programming. *Journal of Software*, 2012, 23(11):2885−2904 (in Chinese). http://www.jos.org.cn/1000-9825/4297.htm

Abstract: Agent-Oriented programming (AOP) is inspired from the concepts and metaphors of multi-agent systems and borrows agent theory and technology to construct software systems. It represents a novel programming paradigm because its method, model, theory, and language are actually different from ones of existing mainstream programming technologies like OOP. As multi-agent system is considered as an effective technology to deal with the development of complex systems in open environment, AOP attracts many researchers and practitioners in the literatures of AI, software engineering and distributed computing. Significant progress has been made in the past twenty years. However, there are still great challenges to widely apply such a paradigm to support the development of complex systems in industry. In addition to using AI as basis, AOP should consider and borrow successful principles and practices of software engineering, especially existing programming paradigms, to promote its wide acceptance by software engineering practitioners. The aim of this paper is to give a systemic introduction of the research roadmap of AOP, investigate its state-of-the-art from a software engineering viewpoint by considering different programming levels of MAS and four research constituents of programming paradigms, including abstraction and model, mechanism and theory, language construct, and facility, supported platform. The survey intends to show the different research focuses and their changes in various stages. Moreover, the study identifies a number of issues and challenges in existing researches and prospect its future researches.

Key words: agent; multi-agent system; agent-oriented programming; organization; agent-oriented software engineering

摘 要: 面向 Agent 程序设计(agent-oriented programming,简称 AOP)基于多 Agent 系统的抽象和思想、借助于 Agent 理论和技术来支持软件系统的构造与实现,其程序设计思想、软件模型、基础理论和语言设施有别于现有主流程序设计技术,如 OOP,代表了一种新颖的程序设计范型.由于多 Agent 系统被视为支持开放环境下复杂软件系统

* 基金项目: 国家自然科学基金(61070034, 61133001, 90818028); 教育部博士点基金(20094307110007); 新世纪优秀人才支持计划

收稿时间: 2012-06-09; 定稿时间: 2012-08-15; jos 在线出版时间: 2012-08-29

CNKI 网络优先出版: 2012-08-29 10:58, http://www.cnki.net/kcms/detail/11.2560.TP.20120829.1058.002.html

开发的一种新颖、有效的技术手段,因而近年来 AOP 受到人工智能、软件工程和分布计算等领域研究学者和工程实践人员的高度关注,并在过去 20 年取得了重要进展.但是,无论在应对复杂多 Agent 系统开发方面,还是在大规模工业化应用等方面,AOP 的研究与实践都面临着严峻的挑战.作为一种程序设计范型,AOP 研究需要在交叉其他学科知识(如人工智能)的基础上,充分借鉴软件工程以及已有程序设计范型的原理、原则和成功实践,从而推动技术走向成熟并为广大工程实践人员所接受.通过对 AOP 研究历程的系统介绍,从软件工程的视点考虑 MAS 程序设计的不同层次,综述 AOP 在程序设计抽象与模型、机制与理论、语言与设施和支撑平台这 4 个方面的研究成果,展示不同时期 AOP 研究关注点的变化以及发展趋势,分析当前 AOP 研究与实践存在的问题和面临的挑战,并展望进一步的研究.

关键词: Agent;多 Agent 系统;面向 Agent 程序设计;组织;面向 Agent 软件工程

中图法分类号: TP18 **文献标识码**: A

随着互联网技术及其应用的迅猛发展,越来越多的软件密集型系统被部署在开放网络环境下,软件形态和特征发生了深刻的变化[1−4].其具体表现如下:

(1) 软件系统驻留在动态、开放、难控的网络环境中,环境变化不受系统完全控制且无法事先确定并对系统产生实质性影响.

(2) 软件系统规模超大[5,6],具有动态、异构、发散、跨域和跨组织等特点,需要具备诸如环境和上下文敏感、自主计算和协同、自适应、动态演化等方面的能力[3].

(3) 系统与环境之间、系统与系统之间、系统内部的各个成分之间将进行多样化、持续、复杂的交互和协同[1].

开放网络环境下软件系统的这些特点对构造这类系统的软件技术,尤其是程序设计技术提出了严峻的挑战[5],如主流的面向对象程序设计中,对象的被动性及交互的静态性和单一性等局限性无法为该类系统的复杂性特点提供有效支持.人们迫切地希望寻求软件新理念、新概念、新模型、新方法和新技术来支持这类系统的工程化开发[3].Zambonelli 等人认为,这种变化不是个别的,而是大范围的,并将对这类软件系统的建模和工程化技术产生实质性的影响[7].

Agent 是指驻留在环境下自主地执行动作并与其他 Agent 进行交互合作以满足设计目标的行为实体,驻留性、自主性和社会性是 Agent 的基本特征.多 Agent 系统(multi-agent system,简称 MAS)由一组相互交互和协同的 Agent 组成[8].由于 Agent 的驻留性和自主性可有效适应环境变化,使得系统具有更强的灵活性;Agent 的社会性是问题求解的有效手段,因而 MAS 被认为是支持复杂系统开发的一种重要而富有潜力的能行技术(enabling technology),提供了诸如高层抽象、问题分解、层次化、系统组织等工程化手段[9].一些应用实践展示了 MAS 在支持一类复杂系统开发方面的优势和潜力,如生产过程、智能控制、分布式网络管理、分布式控制系统、远程医疗系统等[9−12].

自 20 世纪 80 年代以来,Agent 理论与技术的研究在人工智能(artificial intelligence,简称 AI)等领域引起了人们的广泛关注,旨在构造具有自主和社会协同能力的智能计算实体,主要研究集中于智能 Agent 的理性决策理论和相关技术.到了 20 世纪 90 年代初期,人们在 Agent 的意图理论、认知模型、自主决策算法、通信语言及其语义和语用等方面取得了一系列重要进展,并将 Agent 理论和技术用于若干领域(如工业、航空、娱乐等)应用的开发,取得了一些成功实践,积累了经验.在此背景下,1993 年,Shoham 首先提出了面向 Agent 程序设计(agent-oriented programming,简称 AOP)的思想,他认为,AOP 是一种新的程序设计范型,其基本运行实体是自主 Agent,Agent 之间基于言语行为的高层交互方式实现社会协同[13].

随着 Agent 理论和技术研究、应用和实践的不断深入,人们意识到,要充分发挥 Agent 理论和技术在构造和实现开放复杂系统方面的优势和潜力,实现从学术研究到工业实践的过渡:一方面,必须从一些特定的 Agent 技术和具体的应用案例中产生一般性的、具有普遍意义的思想、原理、原则、方法、过程和模型,提供一种系统的、普适的方法以指导 MAS 的开发;另一方面,必须将 Agent 的概念、理论和技术与软件工程的基本思想、原

理和原则相结合,简化并实现高质量 MAS 程序的开发.在此背景下,面向 Agent 软件工程(agent oriented software engineering,简称 AOSE)应运而生[14].然而在 AOSE 提出的初期,人们更多地将注意力集中在 MAS 的需求、分析和设计等问题,经过几年的研究,虽然在面向 Agent 的需求、分析和设计方面取得了一系列的研究成果,但是 AOSE 并没有在开放复杂系统开发方面充分发挥其声称的优势和潜力,没有被工业界和软件工程领域的研究和工程实践人员完全接纳.导致这一状况的主要原因之一就是实现阶段缺乏有效的程序设计理论、语言和支撑平台,造成了软件的高层设计与底层程序设计脱节[2,15],严重阻碍了 AOSE 优势和潜能的发挥.因此,AOP 已成为 AOSE 走向工业化应用的瓶颈和主要技术挑战[16].

从软件工程和程序设计的视点来看,AOP 作为一种新的程序设计范型,需要为 MAS 的开发和实现提供系统的、有效的程序设计理论和技术,包括程序设计理论、模型、语言及开发与运行平台.虽然 AOP 的研究和发展已经历了 20 多年的历程,提出了几十种 AOP 语言,但是其研究与实践并不成熟,尤其是在应对动态开放 MAS 开发方面仍面临着严峻挑战.本文旨在通过对 AOP 研究历程的系统介绍,理清 AOP 的研究脉络,探求 AOP 当前的关注点和发展趋势;同时,从软件工程视点综述 AOP 研究的已有成果、存在问题和挑战,并讨论和展望其未来研究方向,期望对本领域的研究起到一定的指导作用.

本文第 1 节介绍 AOP 的研究历程.第 2 节从 MAS 需求和软件工程角度提出 AOP 现状的分析和评估框架.第 3 节基于我们提出的分析和评估框架对 AOP 的现有研究成果进行介绍和分析.第 4 节分析 AOP 研究与实践面临的问题和挑战,并讨论进一步的研究.最后.第 5 节总结全文.

1 AOP 的研究历程

Shoham 在 1993 年首次提出并设计了第一个 AOP 语言 AGENT-0,旨在推动计算的社会化[13].他认为,AOP 是以计算社会观为基础的新型程序设计范型,其程序设计对象(即 agent)是由各种认知部件(如信念、能力和承诺)构成的认知实体,Agent 间通过言语行为进行交互和合作.Shoham 的工作开启了 AOP 的研究热潮,在随后的几年时间内,人工智能领域的诸多学者提出了一系列 AOP 语言,如 Concurrent MetateM[17],PLACA[18],AOPLID[19],AgentSpeak(L)[20],3APL[21],ConGolog[22]等等,并围绕 AOP 语言的语义模型、交互行为的语义和语用、MAS 形式系统、AOP 语言的解释器和运行支撑环境等开展了深入研究.这一时期,AOP 的研究具有以下特点:

(1) 研究人员来自于人工智能领域;
(2) AOP 理论和语言基于人工智能领域已有成果,如可执行的时序逻辑、BDI 理论和模型、Prolog 语言等;
(3) 尽管具有非常扎实的理论基础,但绝大部分 AOP 语言仍然面向人工智能领域的学术界专业人士,支撑平台是原型化的;
(4) 没有得到充分的认可和接受,很少有工业界或者其他学科领域(如软件工程)的开发者利用 AOP 开发应用,尤其是软件工程领域学者并不认可这一程序设计思想并投入到这一研究方向.

经过若干年的活跃之后,AOP 的研究与实践在 21 世纪初期陷入了低潮.与此同时,为应对开放环境下复杂系统对传统软件工程带来的挑战,21 世纪初,人们开始尝试将 Agent 的概念、理论和技术引入到软件工程领域,并与软件工程的基本思想、原理和原则相结合,进而为开放环境下复杂软件系统的开发提供新颖的而有效的抽象、建模及实现方法和技术.Jennings 于 2000 年提出了 AOSE 的软件开发范型,将软件系统看成一个多 Agent 系统,将 Agent 作为自主软件实体的抽象,借助于组织结构和社会关系等高层抽象来描述软件实体之间复杂动态的交互关系[14].随后,AOSE 领域围绕面向 Agent 需求工程、面向 Agent 建模语言、面向 Agent 方法学、MAS 的软件体系结构和设计模式等内容开展了卓有成效的研究,提出了诸如 MaSE,Gaia,Tropos,i*,AML,AUML,MAS-ML,ODAM 等多达近百种的面向 Agent 建模语言和方法学[23].在 AOSE 提出初期,人们更多地将注意力集中于 MAS 的需求、分析和设计等问题而不是 AOP,但 AOSE 的出现及其研究成果给 AOP 研究带来了新的内涵和机遇,注入了新的活力.

2005 年以来,AOP 的研究再次引起了学者和工程实践人员的关注和重视,并取得了一系列的成果,包括

SLABSp[24]、面向组织的 AOP[25,26]、2APL[27]、面向 Norm 的 AOP[28,29]等.这一时期,AOP 研究的特点出现了一些微妙的变化:

(1) 软件工程领域的一些学者开始关注 AOP 并投入这一方向的研究.

