计算机科学2002Vol. 29№. 11

面向主体的开发方法和可视化建模工具

Agent-Oriented Methodology and Modeling Tools

季 强

(中国科技大学研究生院 北京100039)

Abstract This paper introduces an agent-oriented methodology and modeling tools based on MAGE. The methodology supports analysis, desing and implimentation of multi-agent systems. The modeling tools assist the developer in building multi-agent systems using the methodology through a set of visual model editors.

Keywords Agent, MAS

1 引言

主体(Agent)特别是多主体系统(MAS-Multi Agent System)的理论和技术以主体的概念为核心,为分布式系统的分析、设计和实现提供了一个崭新的途径。当前,面向主体的技术作为一种设计和开发软件系统的新方法已经得到了学术界和企业界的广泛关注。面向主体的技术要想取得成功的应用,工程化的面向主体的开发方法和相应的辅助开发工具是十分关键的。这不仅能保证建造的系统是可靠的和易于维护的,还使得系统的设计和实现可以由广大的软件开发人员进行,而不仅仅是主体技术的研究人员。

关于主体这一概念,最早可以追溯到早期分布式人工智能(DAI)的研究。自从人们开展对 DAI 的研究之时起,多主体系统开发工具就一直是重点的研究方向。从早期的 DVMT 和 MACE 等测试床到近年来的 dMARS 和 Zeus,出现了很多开发工具。其中一些开发工具如 Zeus 采用了 GUI 界面,为多主体系统的实现和调试提供了图形化的工具。这些开发工具往往致力于主体的结构框架以及主体系统的实现和调试,缺乏从需求出发指导系统实现的辅助开发工具。使用面向主体的方法进行系统的开发时,分析阶段和设计阶段缺少工具的支持。相比之下,面向对象的方法得到了众多成熟的商用软件的支持。如 Rational 公司用于面向对象的分析和设计的 UML 可视化建模工具 Rational Rose98i 等。另一方面,关于面向主体的开发方法,虽然到目前为止出现了很多这方面的文献,但是这些文献大多停留在多主体系统的分析和设计阶段,没有对应到实际的系统。

针对这种情况,我们提出了基于 MAGE 中主体结构框架的面向主体的开发方法,以 MAGE 作为最终生成的多主体系统的运行环境,通过为目标系统建立一系列的模型来完成系统的分析、设计和实现。同方法相对应,我们设计并实现了可视化建模工具,辅助用户从需求出发,运用面向主体的方法进行多主体系统的开发。

下面首先介绍 MAGE 和面向主体的开发方法,之后介绍了可视化建模工具,最后通过一个求解四皇后的实例展示了如何通过可视化建模工具运用面向主体的开发方法进行多主体系统的开发。

2 MAGE

MAGE(Multi-AGent Environment)是中科院计算所分布式人工智能研究组开发的多主体环境。MAGE 是基于 Java 语言的,由系统运行框架,主体通信语言 ACL,主体描述语言 ADL,系统服务,安全机制和编程模式等组成。MAGE 为主体提供了基本的结构框架和运行环境,附加上用户提供的功能模块就可以方便地构造多主体系统,为网络计算环境中的多主体系统的快速开发提供了一个平台。

MAGE 中,主体由主体内核(Agent Kernel)、主体通信语言解析器(ACL Parser)、主体描述语言解析器(ADL Parser)、调度器(Scheduler)、通信器(Communicator)、功能模块等部分构成。主体通过主体描述语言(ADL)定义,ADL 是一种开发主体的描述语言,每一个主体都对应一个由主体描述语言写成的文本文件,其中定义了主体的相关属性,如名字、局部地址、熟人地址、功能模块以及行为模式等。主体内核通过主体描述语言解析器获得主体的属性,由调度器按照主体的行为模式执行相应的功能模块以实现主体的功能。通信器和主体通信语言解析器负责完成主体之间基于主体通信语言的交互活动。功能模块则是主体功能的具体实现。

