软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2011,22(4):609−624 [doi: 10.3724/SP.J.1001.2011.03762]
©中国科学院软件研究所版权所有.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

自适应多 Agent 系统的运行机制和策略描述语言 SADL*

董孟高[1+], 毛新军[1], 常志明[1], 王 戟[2], 齐治昌[1]

[1](国防科学技术大学 计算机学院,湖南 长沙 410073)
[2](国防科学技术大学 计算机学院 并行与分布处理国家重点实验室,湖南 长沙 410073)

Running Mechanism and Strategy Description Language SADL for Self-Adaptive MAS

DONG Meng-Gao[1+], MAO Xin-Jun[1], CHANG Zhi-Ming[1], WANG Ji[2], QI Zhi-Chang[1]

[1](School of Computer, National University of Defense Technology, Changsha 410073, China)
[2](National Key Laboratory for Parallel and Distributed Processing, School of Computer, National University of Defense Technology, Changsha 410073, China)
+ Corresponding author: E-mail: mgdong@nudt.edu.cn, http://www.nudt.edu.cn

Dong MG, Mao XJ, Chang ZM, Wang J, Qi ZC. Running mechanism and strategy description language SADL for self-adaptive MAS. *Journal of Software*, 2011,22(4):609−624. http://www.jos.org.cn/1000-9825/3762.htm

Abstract: Self-Adaptive systems have many complex properties such as openness in a situated environment, sensibility to changes, and dynamic of a system. Finding ways to develop such complex systems is still a problem in the literature of software engineering. In this paper, the autonomous entities in self-adaptive systems are abstracted as software agents, and the dynamic binding mechanism for self-adaptation is proposed based on the organization metaphors. A self-adaptive strategy description language called SADL (self-adaptive strategy description language) is presented to describe adaptive strategies that express how agents adapt to the changes. The complier and operating environment for SADL have been developed. This approach enables developers to separate the adaptation logic from functional logic of self-adaptive systems and explicitly describe self-adaptation in order to simplify the development and maintenance of complex self-adaptive systems. This case is studied in detail to illustrate the effectiveness of this proposed approach.
Key words: self-adaptive agent; dynamic binding mechanism; strategy description language; environment; MAS (multi-agent system)

摘 要: 自适应系统具有环境开放性、变化敏感性、系统动态性等复杂性特点,如何支持这类复杂系统的开发和维护是目前软件工程关注的焦点.将自适应系统中的自主运行单元抽象为软件 Agent,借助组织学思想提出了支持自适应系统运行的动态绑定机制,设计了表述 Agent 如何适应环境变化的自适应策略描述语言 SADL(self-adaptive strategy description language),开发了 SADL 的编译器和运行支撑环境.该方法将复杂自适应系统的自适应逻辑和业务逻辑相分离,通过 SADL 语言显式地描述系统的自适应特征,从而简化了复杂自适应系统的开发和维护.通过案例分析,阐述了如何基于上述方法来进行复杂自适应系统的开发,验证了方法的有效性.

* 基金项目: 国家自然科学基金(60773018, 90818028); 国家高技术研究发展计划(863)(2007AA01Z135); 国家重点基础研究发展计划(973)(2005CB321802)
收稿时间: 2009-04-28; 修改时间: 2009-07-21; 定稿时间: 2009-10-10

© 中国科学院软件研究所 http://www.jos.org.cn

关键词: 自适应 Agent;动态绑定机制;策略描述语言;环境;多 Agent 系统
中图法分类号: TP311 **文献标识码**: A

近年来,随着计算机网络,尤其是 Internet 的日益普及和广泛应用,越来越多的软件系统运行和部署在 Internet 之上.Internet 环境的动态、开放和不可预测性导致部署于其上的应用系统需具有更高的自适应能力,能够感知环境的改变,并通过调整自身的结构和行为来适应环境的动态变化,这样的系统被称为自适应系统.目前,自适应系统已在很多领域得到应用,例如长时间运行的关键业务系统、卫星定位导航系统、航空导航系统、金融系统、国家通信基础设施、军事信息系统等[1].如何有效地支持自适应系统的开发、运行和维护已成为软件工程研究关注的焦点,具体表现在:(1) 在自适应机制方面,如何提供有效的机制来指导对系统的自适应性进行描述、分析和实现?(2) 在语言设施方面,如何提供有效的语言来对自适应性进行描述、分析和程序设计?(3) 在开发和运行的支撑环境方面,如何为这类复杂系统的开发提供工具和平台的支持?