(2) AOP 研究的着眼点发生了变化,从单纯关注理性、智能软件 Agent 的构造转向由诸多相互交互的 Agent 所构成的多 Agent 组织的开发[30].

(3) 人们不再单纯基于人工智能技术,而是开始从软件工程视角来研究和分析 AOP 的相关问题[31−33],并思考如何减少理论与实践之间的鸿沟[34],从而更好地推动这一技术在工业范围内的应用与实践.

(4) AOP 的应用范畴和研究背景也超出原先的智能、理性 MAS,转向更为广泛的、与主流计算技术相结合的应用领域,如网络环境下的软件密集型系统、自适应[35]和自组织系统[36]、面向服务计算的系统等等[16,37].

2 AOP 的研究内容和编程支持

本节从程序设计范型包含的研究内容以及 AOP 针对 MAS 特点和要求需提供的编程支持这两个角度来建立 AOP 研究与实践的分析框架,从而指导第 3 节对 AOP 研究现状的分析.

2.1 MAS的程序设计层次

开放环境下的复杂 MAS 通常具有层次化、发散、动态、自主、局部交互、全局涌现等特点[9,36].基于软件工程中分解、模块化和关注点分离等原则,从不同的观察尺度和角度对这类复杂系统进行分析、设计和实现可以简化其开发.文献[38]认为,AOSE 研究与实践应该从 micro,macro 和 meso 这 3 个不同观察尺度来分析和研究 MAS 开发需要解决的关键问题.面向 Agent 建模语言和方法学也通常针对 MAS 不同层次的分析和设计需求提供相应的建模能力[39].同样地,AOP 也需要在不同层次对 MAS 提供编程支持和核心程序设计技术[40−42].针对开放环境下复杂 MAS 的特点和需求,AOP 的研究与实践需提供以下 4 个层次的编程能力:个体 Agent 层、Agent 间交互层、环境层和多 Agent 组织层,从而为实现 MAS 的自主性(autonomy)、Agent 间社会性交互、环境驻留性以及全局 MAS 组织提供程序设计支持.

(1) 个体 Agent 层

AOP 将软件 Agent 作为目标软件系统的基本运行单元,Agent 的自主性是其区别于其他程序设计范型中软件实体(如 OOP 中的对象)的一个重要特征[8].因此,如何描述和实现 Agent 的自主性是 AOP 在个体 Agent 层的主要研究内容,包括:软件 Agent 的自主计算程序设计理论,描述了软件 Agent 自主行为的本质、机理、模型和性质,为软件 Agent 的实现奠定基础;软件 Agent 的体系结构;程序设计语言,提供显式的语言结构和设施并遵循诸如封装、重用、模块化、抽象和分类等软件工程机制来支持和简化对软件 Agent 自主性进行编程和实现;开发和运行支撑环境,为软件 Agent 程序代码的开发和运行提供有效支撑.

(2) Agent 间交互层

MAS 中每个 Agent 不是孤立和封闭的,相反地,由于每个 Agent 所拥有资源、信息、知识和能力的有限性,它们需要通过交互来实现 MAS 的整体设计目标.由于 Agent 的自主性,传统 OOP 中的消息传递机制并不能很好地支持 Agent 的交互.因此在该层次,AOP 主要研究和解决如何支持软件 Agent 间社会性交互的一系列问题:支持 Agent 间社会性交互的机制和机理;建立 MAS 社会性交互和协同的理论体系,包括交互语言的语义和语用,为 AOP 语言设计和支撑平台的实现奠定基础;在 AOP 语言中提供显式的语言结构和设施来支持对 MAS 中 Agent 间的交互进行编程;最后,需要提供有效的平台支持,如基于网络平台的消息和事件机制等.

(3) 环境层

无论是个体 Agent 还是 MAS,它们均驻留在特定的环境中,需要与环境进行双向的交互,即感知环境变化并通过行为实施等手段来影响环境.一方面,环境提供了 Agent 甚至 MAS 的生存条件和上下文,并为 Agent 访问资源以及实现它们之间的交互提供媒介,环境及其变化对个体 Agent 和 MAS 产生实质性的影响;另一方面,环境可以作为一阶的设计抽象和独立的构造块来支持 MAS 的构造,实现了关注点的分离,有助于控制 MAS 开发的复

杂度[43,44].在该层次,AOP 需研究和解决以下一系列问题:环境的抽象和模型;MAS 环境的形式和分类;软件 Agent 与环境的交互理论;环境的封装机制以及相应的程序设计语言结构和设施;支持环境作为一阶实体的运行支撑平台等.

(4) 多 Agent 组织层

MAS 中的软件 Agent 具有行为自主性,为了确保整个 MAS 以一种协调和一致的方式运行,进而实现系统的整体设计目标、获得预期的全局涌现行为,AOP 需要提供有效的机制对自主 Agent 的行为进行组织和管理.也就是说,提供技术手段在软件 Agent 行为的局部自主性和 MAS 的全局约束性两个方面进行权衡和折中.许多具有社会组织特征的软件系统[39]和自组织系统[35,36]对此提出了明确的需求.在该层次,AOP 需要研究和解决以下一系列问题:对 MAS 中个体 Agent 行为进行约束的机制、方式和手段;在多 Agent 组织层对个体 Agent 行为进行约束的程序设计语言结构、推理和表示方法;支持多 Agent 组织运行的模型和支撑平台等.

2.2 AOP的研究内容

从软件工程的视点,程序设计范型的研究主要包含 4 个方面的内容:程序设计的抽象与模型、程序设计机制与理论、程序设计语言与设施以及开发与运行支撑平台.作为一种新的程序设计范型,AOP 的研究同样包含了这 4 个方面的内容来支持 MAS 的构造和实现.

(1) 程序设计抽象与模型

程序设计抽象与模型方面的研究需要回答以下一系列问题:何为软件系统的模型,即构成目标软件系统的基本要素是什么?各个要素的功能以及它们之间的关系?采用什么样的思想、概念和抽象来解释和表示软件模型?等等.实际上,这些问题反映了程序设计的基本思想和指导原则,体现了对软件系统构造的基本理解和认识,从而为程序设计的基础理论、语言设施等方面的研究提供基本哲理.AOP 在该方面需要详细研究如何借助于 MAS 的思想和概念来回答这些问题.

(2) 程序设计机制与理论

程序设计机制和理论方面的研究需要回答以下一系列问题:提供什么样的机制以更好地支持软件系统的程序设计和运行?如何基于程序设计的抽象和模型来解释软件系统的结构和行为?基于特定抽象和模型的软件系统是如何运行的?具有什么样的性质和特征?等等.在该方面,AOP 的研究需要提供有效而多样化的机制来支持 MAS 不同层次的编程需求,并建立相应的程序设计理论基础以支持语言的设计和程序的运行.

(3) 程序设计语言与设施

程序设计语言与设施方面的研究需要回答以下一系列问题:如何提供语言结构和设施来表示程序设计的抽象和模型?如何设计相应的程序设计语言并为其提供严格的语法和语义定义,确保基于程序设计语言所编写的程序能够为计算机所理解,并可以在目标平台上运行?等等.在该方面,AOP 的研究需要提供良定义、具有严格语义和计算机可理解的语言工具来支持 MAS 的编程和实现.

(4) 程序的开发与运行

程序的开发与运行方面的研究需要回答以下一系列问题:如何构造和开发程序(即程序设计方法学)?如何进行程序的调试和测试?基于特定程序设计语言所编写的程序的运行模型是什么?包含哪些基本的运行要素?如何管理各运行要素?采用什么样的方式来支持程序的运行?如何为程序的运行提供支撑平台?等等.在该方面,AOP 的研究需要根据程序设计的基础理论和程序设计语言的形式语义,提供支持 MAS 运行以及对在不同层次对 MAS 软件要素进行开发、管理的基础设施.

3 AOP 的研究现状分析

现有的 AOP 综述文献主要对 AOP 语言的多样性(如声明型、命令型或者混合型)以及平台的支持能力进行分析[42,45].本节从软件工程角度,基于上一节提出的 AOP 研究分析框架对 AOP 的研究现状进行分析.

3.1 抽象与模型

3.1.1 个体 Agent 层

在个体 Agent 层,AOP 程序设计抽象与模型的研究旨在为实现 Agent 自主决策行为提供描述软件 Agent 内部构成的高层概念和抽象,其核心是如何描述 Agent 内部的决策部件,并建立相应的个体 Agent 软件模型.目前,AOP 支持的个体 Agent 软件模型主要有 4 种:知识型、认知型、反应型和混合型.

知识型模型将软件 Agent 视为一个知识系统,基于 AI 领域的信念(belief)、知识(knowledge)、分布式知识等抽象和概念来表示软件 Agent 内部构成.借助于逻辑工具,采用信念修改、知识推理、情景演算等来支持软件 Agent 的自主行为决策.支持知识型 Agent 模型的 AOP 语言包括 Golog,AGTGolog[46],ConGolog[22]等.

认知型模型将软件 Agent 视为一个认知系统,基于认知科学、大众心理学等学科的概念,如目标(goal)、期望(desire)、意图(intention)、规划(plan)等来描述软件 Agent 内部构成,借助于实用推理(practical reasoning)、BDI 和 KARO 等理论支持 Agent 的行为自主决策.至今提出的大部分 AOP 语言集成了这一思想,如 Agent-0[13],3APL[21],PLACA[18],AgentSpeak(L)[20],AOPLID[19],GOAL[47],Dribble[48],CLAIM[49],2APL[27]等.

Agent-0[13]将 Agent 视为是由信念、能力(capability)、承诺(commitment)和动作(action)构成的行为实体.PLACA[18]对 Agent-0 扩展了意图认知部件,以支持目标制导的行为.AgentSpeak(L)[20]和 CLAIM[49]完全基于 BDI 模型.3APL[21]的软件 Agent 模型建立在信念、目标和规划基础之上.另外,AOP 语言的目标概念可以分为过程性的和声明性的.过程性目标对应于具体的规划,强调的是 goal-to-do;声明性目标强调的是 goal-to-be.GOAL[47]引入了声明性目标概念,Dribble[48]和 2APL[27]实现了对两种类型目标的支持.

反应式模型将软件 Agent 视为一个反应式系统.它能感知环境及其自身发生的各种变化,并对这些变化进行及时(time-fashion)的响应和处理.反应型软件 Agent 模型通常由事件和反应式规则两部分构成,基于事件和事件处理机制来支持个体 Agent 的行为自主决策.例如,SLABSp[24]采用行为规则来定义软件 Agent 的行为,刻画了当某个特定场景被满足时软件 Agent 需实施的动作.

混合型软件模型综合上述多种不同的软件模型来支持软件 Agent 的构造与实现,并采用多样化的抽象和概念来描述构成软件模型的各个要素.例如,JAL[50]语言支持反应式模型和认知型模型.

3.1.2 Agent 交互层

在 Agent 交互层,AOP 程序设计抽象与模型的研究旨在解决个体 Agent 间的交互和协同问题,其核心是如何理解和认识 Agent 间的交互,采用什么样的抽象和概念来刻画 Agent 间的交互,借助于什么样的手段来实现 Agent 间的交互.目前,AOP 支持以下几种 Agent 交互方式:

(1) 基于言语行为的直接交互;

(2) 基于事件和消息机制的直接交互;

(3) 基于行为观察的间接交互.

