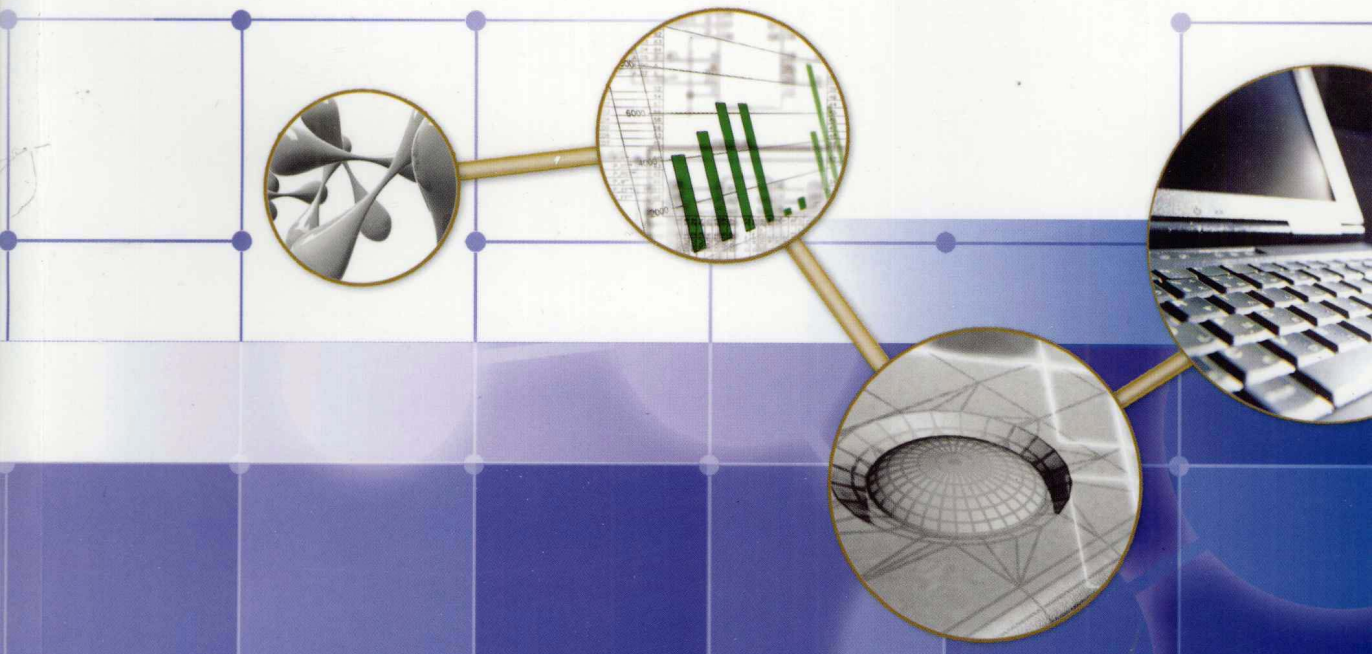
面向Agent的软件设计开发方法

◎薛霄 编著　贾宗璞 主审



電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

面向 Agent 的软件设计开发方法

薛 霄 编 著
贾宗璞 主 审

電子工業出版社
Publishing House of Electronics Industry
北京 • BEIJING

内 容 简 介

在过去几年中，Agent 和多 Agent 系统（MAS）已经成为一种应对各种复杂 IT 情景的强大技术，有大量的研究是关于定义合适的模型、工具和技术以支持开发复杂的 MAS 软件系统。目前关于面向 Agent（AO）方法的科学文献出现得越来越多，遍布在不同的会议、期刊和时事新闻上。因此，无论是新人还是专家，在这个领域进行研究时，都难以操纵所有这些材料。本书试图将各种研究结果和思想有组织地综合在一起，虽然非常多样化，但都以促进复杂 MAS 软件系统的开发作为总体目标，希望能够为研究者和学生了解 AO 方法的发展现状提供线索，而不用在现有的数字图书馆中查阅数以千计的文件，也不会在无尽的搜索中迷失自己。读者同时可以了解到软件工程新的发展趋势，以及如何将 Agent 思想应用于目前软件界所出现的种种新技术（比如 SOA、网格服务等）中。

