ISSN 1000-9825 Journal of Software 软件学报 2000,11(10):1382~1389

A-ADL:一种多智能体系统体系结构描述语言*

马俊涛 傅韶勇 刘积仁

(东北大学软件中心 沈阳 110006)

E-mail: fusy@csc.neu.edu.cn

摘要 多智能体系统(multi-agent system,简称MAS)具有复杂的动态结构及行为特征.传统的体系结构描述语言(architecture description language,简称ADL)在语义和表达能力两方面难以满足要求.提出了一种基于智能体的体系结构描述语言A-ADL(agent-based-ADL).该语言采用计算/联接智能体替代部件/连接器作为体系结构单元,基于体系结构原语、规则及多视图,解决了多智能体系统的动态性和语义问题.通过实例详细阐述了A-ADL的动态体系结构建模机制,并通过相关研究的比较证明了A-ADL对于MAS的适用性.

关键词 智能体,多智能体系统,软件体系结构,体系结构描述语言,体系结构样式.

中图法分类号 TP311

基于文献[1]的弱定义,智能体(agent)是具有心理状态(mental state)的理性决策系统[2],在软件工程领域中被视为继过程、实体和对象之后的资源抽象形态.智能体主动、开放的计算特征及向问题空间映射的直观性决定了它在软件工程领域的广阔研究和应用前景,直接促进了面向智能体软件工程(agent-oriented software engineering,简称AOSE)[1,2]的迅速发展.

通过对国际上AOSE研究的综合考察[3],从软件过程模型的角度可得出以下结论:对多智能体系统(multi-agent system,简称MAS)实现层面的研究较为集中,涌现出大量的AOP语言、框架和实验床系统[1~3],如YAMS,DESIRE,GRATE*,ARCHON,MACE等.但这些系统一般借用面向对象软件工程方法并稍加修改,或者完全基于直觉和经验,缺乏系统的软件工程方法研究[3].

作为MAS软件工程方法研究的一个组成部分,我们设计并使用了一种基于智能体的体系结构描述语言A ADL(agent-based architecture description language).[4]设计A-ADL的必要性主要是依据以下观点:

(1) MAS同样属于软件范畴,尤其是针对MAS的复杂动态结构和行为特征,比如智能体的加入/退出、协作关系的建立/解除、智能体联盟的形成/解体等,更需要显式的体系结构模型来帮助设计者有效地理解、分析和控制MAS的运行形态;

(2) 在现有的技术条件下,描述MAS的体系结构可采用两种途径:体系结构描述语言(architecture description language,简称ADL)和智能体建模语言[5].文献[6]集中分析和对比了目前的ADL,如Unicon,Darwin,Wright,ACME,它们基于部件和连接器(connector)的体系结构单元[7]与智能体之间存在较大的语义断层,而且难以准确地捕获MAS结构和交互的复杂动态特征;某些框架系统(如DESIRE[5])提供的说明性建模语言虽与ADL在功能方面局部重叠,但建模语言侧重于系统的整体行为而不是计算和交互的解耦,ADL的目标则在于设计重用、早期决策、实现质量属性以及作为项目开发的设计蓝图.此外,智能体建模语言缺少某些体系结构基本概念(如连接器和体系结构样式),并且与特定的智能体模型和开发环境相关.

A-ADL以智能体作为体系结构单元,在体系结构规范和MAS之间采用相同的术语和语义,有助于消除"语

* 本文研究得到国家"九五"攻关项目基金(N96-B08-2-1)资助.作者马俊涛,1972年生,博士,讲师,主要研究领域为分布式处理,软件工程.傅韶勇,1974年生,博士生,主要研究领域为基于组件的软件工程,软件体系结构.刘积仁,1955年生,博士,教授,博士生导师,主要研究领域为分布式多媒体,软件重用.

本文通讯联系人:傅韶勇,沈阳 110006,东北大学308信箱软件中心

本文1998-12-14收到原稿,1999-08-30收到修改稿

义断层";同时,智能体之间的分布、协商、合作为动态体系结构的建模提供了比较系统的支持手段,从而可以解决观点(2)中所阐述的问题,适用于 MAS 的体系结构设计.