现有的关于自适应系统的研究绝大多数基于控制学理论,利用反射机制和技术[2],将自适应系统运行视为一个“运行-收集-反馈-决策-调整-再运行”的循环过程,并采用以面向对象技术为基础的软件体系结构技术[3]和中间件技术[4]来加以构造和实现.然而,对象技术具有以下局限性,如无法主动感知环境变化、对象的行为和结构在其生命周期中不可调整、不具有自主性,因而对象技术难以对自适应系统及其特征进行有效抽象、自然建模和分析.此外,现有自适应系统的构造通常将系统的自适应逻辑和业务逻辑缠绕在一起,这样做虽然可以在描述业务逻辑的同时直接定义自适应逻辑,但是由于两个不同关注点的纠缠,导致所开发的系统极为复杂,系统的修改和维护容易引入新的错误.如果系统运行后功能需求或者自适应需求发生改变,系统必须停止运行,对相应位置的代码进行修改,然后重新编译启动运行,这就带来了系统模块难以重用、系统难以在线升级等问题.

针对上述问题,本文旨在研究复杂自适应系统的自然抽象以及相应的核心机制和语言设施,以支持自适应系统的构造和运行.本文将自适应系统中的软件实体抽象、封装和物化为软件 Agent,借助于组织学思想,提出了基于动态绑定[5,6]的自适应机制,并提出了一个基于动态绑定机制的自适应策略描述语言 SADL(self-adaptive strategy description language),以对自适应逻辑进行单独和显式的描述,实现将自适应系统的自适应逻辑和业务逻辑相分离.本文设计并实现了支持 SADL 的支撑环境 SADE(self-adaptive agent development environment),并进行了案例研究.

本文第 1 节提出自适应系统的核心机制和技术框架.第 2 节详述 SADL 语言的语法、SADL 程序的编译和执行过程以及支撑环境.第 3 节通过案例分析,介绍自适应系统的开发过程,说明方法的有效性.第 4 节对相关工作进行对比分析.最后总结全文并展望进一步的研究.

1 自适应系统的构造方法

1.1 自适应机制

自主性是自适应的基础,即具有自适应特征的系统和实体应能感知环境的变化,并具有控制自身结构和行为的能力.Agent 是驻留在某一环境下能够自主、灵活地执行动作以满足设计目标的行为实体,自主性是 Agent 的基本特性.因此,本文将自适应系统中的实体抽象为 Agent,将自适应系统视为由多个 Agent 交互作用所构成的多 Agent 系统(multi-agent system,简称 MAS).

我们注意到,许多社会组织具有自适应的特征,组织中的个体扮演组织中特定的角色,并能随着环境的变化不断调整其所扮演的角色,从而展示他对组织环境的适应性.基于组织学思想,我们将自适应多 Agent 系统看成一个组织,将组织中能够适应环境变化的 Agent 称为自适应 Agent(self-adaptive agent,简称 SA).SA 驻留在环境中能够感知环境变化,并根据自身状态和自适应策略实施一系列自适应行为,从而不断地调整自身的结构和行为,以适应环境改变.图 1 描述了对自适应多 Agent 系统进行分析、描述和构造的组织元模型.

- 组织是指在特定上下文中一组具有共同目标、相互交互作用的 Agent 集合;
- 角色是对 Agent 所处的环境、拥有的属性和行为以及对外提供的服务的抽象表示,反映了 Agent 在组

© 中国科学院软件研究所 http://www.jos.org.cn

织中的地位.角色之间具有一定的组织关系,如对等关系、上下级关系、控制关系等,组织中的个体扮演了不同角色,即具有了这些角色之间的关系;

- 组织结构是从结构视点对组织进行的抽象表示,它描述了组织中的角色以及它们之间的组织关系;
- 角色类是对角色的封装和实现.