3 面向主体的开发方法

复杂的软件系统往往表现为由很多个部件组成,部件之间存在复杂的交互。软件工程的职责之一就是提供适当的模型和技术使开发人员以相对容易的方式处理复杂的系统。面向主体的方法利用主体这种抽象机制,把一个复杂的软件系统分解为若干个具有特定的目标要实现的、交互的、自治的部分(主体),关键的抽象模型以面向主体为中心。由于主体是一个自治的系统,能够同其它的主体交互以实现共同的目标。对于这类由多个部件组成,部件之间存在复杂的交互的问题,面向主体的分解和抽象为软件开发人员提供了更易于理解的建模和开发方法。

随着基于主体的技术越来越多地用于软件开发,出现了面向主体的软件工程。这方面的研究可以分为两大方向:一个方向是使用非形式化的方法分析、设计基于主体的系统;另一个方向是使用形式化的方法进行系统的规范说明、实现、验证等。形式化的方法在主体的理论方面有着重要的意义,非形式化的方法往往是从已经存在的一些软件开发方法发展而来,采用面向主体的方式,把主体作为主要的抽象机制。这些方法中,很多都是从面向对象的方法发展而来,比较有代表性的有 Kinny 等人的 BDI 方法、Odell 等人的 Agent UML 方法、Wooldridge 等人的 Gaia 方法等。非形式化的方法虽然精确性

不高，有很多不确定的人为因素，但是现有的软件开发人员对传统方法的熟悉，使得它们可以很快地被掌握。

通过对上述非形式化方法的分析和总结，我们提出了基于 MAGE 中主体结构框架的面向主体的开发方法，涉及多主体系统的分析、设计和实现三个阶段。分析阶段通过一组面向问题领域的抽象模型来描述目标系统的主要特征。设计阶段根据分析阶段产生的模型，得到描述目的多主体系统的一组模型，完成从描述问题领域的模型向具体系统实现的过渡。最后，在实现阶段、设计阶段产生的模型经过近一步的变换，得到各主体的 ADL 文件以及其它相关文件，附加上功能模块的具体实现后，就生成了一个基于 MAGE 的多主体系统。为了方便对模型的理解和相关操作，在模型的表示方面借鉴了 UML 的图示化模型表示方法，采用了图形的方式。下面简单介绍分析阶段和设计阶段的模型以及各阶段的基本流程。

3.1 分析阶段

分析阶段的主要目标是明确系统的结构和主要功能。从需求出发，按照功能划分和结构划分的方法把目标系统理解为一个由若干个角色组成的组织，每一个角色具有相应的职责，实现系统的一部分功能，系统的功能通过这些角色的交互活动来实现。

组织模型用于描述目标系统的总的功能和结构，也就是系统的宏观方面。组织模型由组织规则和组织结构组成。组织规则说明了系统总的功能和任务，依次展开并分配到具体的角色。组织结构定义了组织中的角色以及角色之间的关系。

角色模型用于刻画组织中每一个具体的角色，也就是系统的微观方面。角色具有两种属性，职责和能力。职责是角色所要实现的功能，能力指角色行使一定的职责所应具有的计算的能力，如某一种特定的算法的实现等。能力的引入主要是为了增强代码的重用性。能力可能是某一个角色专用的，也可能是所有角色通用的。在实现阶段，能力最终对应到 MAGE 中的功能模块。系统提供一个能力库，用于存放那些共同的、常用的能力。角色可以直接从中选择，或者定义自身的能力。

组织模型和角色模型通过组织结构图来表示，参见图2。图中大的图标表示组织中的角色，角色旁边小的图标表示角色的能力，角色之间的连线表明了角色间的关系。组织结构图以图形的方式表示了组织结构以及角色所具有的能力。组织结构图还对应一段文本，用于定义组织规则。每一个角色也对应一段文本，用于定义角色的职责。

