MMAS:基于模型的多 Agent 系统开发方法*)

袁成祥　高　济
(浙江大学人工智能研究所　杭州310027)

摘　要　MAS的复杂性使其应用受到限制。本文提出一个基于模型的MAS开发方法——MMAS。MMAS用任务模型、角色模型、组织模型、交互模型、信念模型和Agent模型来描述一个MAS。本文阐述如何在这六个模型的基础上设计Agent,以实现软件重用,减少MAS开发的复杂性和提高软件开发速度。

关键词　软件Agent,MAS,模型,方法论,软件重用

MMAS:A Developing Method for Model Based Multi-Agent System

YUAN Cheng-Xiang　GAO Ji
(Institute of Artificial Intelligence,Zhejiang University,Hangzhou310027)

Abstract　The complexity of MAS restricts it's utility. This article proposes a Developing Method for Modelbased Multi-Agent System—MMAS. MMAS uses task model,role model,organization model,interaction model,belief model and Agent model to describe a MAS. This article represents how to use these six models to design a MAS to improve software reuse,decrease the complexity of MAS development and promote the software development.

Keywords　Software Agent,MAS,Model,Methodology,Software reuse

1　引言

随着软件Agent得到越来越广泛的应用,多Agent系统(MAS:multi-Agent system)也逐渐成为设计高性能分布式系统的重要技术之一,面向Agent的软件开发变得越来越重要。然而MAS的复杂性影响其质量、可靠性和开发速度。为确保MAS的性能和质量,MAS开发方法论研究很有必要。MAS开发方法论也引起人们注意,并陆续出现了一些方法论[1~8],但这些方法论只提供抽象概念层的设计指导,在如何降低MAS开发复杂性、提高软件重用等方面重视不够。本文提出一种旨在降低MAS开发复杂性、基于模型的MAS开发方法——MMAS(developing Method for Model based multi-Agent System)。

由于Agent在通信语言、互操作协议和编程语言等方面没有标准,这成为异构Agents之间协作的障碍,限制了MAS作用的发挥。解决此问题的前提是对MAS的清晰描述,本文提出用六个模型来描述一个MAS。用模型描述MAS是很多方法论都采用的方法[3,4],不同的是这些方法论对MAS的描述仅停留在概念层,而MMAS提供了描述模版;其次是模型的内容也不尽相同。

本文第2节阐述如何用六个模型描述一个MAS,第3节阐述如何在这六个模型的基础上实现一个Agent,从而组建一个MAS,最后是总结。

2　多 Agent 系统的开发模型

本文从组织角度出发为MAS建立模型。这六个模型分别是:任务模型、角色模型、组织模型、交互模型、信念模型和Agent模型。任务模型描述MAS实现的功能;组织模型描述这些功能如何在Agents和子组织间分配;交互模型描述Agents间的互操作;信念模型描述Agent如何对外界做出反应;Agent模型描述Agent的特征。MAS的特征远不只这些,这些模型已经足够描述一个MAS中Agents间的协作。

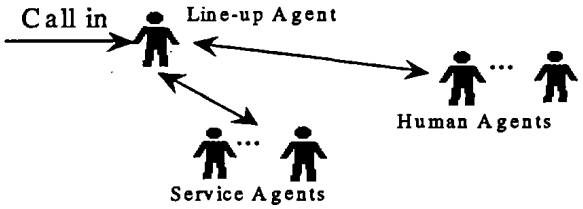

图1　简单的Call Center Agents交互图

本文用一个Call Center(呼叫中心,用电话来给客户提供某种服务)作为例子说明如何描述一个MAS。如图1所示,一个Call Center由3种Agent构成。Service Agent或Human Agent的作用是对外提供服务(Service)。Line-Up Agent的作用是接起呼入电话和调度ServiceAgent或Human Agent处理电话。Service Agent或Human Agent处理完业务后,发送消息给Line-Up Agent。

2.1　任务模型

任务模型描述MAS要完成的功能,例如任务分解、子任务调度等。单个Agent可以完成的任务称为任务单元,用Task-Unit表示。多个Agent才能完成的任务称为复合任务,用Compound-Task表示。任务用BNF定义如下:

〈Task〉::=[〈Compound-Task〉]|[〈Task-Unit〉]

〈Compound-Task〉::=[〈Name〉][〈Task〉]+[〈StatusMapping〉][〈Input〉][〈Output〉][〈PreProcessing〉][〈ExceptionProcessing〉][〈EndProcessing〉][〈PreCondition〉][〈PostCondition〉][〈SchedulingPlan〉]