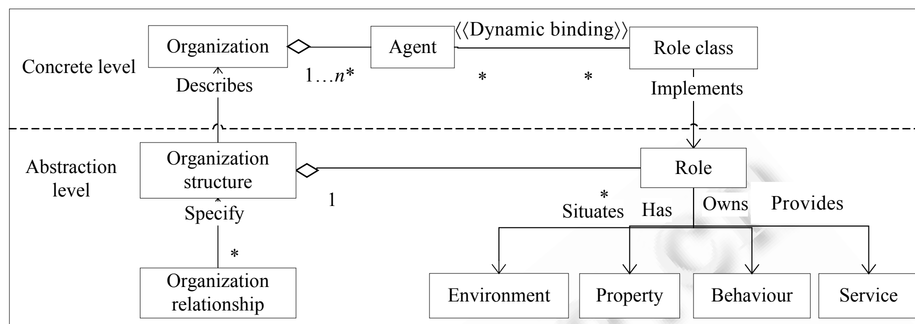

Fig.1 Meta-Modal of the system based organization abstraction

图 1 基于组织抽象的自适应系统元模型

组织中的 SA 可以通过执行一系列的自适应操作(包括加入、退出、激活和钝化)以绑定不同的角色类,从而适应环境变化.SA 通过加入或退出角色类,从而获得或失去角色类所定义的结构和行为特征.当 SA 加入一个角色类时,我们称 SA 绑定了该角色类,从而拥有该角色类的属性、行为、环境和服务.由于不同角色类间存在各种形式的组织关系,所以一旦 SA 改变了它所绑定的角色类,也就改变了它在组织中与其他 Agent 之间的组织关系,进而改变了它在组织中的地位.SA 通过执行激活和钝化动作可以将其绑定角色类的状态在活跃和非活跃之间转换.只有当绑定的角色类处于活跃状态时,SA 才能执行角色类中的行为;否则,角色类中定义的行为不能对 SA 产生影响,但此时 SA 仍能访问已绑定的角色类中的信息.我们将这一自适应机制称为动态绑定机制,并将其视为构建 SA 的基本机制.

1.2 自适应Agent构造的技术框架

自适应 Agent 构造的技术框架如图 2 所示,包含 3 个部分:角色类模块、SA 模块以及自适应策略模块.其中:角色类模块包含一组角色类,是自适应 MAS 规约和实现的基本模块单元;SA 是自适应 MAS 的执行单元,它除了具有自主性以外,还可以根据自身的策略绑定不同的角色类来动态调整自身的结构和行为规约,从而适应环境的改变;自适应策略封装了 SA 在其生命周期内可以采取的一系列自适应行为规则,每条规则说明了 SA 在何种情况下执行何种自适应动作.SA 在运行过程中,通过感知外部环境的变化驱动策略的执行.基于该框架实现的自适应 MAS 具有软件实体主体化、外部环境显式化、自适应逻辑和业务逻辑分离化等特征.系统部署和运行后,可以通过在线更改策略实现对系统的在线升级和维护.