许多 AOP 及其语言将 Agent 间的交互行为视为一种社会性的言语行为(speech act),通过提供多样化的言语行为(如基于 agent 通信语言 KQML)来支持 Agent 间的交互和协同.典型的 AOP 语言包括 Agent-0[13],Agent-K[51]等.Agent-0 提供了 INFORM,REQUEST,UNREQUEST 这 3 个言语行为,支持 Agent 间的通信.Agent-K[51]针对 Agent-0 中 Agent 交互原语进行了扩展,支持基于 KQML 进行通信.

基于事件和消息机制是指个体 Agent 通过接收和发送消息或者事件来进行交互.为此,AOP 语言提供了语言结构来对 MAS 中的消息、事件等进行显式定义.许多 AOP 语言基于事件和消息机制来支持 Agent 间交互.Concurrent MetateM[17]要求显式定义个体 Agent 允许接收和发送的消息,并通过广播方式支持不同 Agent 间的消息通信.ConGolog[22]引入 3 个特殊的动作支持 Agent 间的交互:SENDMSG 用于发送消息,SENSEMSG 用于感知和获取 Agent 消息队列中队首的消息,RMVMSG 用于删除 Agent 消息队列队首的消息.JAL[50]通过外部事件以及“@send”语句来实现 Agent 间的交互.

基于行为观察的交互机制是指个体 Agent 通过观察其他 Agent 的行为来获取用于交互的信息,从而实现间接的交互.SLABSp[25]根据所观察到的行为执行情况来判断特定的场景是否得到满足,进而触发相关行为规则

的执行.

3.1.3 环境层

在环境层,AOP 程序设计抽象与模型的研究旨在讨论何为 Agent 的环境,如何来表示和构造环境,如何来描述个体 Agent 与环境之间的交互等等.任何 Agent 都驻留在特定的环境中,根据观察层次的差异,Agent 驻留的环境可以是物理环境、通信环境和社会环境.早期 AOP 研究与实践很少考虑环境问题,并缺乏显式的程序设计支持,如果有,也是对环境的功能进行隐式的处理,并采用随意(ad-hoc)而不是系统的方式对 MAS 的环境进行编程.例如,ConGolog[22]借助于情景演算描述和建立关于环境的动态模型,将环境的状态描述为 Agent 动作执行的结果.Concurrent MetateM[17]在 Agent 程序的接口规约中,需要显式地定义 Agent 可以接受和发送的消息.

近年来,由于认识到环境在设计和实现复杂 MAS 中扮演着非常重要的角色,有关 MAS 环境抽象和模型以及相应的程序设计支持逐步引起了人们的关注和重视.MAS 规约语言 SLABS 显式地将 Agent 环境定义为 MAS 中与该 Agent 相关的一组 Agent 集合,但是基于 SLABS 的 AOP 语言 SLABSp[24]并没有引入环境的语言结构和设施.2APL[27]基于 Java 对象为 MAS 环境的描述提供了显式的语言结构:对象状态描述了环境状态,对象方法描述了 Agent 可以作用于环境的操作.文献[52]提出了 A&A(agents and artifacts)程序设计模型来支持 MAS 及其环境的建模与编程,基于制品(artifact)对 Agent 环境中的资源、服务及工具等进行抽象,制品可以互联组成 Agent 的工作空间(workspace).制品被看作是动态、被动的实体,Agent 通过构造、共享和使用工作空间中的制品来支持其工作活动(working activity).

3.1.4 多 Agent 组织层

在多 Agent 组织层,AOP 程序设计抽象和模型的研究关注如何提供有效的概念和模型来支持对 MAS 中 Agent 的行为进行有效的组织和调整,确保 MAS 协调和一致的运行,进而获得 MAS 全局涌现行为.从软件开发的角度来看,组织思想为 MAS 的设计提供了一种可行的分解方式,而且多样化的组织结构为 MAS 提供了灵活的结构,包括层次组织、Holonic 组织、联盟、团队、聚集、社会、联邦、市场及矩阵组织等[53].目前,AOP 支持的组织模型包括团队(team)[50,54]、法规化组织[28,29,55−57]、结构化组织[26,58]及混合型组织[29,59,60].

团队将相互合作完成一个复杂任务的多个 Agent 看成是一个团队.该模型往往通过对传统的软件 Agent 的 BDI 模型扩展联合意图、团队规划等要素来对团队进行建模和描述,从而指导或约束 Agent 的决策实现多 Agent 的合作.如,SimpleTeam[50]通过对 Jack 扩展角色、团队能力和团队规划等概念来描述多 Agent 团队,Agent 通过执行团队规划而实现任务合作.EAMCORE[54]基于团队规划和小组(group)信念来实现 Agent 之间的协同.文献[61]对单个 Agent 的 BDI 结构扩展了内容(content)和上下文(context)部件,从而实现和描述小组的概念.

法规化组织将组织看作是一组 Agent 和一个规则集,借鉴于社会学中规则制度等概念,基于法规来定义组织对 Agent 行为和交互的约束和限制.根据性质的不同,法规可分为义务(obligation)、许可(permission)和禁令(prohibition);基于约束的内容不同,法规可分为动作型法规和状态型法规;基于实施机制的不同,法规可分为强制(regimentation)型法规和惩罚(sanction)型法规.目前,支持法规化组织的典型 AOP 语言有 ISLANDER[62]、NOPL[28]和文献[63]所提出的 AOP 语言.ISLANDER[62]基于动作来定义法规,即允许或禁止 Agent 执行的动作,而且 ISLANDER 中的所有法规是不允许违背的.NOPL[28]是一个针对组织管理基础设施(organization management infrastructure,简称 OMI)的程序设计语言,而不是给程序员使用的语言,鉴于 OMI 的特殊性(违背法规会给平台带来致命的错误), NOPL 是一个比较简单的、只支持强制型义务的法规程序设计语言,即 NOPL 的法规只是描述了允许 Agent 执行的动作,而且这些法规都不允许违背.文献[63]基于状态定义法规,支持基于义务、许可、禁令这 3 种法规类型,并对违背法规的 Agent 提供了惩罚机制.

结构化组织将 MAS 看作是一个具有一定结构的组织,基于角色和角色关系来定义组织结构,同时,角色和角色关系也定义了对其扮演者 Agent 的约束和限制.文献[64]对 3APL 扩展了角色的概念,角色封装了一组法规规则、一组期望的目标和 Agent 扮演该角色期望获得的信息,而 Agent 看作是一个动态的角色集,通过 enact, deact,activate 和 deactivate 这 4 个操作来实现角色的动态性.文献[65]使用模块来统一描述角色和 Agent profile,并提供了显式的操作,从而允许程序员显式的进行角色操作.PowerJade[58]对 Jade 扩展了一阶的组织和角色抽

象,组织作为一阶实体为多 Agent 之间提供一种外部协同机制,角色封装了 Agent 在组织上下文中可使用的能力.simpAL[66]基于角色来抽象 Agent 的任务,基于组织来对 Agent 和 Agent 环境实体(制品)进行组织,Agent 为其所扮演的角色提供任务的实现,即规划.

混合型组织综合了法规和组织结构的思想来描述多 Agent 组织对 Agent 的约束和限制,如文献[29]对 2APL 扩展了角色和法规的概念,用来支持 MAS 的开发.其中,角色用来描述针对某一类 Agent 的约束,而法规用来描述不同类型 Agent 或 Agent 交互的约束.S-/J-Moise[60]基于 Moise 模型从功能、结构和义务(obligation)这 3 个方面来描述和规约多 Agent 组织.功能方面,Mosie 基于目标树来描述组织的目标;在结构方面,Moise 基于角色描述 Agent 应该完成的任务及基于角色关系来描述 Agent 之间的交互;在义务方面,基于法规描述了组织目标与角色任务之间的关联关系.S-/J-Moise 允许 Agent 违背法规,但组织没有采取进一步的措施,如惩罚等.2OPL[59]是一个基于规则的程序设计语言,基于角色定义了组织结构,支持义务、许可和禁令这 3 种类型的法规,并提供了强制和惩罚两种法规实施机制.另外,2OPL 还提供了对法规演化的支持来适应不确定的环境.

3.2 机制与理论

基于抽象和模型,AOP 需要提供相关的机制和理论来支持 MAS 的构造、实现与运行,为 AOP 语言的设计以及支撑平台的开发奠定基础.许多 AOP 语言借助于 AOP 理论基础和各种形式工具(如迁移规则、Z 规约)定义语言的形式操作语义,如 2APL[27],3APL[21,67],ConGolog[22],GOAL[47]等.不同于 OOP,AOP 交叉多学科知识开展机制和理论研究,包括认知科学、大众心理学、人工智能、自然语言处理、软件工程等.

3.2.1 个体 Agent 层

在个体 Agent 层,AOP 机制及理论研究 Agent 的自主决策机制和理论.目前,这方面比较有影响的工作包括认知计算[20]、格局演算(situation calculus)[22,68]、情景演算(scenario calculus)[24]、Ambient 演算[49]等理论.

认知计算的机制与理论为认知型 AOP 语言的设计与实现奠定了基础,其典型工作是 BDI 理论[20].早期的工作源于 Cohen 和 Levesque 等人的意图理论.他们建立了理性 Agent 自主决策与诸如信念、目标和意图之间的内在关系.随后,许多学者在此基础上做了进一步的发展:

(1) Singh 认为,意图和 Know-How 是支持理性 Agent 自主决策的两个正交概念,给出了它们的形式语义,建立了 MAS 计算的意图和 Know-How 理论,并用于解释和规约 Agent 交互言语行为的语义和语用.

(2) Shoham 基于一个多模态词的形式化语言对构成 Agent 的信念、承诺、能力和动作等进行了严格的定义,讨论了由这些部件构成的 Agent 具有的性质和特征,如内部一致性、忠信性、自省性、持续性等[13],为 Agent-0 及其解释器的设计奠定基础.

(3) Rao 和 Georgeff 提出 BDI 模型及理论是这方面最有影响的工作之一.它成为诸多 AOP 语言的理论基础,如 AgentSpeak(L)[20],2APL[27],3APL[21]等.

格局演算是由 Lespérance 和 Levesque 等人提出的一种用于表示动态变化世界的一阶逻辑语言及其相应的理论体系,以支持 Agent 对其动作的推理和执行[22],从而为知识型 AOP 语言提供理论基础.格局演算认为:世界变化是动作实施的结果,世界变化的历史对应于动作序列;可以用一阶逻辑中的函词表示动作,项来表示格局.例如,$do(a,s)$表示在情景 s 下执行动作 a 而导致的新格局.在格局演算中,对一个动作的描述主要包括以下 3 个方面:(1) 前件公理(precondition axiom):描述动作执行所需的前提条件;(2) 效应公理(effect axiom):描述动作执行后对世界产生的影响;(3) 后继状态公理(successor state axiom):描述动作执行将不会影响哪些世界状态.格局演算支持将过程性控制语句解释为格局演算中的复杂行动,利用格局演算的推理机制将其分解为一组可执行的原子行动序列.基于格局演算理论,人们提出了一系列具有不同特点的 AOP 语言,包括 GOLOG、GTGolog[46]、CONGOLOG[22]、AOPLID[19]、文献[69]等,并针对不同语言的特点对经典格局演算进行了多样化的扩展.例如,CONGOLOG 引入了并发动作执行机制以及 Agent 间通信机制,以支持 MAS 中 Agent 行为的并发执行以及 Agent 间的交互.AOPLID 提出开放格局演算理论 OSC(open situation calculus).OSC 中的格局是指 Agent 的认知状态,区分两种情景:一种是 Agent 在离线规划时所想象的可能格局,另一种是 Agent 在执行过程中实际改变认知状态后的实格局.文献[69]基于格局演算定义了带优先级目标的程序设计语言的理论基础.