1 设计目标

当前,A-ADL 的主要设计目标是针对多智能体系统复杂的动态结构特征,提供丰富的动态体系结构建模机制.同时,A-ADL 必须实现体系结构单元表示、抽象封装、类型化和通信完整性等 ADL 的基本特征[7].

更系统地,A-ADL 的设计目标可从面向系统、面向语言和面向过程 3 个方面加以阐述.

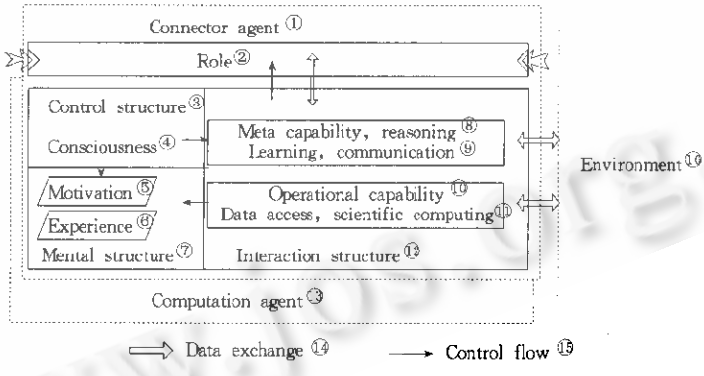
(1) 面向系统特征:A-ADL 主要针对 MAS 目标系统的体系结构建模,通过选择不同的实现路径同样可以支持其他分布、动态类型的应用.A-ADL 本身具有领域无关性;

(2) 面向语言特征:① 语法结构力求简捷有效,同时能够表达丰富的语义;② 复杂动态体系结构建模,捕获事务语义与动态体系结构配置间的互动;③ 基于多视图提供可扩展的显式质量属性,建立视图间的关联和引用.

(3) 面向过程特征:提供使用 A-ADL 创建、生效、分析及精化体系结构过程的辅助设施,通过消除先置约束支持不同的实现路径,并给出几种典型的实现路径方法和工具支持.

2 A-ADL 体系结构模型

ADL 的传统部件/连接器体系结构单元不适合 MAS:① 部件作为数据和处理单元,通常模型化为一组服务,这与智能体模型心理结构的语义差异较大(部件的语义过于简单);② 连接器的抽象层次较低,底层的通信机制(如 RPC)是典型的连接器类型,而 MAS 倾向于假设采取统一的通信语言(ACL)和消息机制.为了提供统一的语义和符号体系,A-ADL 体系结构模型采用计算智能体和联接智能体(如图 1 所示)替代部件/连接器作为体系结构计算和交互单元.



①联接智能体,②角色,③控制结构,④意识,⑤动机,⑥经验,⑦心理结构,⑧元能力,推理,⑨学习,通信,⑩操作能力,⑪数据访问,科学计算,⑫交互结构,⑬计算智能体,⑭数据交换,⑮控制流,⑯外界环境.

Fig. 1 A-ADL architectural element C_2ME model

图 1 A-ADL 结构单元的 C_2ME 模型

定义 1. A-ADL 体系结构模型 $Arch=\bigcup_{i=0}^{n}\mathrm{ArView}_i$,$\mathrm{ArView}=(A_c,A_r,\Gamma_{c\to r})$,其中:

A_c,A_r:计算、联接智能体,两者构成 A-ADL 体系结构单元;

$\Gamma_{c\to r}$:$A_c\to A_r$:计算、联接智能体间的接口(角色)绑定,对应于体系结构形式[7](form);

ArView:体系结构视图.

传统的 ADL 忽略质量属性造成隐式设计标准[6],A-ADL 的多视图结构将这些隐式标准有选择地公开化,视图 ArView_i 暴露 MAS 不同侧面或质量属性,所有视图的叠加构成完整的体系结构模型.A-ADL 缺省定义 ArView_0 为基本视图,$\{\mathrm{ArView}_1,\ldots,\mathrm{ArView}_i\}$ 是扩展视图,扩展视图数量由设计者决定.