〈PreProcessing〉::=[〈函数表达式〉|〈规则〉|〈规则组〉]*

〈ExceptionProcessing〉::=[〈函数表达式〉|〈规则〉|

*)本项目得到国家自然科学基金项目的资助,基金号69773019。袁成祥　博士生。高　济　教授,博导。

〈规则组〉]*

〈EndProcessing〉:=[〈函数表达式〉|〈规则〉|〈规则组〉]*

〈Output〉:=[〈Type〉〈变量名〉]*

〈Input〉:=[〈Type〉〈变量名〉]*

〈SchedulingPlan〉:= {〈PlanStep〉|(LOOP〈SchedulingPlan〉)}*

〈PlanStep〉:= RETURN
| (← 〈TaskSet〉[〈Status〉])
| (OR {(←〈TaskSet〉[〈Status〉])}+)

〈Status-Mapping〉:= $Status_1 \times Status_2 \times \cdots \times Status_n \rightarrow$ Status //$Status_i$ 是子任务状态集

〈Task-Unit〉: = [〈Task-Name〉] [〈Status〉]+ [〈Context〉]+[〈CS-Mapping〉][〈Out-put〉][〈Input〉][〈Pre-Processing〉] [〈ExceptionProcessing〉] [〈EndProcessing〉] [〈PreCondition〉][〈PostCondition〉][〈Constraint〉]

〈Context〉:=[〈Data-Item〉]+

CS-Mapping:= {(〈Context〉→〈Status〉)}+

每个任务都定义了输入、输出、前处理、后处理、先验条件、后验条件、异常处理、一个状态空间。复合任务的状态空间由子任务决定。子任务的状态空间由任务执行后的场景决定。所谓场景(Context),就是程序执行的某一时刻,与程序执行结果有关的数据的快照。复合任务由若干子任务构成,子任务的执行需要调度。子任务的调度采用静态的方法,在事前制订一个个规划步,执行时,根据任务的状态,选取其中的规划步来执行。这里使用的符号→(全函数)等都借用了 Z 语言的约定。

2.2 角色模型

角色是组织成员提供的功能、服务的抽象标识[10]。角色定义了充当此角色的组织成员负担的责任和义务。角色模型定义如下:

Role:=[〈Role-Name〉][〈Task-Name〉]+

在图1的例子中,有3个角色,分别是 Line-Up、Service 和 Human。下面是 Line-Up 角色的描述,我们采用 XML 作为描述语言。

```
〈Role〉
  〈Role-Name〉Line-Up〈/Role-Name〉
  〈Task-Name〉Take-Up(接起电话)〈/Task-Name〉〈Task-Name〉Scheduling〈/Task-Name〉
〈/Role〉
```

2.3 组织模型

组织是具有一定组织结构(Organizational Structure),遵循一定的组织规则(Organizational Rule),实现一定功能的 Agents 的集合。组织中成员是子组织或者 Agent。Agent 的组织结构是组织中成员交互模式的拓扑结构和组织活动的控制机理[9]。组织规则表示 MAS 正确的实例化和正常运转的一般需求,它是成员间、协议间、成员和协议间的关系和限制[9]。角色模型描述了角色要实现的功能,组织规则描述了 Agent 要充当的角色。组织定义如下:

〈Organization〉: = [〈Organization-Name〉] [〈Member〉]+ [〈Organization-Rule〉]*[〈Role-Name〉]*

〈Member〉:= Organization | Agent

Organization-Rule: {(〈Role-Name〉→〈Member〉)}+

Role-Name 是组织要充当的角色名。图1的例子,组织模型描述如下:

```
〈Organization〉
  〈Organization-Name〉Call Center〈/Organization-Name〉
  〈Member〉
    〈Agent-Name〉 Line-Up-Agent〈/Agent_Name〉
    ……
  〈/Member〉
  〈Organization-Rule〉
    〈Role-Name〉Line-Up〈/Role-Name〉〈Agent-Name〉
    Line-Up-Agent〈/Agent-Name〉
    ……
  〈/Organization-Rule〉
〈/Organization〉
```

2.4 互操作模型

组织中的角色相互影响,角色之间存在信息流和控制流。两个或多个 Agent 之间的一个消息序列称为一个会话。本文假定角色之间通过会话而不是黑板互操作。会话有双边和多边会话。按目的,会话可划分为信息查询、消息通知、命令下达和协商。互操作模型定义如下:

〈Conversation〉: = [〈Conversation-Verb〉][〈Sponsor-Description〉][〈Responser-Description〉]

〈Conversation-Verb〉: = "Ask"|"Inform"|"Query"|"Negotiate"|"Reply"

〈Sponsor-Description〉:=[〈Sponsor-Role〉][〈Conversation-Description〉]

〈Responser-Description〉:=[〈Responser-Role〉][〈Conversation-Description〉]

〈Conversation-Description〉: = [〈PreProcessing〉][〈EndProcessing〉][〈Context〉]+[〈Con-versation-Status〉]+[〈Cs-Mapping〉]+ [〈Event〉]+ [〈Translation〉]+ [〈a-operator〉]+[〈Constraint〉]+

〈Event〉: = [〈Conversation-Verb〉][〈Event-Name〉][〈Data-Item〉]+

〈Translation〉: ((Conversation-Status → a-operator)→Conversation-Status)|(Conversation-Status, Event.Event-Name)→Conversation-Status

a-operator 是对会话进行的处理。Constraint 是会话中的限制,例如返回消息的等待时间。Transition 描述会话中的状态转化。会话动词有5个:"Query"、"Inform"、"Ask"、"Negotiate"、"Reply",前4个分别代表查询、通知、任务指派、协商四类会话。会话的第一个消息中的动词标识这个会话的类型。其它消息都用"Reply",它表示这个消息是一个回应消息。会话的处理方式分别如下:

·第一类是 "Query"会话。查询 Agent(Agent$_q$)要知道向哪个 Agent(Agent$_{qd}$)查询。在此用一个称为信息模型的三元组定义 Agent$_q$ 如何找到 Agent$_{qd}$:

(Query-Role,Queried-Role,Conception-Name)

Agent$_q$ 由信息模型和组织规则找到 Agent$_{qd}$,并发送查询请求。Agent$_{qd}$处理查询后将结果向 Agent$_q$ 返回。Agent$_q$ 向 Agent$_{qd}$发送确认消息。会话结束。

·第二类 是"Inform"会话。信息拥有 Agent(Agent$_i$)先要找到被通知的 Agent(Agent$_{id}$)。每个 Agent 都维护一个列表 l,l 的结构如下:

[〈Object-Name〉][〈Agent-Name〉]+

l 中存储 Agent 的名字以及它感兴趣的数据。Agent 可以动态在其它 Agent 处注册它感兴趣的数据项。在任务处理过程中,Agent 对注册在 l 中,并且改变的数据项做一个标记。任务完成后,Agent$_i$ 检查 l 中的每个数据项,如果它发生改变,则通知对它感兴趣的 Agent 改变后的数据。这种会话以 Agent$_i$ 发送通知消息开始,以 Agent$_{id}$发送确认消息而结束。

·第三类是"Ask"会话。复合任务的执行 Agent(Agent$_m$)根据任务模型、角色模型和组织模型找到子任务处理 Agent(Agent$_{md}$),并发送命令。Agent$_{md}$返回确认消息结束"Ask"会话。

·第四类是"Negotiate"会话。协商是一组 Agent 为了在某问题上达成一致而进行的通信过程[11]。协商处理如下:协商开始前,由主控 Agent (Agent$_c$)初始化协商参数,然后将提议发送到协商参与者(Agent$_{cd}$)。Agent$_{cd}$对这个提议进行评估后,决定接受、拒绝这个提议或者提出另外一个提议,并把结果返回给 Agent$_c$。Agent$_c$ 对协商作出和 Agent$_{cd}$相同的处理

后，向 $Agent_{cd}$ 返回结果。如此循环，参数经过多次调整后，最后得到一个协议，或者失败。

$Agent_c$ 处理一个协商需要四个功能：参数初始化、评估、建议产生和状态控制。$Agent_{cd}$ 处理一个协商需要三个功能：评估、建议产生和状态控制。图2[12]是一个双边协商过程状态转化图。它定义了“提议”、“反提议”、“解除”和“结束”等会话名的消息。

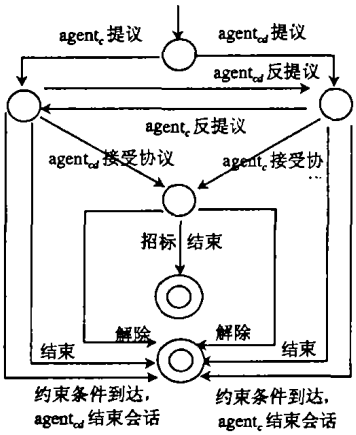

图2 双边协商状态转化图

协商可以用一个四元组描述：

(Neg-Sponsor-Description, Neg-Responser-Description, Parm-List, Neg-Constraint)