本书可以作为计算机软件专业硕士生和博士生的教材和参考用书，对于从事 Agent 理论和技术研究的人员，尤其是从事面向 Agent 软件工程研究的人员以及基于 Agent 技术的工程实践人员均具有较高的参考价值。

图书在版编目（CIP）数据

面向 Agent 的软件设计开发方法 / 薛霄编著. —北京：电子工业出版社，2009.1

ISBN 978-7-121-07984-9

Ⅰ. 面… Ⅱ. 薛… Ⅲ. 软件工具—程序设计 Ⅳ. TP311.56

中国版本图书馆 CIP 数据核字（2008）第 197422 号

策划编辑：张贵芹
责任编辑：张贵芹　田　怡　　特约编辑：李云霞
印　　刷：
装　　订：北京市李史山胶印厂
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　　邮编 100036
开　　本：787×1092　1/16　印张：18.25　　字数：467.2 千字
印　　次：2009 年 1 月第 1 次印刷
定　　价：36.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

前　言

Agent 和多 Agent 系统（MAS）现在已成为一种应对各种复杂 IT 情景的强大技术，如生产过程、Web 服务、基于 Internet 计算的市场和分布式网络管理等。然而，新近出现的理解认为，MAS 不仅是一种有效的技术，还代表了一种新型的软件开发通用范型，即基于自主软件实体（Agent）的设计和开发应用。这种实体位于某个环境中，可以通过高层协议和语言的交互来灵活实现其目标。这些特点非常适合于解决现时情景下的复杂软件开发。事实上，① 自治的应用组件，反映了现代分布式系统内在的分散性，并且可视为系统被不同的利益相关者所拥有，在模块化和封装概念上进行了自然延伸；② Agent 运行和交互（包括相互之间及 Agent 与环境之间）所采取的灵活方式，适应于现今软件在动态和不可预知的情况下运行；③ Agent 的概念为人工智能的成果提供了一个统一的观点，通过使用 Agent 和 MAS 作为存放智能行为的、可靠的和易管理的知识库，从而利用人工智能的成果解决现实世界中的问题。

在过去几年中，基于 Agent 的计算被日益接受为一种新型的软件工程范型，已经有大量的研究是关于定义合适的模型、工具和技术，以支持开发复杂的 MAS 软件系统。这些研究，即面向 Agent 的软件工程（AOSE），不断地提出各种新的建模方法和技巧、新的设计方法和工具，尤其是新型的面向 Agent 的范型。

关于 AOSE 的科学论文在文献中出现得越来越多，遍布在不同的会议、期刊和新闻上。因此，无论是新人还是专家，在这个领域中进行研究时，在操作所有这些材料时总会有困难。本书试图将各种研究结果和建议有组织地综合在一起，虽然非常多样化，但都以促进 MAS 的开发为相同的总体目标。我们的希望就是，这本书能够为研究者和学生了解 AOSE 的发展现状提供线索，而不用在现有的数字图书馆中搜索数以千计的文件，也不会在无尽的搜索中迷失方向。

当然，我们需要清楚地认识到，AOSE 的研究仍处于初级阶段。在 AOSE 被广泛接受，并且在 MAS 复杂软件系统研究中成为实际可用的范型之前，就必须面对所出现的挑战。出于这些原因，本书避免支持特定的技术或方法，而只是给读者介绍不同的设计方法和实现技术，给予他们更多的选择余地。本书的内容共分为五个部分：

- 第一部分（基于 Agent 的软件开发基础知识）是介绍性的，目的在于阐明为什么基于 Agent 是设计复杂软件系统的合适方法（比现有的传统方法优越），同时从不同层面对 AOSE 的研究现状进行调查。
- 第二部分（面向 Agent 的开发方法）阐述了三种经典的方法学（即 Gaia、Tropos 和 MaSE），它们在过去几年中作为通用方法来指导开发复杂的 MAS，在研究者群体中具有巨大的影响；三种特殊的方法（即 ADELFE，MESSAGE 和 Prometheus），虽然影响力不大，但自身的有趣的特性使它们非常适合于 MAS 特殊类（如自适应的 MAS 和基于 Agent 的系统）的设计和特殊的应用领域（如电信应用和智能商场）。