2.1 A-ADL 体系结构单元

A-ADL 本身是抽象的体系结构描述语言,同时,与 MAS 目标系统保持语义的连续性对两者的转换非常关键,因此,A-ADL 试图采纳一种统一的概念和实现模型来描述计算、联接智能体.基于心理状态构筑概念模型是目前智能体研究界的共识[8,9],其中 BDI[8]最具影响力.BDI 模型的信念-愿望-意图认知结构虽然描述了计算的决策过程,但具体执行计算的结构却在模型中没有体现.这虽然有效地简化并有利于智能体理论的研究,但与 A-ADL 中突出计算特征的计算智能体存在矛盾,也不能直接用做智能体实现模型.

A-ADL 将 BDI 模型扩展为控制、心理和交互这样 3 层结构,提出 C_2ME(⟨Consciousness,Capability,Motivation,Experience⟩)模型(如图 1 所示)作为统一的概念和实现模型,C_2ME 的核心语义是控制结构(意识)解释心理结构(动机、经验)作出计算决策,并驱动交互结构(能力)执行计算.交互结构提供 BDI 缺少的计算部件,心理结构可视为简化的 BDI 模型.限于篇幅,C_2ME 模型的深入探讨和语义不在本文给出.

定义 2. 计算智能体是采用 C_2ME 模型的计算单元,$\alpha_c=\langle\varepsilon_0,\gamma_0,\theta_0,\zeta\rangle$,其中 $\varepsilon_0,\gamma_0,\theta_0,\zeta$ 分别是计算智能体的初始经验、初始操作动机、初始操作能力和意识.

定义 3. 联接智能体是包含计算结构的交互单元,$\alpha_r=\langle\varepsilon_0,\gamma_0,\theta_0,\zeta,r,\Gamma_{r\leftrightarrow r}\rangle$,其中

$\varepsilon_0,\gamma_0,\theta_0,\zeta$:联结智能体的计算结构,本质上是一个计算智能体的超集;

r:联结智能体的角色(role)端口,是角色确定交互的合法参与者;

$\Gamma_{r\leftrightarrow r}$:$r\leftrightarrow r$ 是联结智能体的内部角色端口绑定.

定义 4. 角色是可复用的智能体抽象框架,$r_c=\langle\theta_0,\zeta\rangle$,$r_r=\langle\theta_0,\zeta,r,\Gamma_{r\leftrightarrow r}\rangle$,其中 r_c,r_r 是计算、联接角色,对应于计算/联接智能体,其共同特点是缺乏心理结构.

A-ADL 有两个假设:(1) 智能体采用统一的 ACL 和消息传递机制;(2) ACL 支持 A-ADL 计算和联接智能体的通信原语.基于假设(1)和(2),A-ADL 在体系结构模型中保持了智能体协作的本质特征;基于假设(1),联接智能体从更高的抽象层次提取、描述并动态确定计算单元交互模式的结构和语义,而不再表示底层通信机制,如过程或远程过程调用.

抽象层次的提高使 A-ADL 客观上拥有一个重要特征:联接单元比计算单元具有更强的稳定性,交互模式在不同时间和上下文中可能展现出不同的结构和行为.为实现这一特征,联接智能体采用角色确定交互的合法参与者,基于"具有能力充当某种角色"的原则,凡是以直接或间接方式(继承)实现该角色的智能体都可参与联接智能体的交互模式.联接智能体具有计算智能体结构,在系统运行态对参与交互的角色及数据交换进行推理,从而提供附加的数据过滤、处理,改变交互参与方的类型及数量,支持 MAS 的动态体系结构.