分析阶段的工作流程就是从需求说明出发，用逐步求精的方法建立系统的组织模型和角色模型。主要步骤如下：

·依据需求说明，明确目标系统的功能和任务，定义组织规则。

·按照组织规则中功能和任务的划分以及目标系统的结构，确定目标系统中的角色。

·确定角色的职责，并根据角色的职责，决定角色所应具有的能力。

·按照角色的功能确定角色之间的关系，进行组织结构图的编辑。

3.2 设计阶段

设计阶段的目标是产生足够详细的，可以实现的关于目标系统的说明。主体系统模型对应分析阶段的组织模型，用于描述一个具体的多主体系统，定义了系统中存在的主体以及多主体系统的结构。由于分析阶段的角色与实际系统中的主体不一定存在一一对应的关系，而且一个具体的主体系统中，除了实现目标系统功能的主体之外，还会存在一些辅助功能的主体。因而主体系统模型同组织模型是不同的。主体系统模型是面向主体系统的，组织模型是面向问题领域的。主体模型对应分析阶段的角色模型，用于描述系统中每一个具体的主体以及主体同角色的对应关系，包括主体类型和主体实例。主体类型用于描述具有相同功能的一类主体，主体类型具有它所扮演的角色的职责和能力。主体实例对应系统中实际运行的主体。相同主体类型的不同主体实例可以具有不同的初始化条件。

主体类型图对应主体模型，参见图4。其中包括组织结构图中所定义的角色，最下面的一行图标表示系统中存在的主体类型，角色和主体类型之间的连线表明了主体类型对应的角色。主体类型边的标注表示系统开始运行时静态主体实例的个数。主体系统图对应主体系统模型，参见图5。其中大的图标表示系统中的主体实例，主体实例旁边小的图标表示主体实例所扮演的角色以及对应的主体类型。主体实例间的连线表示主体实例之间存在交互活动。每一个主体实例还对应一段文本，用于定义主体实例的初始化条件。

设计阶段的工作流程就是从分析阶段的组织模型和角色模型出发，逐步建立主体模型和主体系统模型，确定实现目标系统的多主体系统的结构以及系统中的每一个主体。主要步骤如下：

·依据分析阶段的组织模型和角色模型，确定系统中的主体类型以及主体实例。

·通过对主体类型图的编辑建立主体模型。

·通过对主体系统图的编辑建立主体系统模型。

3.3 实现阶段

实现阶段从设计阶段的结果出发，得到最终可以运行的主体系统。分为以下几步：

·根据主体模型，编写实现主体行为模式的程序代码。

·编写实现角色能力的功能模块的程序代码。

·编写各主体的 ADL 文件。

·编写多主体系统的 SDL 文件，定义系统中存在的主体以及各主体的初始化条件。

·主体系统运行程序根据 SDL 文件和 ADL 文件创建各主体，激活多主体系统的运行。

4 可视化建模工具的系统结构

同上述方法相对应，我们设计并实现了可视化建模工具，辅助用户运用面向主体的方法进行多主体系统的开发。用户通过可视化建模工具建立分析阶段和设计阶段的各种模型，完成从模型到系统所需描述文件的转换。可视化建模工具采用 GUI 界面，由项目管理器、可视化模型编辑器和自动转换工具组成。系统结构参见图1。

组成一个集成化开发环境的关键是各个工具之间的控制关系和数据交换。项目管理器具有集中控制的作用，用于全局管理多主体系统分析，设计和实现。用户通过项目管理器激活相应的工具来完成多主体系统开发不同阶段的任务。其它工具则对应多主体系统开发的不同阶段。工具之间交互的主要数据就是描述待开发系统的模型。