Zhu 在其 MAS 形式规约语言 SLABS 及其受限的程序设计语言 SLABSp[24]中引入了情景概念和情景演算,以支持对 MAS 的格局进行规约,进而触发 Agent 程序代码中相关行为的实施.情景演算提供了足够的表达能力来描述 Agent 及其环境中其他 Agent 的行为模式,如,至少有一个扮演特定 Caste 的 Agent 实施了某个行为、扮演某个 Caste 的 Agent 具有某种行为等.

Fallah-Seghrouchni 和 Suna 在 CLAIM[49]中基于 Ambient 演算的理论支持移动 Agent 的迁移.Ambient 被认为计算发生的有效空间,一个 Ambient 包含了用于控制访问的名字、一组局部进程和一级子 Ambient.Ambient 层次的组织成一个树型结构,并提供了一系列支持移动性的能力:in,out,open(·),acid(·),mv in(·)和 mv out(·).

3.2.2 Agent 交互层

在 Agent 交互层,AOP 机制及理论研究需要针对具体的 Agent 交互抽象和模型,基于个体 Agent 的机制与理论,建立支持 AOP 的 Agent 交互理论和方法,尤其是给出交互行为的形式语言,从而为 Agent 解释和分析所接收的交互行为的语义和语用奠定基础.

针对基于言语行为的交互形式及其语言设施,人们根据言语行为理论提出了言语行为三分说,即一个言语行为涉及 3 方面的活动,包括以言指事(locutionary act),指会话的物理动作;以言行事(illocutionary act),指言者传递给听者的意图;以言成事(perlocutionary act),指言语行为的结果.这方面人们通常基于 Agent 的知识模型和认知模型,通过分析言语行为对参与交互 Agent 内部状态的影响来解释言语行为的语义和语用,典型的工作包括 FIPA ACL 和 KQML 的语义定义、Singh 关于一组交互言语行为的语义定义等.Agent-0 基于 Agent 内部的信念、承诺等来解释和定义诸如 INFORM,REQUEST 等交互行为的语义.

在基于事件/消息通知机制的交互方式中,人们在 AOP 语言设计中提出了一系列的机制来支持 Agent 之间的交互:(1) 消息广播和过滤机制.例如,Concurrent MetateM[17]支持 Agent 之间通过广播消息进行交互,在对 Agent 进行编程时,需要在其接口规约中显式定义它可以接收的消息和可以发送的消息.每个 Agent 都有一个消息队列,如果 Agent 接收的消息在其接口规约中做了定义,那么该消息将被置于消息队列中等待进一步处理;否则,Agent 将抛弃该消息;(2) 事件/消息订阅机制.Agent 可以通过订阅机制来显式地订阅/退订它所感兴趣的事件和消息.这一机制广泛应用于诸多 AOP 语言及其实现平台的设计,如 JAL[50]等等.

3.2.3 环境层

在环境层,AOP 机制及理论的研究需要为环境的管理与实现、MAS 与环境之间的交互提供方法和手段.尽管越来越多的研究认识到环境在 MAS 中的重要性,并日益强调需要将环境作为一阶(first-class)的编程对象来支持 AOP 语言的设计与实现,但是目前,围绕环境机制和理论的研究仍然非常有限.

2APL[27]引入了环境接口(environment interface)机制来支持环境实体的创建,任何实现了环境接口的对象类均被视为环境.A&A 程序设计模型借鉴人类社会学中的活动理论(activity theory),引入了环境制品的概念来支持 MAS 环境描述和构造[52],并采用工作空间的方式来支持对环境制品进行组织和管理.SLABSp[24]通过引入 Scenario 来刻画环境中各个 Agent 的可观察行为,进而来描述环境的变化.

在 Agent 与环境交互方面,典型的理论机制是影响-反应(influence-reaction)模型[70,71],其基本思想是,Agent 不能直接改变环境的状态,只能影响环境的动态性.即,Agent 决定做什么动作,而环境来决定动作的结果. Agent 与环境的交互包含了两个方面:(1) Agent 可以作用于环境的动作,包括感知动作;(2) 环境通过事件与 Agent 进行交互.

3.2.4 多 Agent 组织层

在多 Agent 组织层,AOP 的研究需要提供显式的机制和理论来解释、分析、规约以及实现多 Agent 组织层的结构和行为、组织层对 Agent 的管理和调整以及 Agent 对组织层的感知和推理;同时,为 AOP 语言的设计以及支撑平台的开发提供理论基础.早期的 AOP 主要基于共同目标、联合意图等心智概念来实现多 Agent 的协同和合作,即,对传统的 BDI 方法扩展描述团队心智状态.一组 Agent 通过拥有共同的信念和承诺来合作完成团队任务,如 TEAMCORE[54].在这类方法中,组织是隐式的,Agent 通过共同目标或联合意图来实现 Agent 的合作和协同.

近年来,越来越多的研究人员认为,组织概念应该作为一阶的编程对象来支持 MAS 的开发和实现.目前,在组织层,AOP 机制及理论研究主要关注组织敏感(organization-aware)Agent 的实现机制以及法规的实施机制等.

组织参与(organizational participation)和组织规约的理解是实现组织敏感 Agent 的两个关键性问题[72].组织参与的主要机制之一是角色扮演机制,即,一个 Agent 通过扮演组织中的角色而实现加入组织.角色扮演动作是一个复杂动作,由组织和 Agent 共同完成[64].目前,AOP 中角色扮演机制主要存在两种方式:一是无约束的角色扮演,在这种方法中,角色对 Agent 没有要求,Agent 可以直接扮演角色,如 2OPL[59],S-/J-Moise[60];二是基于协商的角色扮演,角色对其扮演者提供了一系列的权力(power)和提出了一系列的能力(capability)需求,Agent 只有具备了角色的需求才能扮演角色,Agent 一旦扮演了某个角色则拥有了该角色定义的所有权力,如 powerJade[58].组织规约的理解主要指组织中定义的对 Agent 行为和交互的约束如何影响 Agent 的决策.在 AOP 中,Agent 对组织规约的理解存在两种方式:一种是将组织规约转换为 Agent 的内部状态,如,J-Moise[60]将组织状态转换为 Agent 的信念,组织状态的变化将会引起 Agent 信念的变化;另一种是修改 Agent 的决策逻辑,如在 BDI 模型中增加组织部件.就我们所知,目前还没有支持该方法的运行平台和程序设计语言.

法规的实施方式是法规化系统(normative system)的关键技术理论,目前,AOP 中主要存在两类实施方式:强制(regimentation)型和惩罚(sanction/enforcement)型.强制型法规不允许 Agent 对法规的违背,其实现机制存在两种:一种是 Agent 将法规转换为内部状态,如信念、目标,即,Agent 具有法规感知能力;另一种是由组织或中间件对 Agent 的行为进行监控,如果发现违规行为,则通过事件机制通知 Agent.惩罚型法规允许 Agent 对法规的违背,保证了 Agent 的自主性,其基本理论主要是基于 John Searle 的社会实体理论(social reality theory)[73].社会实体理论的核心思想是:将系统的状态分为物理(brute)状态和制度(institutional)状态,其中,物理状态是指世界上可观测的事实,制度状态是指特定社会中存在的风俗习惯、规则和法规等;然后,通过引入 count-as 操作算子来判断当前的物理状态或制度状态是否违背了法规.该工作的典型代表包括 2OPL[59],NOPL[28]等.

另外,为了支持可计算的组织,目前,一些研究人员对组织的形式化模型和理论展开了相关的研究工作.如:2OPL[59]基于义务逻辑(deontic logic)给出了职责、允许和禁令的形式化操作语义,并对义务逻辑进行扩展支持带条件的职责和禁令;文献[74]基于事件演算(event calculus)来规约法规化系统,为带条件的职责、允许和禁令提供了直接的描述;文献[75]将组织规则或法规转换为 Agent 决策推理的事件处理(event-processing)规则,从而实现了组织对 Agent 的管理与约束.

3.3 语言与设施

根据观察角度和研究目的的差异,下面从多个不同的方面来分析现有的 AOP 语言及其设施.

第一,根据语法、语义和语用的差异性,现有的 AOP 语言主要分为两类:(1) 基于逻辑程序设计语言,如 Concurrent MetateM[17],ConGolog[22],Agent-0[13]等.它们在语法上采用诸如 LISP 和 Prolog 的表示形式,在语义上建立在逻辑程序设计和 AOP 程序设计理论基础之上.(2) 基于 OOP 语言(如 Java),如 JAL[50]等.它们在语法上采用与 Java 语言类似或者相融合的表示形式,在语义上借助于 AOP 的程序设计理论,采用诸如迁移系统(transition system)等给出系统的操作语义,如 CLAIM[49].

第二,根据语言设施表示程序形式的差异,现有 AOP 语言可划分为声明型语言、命令型语言和混合型语言[42,45].声明型 AOP 语言支持对个体 Agent 层、环境层、MAS 组织层的软件模型要素进行表示和推理,如信念、知识、目标、意图、组织法规等,它们在语法上通常采用类 LISP 或者 Prolog 的形式,具有非常严格、坚实的形式语义基础(如 BDI Agent 理论[20]、Ambient 演算[49]、Situation 演算[22]等),典型语言包括 Concurrent MetateM[17],CLAIM[49],AgentSpeak(L)[20],ConGolog[22]等.命令型语言支持个体 Agent 层、Agent 交互层、环境层和 MAS 组织层的软件模型要素进行表示与实现,如任务、服务、行为、过程的表示和实现.它们在语法上通常是对现有主流命令型程序设计语言如 Java 的扩展,在语义上通常具有非常严格的操作语义,典型语言包括 JAL[50],SLABSp[24]等.混合型的 AOP 语言同时兼有声明型语言和命令型语言的成分,分别采用不同的方式对个体 Agent 层、Agent 交互层、环境层、MAS 组织层的软件模型要素进行表示、推理和实现,它们通常将多种不同的程序设计抽象、语言结构和设施融合在一个 AOP 语言之中,并提供相应的语义基础,如,基于 Z 规约的 3APL

形式框架[67]、基于迁移系统的 2APL 操作语义[27].

第三,根据所编写的 MAS 程序执行方式的差异,现有 AOP 语言可分为解释型和编译型.解释型 AOP 语言借助于解释器对 AOP 语言所编写的程序进行分析、解释和执行,声明型 AOP 语言通常采用解释的方式来执行程序.典型的解释型 AOP 语言包括 Agent-0[13],Concurrent MetateM[17]等.编译型 AOP 语言首先对编写的程序进行编译,生成目标机器代码或者某些中间代码,然后借助于运行平台支撑目标软件的运行.命令型 AOP 语言通常采用编译的方式来执行程序,典型的编译型 AOP 语言包括 SLABSp[24],JAL[50],2APL[27],3APL[21]等.

此外,根据语言的应用领域与范围,AOP 语言可分为通用型和专用型.通用型 AOP 语言所提供的语言设施和编程能力使得它们可以应用于诸多应用领域,通常声称不限定领域,但实际上,由于受程序设计的抽象和软件模型、语言结构和设施等的限制,它们仍然受限于或者更适合于某些特定的应用领域.例如,Agent-0[13],Concurrent MetateM[17],ConGolog[22]等似乎更适合于智能软件的开发,而不适合涉及大量数据处理的应用.专用型 AOP 语言考虑了特定应用领域的特点及其对程序设计的需求,并在语言设计中引入了相关的抽象、模型要素、语言设施和实现机制等,如支持自适应编程的语言 GTGolog[46]、面向移动计算的语言 CLAIM[49]等.