2.2 类型系统

A-ADL 类型系统支持两种机制:(1) 角色的实现:智能体可实现角色,角色之间也可互相实现;②智能体的继承,能够重载或增加父类的心理结构.A-ADL 类型系统定义了以下约束:

① 计算智能体可实现 $n(n\geqslant 1)$个计算角色,只能继承一个计算智能体,不允许实现(继承)联接角色(智能体).角色无心理结构,因此,多角色实现间接提供了多继承机制,同时,可有效避免冲突(交互结构必须由心理结构驱动,心理结构在 A-ADL 中认为是唯一的冲突来源).仅从比较的角度而言,角色在 A-ADL 中类似于 Java 编程语言中的接口;

② 联接智能体可以同时实现 $n(n\geqslant 1)$个计算/联接角色,但只能继承一个计算/联接智能体.A-ADL 建议保持计算和交互逻辑的"事务分离",计算智能体不允许继承联接智能体则体现了这一原则.

3 A-ADL 体系结构样式和实例

体系结构样式表示一族系统的共性,本质上为面向设计重用的体系结构模板或骨架.体系结构实例是体系结构样式的特化,描述特定领域目标系统的软件体系结构.

定义 5. 体系结构样式是由角色确定的抽象计算和交互模式,$S=\langle R_c,R_r,\Gamma_{c\leftrightarrow r}\rangle$,其中 R_c,R_r 是计算、联接角色集合;$\Gamma_{c\leftrightarrow r}:R_c\rightarrow R_r(R_r)$是计算角色向联接角色中角色端口的绑定.

定义 6. 体系结构实例 $I=\langle S, I_s, A_c, A_r, \Gamma_{c\to r_c}, \Gamma_{r\to r_r}\rangle$,其中:

I:体系结构实例,可实现多个体系结构样式 S,并继承唯一的父类 I_s;

A_c,A_r:计算、联接智能体集合;

$\Gamma_{c\to r_c}: A_c \to R_c(S)$:计算智能体向计算角色的实现映射;

$\Gamma_{r\to r_r}: A_r \to R_r(S)$:联接智能体向联接角色的实现映射.

从过程模型的角度出发,体系结构样式和实例都是体系结构设计不同阶段的软件产品,并形成了增量关系.体系结构建模过程可抽象为函数 $\Gamma_m(R\to I)$,输入参数为需求规范,输出体系结构实例.在 A-ADL 中,这个过程通过以下步骤来实现:

① 根据需求模型,选择可重用的体系结构 S' 和体系结构实例 I_s;

② 适应 S' 得到体系结构样式 S;

③ 继承 I_s 并实现 S,设计智能体集 A_c 和 A_r,完成 A_c 和 A_r 向样式中角色 $R_c(S)$ 和 $R_r(S)$ 的映射 $\Gamma_{c\to r_c}$ 和 $\Gamma_{r\to r_r}$,得到体系结构实例 I'.此步骤可视为向角色分配心理结构而实例化的过程;

④ 通过定义新的角色和智能体,继承、重载父类的心理结构和角色绑定,精化 I' 为最终的体系结构实例 I.

从以上过程可以看出,可复用的体系结构样式库是体系结构建模过程的重要参数.

4 动态体系结构建模

对 MAS 而言,不同时刻和上下文,尤其是目标的差异可能导致设计阶段难以预测的动态结构和行为.为简化问题,A-ADL 将动态体系结构分解为两种类型:

① 质量属性在特定条件下决定的动态结构,此为外因.质量需求(即所谓 NFR,或称非功能需求)是设计空间选择决策的重要参数,决定目标应用的系统级特征(如可靠性、可管理性、可伸缩性等),这些特征包含了在特定上下文条件下进行系统动态重构的策略;

② 事务逻辑本身需要改变系统拓扑结构,此为内因.这在 MAS 中体现为智能体间动态的协商和合作、智能体的引入/退出、合作关系的建立/失效等.反过来,系统的结构同样影响智能体的问题求解过程,因此,这种动态性在本质上体现为体系结构配置与事务逻辑间的互操作.

4.1 动态体系结构类型的转化

A ADL 的视图被用于建模类型①的动态体系结构.A-ADL 把体系结构实例组织为数量可变的多个视图,每个视图刻画相对独立的功能或质量属性侧面.为了对类型①、②及不同质量属性所导致的动态结构进行解耦,A-ADL为不同的质量属性分配独立的视图,然后指定视图间(尤其是与基本视图)的关联和冲突消解原则,扩展视图内可以分配新的心理结构,或者增加体系结构单元.