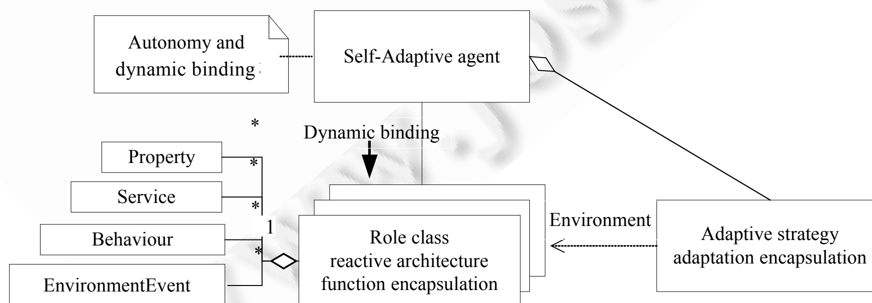

Fig.2 Technical framework to construct self-adaptive agent

图 2 自适应 Agent 构造的技术框架

1.3 环境的显式化

对自适应系统的环境进行显式建模和分析是开发自适应系统的一项重要工作[7].自适应 Agent 的环境包括系统上下文(如网络带宽、CPU 占用率、空间位置等)以及能够对该 Agent 产生影响的其他 Agent.为了提供统一的视点来描述和分析环境,简化自适应系统开发,我们将系统上下文封装为相应的 Agent(这类 Agent 被称为环境 Agent).因此,自适应 Agent 的环境是指其生存所必须依赖的、与之发生交互作用的所有 Agent 集合.图 3 展示了自适应 Agent 的环境模型,其中,环境 Agent 对不同的系统上下文进行封装,环境 Agent 中定义了需要感知的系统上下文的相关属性,如名为 SystemResource 的环境 Agent 包含 CPU 负载(CPU_LOAD)、存储容量(MEMORY_SIZE)等属性.环境 Agent 通过调用软/硬件传感器获得系统上下文相关的属性值.

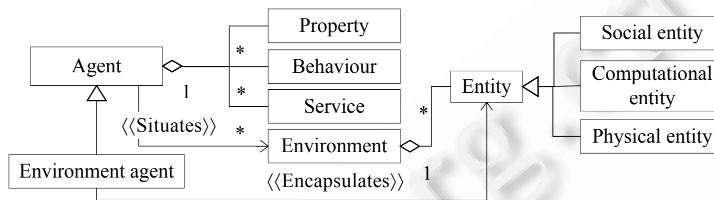

Fig.3 Environment model of self-adaptive agent

图 3 自适应 Agent 的环境模型

自适应 Agent 环境的变化包括环境中 Agent 的内部状态变化、行为实施、对外提供服务变化以及生命周期变化.自适应 Agent 通过感知环境事件的发生来感知环境的变化.在我们提出的方法中,一个环境事件的描述由 3 部分组成:事件类型(type)、事件内容(content)和对事件内容的约束(constraint).根据 Agent 感知的环境变化范围的不同,我们将环境事件分为 3 类,分别是 SERVICE,AGENT 和 ROLE,每种事件类型及其含义见表 1. Content 表示发生变化的环境对象,Constraint 用来对事件表示的环境变化范围加以限定.比如对于一个 AGENT 类型的环境事件,如果 Constraint 为 AgentBehaviour,则表示该环境事件对应于环境中某个 Agent 的行为执行事件.因此,环境事件对应于一个三元偶 $EnvironmentEvent=\langle Type,Content,Constraint\rangle$.

Table 1 Type and meaning of environment event

表 1 环境事件的类型及含义

Type	Meaning	Example
SERVICE	Changes of service provided by agents	⟨*SERVICE*,“*Sell Books*”,“*AI*”⟩ means the environment event which is the service registering one for selling books related to AI
AGENT	Changes of states, behaviors and lifecycle state of a particular agent	⟨*AGENT*,“*Tom*”,“*AgentState*”⟩ means the environment event about that the agent who is called “Tom” has changed its states
ROLE	Changes of states, behaviors and lifecycle state of the agents that bound to the role	⟨*ROLE*,“*Buyer*”,“*RoleState*”⟩ means the environment event about that the agents who have bound to “Buyer” have changed their states

为了使得自适应 Agent 能够感知环境事件、环境中 Agent 能够向外发布环境事件,需要提供环境事件的注册器和发布器.为此,我们实现了对应于不同环境事件类型的订阅器和发布器.SA 通过环境事件订阅器注册或者订阅要感知的环境内容,通过环境事件发布器向外发布自身的变化,从而影响环境.开发人员将环境事件定义在角色类中,SA 绑定了角色类就会获得其中定义的环境事件,从而自动加载对应的环境事件订阅器来感知环境事件的发生,进而感知环境的变化.

2 自适应策略描述语言 SADL 及支撑环境 SADE

自适应系统是一类复杂的系统,为了简化这类系统的开发和维护,我们认为需要将自适应系统的自适应逻辑与业务逻辑相分离,从而提高软件系统的模块化程度,增强系统模块的可重用性.为了简化自适应逻辑的表述,我们提出基于动态绑定机制的自适应策略描述语言 SADL,以清晰地描述和定义 SA 如何根据环境的变化动

© 中国科学院软件研究所 http://www.jos.org.cn

态地调整自身的结构和行为.开发和运行支撑平台 SADE 提供了对 SADL 程序进行编译和执行的支持.