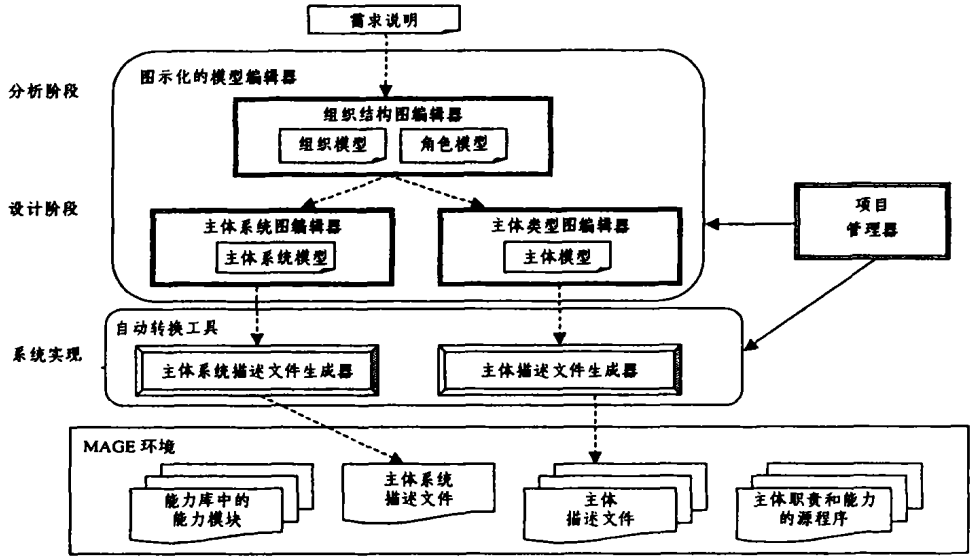

图1 系统结构图

4.1 项目管理器

项目管理器通过一个树形结构表示待开发的主体系统。用户点击其中的相关部分来激活相应的工具。树形结构分为以下几部分：

Analysis Phase	分析阶段的模型和相关信息
Design Phase	设计阶段的模型和相关信息
Implementation Phase	实现阶段自动转换工具生成的描述文件等。
Application	待开发多主体系统的相关信息，各种文件的路径等。
System	MAGE 的相关信息，系统文件的路径等。

项目管理器还负责读取和保存与待开发多主体系统相关的所有信息，包括各阶段的模型、生成的结果、当前的状态等。这些信息存储在一组项目文件中。

4.2 可视化模型编辑器

可视化模型编辑器用于辅助用户进行模型的建立和修改等操作，包括组织结构图编辑器、主体类型图编辑器和主体系统图编辑器。可视化模型编辑器类似于一个窗口环境下的绘图工具，但又有所不同。首先图示化的模型用矢量图形的方式来表示，而不是位图的形式，以便进行图形的增删、放缩和移动等操作；其次图示化的模型中的元素具有自动布局的能力，当用户移动一个元素时，与之相关的其它元素也会做出相应的移动。为了方便地对模型中的元素进行插入、删除和搜索等操作，模型采用多级链表的形式存储。链表的形式也使得模型之间的交叉引用可以方便地进行。

4.3 自动转换工具

自动转换工具用于设计阶段完成从模型到实际系统所需的描述文件的转换工作，包括主体描述文件生成器和主体系统描述文件生成器。主体描述文件生成器根据主体模型生成各主体的 ADL 文件。主体系统描述文件生成器根据主体系统模型生成多主体系统的 SDL 文件，自动转换工具通过对相关模型以及模型中元素的搜索和遍历，搜集所有必要的信息，生成 ADL 文件和 SDL 文件。

5 开发实例

下面以一个求解四皇后的问题为例，简单介绍如何通过可视化建模工具进行多主体系统的开发。求解四皇后的问题被看作一个由四个皇后和一个棋盘组成的组织，每一个皇后的职责是沿水平方向移动并保证自身在棋盘中的位置不与其它的皇后冲突。棋盘的职责是显示皇后移动的过程和结果。求解的过程就是4个皇后不断地移动并协商解决它们的布局的过程。

5.1 分析阶段

打开组织结构图编辑器，以鼠标拖放的方式在图中放置五个角色。其中四个角色命名为 Queen1、Queen2、Queen3、Queen4，另一个角色命名为 Display，用于显示棋盘的布局。Queen1～4和 Display 选用不同图标以便于区分。之后，连接存在交互的角色，标明角色间的关系。对于 Queen1—4，添加寻找不冲突位置的能力 NextPosition。对于 Display，添加显示棋盘布局的能力 DisplayChess-board，见图2。完成组织结构图的绘制后，项目管理器的应用程序窗口显示了组织结构图中的角色、职责和能力等，见图3。这时可以通过点击相应的图标进行详细的定义和设置，如实现能力的模块，角色职责的描述等。