- 第三部分（改进面向 Agent 的软件开发方法）针对目前 AO 方法所面临的困境，提出了层次开发框架 HDA，为将 Agent 技术变为复杂系统建模的有力工具提供了一种可行的途径。HDA 能够基于现有的各种 AO（Agent Oriented）元模型定制出符合特定项目需求的方法，并为设计模型的实现提供了一整套的方案，解决了设计模型同软件实现相脱节的问题。
- 第四部分（面向 Agent 的软件工程工具和基础设施）将焦点从方法学转换到基础设施和工具上。事实上，尽管方法学驱动着构建 MAS 的过程，但只有提供了适当的工具和软件基础设施，才能最终实现一个良好的工程化软件系统。概念性工具列出了 FIPA 标准和 AUML，软件基础设施则介绍了截至目前可供开发者使用的、最有前途的工具 JADE。
- 第五部分（新兴的趋势和前景）更具有应用导向的性质，集中在 MAS 技术未来的假定应用上，如信息服务和普适计算。在这些新出现的技术背景下，基于 Agent 的系统有可能得到广泛利用，为了促进开发的可靠性和有效性，对 AOSE 的技术的需求也可能会更迫切。在此基础上，给出了该领域的 AOSE 研究路线图。

在国家科技支撑计划重大项目课题“三峡库区枢纽港经济圈物流服务示范工程（2006BAH02A20）”、国家科技支撑计划重点项目课题“生殖健康公共服务架构设计与信息标准研制（2008BAH24B01）”和北京市自然科学基金项目“现代服务理论及其在现代物流服务业中的应用研究（4082017）”和河南理工大学博士基金课题“Agent 软件理论与关键技术研究（648227）”的支持下，作者进行了有关 Agent 建模技术在现代信息服务中的应用研究和开发工作，取得了一些初步的成果，希望通过本书与广大科技人员和读者交流与共享，从而推动 Agent 技术的研究和实施。

首先我想强调，这本书凝聚了 AOSE 领域很多研究人员的努力成果，在这里感谢他们为 AOSE 的发展所作出的显著贡献。没有他们，我们将永远不会形成对该领域研究的宽广视野。另外，这里要特别感谢河南理工大学计算机学院的贾宗璞教授在百忙之中抽出时间审阅了全书，提出了很多有价值的修改意见，对于全书的形成至为重要。

另外，本书的完成还要感谢我的妻子王淑芳，她对本书所参阅的文献资料做了大量的整理工作；感谢河南理工大学计算机学院的研究生朱红磊、李慧琴、张亚婷、潘亚峰、王娟、杨玲、车平、张纯等，他们为本书的插图和校对进行了大量辛苦的工作；感谢河南理工大学智能科学实验室的同事刘小燕、马永强、罗军伟、侯艳芳、王志衡和李会军，他们为本书的写作提出了很多宝贵的建议。感谢山东科技大学的博士生李东民和湖北工业大学的研究生朱传鸿，清华大学 CIMS 工程中心实验室的朱鹏、王婷、刘志宇和李梦生，他们为本书的完稿提供了很多力所能及的帮助。最后，我真心感谢本书的编辑，是她付出的辛勤劳动，使得这本书能最终面世。