限于篇幅,本节不对现有 AOP 语言的细节(如程序设计抽象、语言设施和机制、语言的语义等)进行详述,而是列举现有主要 AOP 语言及其在个体 Agent 层、Agent 交互层、环境层、多 Agent 组织层的编程支持程度(见表 1),从中发现不同时期 AOP 语言研究关注点的变化及其对不同层次编程能力的支持.

Table 1 Lists of current AOP languages

表 1 现有的 AOP 语言列表

名称	年份	提出者代表	各层次支持				说明
			A	I	E	O	
Agent-0[13]	1993	Yoav Shoham	+	–	/	/	Agent 认知模型,简单的言语行为交互
Concurrent MetateM[17]	1994	Michael Fisher	+	–	–	/	Agent 认知模型,可执行线性时序逻辑语言,基于接口的交互
Agent-K[51]	1994	W.H.E. Davies	+	+	/	/	对 Agent-0 扩展,增强了基于 KQML 的交互和通信
PLACA[18]	1995	S.R. Thomas	+	–	/	/	扩展 Agent-0,引入了动机类别的认知部件,强化目标制导决策
AgentSpeak(L)[20]	1996	A. Rao	+	–	/	/	Agent 的 BDI 软件模型,受限的一阶逻辑语言
3APL[21]	1999	Koen. Hindriks	+	–	/	/	Agent 的 BDI 模型,结合命令式和逻辑程序设计思想
ConGolog[22]	2000	Yves Lespérance	+	–	–	/	基于情景演算理论,融合过程式与逻辑程序设计
GOAL[47]	2001	K. Hindriks	+	–	/	/	Agent 认知模型,引入描述性目标,区分知识和信念
DALI[76]	2002	S. Costantini	+	+	–	/	基于Horn Clauses和Least Herbrand Models,支持主动和反应式行为,基于事件的交互
FLUX[77]	2002	M. Thielscher	+	/	/	/	基于Agent认知模型的逻辑程序设计语言,流演算(fluent calculus)
CLAIM[49]	2003	A. El Fallah-Seghrouchni	+	+	–	/	基于 Ambient Calculus 理论,采 BDI 模型,支持移动、自适应和重配置
Dribble[48]	2003	Birna van Riemsdijk	+	–	/	/	Agent 认知模型,区分过程性目标和描述性目标
AOPLID[19]	2003	郭磊	+	–	–	/	Agent 的认知模型,对 GOLOG 语言的扩展
Go![78]	2004	K. Clark	+	–	/	/	多线程、强类型,基于行为规则的软件模型
AF-APL[79]	2004	R. Collier, R. Ross	+	–	–	–	认知Agent模型(信念、目标和承诺),显式的环境操作算子
SPARK[80]	2004	David Morley	+	/	–	/	BDI 模型和反应式模型
JAL[50]	2005	Winikoff. M	+	+	–	/	BDI和反应式的软件Agent模型,对OOP语言的扩展,采用事件方式进行交互
Jason[81]	2005	Rafael H. Bordini	+	+	+	/	BDI 模型,言语行为交互,支持环境编程
SLABSp[24]	2005	Ji Wang, Hong Zhu	+	–	+	/	基于caste和scenario机制,采用行为规则描述个体Agent行为
AGTGolog[46]	2006	Alberto Finzi	+	–	/	/	对 Golog 的扩展,支持自适应 MAS 的编程
2APL[27]	2008	Mehdi Dastani	+	–	+	/	Agent 的 BDI 模型,支持环境编程,融合命令式和描述式,支持个体 Agent 和 MAS 二层的关注点分离
2OPL[59]	2009	N. Tinnemeier	+	–	–	+	基于规则的程序设计语言,支持基于角色和 Norm 的 MAS 组织编程

Table 1 Lists of current AOP languages (Continued)

表 1 现有的 AOP 语言列表(续)

名称	年份	提出者代表	各层次支持				说明
			A	I	E	O	
NOPL[28]	2010	Jomi F. Hübner	/	/	/	+	Normative程序设计语言,基于组织管理基础设施ORA4MAS
Meta-APL[82]	2011	Thu Trang Doan 等人	+	/	–	/	BDI 模型,显式的环境操作,支持对决策策略的编程
simpAL[66]	2011	Alessandro Ricci 等人	+	–	+	+	集成了 Agent、环境和组织的程序设计语言,对 Java 的扩展
Oragent[83]	2011	胡翠云等人	+	–	/	+	反应式 Agent,支持基于角色的 MAS 组织编程

说明:–,+,/分别表示弱、强、不支持或者不可知.

3.4 开发与运行

开发支持环境和运行时环境,是 AOP 语言应用和实现的基础和前提.AOP 的开发支持环境是指,借助于软件工具或者集成开发环境(integrated development environment,简称 IDE)等形式,在软件设计阶段为程序员的软件开发活动如编码、调试、部署等提供工具支持,从而简化了程序员的开发和维护任务,提高面向 Agent 程序设计的质量和效率.开发支持环境的功能主要包括程序编辑、管理(如以项目的方式组织程序、程序的版本管理、可重用软件的部署和检索、程序的 Check-in 和 Check-out 等等)、AOP 程序的部署、测试和调试等等.AOP 的运行时环境是指为 AOP 语言所编写程序的运行提供支持,包括程序代码的编译器或解释器、MAS 软件模型要素的管理系统、支持软件要素交互的通信设施以及运行引擎等.

3.4.1 AOP 开发支持环境

文献[45]将现有支持 AOP 的 IDE 功能分为以下 5 种类型:(1) 项目管理,即根据开发者的需要组织项目结构;(2) 创建和编辑源文件,即提供程序的结构视图、支持快速和便捷的导航、程序编辑、在线的语法错误检查、程序代码的存储等;(3) 重构,即支持快速和可靠的代码重组织;(4) 配置和运行,即 IDE 集成了 AOP 运行时支持平台,允许在 IDE 内部运行 AOP 程序;(5) 测试和调试,即支持对编写的程序进行单元测试以发现程序中的错误,并通过调试的方式发现和纠正错误.

尽管目前涌现出大量的 AOP 语言,但只有很少一部分 AOP 语言提供了 IDE,典型的工作有 3APL,2APL,Jason[81],JACK[50]和 AgentFactory[84].3APL 和 2APL 的 IDE 都提供了代码编辑器、Agent 内部状态和 Agent 通信消息的监测器.Jason[81]的 IDE 也提供了代码编辑器(包括多 Agent 的配置文件和单个 Agent 的代码),Agent 内部状态的监测器并支持 Agent 的分布式管理.JDE(JACK development environmnet)[50]是为 JACK 平台提供的一个比较完整的商业化 IDE,提供了项目管理、代码编辑以及调试等工具支持.Agent Factory[84]支持语言 AF-APL[79]的 IDE,提供了基本的项目管理、代码编辑器和 Agent 要素的配置等功能,同时集成了 AF-APL 解释器.DECAF(distributed, environment-centered agent framework)(http://www.few.vu.nl/~wai/demas/tools2.html)是一个通用的 Agent 开发平台,独立于具体的 AOP 语言,采用中间件的形式并提供了一组工具(如规划编辑器、Agent 构造工具、测试生成工具)以支持 Agent 初始化、分发、执行、规划、名字服务、匹配和代理等功能.

3.4.2 AOP 运行时环境

AOP 运行时环境包括两个部分:一部分是用来处理 AOP 代码的编译器或解释器;另一部分是实现了 AOP 语言语义的运行支撑平台[45],主要负责为 AOP 程序的运行提供管理和通信等基础设施.

目前,针对个体 Agent 的程序设计语言大多数都提供了解释器或编译器来运行或处理其源代码.例如,AgentSpeak(L)的解释器 Jason 支持 Agent 之间基于言语行为的交互、基于 AgentSpeak(L)软件模型的自主行为决策、描述和实现 MAS 环境模型等.JAL[50]和 SLABSp[24]运行支撑环境都建立在 Java 虚拟机之上,AOP 语言编写的程序经过编译后生成相应的 Java 代码,通过对 Java 虚拟机进行必要的扩展或引入 MAS 的运行引擎来支持 MAS 程序的执行.在环境层,目前缺乏专门的程序设计语言,如 2APL[27],Jason[81]等语言直接采用 Java 语言来描述环境,而 CARTAGO[52]则是基于 Java 提供了一些支持环境实体——制品的类库.在组织层,AOP 语言一般采用解释执行的方式,一种是由 Agent 对组织描述进行解释执行,如 J-Moise[60];另一种是开发专门的中间件或解释

器对组织代码进行解释执行,如 2OPL[59]则是基于解释器来解释执行组织、角色和法规代码.

AOP 运行支撑平台是 AOP 程序运行的基础,其主要功能可以分为如下 3 类:

(1) 软件实体管理

不同于 OOP 程序设计,AOP 程序中可能包括了多种类型的软件实体,如个体 Agent,环境以及组织层的角色、法规等,因此,AOP 运行支撑平台需要为 AOP 程序中的各类软件实体的运行提供多种形式的管理和维护支持,包括生命周期管理、命名服务和实体的发现、查找(如黄页、白页服务)以及各类软件实体的组织等.此外,一些复杂 MAS 中的软件 Agent 还具有诸多自适应、自管理、移动等方面的特点,为此需要对这类特殊软件 Agent 的适应性行为、变化的状态(如移动导致的地址、位置和环境的变化)等提供有效的管理.

(2) 通信基础设施

MAS 中存在多种形式的交互,包括 Agent 与 Agent 的交互、Agent 与环境的交互和 Agent 与组织层的交互,因此,AOP 运行支撑平台需要为 MAS 的运行提供一组基础设施,包括事件管理和服务设施(如事件订阅)、Agent 间交互和协同设施(如支持 Agent 间基于 FIPA ACL 或者 KQML 的通信)、Agent 与环境的交互设施、Agent 与组织之间的交互和感知设施等.CLAIM[49]借助于 SyMPA 平台支持 CLAIM 程序的运行,SyMPA 平台提供了必要的机制和功能来实现对 Agent、通信、移动、安全和容错等进行管理,它遵循 OMG 的 MASIF 标准.

(3) MAS 程序运行

AOP 运行支撑平台需要基于 AOP 语言的形式语义为 MAS 程序的运行提供支持:在个体 Agent 层,需要支持对构成个体 Agent 内部各个成分的解释和推理,如支持信念、目标和意图的修改、规划的生成和执行、行为规则的触发、选择和实施;在 Agent 交互层,需要支持对 Agent 交互内容的解释和分析,并引发个体 Agent 内部状态的变化;在环境层,需要支持 Agent 对环境的感知、Agent 对环境的适应性行为和对环境的影响;在 MAS 组织层,需要对 MAS 的组织法规进行解释和推理,并对 MAS 组织中个体的行为进行监控.如果需要,对违背法规的 Agent 实施惩罚措施等.

目前,支持个体 Agent 的 AOP 语言大多数都提供了平台支撑,其中,JADE[85]平台作为开源的 MAS 运行平台,提供了较好的 Agent 管理服务和通信基础设施,很多语言的支撑平台都是基于 JADE 进行扩展,如 2APL[27],Jason[81]等.环境层和组织层的程序设计往往独立于个体 Agent 层,即描述环境和组织的程序设计语言与个体 Agent 语言缺乏统一的语法和语义,因此,目前研究工作主要基于中间件的技术实现个体 Agent 与环境层或组织层进行交互.例如,S-MOISE+[60]是一个支持 MAS 组织层次的中间件,提供了通信层、组织层的功能,允许 Agent 改变其组织,ORG4MAS[86]是 S-MOISE+是基于制品的实现版本;JoCoMo[87]则是集成了个体 Agent、环境和组织等 3 个维度的中间件支持平台,基于 CARTAGO[52]提供了对环境的生命周期和动态性的管理,基于 S-/J-Moise[60]实现了对组织层的描述和推理,基于 Jason[81]实现了个体 Agent 的管理和通信.