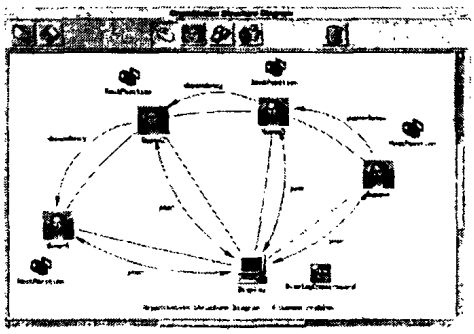

图2 组织结构图

5.2 设计阶段

打开主体类型图编辑器，定义角色和主体类型的对应关系。主体类型 QF 对应 Queen1，也就是第一个皇后。主体类型 QM 对应 Queen2和 Queen3，也就是中间的皇后。主体类型 QL 对应最后的皇后 Queen4，主体类型 D 对应 Display。运行时，QM 对应两个主体实例。其余的主体类型只对应一个主体

实例，见图4。

定义完主体类型后，打开主体系统图编辑器。放置一个QF类型的主体q1，两个QM类型的主体q2和q3，一个QL类型的主体，一个D类型的主体d。以鼠标拖放的方式指定主体扮演的角色，相应的角色间的交互活动体现在了主体之间，见图5。完成主体类型图和主体系统图的绘制后，项目管理器的应用程序窗口显示了系统中的主体类型和主体实例等。这时可以进行进一步的设置，包括主体类型的实现模块，主体实例的初始条件等。