在多视图构造之后,类型①的动态结构转化为多个视图内部的类型②的动态结构.

4.2 体系结构原语及语言特征

为描述 A-ADL 如何建模类型②的动态体系结构,有必要分析传统 ADL 在这方面存在不足的本质原因.对 Darwin,Wright,UniCon 的研究表明,它们普遍假设存在集中的控制结构,在体系结构层次实施集中的动态配置,构件只以被动的方式参与其中.这种方式的动态体系结构由设计者决定,具有某种规律性,在设计态可以预见.

由于智能体的主动性和多智能体系统的松耦合特征,所有导致系统动态特征的全局行为都可以分解为多个智能体的局部动作,这使 A-ADL 取消上述假设成为可能.基于这个思想,A-ADL 假设由智能体分散地承担动态重构的责任,通过合作实现全局的动态体系结构.具体方法是,在智能体心理结构中引入两个体系结构原语 new 和 bind,分别描述智能体能够执行的两种体系结构操作:体系结构单元的动态创建和绑定.基于 new 和 bind,可以实现以下类型复杂的动态体系结构的建模:

① 动态体系结构交互单元.A-ADL 体系结构原语以体系结构配置为作用对象,使计算智能体能够动态地创建联接智能体,分配初始的角色绑定,同时,联接智能体可以自主确定角色绑定.而传统 ADL 在体系结构层次

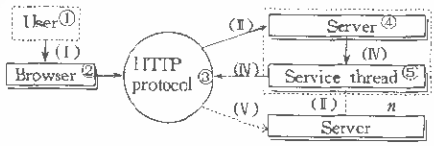
的集中配置很难描述动态交互单元的角色绑定;

② 由计算单元或交互单元发起的体系结构进化.A-ADL 的体系结构原语允许智能体自主地决定自身相关的体系结构配置.计算智能体能够终止现有的协作关系,转而与其他智能体合作.在满足约束的前提下,联接智能体可以动态地确定参与交互的计算智能体类型、数量及角色绑定.

在语言特征方面,A-ADL 采用规则语言 C-CLIPS[14]描述心理结构,并以 C-CLIPS 函数实现了 A-ADL 的体系结构原语.A-ADL 支持参数化描述数量不确定的结构单元,可定义为向量,维数通过参数传递确定,适合于初始化阶段确定的动态结构.A-ADL 的类型系统支持动态的结构单元类型,只要实现了合法的角色,不同类型的智能体都可能与联接智能体建立动态角色绑定.值得说明的是,根据 A-ADL 的假设(2),实现体系结构动态配置的具体协议不在 A-ADL 中捕获.

5 实例分析

为了便于和 Wright[10]进行比较,本文采用 A-ADL 描述一个如图 2 所示的 WWW 动态体系结构.它属于客户/服务器模式,有以下几点动态特征:(I) 在运行态,浏览器由用户创建,其数量和类型不受限制;(II) 系统中存在 n 个服务器,n 在初始化时确定;(III) 根据浏览器输入的 URL,动态地建立浏览器和服务器的联接;(IV) 服务器每接收到一个客户请求,就为其创建一个独立线程来提供响应;(V) 在建立浏览器和服务器联接时,能够考虑服务器性能的质量属性,如服务的延时,来动态地选择服务器.



①用户,②浏览器,③HTTP协议,④服务器,⑤服务线程.

Fig. 2　A dynamic WWW architecture

图2　动态的WWW体系结构

图 3 首先定义了 WWW 体系结构中的角色和智能体类型,从图 2 所示的问题空间将浏览器、服务器和服务线程抽象为 Browser,Web_server,Url_provider 计算智能体,将 HTTP 协议抽象为 HTTP 联接智能体,分别实现 Client,Server 计算角色和 C/S 联接角色.图中加重表示的为体系结构原语.