Neg-Sponsor-Description: =[⟨initialization⟩][⟨Evaluation⟩][⟨Su-ggest⟩][⟨Policy⟩]

Neg-Responser-Description: = [⟨Evaluation⟩] [⟨Suggest⟩][⟨Policy⟩]

Parm-List: =[⟨Item⟩]$^+$

协商定义中的“Neg-”前缀表示用来描述协商。initialization，Evaluation，Suggest 是算法，Policy 是策略。Parm-List 是协商的参数列表。Item 是协商项。算法使用 Policy 来调整或者确定参数。

“Query”、“Inform”和“Ask”会话处理流程简单，它们的描述不需存储在知识库中。“Negotiate”会话因为复杂，需要保存于知识库中，用来控制 Agents 完成协商。

2.5 信念模型

信念模型描述 Agent 如何对外界做出反应。例如 Agent 何时发起会话，何时执行任务。

Conversation-Trigger: (Conversation-Verb, Content, Role)← (Task-Name, Dest-Context)

Task-Trigger: Other-Task-Name ←(Task-Name, Dest-Context)

Belief: = Conversation-Trigger | Task-Trigger

任务完成后，可能会触发其它任务或者会话。在上面的信念模型中，Task-Name 是执行过的任务名，Dest-Context 是目标场景。任务完成后，Agent 将任务结束后的场景与目标场景比较，如果这两个场景符合，就发起由目标场景决定的会话或者执行一个任务。如果发起会话，Content 是要发送的内容，Role 是会话参与角色；如果要执行一个任务，Other-Task-Name 就是要执行的任务名。

下面是 Service Agent 的信念模型描述。

```
⟨Belief ⟩
  ⟨Conversation-Trigger⟩
   ⟨Conversation-Verb⟩Inform⟨/Conversation-Verb⟩
   ⟨Content⟩"Status = Free"⟨/Content⟩
   ⟨Role⟩Line-Up⟨/Role⟩
   ⟨Task-Name⟩Service⟨/Task-Name⟩
   ⟨Dest-Context⟩⟨/Dest-Context⟩
   ……
  ⟨/Conversation-Trigger⟩
⟨/ Belief ⟩
```

2.6 Agent 模型

描述 Agent 的特征，例如：标识、可以提供的服务、支持的本体论、通信方式等。Agent 能够提供的服务和其通信方法等对其它 Agents 透明。Agent 定义如下：

⟨Agent⟩: =[⟨Agent-Name⟩][⟨Service-Name⟩]$^+$[⟨Ontology ⟩]$^+$ [⟨ Communication-Method ⟩]$^+$ [(⟨ Task-Unit ⟩⟨Function⟩)]$^+$

Service 是 Agent 可以提供的服务。Ontology 是 Agent 支持的本体论。下面是 Line-Up-ag 的描述。Task-Unit 和 Function 的对应关系是 Agent 调用对应的函数完成实现指定功能的依据。

```
⟨Agent⟩
  ⟨Agent-Name⟩Line-Up-ag⟨/Agent-Name⟩
  ⟨Service-Name⟩Line-Up⟨/Service-Name⟩⟨Service-Name⟩
    Scheduling⟨/Service-Name⟩
  ……
⟨/Agent⟩
```

3 系统实现

MMAS 的六个模型在分析阶段提出了用 MAS 实现一个分布式系统时需要考虑的问题。在实现阶段，这些模型要用本体论表示语言描述，并存储在知识库中。

3.1 知识项和对象间的双向转化—KIOBE

Agent 要提供的服务可以用容器或镶嵌对象中的方法来实现。这种方法面临如下问题：Agents 间语义通信的消息中使用的是知识项，任务模型、场景等都以知识项的形式存储于知识库中，而 Agent 中的数据用对象表示。因此 Agent 要能完成对象和知识项间的双向转化，这样才能使用知识库或者进行语义通信。

对象和知识项间的双向转化由 KIOBE(Knowledge Item and object bi-directional exchanger)完成。Agent 中需要进行这种双向转化的对象是 Agents 间共享的数据以及与推理相关的数据。特定 Agent 编程语言的 KIOBE 实现代码可以由程序辅助产生。实现 KIOBE 可以先建立编程语言和知识库使用的本体论定义语言基本数据类型间的双向转化函数，如表1所示；然后为需要转化的对象产生转化方法。如图3所示，Agent 对象中的转化方法以及一个控制器构成 KIOBE。