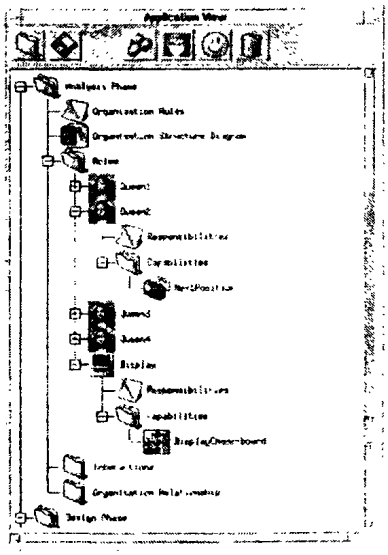

图3　项目管理器的应用程序窗口

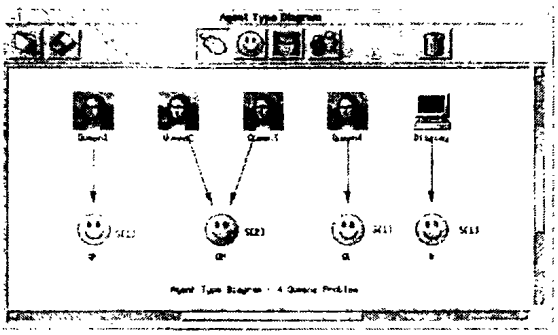

图4　主体类型图

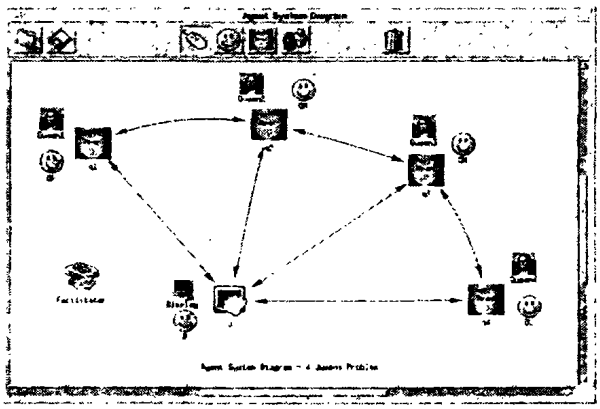

图5　主体系统图

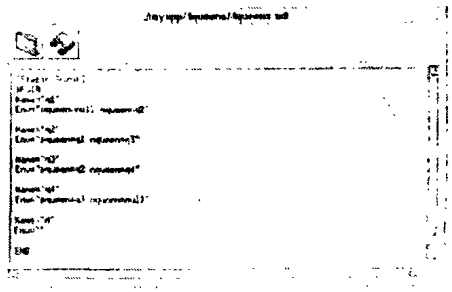

图6　主体系统描述文件

5.3　实现阶段

主体行为模式的程序代码和功能模块的程序代码编写完成后，调用转换工具生成主体系统描述文件和主体描述文件，见图6。通过主体系统描述文件和主体描述文件运行这个求解四皇后问题的主体系统。运行的结果和扮演皇后的主体之间的消息传递见图7。

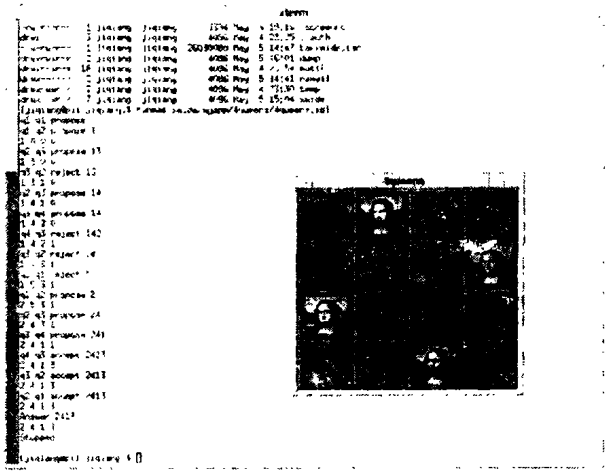

图7　求解四皇后问题的运行

结束语　本文介绍了基于MAGE的面向主体的开发方法和相应的可视化建模工具。可视化建模工具辅助用户使用面向主体的开发方法进行多主体系统的开发。在分析阶段和设计阶段，可视化的模型编辑器方便了用户进行模型的建立和修改等操作。系统的实现阶段，自动转换工具使得模型到实际系统的转换工作更容易进行。目前虽然已经可以使用可视化建模工具进行多主体系统的开发工作，但距离真正的实用还有一定差距。在开发方法中缺乏描述主体交互活动的模型，对于主体之间存在复杂交互活动的多主体系统的刻画不够完整。各工具的关联操作和GUI界面也存在一定的问题。这些都是今后的努力方向。

参考文献

1　Jennings N R, Wooldridge M. Agent-oriented software engineering. In: Handbook of Agent Technology, AAAI/MIT Press, 2000

2　Iglesias C A, Garijo M, Gonzalez J C. A survey of agent-oriented methodologies. In: J. P. Muller, M. P. Singh, A. S. Rao, eds. Intelligent Agents V (LNAI Volume 1555). Springer-Verlag, Berlin, Germany, 1999

3　Wooldridge M, Jennings N R, Kinny D. The Gaia methodology for agent-oriented analysis and design. Journal of Autonomous Agents and Multi-Agent Systems, 2000, 3(3)

（下转第160页）

(3)受限的细粒度可扩展性 当一个系统替换内核里的扩展并使用软件保护,细粒度的可扩展性常常是不切实际的,因为它意味着在内核里的每一个例程必须被保护。如果任何的程序能够被替换,那么任何内核对预想的安排在调用一个程序前或者在调用一个程序后都是无效的。结果是,软件保护系统通使用一个细粒度可扩展性的特殊情况——受限的细粒度可扩展性,它只允许内核功能的一个子集可扩展。在这样的系统里,难题是识别所有有用的可扩展点并确保它们的替换不会使系统的预想无效。

5. 扩展操作系统方法的归纳

表2是我们对几个典型的可扩展操作系统用四个关键设计点进行的技术分析总结。由于可扩展操作系统在具体实现时是多种多样的,可能同时采取多种设计方法和技术,对具体分析带来了无比的复杂性,因此,为突出重点,这里只列出了它所采用的主要方法。