为了描述动态特征(III),HTTP 联接智能体具有动机 on_request 分析客户角色的 URL 请求,on_request 首先驱动能力 find_server 返回能够提供服务的服务器标识,然后采用 bind 体系结构原语,将服务器角色端口与返回的服务器智能体动态绑定,并转发来自客户的服务请求.

Web_server 实现了动态特征(IV).接到 URL 请求时,Web_server 已由动态特征(III)确定了与 HTTP 联接智能体 x 的角色绑定.为了提高服务性能,Web_server 采用体系结构原语 new 动态创建服务线程智能体 Url_provider 并返回标识,然后用原语 bind 取消自身与联接智能体 x 的角色绑定,将新建的 Url_provider 与 x 动态绑定,转由 Url_provider 提供服务.基于 A-ADL 的类型系统,Web_server 和 Url_provider 虽然类型不同,但都实现了 Server 角色,因此,上述针对 HTTP 的动态绑定是合法的.

图 4 在图 3 的基础上定义了两个视图:基本视图 PRIMARY 和扩展视图 EXTENDED,在视图中给出了计算和联接智能体类型的实例和初始联接.从图 4 可以看出,基本视图以参数传递的形式实现动态特征(II),A-ADL 的体系结构初始配置分散于不同的智能体.虽然也可借用传统 ADL 的集中配置模式,但 A-ADL 自动将其分解.

扩展视图中用于实现动态特征(I)和(V).由于浏览器的数量即使在运行态也难以确定,扩展视图将用户抽象定义为计算智能体 User,它展示了 A-ADL 体系结构交互单元的动态创建.User 首先采用原语 new 创建一个 Browser 计算智能体和一个 HTTP 联接智能体,然后采用原语 bind 将 Browser 和 HTTP 的客户角色端口动态绑定,HTTP 的 Server 角色绑定则根据 URL 请求动态地确定.

为了描述动态特征(V),本例假设若服务器超时则放弃,建立新的服务器绑定并重发服务请求.扩展视图以

客户端透明的方式,为HTTP联接智能体增加容错的动机on_timeout,延时超过阈值的条件一旦满足,HTTP将重新确定角色绑定,重发服务请求.

```
//definition of roles in C/S①
CROLE Client {  //client role②
  CAPABILITY send_request,accep:_response}
CROLE Server :   //server role③
  CAPABILITY accept_request, send_response}
RROLE C/S {  //C/S connector role④
  ROLE Client, Server}

//definition of agents⑤
CAGENT Browser {//browser agent⑥
  IMPLEMENT Client;
  MOTIVATION
    Web access: {
      (input &url)→(send_request &url);
      (accept_response &c)→(render &c)},
  CAPABILITY input, render;}

CAGENT Web_server {//server agent⑦
  IMPLEMENT Server;
  EXPERIENCE {(ragent x)};
//connector agent to bind⑧
  MOTIVATION web_service: {
    (accept_request &url)→//create thread dynamically⑨
    (bind x (new Url_provider&url))}}

CAGENT Url_provider {//thread agent⑩
  IMPLEMENT Server;
  MOTIVATION provide_content: {//provide service⑪
    (→(translate &url,&c)//run automatically, then exit⑫
    (send_response &c) (exit))}
  CAPABILITY translate;}

RAGENT HTTP{//HTTP connector agent⑬
  IMPLEMENT C/S;
  MOTIVATION
    on_request: {//dynamically find server on request⑭
    (→(bind Server (find_server &url))
    (send_request &url))}
  CONNECT Client.send_request→ Server.accept_request;
        Client.accept_response→Server.send_response;
  CAPABILITY find_server,}
```

①客户/服务器结构的角色定义,②客户角色,③服务器角色,④联接角色,⑤计算和联接智能体类型,⑥浏览器智能体,⑦服务器智能体,⑧绑定的联接智能体,⑨动态创建服务线程,⑩服务线程智能,⑪提供服务,⑫自动执行,然后结束,⑬联接智能体,⑭接到请求后动态确定服务器.