表1 基本数据类型转化函数

XML to C++	XML 基本数据类型	C++基本数据类型	C++ to XML
IntegertoInt()	Integer	Int	InttoInteger()
PcDatatoCTime()	PCDATA	Ctime	CtimetoPcData()
PcDatatoCString()	PCDATA	Cstring	CstringtoPcData()
……	……	……	……

3.2 Agent 微内核

角色是系统功能中等粒度的聚合，将角色聚合成 Agent 仅仅是对角色类进行简单配置。Agent 的实现可以分成七个部分：协商引擎、推理引擎、执行引擎、KIOBE、对象集合、方法(Method)集合和用户操作接口。这种划分方法，因强调重点不同，存在重叠。这七个部分的关系如图3所示。

一、对象集合 在分析 Agent 要完成的每个任务和会话的时候，定义了与之相关的概念。系统设计时，这些概念转化

成 Agent 中的对象。Agent 是容器对象，这些对象是 Agent 中的镶嵌对象。

二、方法(Method)的集合　Agent 是一个对象，它通过对象中的方法为其它 Agent 提供服务。

三、KIOBE　其处理知识项和对象数据之间的转化。KIOBE 由一个控制器和各对象中的转化成员函数组成。支持多个本体论的 Agent 需要多套 KIOBE。

四、推理引擎　Agent 访问知识库、进行推理都由推理引擎完成，它是 Agent 的控制中心。推理引擎先获取任务描述，在任务执行前，先处理 PreProcessing，再调用执行引擎执行对应的功能。最后处理 EndProcessing。如果执行的过程中出现异常，处理 ExceptionProcessing。

五、互操作引擎　其处理 Agents 间的会话。具体完成如下功能：发送和接收消息；从消息中提取知识项；将 KIOBE 控制器传来的知识项包装成消息的格式；协调多个会话；判断会话的约束条件是否到达，检查会话的合法性等。

Agent 对会话的处理，在互操作模型中已经有所阐述。如果 Agent 接收到协商请求，根据图2的状态转化图，互操作引擎先调用参数评估函数计算参数，如果可以接受，就发送接受消息。否则调用建议函数产生一个建议，并将它发给协商请求者。

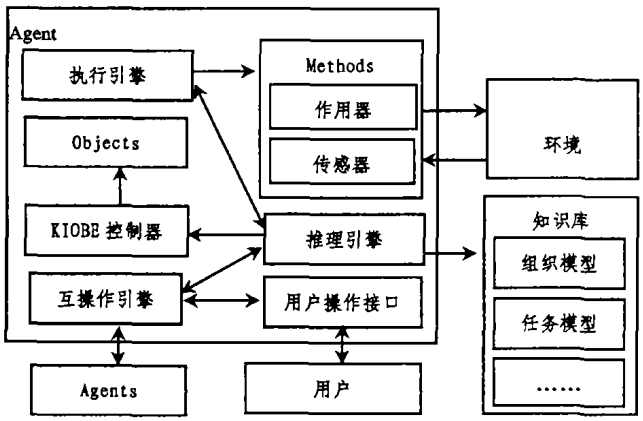

图3　Agent 结构图

Agent 也可以发起会话。为发起一个协商会话，先由推理引擎找到协商者，然后调用初始化函数初始化协商参数，再由会话引擎发送协商请求。收到回答后，互操作引擎将接受项从下轮协商中去掉，再调用评估函数计算参与者的建议，并由评估函数决定是不是要接受某些建议，最后推理引擎调用反建议产生函数产生另一建议。互操作引擎将这些结果发送给参与者。这个过程反复进行，直到协商的约束条件到达，或者就所有的参数达成一致。

六、执行引擎　其作用是调用 Agent 中的方法实现指定的功能，具体如下：调用 Agent 中的方法；作为复合任务的承担者，协调多个任务；判断任务的约束条件是否到达等。

七、用户操作接口　是 Agent 和操作者之间的中介。它处理 Agent 和操作者间的交互。Agent 和操作者间的通信也是语义通信。操作者可以看作是一个特殊的 Agent，此 Agen 既不会封装消息，也不会解释消息。所以操作者和 Agent 交互的时候，操作者发送给 Agent 的消息由 Agent 来封装，Agent 发送给操作者的消息也由 Agent 来解释。接口处理消息封装和解释。