2.1 自适应策略描述语言SADL

SADL 程序由一组声明语句和自适应策略的定义组成,图 4 展示了 SADL 中声明语句和自适应策略的部分语法.声明语句中包的声明用于指定自适应程序编译后所处的包;角色类的声明用于指定策略中使用的角色所属的类;自适应 Agent 的声明用于指定该策略是被哪个自适应 Agent 所使用,SADL 规定一个自适应策略只能被一个自适应 Agent 使用,但是一个自适应 Agent 可以使用多个自适应策略;程序声明是用来定义自适应程序的名字.值得注意的是,使用该策略的自适应 Agent 需要绑定的角色类一定要在 SADL 程序的“import”语句中声明,只要进行了正确声明,自适应 Agent 就可以绑定 SADL 程序的“import”语句可导出的任何一个角色类.

```
Program::=
        ('package' PackageName ';')?
        ('import' ((PackageName '.' '*') | ((PackageName '.')? RoleClassName)) ';')*
        'use' ((PackageName '.')? AgentClassName ';')
        'program' ProgramName '{' (InitStrategy)? (AdaptiveStrategy)+'}' ⟨EOF⟩
InitStrategy::='InitStrategy' '{' (IfStatement|AdaptationStatement)* '}'
AdaptiveStrategy::='strategy' StrategyName '{' (AdaptiveRule)+'}'
AdaptiveRule::='when' '(' EventExpression ')' '{' (IfStatement|AdaptationStatement)* '}'
EventExpression::=InternalEvent|ExternalEvent | '(' EventExpression ')'
        | EventExpression '&&' EventExpression | EventExpression '||' EventExpression
IfStatement::='if' '(' StateExpression ')' '{' (IfStatement | AdaptationStatement)? '}'
        ('else if' '(' StateExpression ')' '{' (IfStatement | AdaptationStatement)? '}')*
        ('else' '{' (IfStatement|AdaptationStatement)? '}')?
AdaptationStatement::=(AdaptiveAction '(' RoleIdentifier (','Expression)* ')' ';')+
```

Fig.4 Part of the SADL syntax

图 4 SADL 的部分语法

自适应策略定义了 SA 的自适应逻辑,包括初始策略和一般策略.其中,初始策略定义了 SA 在加载策略时就需要执行的行为.一般策略的定义包括策略的声明和自适应规则的定义,其中,策略的声明用来定义策略的名字,自适应规则对 SA 在不同的场景下执行的自适应行为进行定义,一般形式如:“**when** (ChangeExp) [**if** (StateExp)] {(AdaptiveAction())*}”.其中,ChangeExp 是对 SA 所处环境变化的描述,StateExp 描述了 SA 的状态,AdaptiveAction 定义了在满足这两个条件时 SA 执行的自适应动作序列.

环境事件显式定义在角色类中,SA 对环境变化的感知通过接收环境事件来实现.在 SADL 中,对 SA 所处环境变化的判断是通过对事件表达式的判断来完成的,事件表达式由单个事件或由连接子(与(“&&”)、或(“||”))连接起来的多个事件表示.SADL 中的事件包括内部事件和外部事件,内部事件描述 Agent 自身发生的状态改变或行为执行事件,外部事件描述环境中其他 Agent 发生的变化.事件表达式的部分语法如图 5 所示.

```
EventExpression::=InternalEvent | ExternalEvent | '(' EventExpression ')'
        | EventExpression '&&' EventExpression | EventExpression '||' EventExpression
InternalEvent::='INTERNAL_EVENT' '(' InternalEventName ',' (RoleName | 'self') (',' ValueExpression)* ')'
ExternalEvent::=LifeCycleEvent | AdaptationEvent | StateChangeEvent | BehaviourEvent | ServiceEvent
LifeCycleEvent::='LIFECYCLE_EVENT' '(' AgentName ',' RoleName ',' LifeCycleName ')'
LifeCycleName::='BORN' | 'DEAD' | 'MOVED' | 'RESUMED' | 'SUSPENDED' | 'FROZEN' | 'THAWED'
AdaptationEvent::='ADAPTATION_EVENT' '(' AgentName ',' AgentAdaptationName ',' RoleName')'
AgentAdaptationName::='JOIN' | 'QUIT' | 'ACTIVATE' | 'DEACTIVATE'
StateChangeEvent::='STATECHANGE_EVENT' '(' AgentName ',' RoleName ',' PropertyName (',' ValueExpression)? ')'
BehaviourEvent::='BEHAVIOUR_EVENT' '(' AgentName ',' RoleName ',' BehaviourName (',' ValueExpression)? ')'
ServiceEvent:: ='SERVICE_EVENT' '(' ServiceName (',' ServiceContent (',' ServiceProvider)* )? ')'
```