Fig. 3 Definition of roles and agents in WWW architecture

图3 WWW体系结构的角色和智能体类型

```
//primary view, the inheritance of
    architecure style is omied①
ARCHITECTURE WWW (m) PRIMARY {
    //primary view②
  CAGENT Browser, Web_server[m];
  RAGENT HTTP;
//initial connection, don't bind server statically③
Browser {EXPERIENCE {(ragent HTTP);}
HTTP {CONNECT Client Browser; }
//traditional configuration style④
//CONNECT Browser →HTTP. Client;
}
//extended view⑤
ARCHITECTURE WWW EXTENDED {
CAGENT User {
  MOTIVATION create_browser: {
        //create new browser⑥
    ( →(bind (new Browser) (new IITTP). Client))}}

//change server if timed out⑦
RAGENT HTTP {
  EXPERIENCE {(timeout 200)};
  MOTIVATION
    on_timeout: {//find and bind new server dynamically⑧
      (>waittime timeout) →
        (bind Server (find_server &url))
        (send_request &url)}}
}
```

①基本视图,限于篇幅,略去继承体系结构样式的步骤,②基本视图,③初始联接,不静态联接服务器,④初始联接的集中配置方法,⑤WWW体系结构的扩展视图,⑥新建浏览器,⑦若反应时间超过指定时间,则改变服务器,⑧若超时,则改变服务器.

Fig. 4 Definition of basic and extended views of WWW architecture

图4 WWW体系结构的基本视图和扩展视图

6 与其他ADL的对比分析

Unicon,Darwin和Wright在传统ADL中具有一定的代表性,它们是通用性的,没有领域特定的语法结构,

对目标系统的领域特征不作特殊约束.在适用目标方面,Darwin,Wright 分别基于 π 演算和 CSP 过程代数的语义,使二者本质上适合于分布、并发类型的应用,Wrigh: 尤其强调体系结构的形式化以及体系结构性质的推理,为正确性要求比较严格的系统设计提供了工具.

此外,还出现了一种 ACME 体系结构交互(interchange)语言,支持不同类型 ADL 规范之间的映射,被用于支持不同 ADL 之间的设计和工具的集成.A-ADL 与 Unicon,Darwin,Wright 相比,具有与 ACME 类似的"专用性"以及领域无关性.但主要针对 MAS 类型的目标系统.

A-ADL 面向 MAS 的设计目标决定了其采用 C_2ME 智能体作为体系结构单元,既继承了 BDI 模型的心理结构和语义,又显式地提供了控制和计算结构,比传统的部件和连接器具有更丰富的语义和表达能力,同时和智能体的设计与实现保持了连续的语义.

在动态体系结构建模方面,Unicon 只提供有限的机制定义新的连接器类型,不能描述运行态发生的体系结构变化.Darwin 具有惰性和动态实例化两种动态机制[11],但它只涉及计算结构单元,对于事务逻辑和体系结构配置的互动没有考虑[10].Wright 基于 CSP 语义实现动态体系结构[10],表现在通过控制事件和控制视图决定重新配置的条件和如何触发重构,具有较强的动态体系结构描述能力.但它侧重于动态的体系结构形式,缺乏很强的结构单元动态特征,同时,引入外界不透明的控制事件也影响了它的可复用性.

A-ADL 联接智能体在更高的层次上提取、抽象智能体交互模式,并具有心理和交互结构使之承担交互的控制责任.与 Darwin,Unicon,Wright 不同,A-ADL 取消了集中控制结构假设,在心理结构中定义了体系结构原语,支持了事务逻辑与体系结构配置的直接互操作,能够实现复杂的动态体系结构建模(如动态交互单元)以及由计算或交互单元发起的动态配置,这些动态特征在 MAS 中都是很常见的.

A-ADL 的多视图借鉴了 Wright 的控制视图思想,但通过提高抽象层次,不在体系结构中捕获智能体协商/合作的具体协议,从而避免引入控制事件,降低了体系结构单元间的耦合度.