4 AOP 研究与实践面临的问题与展望

本节讨论当前 AOP 研究与实践面临的问题和挑战,分析这一新颖程序设计范型要为软件工程实践人员所接受并走向大规模工业化应用所面临的主要障碍,在此基础上展望 AOP 未来的研究.

4.1 问题和挑战

尽管在过去 20 多年里,人们围绕 AOP 的抽象与模型、机制与理论、语言与设施以及开发与运行等方面开展了卓有成效的研究,取得了一系列的成果和进展,提出了几十种 AOP 语言,并开发了诸多支撑软件工具集和平台,但无论是在理论和技术的成熟度方面,还是在应对复杂 MAS 开发以及大规模工业化应用等方面,AOP 的研究与实践都面临着严峻的挑战:

(1) AOP 抽象与模型的多样性导致 AOP 技术难以标准化、集成和互操作

不同于 OOP,当前 AOP 所提供的抽象、模型、语言等具有多样化的特点.例如,软件 Agent 模型可以是知识型、认知型、反应型和混合型等,并基于认知科学、人工智能、社会组织学等学科的诸多异构概念来抽象 AOP 程序模型和运行模型,有些 AOP 是声明型,有些则是集成了声明型和命令型.这一状况源自 Agent 理论与技术的

研究,同时也反映了目前没有一种 AOP 抽象和模型占主导地位,这势必导致 AOP 技术(例如语言和平台)的多样性,使得开发人员难以选择合适的 AOP 语言,同时也影响了由不同 AOP 语言所编写软件之间的互操作性.

(2) AOP 语言对 MAS 不同层次的支持和融合有限

描述个体 Agent、Agent 之间的交互、环境层和组织层的 AOP 语言各有侧重点,往往独立发展,如描述个体 Agent 的 BDI 语言,Agent 通信语言 FIPA ACL 和 KQML,描述环境的 Java 和描述组织的 XML.这些不同层次的语言缺乏统一的语法和语义,一方面增加了对它们进行集成的难度,另一方面加重了程序员的学习负担和影响程序的可理解性.

(3) AOP 的 IDE 缺乏有效的调试和测试工具

调试是程序设计过程中的一个重要环节,主要是发现和解决程序中的错误或可能的失效.在编码过程中,程序员不可避免地会人为地引入语法方面和语义方面的 bug,编译器或词法分析器及时发现并定位该类错误是保证程序正确性和减轻程序员工作量的重要手段.另外,由于个体软件 Agent 的自主性以及局部 Agent 交互引发的全局行为具有不确定、不可预知和不可(完全)控制的特点,从而使得面向 Agent 的软件测试有其特有的复杂性[88].目前,只有大约 20%的 AOP 研究工作提供了 IDE,其主要功能包括了代码编程、程序的创建与运行、项目管理、Agent 状态和交互状态跟踪和监测等,然而这些 IDE 对调试、测试的支持仍然非常有限,对代码正确性的保证还主要依靠程序员人为的实现.

(4) AOP 尚未在支持复杂 MAS 开发方面充分展示其技术潜力

尽管在理论层面上人们一直声称 MAS 在支持一类具有动态、开放、自适应等复杂特征的应用开发方面具有潜力,但是具体到技术层面和工程实践环节,现有的 AOP 语言及其平台缺乏有效的机制和设施来展现其优势,从而使得开发人员从中受益.这也是导致 AOP 范型未能被人们所广泛接受的主要原因之一.当前,AOP 的研究无论在抽象、模型、机制还是语言设施等方面都缺乏针对复杂 MAS 的深层次理解和支持.例如,如何支持开放环境下 MAS 的动态调整和适应性?如何确保基于自主 Agent 的软件系统具有更强的灵活性和健壮性?如何通过大粒度的软件 Agent 模块提高重用和开发效率?等等.

(5) AOP 未能充分借鉴和体现软件工程的原理、原则和思想

在长期的工程实践过程中,人们总结出一系列行之有效的软件工程原理、原则和思想,以指导软件系统的工程化开发,如抽象、分解、组合、重用、模块化、信息隐藏等等.当前,主流的程序设计技术如 OOP 等充分考虑和融合了这些原理、原则和思想.现有 AOP 语言尽管具有形式的程序语义和严格的理论基础,具备知识表示和推理能力,但这些研究主要从应用角度出发研究 Agent 的自主性、驻留性和社会性等特点,缺乏从软件开发的角度来研究如何提高 MAS 的重用性、可维护性等,即缺乏现代程序设计所倡导的一些重要机制,如模块化、重用和封装等等[40].

(6) AOP 缺乏程序设计方法学

目前,AOP 研究集中聚焦于如何提供模型、语言和平台来支持 MAS 的构造与实现,它并没有提供有效的策略、原则等来指导程序员如何利用 AOP 来编写高质量的 MAS 程序.面向 Agent 程序设计方法学的研究,是目前 AOP 的薄弱环节.程序设计方法学的缺乏,导致软件工程实践人员无法更为深入地理解、认识和接受 AOP,缺乏系统的方法指导程序设计人员编写高质量的 MAS 程序,并直接影响程序设计的效率.

4.2 研究展望

基于上述问题和挑战,未来 AOP 的研究与实践需要着重解决以下 3 个方面的问题:(1) 程序设计能力问题.与当前主流的程序设计技术(如 OOP)相比,AOP 需要提供更为有效的思想、模型、机制、语言等来促进复杂软件系统的开发,在提高软件系统质量和开发效率、应对复杂软件系统所带来的挑战等方面充分展示其技术优势.(2) 工程化能力问题.AOP 的研究需要从工程化开发的角度更多地借鉴软件工程领域的原理和原则以及成功的经验,从而为 AOP 的大规模应用与实践提供更为友好、高效的支持,如面向 Agent 程序设计方法学、支撑软件平台、可重用的软件构件库、面向 Agent 的测试和调试、设计模式和软件体系结构等等.(3) 技术统一化问题.针对当前 AOP 多样化的技术形态和成果,需要加强在 AOP 概念、模型、语言、机制等方面研究的认同性,

推动 AOP 技术的统一化工作,从而促进 AOP 的技术标准化以及不同 MAS 软件的互操作等.

我们认为,未来 AOP 的研究与实践涉及以下几个方面的内容:

(1) 统一 AOP 抽象和软件模型

需要从多个不同的方面来考虑 AOP 抽象和模型的统一化问题.首先,统一 AOP 的抽象、概念及其内涵.当前, AOP 研究领域所涉及的概念繁杂,并且来自多学科和研究领域,如人工智能、大众心理学、认知科学、社会组织学等等.相同的概念在不同 AOP 语言中有不同的内涵,不同的概念又可能具有相同的内涵.概念和抽象的统一化将有助于就 AOP 研究与实践的本质内涵达成共识,并为模型和语言设计奠定基础.其次,统一 MAS 软件模型,就 MAS 软件的内部组成、各个要素功能、它们之间的相互关系达成共识.

(2) 集成与融合的 AOP 语言和工具

第一,AOP 及其语言的研究需要结合 MAS 不同层次的编程需求进行综合和集成,从而为不同层次的程序设计提供统一的语法和语义.第二,AOP 语言的研究与实践需要集成声明型和命令型程序设计的基本要素.第三,AOP 的研究与实践还需要与当前主流程序设计技术进行集成和融合.AOP 并不是一个全新的概念,可以认为是 OOP 的演化.第四,AOP 研究与实践应该与现有的热门技术进行集成和融合,如面向服务计算、移动计算等.

(3) 支持 AOP 的软件测试和确认技术

面向 Agent 的软件测试和确认可以从两个方面展开研究:一是基于形式化方法,如模型验证、定理证明等;二是在 AOP 的 IDE 中提供调试、跟踪、监测等辅助工具.面向 Agent 的软件测试和确认不仅要保证 Agent 代码、Agent 运行和交互的正确性,而且需要考虑环境层和组织层代码的正确性、个体 Agent、环境和组织之间语义及运行时状态的一致性.因此,寻求支持 MAS 软件测试的新理论和方法,如面向目标的测试,并在此基础上开发相应的自动测试和验证软件工具,是推动 AOP 走向应用的主要环节之一.

(4) 对复杂环境和系统的编程支持

要充分发挥 AOP 在开发复杂 MAS 方面呈现的诸如自主化、组织化、抽象化等技术优势,Agent 技术必须在程序设计层针对驻留环境的开放性、系统的动态性、适应性、演化性、成长性、灵活性、自组织、全局涌现等复杂性特点提供 AOP 软件模型、程序设计机制、基础理论和程序设计语言设施等方面的支撑.目前,一些研究学者已开始借助于环境显式化和组织的思想来应对环境的动态性和系统的开放性,并实现了与个体 Agent 模型和理论的集成,如 JoCoMo[87].未来 AOP 的研究还需要支持更多的复杂性特点,尤其是在程序设计语言层提供简单而有效的语言设施和平台支持.

(5) 从软件工程角度来开展 AOP 语言和机制设计

如果说早期 AOP 的研究与实践更多地借助于 AI 的理论和技术,关注的是如何设计出一个 AOP 语言或者平台来支持 MAS 的开发,那么在今后一段时间内,AOP 的研究与实践需要更多地借鉴软件工程的思想,关注如何设计出一个能够更好地支持工程化开发的 AOP 语言与其支撑平台,包括遵循软件工程的基本原理与原则(如抽象、封装、模块化、低耦合高内聚)、促进软件重用、模式运用和关注点分离、实现从现有设计方法学及其模型到 AOP 程序模型的平滑转换、加强 AOP 语言的简洁性、表达性等方面问题的考虑,解决学术研究与工业应用脱节的问题等等.目前,一些研究研究开始关注这些问题,如基于角色的继承和软件重用机制、角色的组合和聚合[89]、互操作问题[41].

(6) 关注 AOP 方法学研究

需要研究 AOP 的程序设计方法,从而为开展 AOP、构造高质量的 MAS、提高程序设计的效率等提供系统的方法学指导.例如,如何设计和组织模块单元,如何封装基本的软件 Agent 模块,如何有效实现重用,如何提高软件系统的可维护性,等等.

5 结 论

自 1993 年 Shoham 提出面向 Agent 程序设计概念和思想以来,AOP 的研究与发展已经过了 20 年的历程.根据 AOP 研究特点的差异性,这一历程大致可分为以下两个不同阶段:(1) 前 10 年,主要是 AI 领域的学者关注

AOP 的研究与实践,其研究特点是交叉 AI、DAI、认知科学、大众心理学等学科的知识,借助于 Agent 理论和技术的具体成果开展 AOP 模型、语言、理论和工具的研究,所提出的 AOP 语言主要侧重于对个体 Agent 层和 Agent 交互层的编程提供支持,具有明显的 AI 程序设计语言特征.(2) 后 10 年,更多领域的专家和学者加入到 AOP 的研究与实践行列,包括 AI、DAI、软件工程、分布计算等.AOP 的研究交叉更多学科的知识,包括社会组织学、软件工程、服务计算等.尤其是越来越多的学者认识到,AOP 是软件工程(尤其是面向 Agent 软件工程)的重要组成部分,开始从软件工程的视点来思考 AOP 的研究与实践问题,并充分借鉴软件工程的原理、原则和思想等来指导 AOP 语言的设计,希望推动 AOP 在工业范围内的应用与实践.这一时期,人们加强了对 MAS 环境层和组织层的编程能力支持,组织抽象和概念(如 Norm,Rule,Role 等)被引入到程序设计抽象和模型中,AOP 语言开始考虑对现代程序设计的机制和结构支持,如重用、封装、信息隐藏等,并与一些主流的计算技术(如面向服务计算等)相结合,以支持开放环境下复杂软件系统的开发.