表2 典型的可扩展操作系统设计

系统	位置	程序生成	保护	粒度
Apertos	内核	替换	软件	细粒度
	库	替换	硬件	细粒度
Exokernel	内核	替换	软件	程序上受限的
	库	替换	硬件	细粒度
Cache Kernel	服务器	替换	硬件	粗粒度
Choices	内核	替换	解释	粗粒度
Hydra	内核	选择	硬件	粗粒度
Scout	内核	选择	软件	细粒度
SPIN	内核	替换	解释	受限的细粒度
Spring	服务器	替换	软件	粗粒度
Synthetix	内核	推理	解释	细粒度
VINO	内核	替换	软件	受限的细粒度

结束语 虽然可扩展操作系统的设计方法不尽相同,但它们一般都能归纳到本文提出的可扩展操作系统四个关键设计点中,对于特定的应用需求,需要在继承传统操作系统优秀成果的基础上,通过综合考虑和利用这四个关键设计点,解决操作系统的性能和柔性问题。但是,今天的可扩展操作系统在实际应用中还面临着一系列的难题,最主要是系统的自适应性和可靠性,这都是今后操作系统研究中需要重点解决的问题。

参 考 文 献

1 Bershad B, et al. Extensibility, Safety, and Performance in the SPIN Operating System. In: Proc. of the Fifteenth Symposium on Operating Systems Principles, Copper Mountain, CO, Dec. 1995. 267～284

2 Chen B, et al. The Measured Performance of Personal Computer Operating Systems. ACM Transactions on Computer Systems, Feb. 1996. 3～40

3 Cowan C, et al. Adaptable Operating Systems. reference to paper in this same collection

4 Cao P, et al. Application-controlled File Caching Policies. In: Proc. of the 1994 Summer Usenix Technical Conf. Boston, MA, June 1994. 171～182

5 Campbell R, Tan S M. mChoices: An Object-Oriented Multimedia Operating System. In: Proc. of HotOS V, Orcas Island, WA, May 1995. 90～94

6 Engler D, Kaashoek F, OToole J. Exokernel: An Operating System Architecture for Application-level Resource Management. In: Proc. of the Fifteenth ACM Symposium on Operating Systems Principles, Copper Mountain, CO, Dec. 1995. 251～266

7 Fall K, Pasquale J. Exploiting In-Kernel Data Paths to Improve Throughput and CPU Availability. In: Proc. of the 1993 Winter Usenix Technical Conf San Diego, CA 1993, 327～334

8 Fiuczynski M, Bershad B. An Extensible Protocol Architecture for Application-Specific Networking. In: Proc. of the 1996 USENIX Technical Conf. San Diego, CA, 1996. 55～64

9 Ganger G R, Kaashoek M F. Embedded inodes and explicit grouping: Exploiting disk bandwidth for small files. In: USENIX 1997 Annual Technical Conf. Jan. 1997

10 Golub W, et al. Mach: A New Kernel Foundation for UNIX Development. In: Proc. of the Summer 1986 USENIX Conf. July 1986. 93～12

11 Liskov B, et al. Theta Reference Manual. MIT LCS Programming-Methodology Group Memo 88, Feb. 1995

12 Seltzer M I, Endo Y, Small C, Smith K A. Dealing with disaster: Surviving misbehaved kernel extensions. In: Proc. 2nd Symp. on Operating Systems Design and Implementation, Seattle, WA, Oct. 1996

13 Wahbe R, Lucco S, Anderson T E, Graham S L. Efficient software-based fault isolation. In: Proc. of the Fourteenth ACM Symposium on Operating Systems Principles, 1993. 203～216

(上接第153页)

4 Kinny D, Georgeff M, Rao A. A methodology and modelling technique for systems of BDI agents. Proceed-ings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World, (LNAI Volume 1038). Springer-Verlag, Berlin, Germany, 1996

5 Bauer B, et al. Agent UML: A formalism for specifying multiagent software systems. In: Proc. of the First Intl. Workshop (AOSE-2000). Springer-Verlag, Berlin, Germany, 2000

6 刘大有, 杨鲲, 陈建中. Agent 研究现状与发展趋势. 软件学报 2000(11)