Fig.5 Part of the syntax of the event expressions in SADL

图 5 SADL 事件表达式的部分语法

© 中国科学院软件研究所 http://www.jos.org.cn

SA 的状态包括私有状态、自演化状态和自适应状态.私有状态是指 SA 自身具有的属性对应的状态,如 SA 的名字.在 SADL 中,对私有状态的判断是通过对 SA 私有属性值的判断来完成的.自适应状态和自演化状态是自适应 Agent 特有的两种状态.自演化状态是指 SA 在演化过程中绑定角色类的状态,即 SA 是否绑定了某个角色类,所绑定的角色类是否处于活跃状态等.在 SADL 中,对自演化状态的判断是通过角色名加“@”符号,再加状态来表示的(如 Buyer@BOUND 表示角色类 Buyer 已被绑定).

自适应状态是指在 SA 运行过程中自适应属性值的状态.自适应属性是 SA 特有的一种属性,不同于私有属性,自适应属性是在 SA 绑定角色类后才具有的属性.这种属性在角色类中声明,在 SA 绑定角色类后可以对其访问和修改.SA 的私有属性在 SA 生命周期中一直存在,而自适应属性会随着 SA 加入某个角色类而产生,退出某个角色类而失去.比如,在网上交易系统中,Agent 会根据需要加入买者(buyer)角色类,从而具有了待购商品名、待购商品数量等自适应属性,当它购买完成退出买者角色类后,将不再拥有这些属性.

SA 在运行过程中因为加入角色类而具有了自适应属性,从而同时具有私有属性和自适应属性,这就有可能产生属性不一致问题.所谓属性不一致是指 SA 的某个自适应属性 a 与某个私有属性 a'或者其他的自适应属性 a''指向同一个对象(a,a'和 a''可同名也可不同名),在 SA 运行过程中,如果这几个指向同一个对象的属性中的一个发生了变化,而其他的没有进行相应的改变,则产生了属性不一致现象.例如,在网上交易系统中,一个名叫 Tom 的 SA 具有私有属性 money,买者角色类也具有 money 属性,这两个属性都是指购买商品的资金这个对象.在 Tom 绑定了买者角色类后,同时具有了 money 这个私有属性和自适应属性.Tom 执行了买者角色类中的购买行为后,自适应属性 money 的值减少.此时,如果不改变对应的私有属性 money 的值,就会造成属性不一致.

为了解决属性不一致问题,我们在 SADE 中增加了维护属性一致性的语言设施,为开发人员提供了维护属性一致性的操作接口.该接口通过反射机制实现,开发人员使用该接口只需指明要维护一致性的属性名,其他关于属性所属的角色类或者 SA 的判断、属性类型的获取以及属性的赋值都由底层语言设施实现,对开发人员透明.