作为一种新颖的程序设计,AOP 对开放环境下复杂软件系统的开发提供高层的抽象和模型以及有效的分解和构造机制.至今,人们在 AOP 领域开展了一系列的研究工作并取得了一定的研究成果.然而,AOP 的理论、语言及平台并没有得到广泛认可和应用.本文从程序设计范型的 4 个方面——抽象与模型、机制与理论、语言与设施、开发与运行,介绍和分析了现有 AOP 研究工作对个体 Agent、Agent 交互、环境、MAS 组织等不同层次的编程支持,分析了现有研究工作存在的不足,指出了 AOP 未来的研究方向.

References:

[1] 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 [doi; 10.1360/jos160001e]

[2] Mao XJ. The state-of-the-art, challenges and perspectives of agent-oriented software engineering. Computer Science, 2011,38(1): 1−7 (in Chinese with English abstract).

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

[4] Boehm B. A view of 20th and 21st century software engineering. In: Proc. of the Int'l Conf. on Software Engineering 2006. New York: ACM Press, 2006. 12−29. [doi: 10.1145/1134285.1134288]

[5] Mao XJ, Wang HM, Qi ZC, J. Wang J, Chen ZB. Software engineering for ultra-large scale system: Challenge and prospect. Communications of the China Computer Federation, 2010,6(7):60−65 (in Chinese with English abstract).

[6] Northrop L. Ultra-Large-Scale Systems: The Software Challenge of the Future. Pittsburgh: Software Engineering Institute, Carnegie Mellon University, 2006.

[7] Zambonelli F, Parunak V. 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]

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

[9] Jennings NR. An agent-based approach for building complex software systems. Communications of the ACM, 2001,44(4):35−41. [doi: 10.1145/367211.367250]

[10] Luck M, McBurney P, Shehory O, Willmoff S. Agent technology roadmap. 2005. http://www.agentlink.org

[11] Luo X, Miao C, Jennings NR, He M, Shen Z, Zhang M. Kemnad: A knowledge engineering methodology for negotiating agent development. Computational Intelligence, 2012,28(1):51−105. [doi: 10.1111/coin.2012.28.issue-1]

[12] Garcia E, Tyson G, Miles S, Luck M, Taweel A, Van Staa T, Delaney B. An analysis of agent-oriented engineering of e-health systems. In: Proc. of the AOSE 2012. Valencia, 2012.

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

[14] Jennings NR. On agent-Based software engineering. Artificial Intelligence, 2000,17(2):277−296.

[15] Bordini1 RH, Dastani M, Winikoff M. Current issues in multi-agent systems development. In: Carbonell JG, Siekmann J, eds. Proc. of the 7th Int'l Wrokshop on Engineering Societies in the Agents' World (ESAW). LNAI 4457, Heidelberg: Springer-Verlag, 2007. 38−61.

[16] Winikoff M. Future directions for agent-based software engineering. Int'l Journal of Agent-Oriented Software Engineering, 2009, 3(4):402−410. [doi: 10.1504/IJAOSE.2009.025319]

[17] Fisher M. A survey of concurrent METATEM—The language and its applications. In: Goos G, Hartmanis J, eds. Proc. of the Int'l Workshop on Temporal Logic. LNCS 827, Heidelberg: Springer-Verlag, 1994. 480−505.

[18] Thomas SR. The PLACA agent programming language. In: Proc. of the Intelligent Agents. LNAI 890, Heidelberg: Springer-Verlag, 1995. 355−370. [doi: 10.1007/3-540-58855-8_23]

[19] Guo L, Ge YT, Chen SF, Zhang DM. An agent-oriented programming language with intention driver. Journal of Software, 2003, 14(3):383−391 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/14/383.htm

[20] Rao AS. AgentSpeak(L): BDI agents speak out in a logical computable language. In: Van de Velde, Perram JW, eds. Agents Breaking Away: Proc. of the 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World. LNAI 1038, Berlin: Springer-Verlag, 1996. 42−55.

[21] Hindriks KV, De Boer FS, Van Der Hoek W, Meyer JJC. Agent programming in 3APL. Autonomous Agents and Multi-Agent Systems, 1999,2(4):357−401. [doi: 10.1023/A:1010084620690]

[22] Giacomo GD, Lespérance 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]

[23] Akbari OZ. A survey of agent-oriented software engineering paradigm: Towards its industrial acceptance. Journal of Computer Engineering Research, 2010,1(2):14−28.

[24] Wang J, Shen R, Zhu H. Agent oriented programming based on SLABS. In: Xie F, Lei J, eds. Proc. of the Int'l Computer Software and Applications Conf. (COMPSAC). Washington: IEEE Computer Society, 2005. 127−132. [doi: 10.1109/COMPSAC.2005.41]

[25] Wester-Ebbinghaus M, Moldt D, Reese C, Markwardt K. Towards organization oriented software engineering. In: Böttinger S, Theuvsen L, Rank S, Morgenstern M, eds. Proc. of the Software Engineering. 2007. 205−217.

[26] Boissier O, Hübner J, Sichman JS. Organization-Oriented programming: From closed to open systems. In: O'Har G ed. Proc. of the 7th Int'l Wrokshop on Engineering Societies in the Agents' World (ESAW). LNCS 4457, Heidelberg: Springer-Verlag, 2007. 86−105. [doi: 10.1007/978-3-540-75524-1_5]

[27] Dastani M. 2APL: A practical agent programming language. Autonomous Agent and Multi-Agent System, 2008,16(3):214−248. [doi: 10.1007/s10458-008-9036-y]

[28] Hübner JF, Boissier O, Bordini RH. A normative organisation programming language for organisation management infrastructures. In: Padget J, Artikis A, Vasconcelos W, Stathis K, Silva VT, Matson E, Polleres A, eds. Proc. of the Coordination, Organization, Institutions and Norms in Agent Systems (COIN). LNAI 6069, Heidelberg: Springer-Verlag, 2010, 114−129. [doi: 10.1007/978-3-642-14962-7_8]

[29] Tinnemeier N, Dastani M, Meyer JJ. Roles and norms for programming agent organizations. In: Proc. of the 9th Int'l Conf. on Autonomous Agents and Multiagent Systems (AAMAS). New York: ACM Press, 2009. 121−128.

[30] Ferber J, Gutknecht O, Michel F. From agents to organizations: An organizational view of multi-agent systems. In: Giorgini P, Muller J, Odell J, eds. Proc. of the 4th Int'l Conf. on Agent-Oriented Software Engineering (AOSE). LNCS 2935, Heidelberg: Springer-Verlag, 2003. 443−459.

[31] Weyns D, Helleboogh A, Holvoet T. How to get multi-agent systems accepted in industry? Int'l Journal of Agent-Oriented Software Engineering, 2009,3(4):383−390. [doi: 10.1504/IJAOSE.2009.025316]

[32] Juneidi SJ. Toward programming paradigms for agent oriented software engineering. In: Hamza MH, ed. Proc. of the IASTED Conf. on Software Engineering. Anaheim: IASTED/ACTA Press, 2004. 428−432.

[33] Ricci A, Santi A. Agent-Oriented computing: Agents as a paradigm for computer programming and software development. In: Nygard KE, Lorenz P, eds. Proc. of the 3rd Int'l Conf. on Future Computational Technologies and Applications. Wilmington: Xpert Publishing Services, 2011. 42−51.

[34] DeLoach SA. Moving multi-agent systems from research to practice. Int'l Journal of Agent-Oriented Software Engineering, 2009, 3(4):378−382.

[35] Cheng BHC, de Lemos R, Giese H, Inverardi P. Software engineering for self-adaptive systems: A research roadmap. In: Cheng BHC, de Lemos R, Giese H, *et al*., eds. Proc. of the Software Engineering for Adaptive Systems. LNCS 5525, Heidelberg: Springer-Verlag, 2009. 1−26. [doi: 10.1007/978-3-642-02161-9_1]

[36] Nagpal R, Yu C, Yamins D. Engineering self-organizing multi-agent systems. In: Padgham L, Parkes D, Muller J, Parsons S, eds. Proc. of the 7th Int'l Conf. on Autonomous Agents and Multi-Agent Systems. New York: ACM Press, 2008. 1717−1718.

[37] Ricci A, Denti E. simpA-WS: A simple agent-oriented programming model & technology for developing SOA & Web services. In: Baldoni M, Boccalatte A, Paoli FD, Martelli M, Mascardi V, eds. Proc. of the AI*IA/TABOO Joint Workshop From Objects to Agents (WOA 2007). 2007. 140−156.

[38] Zambonelli F, Omicini A. Challenges and research directions in agent-oriented software engineering. Autonomous Agents and Multi-Agent Systems, 2004,9(3):253−283. [doi: 10.1023/B:AGNT.0000038028.66672.1e]

[39] Mao XJ, Yu E. Organizational and social concepts in agent oriented software engineering. In: Odell J, Giorgini P, Müller JP, eds. Proc. of the Agent-Oriented Software Engineering. LNCS 3382, Heidelberg: Springer-Verlag, 2005. 1−15. [doi: 10.1007/978-3-540-30578-1_1]

[40] Novák P, Dix J. Adding structure to agent programming languages. Technical Report, Series IfI-06-12, Clausthal-Zellerfeld: Clausthal University of Technology, 2006.

[41] Luck M, McBurney P, Gonzalez-Palacios J. Agent-Based computing and programming of agent systems. In: Bordini RH, Dastani M, Dix J, Fallah-Seghrouchni AE, eds. Proc. of the 3rd Int'l Workshop Programming Multi-Agent Systems. LNAI 3862, Heidelberg: Springer-Verlag, 2006. 23−37. [doi: 10.1007/11678823_2]

[42] Dastani M. Programming multi-agent systems. In: Weyns D, Müller JP, eds. Proc. of the 12th Int'l Workshop on Agent-Oriented Software Engineering. Heidelberg: Springer-Verlag, 2012. 23−52.

[43] Weyns D, Omicini A, Odell JJ. Environment as a first-class abstraction in multi-agent systems. Autonomous Agents and Multi-Agent Systems, 2007,14(1):5−30. [doi: 10.1007/s10458-006-0012-0]

[44] Weyns D, Parunak VD, Michel F, Holvoet T, Ferber J. Environments for multiagent systems: State-of-the-Art and research challenges. In: Parunak VD, Michel F, eds. Proc. of the Environments for Multi-Agent Systems 2004. LNAI 3374, Heidelberg: Springer-Verlag, 2005. 1−47. [doi: 10.1007/978-3-540-32259-7_1]

[45] Bordini RH, Braubach L, Dastani M. A survey of programming languages and platforms for multi-agent systems. Informatica, 2006,30(1):33−44.

[46] Finziand A, Lukasiewicz T. Adaptive multi-agent programming in GTGolog. In: Freksa C, Kohlhase M, Schill K, eds. Proc. of the German AI Conf: Advances in Artificial Intelligence. LNCS 4314, Heidelberg: Springer-Verlag, 2007. 389−403.