A-ADL 与 Unicon,Darwin,Wright 相比存在以下不足:(1) 仍处于研究阶段,缺乏足够的工具支持,目前仅提供 A-ADL 支持系统装配的辅助工具,不能直接由 A-ADL 规范生成目标系统,而 Unicon,Darwin 已经提供了向可执行系统的转换工具;(2) A-ADL 语言采用基于 BDI 的语义,同时可以基于 CLIPS 描述计算单元和联结单元的语义,但是,在体系结构特性的推理上弱于 Wright 和 Darwin,尤其是 Wright 以 CSP 同时作为 ADL 以及其中部件的语义描述工具,并且可以推理系统的一致性和正确性;(3) A-ADL 是面向 MAS 的 ADL,虽然 A-ADL 被设计成可以具有面向对象、过程的实现路径,但目前在非 MAS 应用方面仍然缺乏有效的过程支持.

7 结束语

目前,我们已经为 A-ADL 提供了初步的支持工具,并且把它应用于基于智能体应用系统的开发之中.实践表明,A-ADL 对 MAS 具有足够的表达能力,而且保持了语义的连续性,在指导 MAS 系统的设计、实现以及设计复用方面达到了设计目标.在以后的一定时间之内,我们计划进一步完善 A-ADL 的工具支持,以提高 A-ADL 的应用价值.

参考文献

1 Wooldridge M, Jennings N R. Intelligent agents: theory and practice. The Knowledge Engineering and Review, 1995,10(2):115~152

2 Wooldridge M. Agent Based software engineering. IEEE Transactions on Software Engineering, 1997,144(1):26~37

3 Jennings N R, Sycara K P, Wooldridge M. A roadmap of agent research and development. Journal of Autonomous Agents and Multi-Agent Systems, 1998,1(1):7~36

4 MA Jun-tao. Research on agent-oriented compositional software architecture-AOCS [Ph.D. Thesis]. Shenyang: Northeastern University, 1999
(马俊涛.智能体组合软件体系结构 AOCS 研究[博士学位论文].沈阳:东北大学,1999)

5 Brazier F M T, Dunin-Keplicz B M, Jennings N R. DESIRE: modeling multi-agent systems in a compositional formal

framework. Journal of Cooperative Information Systems, 1997,6(1):67～94
6 Garlan D, Perry D E. Introduction to the special issue on software architecture. IEEE Transactions on Software Engineering, 1995,21(4):269～274
7 Perry E, Wolf A L. Foundations for the study of software architecture. ACM Software Engineering Notes, 1992,17(4):40～53
8 Rao A, Georgeff M. Modeling rational agents within a BDI-architecture. In: Fikes R, Sandewall E eds. Proceedings of Knowledge Representation and Reasoning (KR&R'91). San Mateo, CA: Morgan Kaufmann Publishers, Inc., 1991. 473～484
9 Shoham Y. Agent-Oriented programming. Artificial Intelligence, 1993,60(1):51～92
10 Allen R, Douence R, Garlan D. Specifying and analyzing dynamic software architectures. In: Egidio Astesiano ed. Proceedings of Fundamental Approaches to Software Engineering. Libson Portugal: Springer-Verlag, 1998. 21～37
11 Magee J, Kramer J. Dynamic structure in software architectures. ACM Software Engineering Notes, 1996,21(9):3～14

A-ADL: an Architecture Description Language for Multi-Agent Systems

MA Jun-tao FU Shao-yong LIU Ji-ren

(*Software Center Northeastern University Shenyang* 110006)

Abstract Multi-Agent systems (MAS) generally exhibit complicated dynamic structure and behavior characteristics. If traditional architecture description languages (ADLs) are applied, semantic gap and expression problems are difficult to overcome. An agent-based architecture description language A-ADL is proposed in this paper, which takes computation/connector agent as architectural elements instead of component/connector in traditional ADL. By introducing architectural primitives, rule-based agent description and multi-view mechanism, it can be effectively used to model complicated dynamic architectures and overcome semantic gap mentioned above. This paper also describes dynamic architecture modeling method of A-ADL in detail with an example, and justifies the suitability of A-ADL for MAS by comparing A-ADL with several other traditional ADLs.

Key words Agent, MAS (multi-agent systems), software architecture, ADL (architecture description language), architectural style.