SA 的属性(包括私有属性和自适应属性)既有简单类型也有复杂类型,简单类型包括整型、字符串型、长整型、布尔型等,复杂类型包括容器类型(如 List 类型、Set 类型、Stack 类型等)和数组类型.在 SADL 中,对 SA 简单类型私有属性的访问用“**self.**”加属性名完成.对复杂类型私有属性的访问不能采用与简单类型属性访问相同的方法,因为复杂类型属性包括容器类属性,该属性本身就是一个对象,属性有其包含的元素.为了支持对复杂类型属性的访问,SADL 提供了专门用于判断容器类属性长度、内容以及属性是否为空的语法,如图 6 所示.例如,访问容器类私有属性的内容可以通过“self.property.include(obj1,obj2)”来实现.其中,self 表示使用该策略的 SA 实例的标识符,property 是指容器类属性的名字(使用时用具体的属性名字),include 是 SADL 提供的访问属性内容是否包含 obj1 和 obj2 对象值的方法.SADL 语法也支持直接访问数组类属性.对自适应属性的访问与私有属性的访问方式不同之处在于,私有属性的访问是通过“**self.**”加属性名或操作来完成的,而自适应属性的访问是通过“**roleName.**”加属性名或操作来完成的.roleName 是指角色类名字,且值不能为“self”.

PrivateProperty::=‘**self**’ ‘**.**’ SimpleProperty | ‘**self**’ ‘**.**’ ComplexProperty
SelfAdaptationProperty::=RoleIdentifier ‘**.**’ SimpleProperty | RoleIdentifier ‘**.**’ ComplexProperty
SimpleProperty::=Identifier
ComplexProperty::=Identifier ‘**.**’ ‘**length**’ | Identifier ‘**.**’ ‘**include**’ ‘(‘ Object (‘,’ Object)* ‘)’
  | Identifier ‘**.**’ ‘**exist**’ ‘(’ Object ‘,’ DecimalIntegerLiteral ‘)’
  | Identifier ‘[’ DecimalIntegerLiteral ‘]’ (‘[’ DecimalIntegerLiteral ‘]’)*
SelfEvolutionState::=RoleIdentifier ‘@’ AdaptiveRoleState
AdaptiveRoleState::=‘**ACTIVELY-BOUND**’ | ‘**INACTIVELY-BOUND**’ | ‘**UNBOUND**’ | ‘**BOUND**’

Fig.6 Part of the SADL syntax for accessing the state of self-adaptive agent
图 6 SADL 中对自适应 Agent 状态访问的部分语法

每个 SA 可实施 4 种自适应行为 join,quit,activate 和 deactivate.通过执行不同的自适应行为,SA 可以改变自身的结构和行为,从而适应环境的改变.

© 中国科学院软件研究所 http://www.jos.org.cn

2.2 SADL程序的编译和执行

SADL 程序经过编译后将生成相应的 Java 语言程序.SADL 程序中定义的不同事件编译后生成不同的事件对象,对象的实例化参数从 SADL 程序的事件参数中解析获得,编译前的事件描述和编译后的事件对象的对应关系见表 2.在 SADL 定义的事件中,如果事件的某个参数为空,表示在事件匹配时不需要关心该参数;如果事件的参数中含有通配符("*"),表示在事件匹配时通配符的位置可以是任意的值;如果事件参数中含有比较关系符(如">"等),表示在事件匹配时对应于该参数的值应该符合关系运算符规定的关系.对于编译后的各事件及其参数的解析,由开发包提供的各个事件类完成.

Table 2 Event taxonomy and the event class name after the strategy is complied

表 2 事件分类及事件编译后对应的事件类名

Event taxonomy	Event type	Event identifier in SADL	Event class name after the strategy is complied
External event	AGENT	ADAPTATION_EVENT	AdaptationMessage
	AGENT/ROLE	LIFECYCLE_EVENT	AgentLifeCycelMessage
		STATECHANGE_EVENT	AgentStateMessage RoleStateMessage
		BEHAVIOUR_EVENT	AgentBehaviourMessage RoleBehaviourMessage
	SERVICE	SERVICE_EVENT	ServiceMessage
Internal event		INTERNAL_EVENT	InternalMessage

SADL 程序中定义事件的目的就是为了判断 SA 的环境有没有发生事件描述的变化,为此,事件编译后的基本形式如 **checkEnvMsg**(new XEvent(…),**msg**),意思是判断当前 SA 实例感知到的消息(msg)是否符合 XEvent 的描述.如果符合,则说明当前环境中发生了 SADL 程序描述的事件,即环境中发生了相应的变化.

我们以生命周期改变事件编译前后的比较为例对事件进行解释,下面是该事件编译前后的对比.