[47] Hindriks KV, de Boer FS, van der Hoek W, Meyer JJC. Agent programming with declarative goals. In: Castelfranchi C, Lespérance Y, eds. Proc. of the 7th Int'l Workshop on Agent Theories, Architectures, and Languages (ATAL 2000). LNCS 1986, 2001. 248−257. [doi: 10.1007/3-540-44631-1_16]

[48] van Riemsdijk B, van der Hoek W, Meyer JJC. Agent programming in dribble: From beliefs to goals with plans. In: Proc. of the Int'l Conf. on Autonomous Agents and Multiagent Systems. 2003. 393−400. [doi: 10.1007/978-3-540-45133-4_32]

[49] Fallah-Seghrouchni AE, Suna A. CLAIM: A computational language for autonomous, intelligent and mobile agents. In: Dastani M, Dix J, Fallah-Seghrouchni AE, eds. Proc. of the Int'l Workshop Programming Multi-Agent Systems. LNCS 3067, Heidelberg: Springer-Verlag, 2004. 90−110. [doi: 10.1007/978-3-540-25936-7_5]

[50] Winikoff M. JACK^{TM} intelligent agents: An industrial strength platform. In: Bordini R, Dastani M, Dix J, Fallah-Seghrouchni AE, eds. Proc. of the Multi-Agent Programming: Languages, Platforms and Applications. Heidelberg: Springer-Verlag, 2005.

[51] Davies WHE, Edwards P. AGENT-K: An integration of aop and kqml. In: Proc. of the ACM Int'l Conf. on Information and Knowledge Management (CIKM'94) Workshop on Intelligent Agents. 1994.

[52] Ricci A, Viroli M, Omicini A. A general-purpose programming model & technology for developing working environments in MAS. In: Dastani M, Fallah-Seghrouchni AE, Ricci A, Winikoff M, eds. Proc. of the Int'l Workshop Programming Multi-Agent Systems (PROMAS 2007). Heidelberg: Springer-Verlag, 2007. 54−69.

[53] Horling B, Lesser V. A survey of multi-agent organizational paradigms. Knowledge Engineering Review, 2004,19(4):281−316. [doi: 10.1017/S0269888905000317]

[54] Pynadath DV, Tambe M, Chauvat N, Cavedon L. Toward team-oriented programming. In: Jennings NR, Lespérance Y, eds. Proc. of the Int'l Workshop on Agent Theories, Architectures, and Languages. LNCS 1757, Heidelberg: Springer-Verlag, 2000. 233−247.

[55] García-Camino A, Rodríguez-Aguilar JA, Sierra C, Vasconcelos W. Norm-Oriented programming of electronic institutions. In: Nakashima H, Wellman MP, Weiss G, Stone P, eds. Proc. of the Int'l Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2006). New York: ACM Press, 2006. 670−672. [doi: 10.1145/1160633.1160750]

[56] García-Camino A, Rodríguez-Aguilar JA, Sierra C, Vasconcelos W. Constraining rule-based programming norms for electronic institutions. Journal of Autonomous Agents and Multi-Agent Systems, 2009,18(1):186−217. [doi: 10.1007/s10458-008-9059-4]

[57] Tinnemeier N, Dastani M, Meyer JJC, van der Torre L. Programming normative artifacts with declarative obligations and prohibitions. In: Proc. of the Int'l Joint Conf. on Web Intelligence and Intelligent Agent Technology. Washington: IEEE Computer Society, 2009. 145−152. [doi: 10.1109/WI-IAT.2009.144]

[58] 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]

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

[60] Hübner J, Sichman JS, Boissier O. Developing organised multiagent systems using the MOISE+ model: Programming issues at the system and agent levels. Int'l Journal of Agent-Oriented Software Engineering, 2007,1(3-4):370−395. [doi: 10.1504/IJAOSE.2007.016266]

[61] Dennis L, Fisher M, Hepple A. Language constructs for multi-agent programming. In: Sadri F, Satoh K, eds. Proc. of the 8th Workshop on Computational Logic in Multi-Agent Systems (CLIMA 2007). LNAI 5056, Heidelberg: Springer-Verlag, 2008. 137−156. [doi: 10.1007/978-3-540-88833-8_8]

[62] Esteva M, Rodríguez-Aguilar JA, Rosell B, Arcos JL. 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.

[63] Dastani M, Grossi D, Meyer JJC, Tinnemeier N. Normative multi-agent programs and their logics. In: Meyer JJC, Broersen JM, eds. Proc. of the 1st Int'l Workshop on Knowledge Representation for Agents and Multi-Agent Systems. LNAI 5605, 2009. 16−31.

[64] Dastani M, van Riemsdijk MB, Hulstijn J. Enacting and deacting roles in agent programming. In: Odell J, Giorgini P, Müller JP, eds. Proc. of the 4th Int'l Workshop on Agent-Oriented Software Engineering. LNCS 3382, Heidelberg: Springer-Verlag, 2005. 189−204. [doi: 10.1007/978-3-540-30578-1_13]

[65] Dastani M, Mol CP, Steunebrink BR. Modularity in agent programming languages. In: Bui TD, Ho TV, Ha QT, eds. Proc. of the Intelligent Agents and Multi-Agent Systems. LNCS 5357, Heidelberg: Springer-Verlag, 2008. 139−152. [doi: 10.1007/978-3-540-89674-6_17]

[66] 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]

[67] d'Inverno M, Hindriksy K, Luck M. A formal architecture for the 3APL agent programming language. In: Bowen JP, Dunne S, Galloway A, King S, eds. Proc. of the Formal Specification and Development in Z and B. LNCS 1878, Heidelberg: Springer-Verlag, 2000. 168−187. [doi: 10.1007/3-540-44525-0_11]

[68] Lespérance Y, Levesque HJ, Lin FZ. Foundations of a logical approach to agent programming. In: Proc. of the Int'l Workshop on Agent Theories, Architectures, and Languages 1996. LNCS1037, Heidelberg: Springer-Verlag, 1997. 331−346.

[69] Khan SM, Lespérance Y. Logical foundations for a rational BDI agent programming language. In: Dennis LA, Boissier O, Bordini RH, eds. Proc. of the Int'l Workshop Programming Multi-Agent Systems (ProMAS 2011). Heidelberg: Springer-Verlag, 2011. 1−20. [doi: 10.1007/978-3-642-31915-0_1]

[70] Ferber J, Müller JP. Influences and reaction: A model of situated multi-agent Systems. In: Proc. of the Int'l Conf. on Multi Agent Systems (ICMAS'96). Cambridge: MIT Press, 1996. 72−79.

[71] Helleboogh A, Vizzari G, Uhrmacher A, Michel F. Modeling dynamic environments in multi-agent simulation. Journal of Autonomous Agents and Multi-Agent Systems, 2007,14(1):87−116. [doi: 10.1007/s10458-006-0014-y]

[72] van Riemsdijk MB, Koen R, Jonker HC. Programming organization-aware agents, a research agenda. In: Aldewereld H, Dignum V, Picard G, eds. Proc. of the Engineering Societies in the Agents World X. LNCS 5881, Heidelberg: Springer-Verlag, 2009. 98−112. [doi: 10.1007/978-3-642-10203-5_9]

[73] Searle J. The Construction of Social Reality. New York: Free, 1995.

[74] Artikis A. Dynamic protocols for open agent systems. In: Decker, Sichman, Sierra, Castelfranchi, eds. Proc. of the Int'l Conf. on Autonomous Agents and Multiagent Systems. New York: ACM Press, 2009. 97−104.

[75] van der Vecht B, Dignum F, Meyer JJC. Autonomous Agents Adopting Organizational Rules. In: Dignum V, ed. Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models. Hershley: IGI Global, 2009. 314−333.

[76] Costantini S, Tocchio A. A logic programming language for multi-agent systems. In: Flesca S, Greco S, Leone N, Ianni G, eds. Proc. of the Logics in Artificial Intelligence, European Conf. (JELIA 2002). LNAI 2424, Heidelberg: Springer-Verlag, 2002. 1−13.

[77] Thielscher M. FLUX: A logic programming method for reasoning agents. Theory and Practice of Logic Programming, 2005,5(4–5): 533−565. [doi: 10.1017/S1471068405002358]

[78] Clark K, McCabe F. Go!—A multi paradigm programming language for implementing multi-threaded agents. Annals of Mathematics andArtificial Intelligence, 2004,41(2-4):171−206. [doi: 10.1023/B:AMAI.0000031195.87297.d9]

[79] Ross R, Collier R, O'Hare G. Af-apl: Bridging principles & practices in agent oriented languages. In: Bordini RH, Dastani M, Dix J, Fallah-Seghrouchni AE, eds. Proc. of the ProMAS 2004. LNCS 3346, Heidelberg: Springer-Verlag, 2004. 66−88. [doi: 10.1007/978-3-540-32260-3_4]

[80] Morley D, Myers K. The SPARK agent framework. In: Proc. of the Int'l Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2004). Washington: IEEE Computer Society, 2004. 714−721. [doi: 10.1109/AAMAS.2004.267]

[81] Bordini RH, Hubner JF. BDI agent programming in AgentSpeak using *Jason*. In: Toni F, Torroni P, eds. Proc. of the Computational Logic in Multi-Agent Systems. Heidelberg: Springer-Verlag, 2005. 143−164. [doi: 10.1007/11750734_9]

[82] Doan TT, Alechina N, Logan B. The agent programming language meta-APL. In: Dennis LA, Boissier O, Bordini RH, eds. Proc. of the Int'l Workshop Programming Multi-Agent Systems (ProMAS 2011). Heidelberg: Springer-Verlag, 2011. 72−87.

[83] Hu CY, Mao XJ, Cheng Y, Zhou HP. OrgMAP: An organization-based approach for multi-agent programming. In: Conitzer, Winikoff, Padgham, van der Hoek, eds. Proc. of the Int'l Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2012). New York: ACM Press, 2012. 1437−1438.

[84] Collier RW. Agent factory: A framework for the engineering of agent-oriented applications [Ph.D. Thesis]. Dublin: University College Dublin, 2001.

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

[86] Hübner JF, Boissier O, Kitio R, Ricci A. Instrumenting multi-agent organisations with organisational artifacts and agents: Giving the organisational power back to the agents. Int'l Journal of Autonomous Agents and Multi-Agent Systems, 2010,20(3):369−400. [doi: 10.1007/s10458-009-9084-y]

[87] Boissier L, Bordini RH, Hubner JF, Riccid A, Santi A. Multi-Agent oriented programming with JaCaMo. Science of Computer Programming, 2012. http://www.sciencedirect.com/science/article/pii/S016764231100181X [doi: 10.1016/j.scico.2011.10.004]

[88] de Cerqueira Gatti MA, von Sta A. Testing & debugging multi-agent systems: A state of the art report. Monografias em Ciência da Computação, 2006. ftp://ftp.inf.puc-rio.br/pub/docs/techreports/06_04_gatti.pdf

[89] Collier R, Ross R, O'Hare GMP. A role-based approach to reuse in agent-oriented programming. In: Proc. of the AAAI Fall Symp.: Roles, an Interdisciplinary Perspective. Arlington, 2005.

附中文参考文献:

[1] 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 [doi; 10.1360/jos160001e]

[2] 毛新军.面向 Agent 软件工程:现状、挑战和展望.计算机科学,2011,38(1):1−7.

[3] 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(F 辑:信息科学),2006,36(10):1037−1080.

[5] 毛新军,王怀民,齐治昌,王戟,陈振邦.面向超大规模系统的软件工程:挑战与展望.计算机学会通信会刊,2010,6(7):60−65.

[8] 毛新军.面向主体软件开发.北京:清华大学出版社,2005.

[19] 郭磊,戈也挺,陈世福,张东摩.一种意向驱动式面向 agent 程序设计语言.软件学报,2003,14(3):383−391. http://www.jos.org.cn/1000-9825/14/383.htm

毛新军(1970－),男,浙江江山人,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.

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

孙跃坤(1985－),男,硕士生,主要研究领域为基于组织的软件自适应.

王怀民(1962－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为智能 Agent 理论和技术,分布式计算.