Agent 的这七个部分中，互操作引擎、推理引擎、用户接口和执行引擎这四个部分与领域问题无关，这里称它们为 Agent 微内核。

采用 MMAS 开发的 MAS 由 Agents 和知识库两部分构成。知识库中存储模型描述。Agent 被其它 Agent 调用或者被操作者启动而执行。Agent 被要求执行一个任务后，它从知识库中获取此任务的描述。如果任务是复合任务，Agent 根据组织模型、角色模型以及 Agent 所处的场景对任务进行分配和调度；否则 Agent 调用对应的方法完成任务。任务完成后，Agent 根据信念模型确定是否要执行另一个任务或者是要发起一个会话。当 Agent 需要和其它 Agent 协商的时候，它参照知识库中的协商模型。

总结　MMAS 用七个模型来描述一个 MAS，使之对外界透明，从而异构 MAS 间也可以互操作。MAS 的描述是用来控制 Agent 的，描述的修改导致业务流程重构。Agent 的实现部分被分成七个部分，这七个部分中，有四个部分和领域问题无关，作为 Agent 的微内核，可以重复使用，这对提高 MAS 的质量，减少开发难度，提高开发速度有重要作用。

参 考 文 献

1 Iglesias C A, Garijo M, Gonzalez J C. A survey of Agent-oriented Methodologies. In: J. P. Müller, M. P. Singh, A. S. Rao, eds. Intelligent Agents V - Proc. of the Fifth Intl. Workshop on Agent Theories, Architectures, and Languages (ATAL-98), Lecture Notes in Artificial Intelligence. Springer-Verlag, Heidelberg, 1999

2 Sabas A, Badri M, Delisle S. Applying a New Multidimentional Framework to the Evaluation of MultiAgent System Methodologies. Intl. Symposium on Information Systems and Engineering (ISE 2002), San Diego (California, USA), Simulation Series, 2002, 34(2): 37～44

3 Iglesias C A, Garjo M, Gonzàlez J C, Velasco J R. Analysis and design of multiAgent systems using MAS-CommonKADS. In: AAA'97 Workshop on Agent Theories, Architectures and Languages, Providence, RI, ATAL. An extended version of this paper has been published in INTELLIGENT AGENTS IV: Agent Theories, Architectures, and Languages, Springer-Verlag, 1998

4 Wooldridge M, Jennings N J, Kinny D. The Gaia Methodology For Agent-Oriented Analysis and Design. Journal of Autonomous Agents and Multi-Agent Systems, 2000, 3(3): 285～312

5 Wooddridge M, Jennings N R. Pitfalls of Agent-oriented development. In: Proc. of Second Intl. Conf. on Autonomous Agents (Agent98), Minneapolis, MN, May 1998

6 Kinny D, Georgeff M, Rao A. A Methodology and modelling technique for systems of BDI Agents. In: W. van der Velde J. Perram, eds. Agents Breaking Away: Proc. of the SEventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW'96 (LNAI Volume1038), Springer-Verlag, Heidelberg, Germany, 1996

7 Sabas A, Badri M, Delisle S. A Multidimentional Framework for the Evaluation of Multiagent System Methodologies. 6th World Multiconf. on Systemics, Cybernetics and Informatics (SCI-2002), Orlando (Florida, USA), 2002, I: 211～216

8 Wooldridge M, Jennings N R, Kinny D. A Methodology for Agent-oriented analysis and design. In: Proc. of the Third Intl. Conf. on Autonomous Agents (Agents 99), Seattle, WA, May 1999. 69～75

9 Zambonelli F, Jennings N R, Wooldridge M. Organisational Rules as an Abstraction for the Analysis and Design of Multi-Agent Systems. Int J. of Software Engineering and Knowledge Engineering, 2001, 11(3): 303～328

10 Ferber J, Gutknecht O. A meta-model for the analysis and design of Organizations in multi-Agent systems. In: Proc. of the 3rd Intl. Conf. on Multi-Agent Systems (ICMAS 98). IEEE CS Press, June 1998

11 Bussmanm S, Muller J. A Negotiation framework for Cooperating Agents. In: Deen S M, ed. CKBS-SIG', Dake Centre, University of Keele, 1992. 1～17

12 Schreiber A T, Wielinga B J, Akkermans J M, van de Velde W. CommonKADS: A comprehensive Methodology for KBS development. Deliverable DM1. 2a KADSII/M1/RR/UvA/70/1. 1, University of Amsterdam, Netherlands Energy Research Foundation ECN and Free University of Brussels, 1994