作　者

河南理工大学

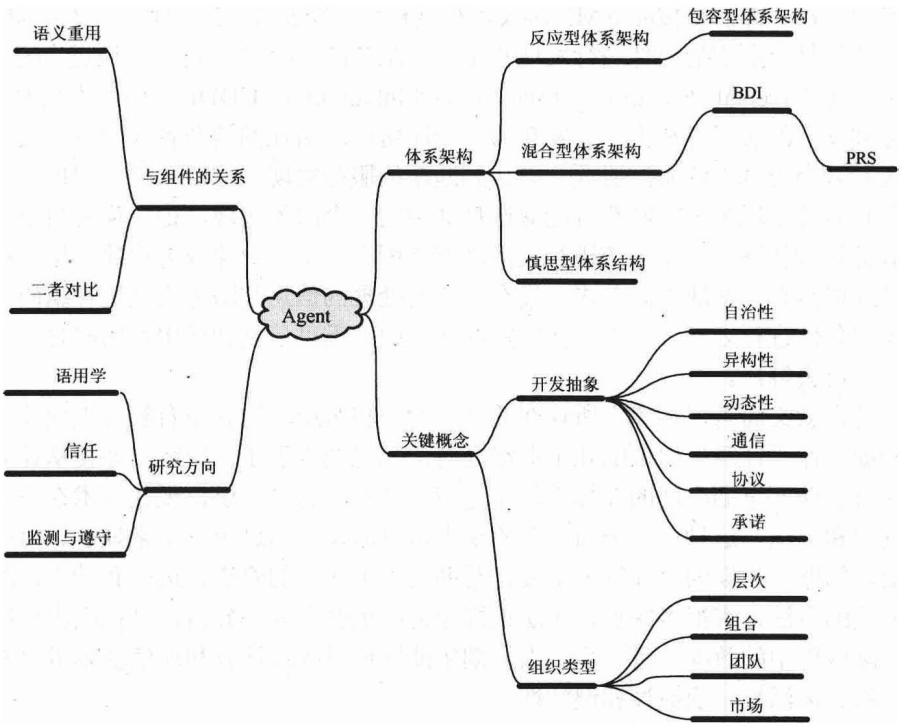
2008 年 10 月

目　录

第 1 章　面向 Agent 的软件开发抽象

【本章概要】本章提供了必需的历史知识和基础概念，以帮助读者完整了解面向 Agent 的软件开发工具和技术。我们首先回顾了软件技术发展的历史（从独立系统到开放系统）；然后讨论了由于软件系统变得日益庞大和开放，这种不断增长的复杂性所带来的挑战。本章展示了为什么 Agent 技术抽象是开发大规模开放式系统的理想选择；与组件相比较，Agent 抽象所具有的相似性和差异性；以及它们在软件实践中所得到的结果。

【本章知识结构图】



1.1　引　　言

最初的计算机研究是基于集中式架构的，很少考虑编程方面所遇到的挑战，但是在相关技术方面已经取得了很大进步，例如以健壮的、可重用的模式来设计数据结构和算法。虽然目前的企业级系统大多采取分布式的计算结构，但是所有的系统架构都是由组件构成，每个组件都必须以健壮的方式进行开发，所以已经提出的方法仍然是有价值的。

分布式系统技术能够使企业中相互隔离的信息组合起来，从而被更好地使用。但是，在克

服了网络方面的低层次挑战之后，系统在向分布式迁移时面临着一个痛苦的事实：难以将目前所存在的各个信息孤岛连接起来以获取额外价值。这里的一个关键问题就是异构性，也就是说，在不同信息源中存储的信息会出现语义不匹配。最为常见的是，根本无法对语义进行建模，协调各种不同的资源十分困难，甚至是不可能的。处理上述各种异构设置的最佳方法，就是采用静态信息模型或数据库模式对各种不同的资源进行建模，然后将这些模型集成起来。模型集成在处理异构信息系统时很少起作用，因为任何单个模型的变化都会导致整个集成模式的失效。

我们这里所讨论的系统是开放的，因为它们反映了组织本身所具有的开放性。在开放环境中，组件模式不但是异构的，而且是自治的，可能会快速地改变，各个组件也没有被要求一定要进行合作。开放架构的特点是可以对自主异构的组件进行动态增加和删除。随着电子商务的普及，开放架构正在变得越来越普及，其中最经典的一个应用就是 Web 服务。

Web 服务架构把每个组件看做一个服务，服务所提供的一系列能力被定义为方法（Curberaet 等，2002a）。服务可以被调用，也可以交换文档。在 Web 服务中，被调用的参数、结果和交换的文档，都需要按照 XML 协议进行结构化（Box 等，2000）。开放架构所面临的一个独特挑战就是对所需的组件进行发现和定位。在 Web 服务中，这一点通过“统一描述发现和集成标准（Universal Description Discovery and Integration，UDDI）”中的“注册”概念来实现。服务被它们的实现者和推广者发布为一个注册项。潜在的消费者（即寻求支持特定接口服务的人）就会尝试着联系注册项，以找到匹配的服务实现。注册项将会提供一个满足给定要求的所有服务实现列表。消费者能够选择其中的一个服务实现，把它作为自己的合适选择。Web 服务架构能够对大型系统进行灵活动态的配置，正在快速成为构建大型系统的事实标准，但是它仍然有一些缺点。首先，没有对所要处理的信息和服务语义进行编码，匹配过程是依据语法条件进行处理。其次，在特定的环境中缺乏可信度和适用性的概念，这些问题被留给用户自己去解决。

就某种特定意义而言，“计算”所存在的基本商业和组织环境并没有随着现代技术的发展而改变。例如，在“计算”出现的几个世纪之前，就已经出现了供应链。即使从计算的角度出发，供应链也能同早期出现的计算系统相适应。从实际意义上讲，现代技术在很大程度上也影响了商业和组织。这种“一分为二”的观点可以通过下面这个例子来理解。以前供应链中存在着很多问题，早期的“计算”主要是帮助改善供应链的建造、运行和效率检测。但这些是比较死板的方法，有很多缺点，无法很好地把握机会和处理异常，这在很大程度上限制了它们在实际应用中的效率。进一步，人们期望能够通过改进计算和通信基础机制来获得下列属性：效率、灵活性、敏捷性和健壮性。

面对当前环境下的实际挑战，传统的计算机科学无法提供强有力的抽象和灵活的技术。这些问题要求采用一种更为宽广的视野来看待“计算”，不仅要坚持科学原则，还要考虑计算发生时所处的组织环境，理解用户的需求，即计算所服务的对象。换言之，真正的世界在某种意义上一直都是开放的，只是我们采用构建和管理过程的技术对它们施加了一定的限制。现在所需的就是引入一种技术来避免这些限制，这就是 Agent 出现的原因。Web 服务的层次架构在许多方面都很自然地体现了 Agent 的重要特征。如果把 Agent 系统开发看做是 Web 服务的上层结构，对于探索它们的现有特征，改进 Web 服务架构都是很有价值的。值得注意的是，应该避免在 Web 服务中试图复制 Agent 的每一个思想，这样会使系统出现冗余，变得难以理解，尤其是对于大量的软件开发人员而言。

1.2　Agent 的开发抽象

近些年，人们已经提出了很多有关 Agent 的定义。有些定义侧重于 Agent 在用户界面中的应用，使图形界面具有一些典型的人性化特征，能够根据用户特征实现自适应变化。这种 Agent 同本书并没有什么直接的关联。有些定义阐述了 Agent 有能力同其他各方进行交互，或者同某些特定技术紧密相关，例如基于定理证明器，或者使用思维概念中的数据结构；又如信念、知识、目标、期望、意图等。这些定义的局限性在于考虑 Agent 的内部结构过多，限制了它们的通用性，即运用于开放环境的可能性。还有一些定义先假定 Agent 具有一定的理性，这可能会对 Agent 的自治性产生一些不合时宜的限制。

Wooldridge 和 Jennings 描述了有关 Agent 的两种观点（Wooldridge 和 Jennings，1995a）：一个是弱概念，另一个是强概念。Agent 的弱概念在主流计算中很流行，尤其是软件工程师。这种观点认为 Agent 类似于 UNIX 进程，具有自治、社交、反应和预动等属性。自治是指 Agent 在没有人为干预下的工作能力，可以控制它们自己的状态和行为。社会能力是指与其他 Agent 进行高层对话的通信能力。反应是指对外界变化做出及时的感知和响应。预动是指 Agent 选择本身的目标，并且依照目标进行动作的能力。相比之下，Agent 的强概念在人工智能领域中很常见，认为 Agent 是一个计算机系统。除了具有上面提到的所有属性外，还可以被概念化或模型化，具有人类的特征，例如知识、信念、目的和义务等思维概念。

Agent 技术近几年的发展努力使这项技术走向一个合理、务实的方向。几年前，Agent 领域内部还在为 Agent 的定义争论不休。然而，当研究人员意识到不可能在 Agent 的定义上达成一致，或者说是不需要时，这场争论就变得悄无声息了。定义的关键在于“提出这样一个定义的目的”。如果这个目的是为了说明如何构造单个的 Agent，那么定义 Agent 的内部设计就是合理的。然而，如果这个目的是想定义一个开放系统，保证不同的 Agent 都能够参与进去，那么重点就应该放在 Agent 之间的交互上，而不是如何去构建它们。按照上述想法，一个更为激进的立场是，不要为形式化的定义去烦恼，而应该关注如何去测试是否为 Agent（Huhns 和 Singh，1999）。Huhns 和 Singh 所提议的测试主要考虑了一个 Agent 如何同其他 Agent 进行交互，在其他 Agent 的影响下如何改变自身行为。这个定义是从外部特征来描述 Agent，留下了一些不确定的事实，例如何构建一个 Agent，Agent 是否具有信念和意图，是否具有理性等。这样定义的 Agent，可以用于开放环境中的各种实际计算，能够反映所代表参与者（在一个信息环境中）的自治性和异构性（在现实世界中）。

当开始强调交互时，我们的兴趣点很自然地就从单个 Agent 转移到了多 Agent 系统上来。这是一个合理的转变，因为事实上单个 Agent 系统并不是那么令人感兴趣。如果你一定要拥有 Agent，那么就必须把它们作为多 Agent 系统（Multi-Agent System，MAS）的一部分。这么说的理由在于，如果没有交互性和开放性，那么，传统的计算机科学应该能够很好地满足你的要求。如果你不打算从 Agent 的特殊属性中受益，为什么要引入它们呢？

开放系统的挑战和 Agent 的潜在灵活性都表明 Agent 将为现实应用提供一个优秀的解决方案基础。当然，这个实践的整体意义是展示 Agent 如何匹配开放系统的要求。回顾一下上面所提到的，开放系统一般都是由自治、异构和动态的组件构成。从广义上讲，基于 Agent 的软件开发是能够充分挖掘 Agent 主要特征的技术与方法学。现在让我们考虑一下与 Agent

概念相关的关键抽象，研究如何把它们捕捉为计算抽象，并给出这么做的一个典型手段。通过下面的简要讨论，可以为后续章节打下基础。

1．**自治性**（Autonomy）

自治性也就是指一方具有独立性，能够根据自己的意愿来行动。从广义上讲，商务交易中参与者的自主决策就反映了自治性。没有人能够强迫你买卖任何东西；也没有人能够强迫你听命于另一方，或者与之妥协；更没有人能够强迫你使用某种特定的推理策略。特别地，一个自治个体甚至不需要任何外部意义下的“理智”，因为这样的要求会限制它的自治性。

如果某种方法能够使交互各方的自治性呈现出良好的效果，那么这种方法应该很容易适用于几乎所有的场合。值得注意的是，有些场合所呈现出的自治现象，真实情况或许并不是由于自治性，而是由其他一些原因引起的。例如，参与者没有对消息做出反应，可能并不是它做出的自治性反应，而是因为基础机制失效。当然，在相同的框架下，实现自治的参与者也可以自如地处理基础机制的变化。因此，判断某个行为是自治性的体现，还是由于基础机制的失效而引起的，这点可能很重要。

一般来说，不受约束的计算自治性就如同现实世界中不受约束的自治性一样，将会产生不可预期的结果。并且，为了给出完整的计算模型，或者做出可靠的预测，我们都必须假设参与者的自治性在某种程度上受到约束。典型地，根据协议进行交互就可以实现对自治性的限制。

2．**异构性**（Heterogeneity）

异构性源于设计者构建组件的方式相互独立，导致组件的信息模型或过程模型各不相同。一般来说，在功能系统中，异构性的产生是由于历史原因造成的。没有人开始就想设计一个异构系统，但是构建大型系统的最终结果往往是异构的。反之，在设定一个开放系统的参数时，不假定内部结构的同质性是非常重要的。最终，为了能够让这些组件一起运作，就要对它们之间的异构性施加一定的限制。也就是说，一定要有对共同性的说明。就信息模型而言，通过一个共享本体来捕捉共同性（Gruber，1991）；这是同 Agent 相关的，但是并不特指 Agent，因为它的产生也是源于将异构信息源组合起来。就过程模型而言，可以通过确定典型外部事件的方法来捕捉共同性（Singh，2003）。这个思路是，Agent 的行为标记是指对外界的响应结果，而不需要暴露内部的构造细节。这些标记具有被标准化的潜力，事实上是分布式数据库事务处理中两阶段提交协议所采用的方法（Gray 和 Reuter，1993）。任何按照特定标准实现的 Agent，都被要求公布合适的特定事件，但并不需要暴露内部的实现细节。

3．**动态性**（Dynamism）

动态性是指管理者能够独立对系统进行灵活配置，并且可以根据需要改变其配置，而不需要明确通知相关各方（系统中的其他成员）。开放系统具有最大程度的动态性，因为原则上它们根本不需要管理者。事实上，它们出于特定目的可能会需要一些管理功能，如监测和安全、根据应用限制潜在的成员参与等。原则上，这些功能可以分布于成员之间，不需要管理者，但是出于社会政治原因，系统中经常需要有一方来负责。

Agent 方法所倡导的动态配置技术有一些变种，已经被广泛使用，例如在 UDDI 中。然而，更深层次的挑战并不在于服务发现，也就是 UDDI 所强调的，而是在于如何从几个可能发现的服务中选择一个最合适的服务实现。一些与之最相关的 Agent 方法，在很大程度上还

没有被吸收到商业方法中，包括如何进行匹配。工程化的 Agent 系统经常包含一个进行匹配的组件。有一些方法考虑基于 Web 服务的语义描述进行匹配（Trastour 等，2001）。另一类方法主要是考虑所获得服务的质量，引入了服务质量的概念模型，并把它嵌入到了一个框架中，其中服务选择被看做是一个集成组件（Maximilien 和 Singh，2002）。还有一些方法则是改进传统的推荐系统，不仅可以处理产品选择，还能处理服务选择（Sreenath 和 Singh，2003）。

4．**通信**（Communications）

任何组件都可以通过它们所共享的环境进行交互。当一方对环境做出改变时，其他各方能够看得到。当采用 Agent 对组件进行概念化时，它们之间的交互呈现出一种不同的风格。很明显的，Agent 能够通过环境进行持续交互：如果它们是机器人，可以采用彼此碰撞对方的方式；如果它们是信息 Agent，可以通过修改文件或数据库来实现。然而，通过环境交互可能会影响交互各方的自治性，也就是说，各方除了交互之外没有其他选择。例如，机器人除了碰撞可能没有别的选择，而这种碰撞可能导致一方或双方打断原来的程序。类似的，一个信息 Agent 或许需要修改一些数据项来完成一项任务，另一个 Agent 或许需要读这个数据项，如果观察到数据项被修改，那么就需要特别注意这些被修改的值。

我们把通信定义为“保留参与各方自治性的交互”。通信并不要求采用计划或“信念—意图”结构来支持。从这个意义上讲，“通信”概念是基于自治的；如果交互能够保留相关各方的自治性，那么它就是符合要求的通信。很显然，一个给定环境中的碰撞和修改数据都可以作为传输信息的手段。在低层次上，通信是借助于某些物理手段来实现（也就是环境），如通过一个数据链路来发送数据包。但是，如果上升到各方都能够根据自己的意愿来进行通信（也就是发送数据和接收数据），那么就可以把通信看做是一个交互。

通信语言被很典型地划分为三个主要部分：传输层提供消息，在有限程度上是可靠的；内容层提供了表达相关领域细节的一个手段，与一个本体典型关联，在语言本身之外就被确定了；通信行为层确定了通信态度，如这个通信是否为断言、指令、许诺等。依靠这个理论，这组原语可能会发生变化，但是原语的数目一般很小（典型的是小于 10）。虽然在何种情况下选择何种通信行为是很清楚的，但是希望参与者在任何情况下都做出正确的行为选择还是有些困难的。因此，就有了一个趋势，仅仅选择一个通信行为（通常这对应于断言一个事实的行为，被称为“断言”或“告知”），为它加载所有其他通信行为的含义。也有一些反对这种倾向的尝试，采用更为丰富的对话和论证模型，也取得了一定的成功（Pasquier 和 Chaib-draa，2003），但是这些还没有体现在核心的软件方法学中。

通信是最重要的一个 Agent 概念抽象。大家经常想当然地，把这个概念和底层的数据传输（这个仅仅属于通信机制）相混淆。通信语义的研究目前主要有两种方法，分别基于心理概念和社会概念，已经足以说明问题了。如上所述，心理概念主要处理 Agent 的内部结构，因此并不适用于开放环境。社会概念则有着更广泛的应用，文献（Singh，1998）对此进行了更为严格的论述。

5．**协议**（Ptotocols）

很难采用一种独立的方式对通信进行研究。当没有办法去查看通信 Agent 的内部结构时，对通信行为的组合进行标识可能是更简单和更为合适的选择。这些组合就是协议，也就是前面所暗示的“对于 Agent 行为所采取的限制”。简言之，协议阐述了一个 Agent 应该何时，以

及如何同其他 Agent 进行通信。举一个简单的商业案例，一个 Agent 在受到询问时给出报价，在收到所订购物品时完成付款等。

计算机科学的其他分支也对协议进行了研究，尤其是在网络领域。例如，网络协议就限制了通信各方所能发送的消息和响应。这些协议定义了在各种情况下如何为需要通信的数据进行编码，如报头。严格的定义对于保证可靠的实现是非常重要的。然而，我们希望 Agent 的灵活性能够最大化（换言之，就是尽可能少的限制它们的自治性）。要兼容灵活性是非常有挑战性的，尤其是要确保 Agent 的行为能够正确符合一个协议的规定。

6．**承诺**（Commitments）

尽管传统的通信研究是基于心理概念的，而对协议的研究却是在社会概念的框架中形成的。我们这里就使用“承诺”这个典型的社会概念来实现目的（Singh，1999a）。虽然也可以定义别的概念，但是“承诺”对于实现当前目标已经足够了。“承诺”涉及债务人、债权人、条件或行为，以及上下文。基本含义就是债务人有义务帮助债权人实现所陈述的条件，或者执行所要求的行为；这些行为甚至经常被当成前提条件。给定的承诺可以在一定的上下文中获得，例如虚拟企业这样一个组织，或是供应链这样一个商业抽象。承诺可以通过以下手段进行操作，如委派（变更债务人）、转让（变更债权人）、取消（债务人违反承诺）、豁免（债权人豁免债务人）。像“取消”这一类操作，很明显是具有一定风险的。需要通过进一步的“元承诺”（用于定义基本层次承诺的操作环境）来限制它们。更重要的是，增加“元承诺”可以使我们为应用中的许多实际协议进行建模（Yolum 和 Singh，2002）。

关于承诺的早期研究工作假定：根据对所期望交互的理解，直接由设计者来决定所期望的承诺和元承诺。最近的工作已经产生了对于承诺更为精确的阐述，可以用做构建系统的基础。目前的时序逻辑方法中有可以表达承诺的语义，可以将“承诺的可重用交互模式”形式化（Fornara 和 Colombetti，2002）。这些模式可以用做设计多 Agent 系统的基础，并确保最终产生的交互具有某些属性。另一些工作则是基于对所期望交互的分析，考虑提出正确承诺的方法要素（Wan 和 Singh，2003），这体现了文献（Huhns 等，2002）的精神，但目标却是针对承诺。

1.3 Agent 的体系架构

除了 1.2 节所给出的 Agent 概念之外，还需要强调一种不同的观点。这种观点把 Agent 看做是软件系统的建模抽象，面向 Agent 的软件工程就是基于这种观点。在这里，Agent 通常用于模拟现实中的实体，软件系统被看做是为了实现预期功能而相互作用的一系列 Agent。Agent 凭借自主性、预动性和/或反应性等一系列特征，为应用程序开发人员描绘了一种很有前景的方法。更一般的，这意味着复杂应用程序可以在动态环境中运行。

基于这种观点，可以把 Agent 体系架构看做是软件模块的详尽集合，用带箭头的方框来表示数据和模块之间的数据流控制。Agent 体系架构可以分为慎思型、反应型和混合型。慎思型体系架构是最先出现的，其标志就是采用明确表示的世界符号模型，基于模式匹配的决策进程和符号操作技术。因此，体系架构需要解决的问题就是：如何把真实世界转换为符号，以及如何有效准确地描述决策进程。一般来说，解决这些问题需要大量的计算，这使得慎思