LIFECYCLE_EVENT(*AgentName*,*LifecycleName*)→

checkEnvMsg(*new AgentLifeCycleMessage*("*AgentName*","*LifecycleName*"),*msg*)

LIFECYCLE_EVENT 事件的语法规定 AgentName 和 LifecycleName 可以是空、字符串或者带有通配符的字符串,下面对包含不同参数事件的含义进行解释:

(1) AgentName 为某个字符串或者带有通配符的字符串,LifecycleName 为空.此生命周期改变事件是指某个名字符合 AgentName 描述的 Agent 发生了生命周期改变;

(2) AgentName 为空,LifecycleName 为某个字符串或者带有通配符的字符串.此生命周期改变事件是指环境中某个 Agent 发生了名为 LifecycleName 的生命周期改变事件;

(3) AgentName 和 LifecycleName 都为某个字符串或者带有通配符的字符串.这种情况下的生命周期改变事件是指某个名字为 AgentName 的 Agent 发生了名为 LifecycleName 的生命周期改变事件.

SADL 程序中定义的自适应动作编译后生成 SA 实例调用的自适应动作的形式,即将 SADL 中的 AdaptiveAction(roleName)编译成为 agent.AdaptiveAction("roleName").其中,Agent 是指使用该策略的 SA 实例,AdaptiveAction 是第 2.1 节中所述的 4 种自适应行为,roleName 是要被实施自适应行为的角色类名.Agent 加入某个角色类,即对该角色类进行实例化,并将其置于活跃状态.SA 实例通过调用角色类的 getBehaviours 方法得到该角色类的行为规约,并将其放到 SA 实例的行为队列中以供调度;通过调用角色类的 getEnvEvents 方法得到角色类中声明的环境事件,根据事件类型加载对应的环境事件订阅器,从而感知环境的变化.Agent 退出某个角色类,即从 Agent 的行为队列中删除对应于该角色类的行为,停止感知该角色类对应的环境变化,将该角色类从 Agent 绑定的角色列表中删除.Agent 激活某个角色类,即将角色类从非活跃状态置为活跃状态,当角色类处于不同的状态时,对 Agent 的作用如第 1.1 节所述.Agent 钝化某个角色类,即将角色类从活跃状态置为非活跃状态.

为了将 SADL 程序编译成相应的 Java 程序,我们设计和实现了 SADL 编译器.该编译器借助 JavaCC(Java compiler compiler)[8]构造,JavaCC 是一个非常灵活且稳定的语法解析器的生成器.基于 JavaCC 的 SADL 编译器的实现过程如图 7 所示,首先通过 jjTree 命令对 SADL 的语法文件(SADL.jjt)进行预处理,得到可以生成语法树

© 中国科学院软件研究所 http://www.jos.org.cn

的 SADL.jj 文件;然后再用 JavaCC 的命令对 SADL.jj 文件进行处理,得到包括词法分析器、语法分析器以及语法树等多个 Java 文件;最后使用 Java 编译器对所有的 Java 文件进行编译,生成 SADL 编译器.

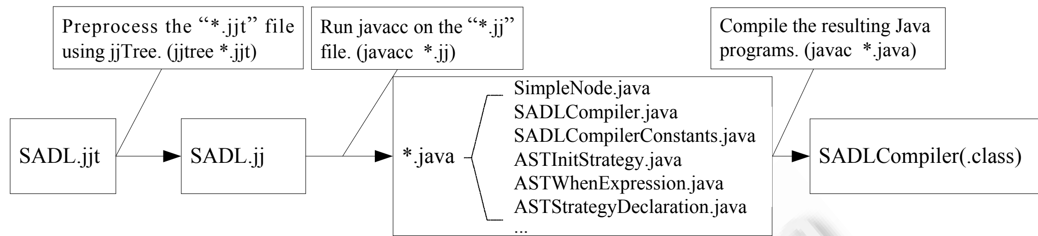

Fig.7 Process of implementing SADL compiler
图 7 SADL 编译器的实现过程

SADL 程序的编译和执行过程如图 8 所示.使用 SADL 编译器将 SADL 程序编译成 Java 代码(即策略类),然后与 SA 和 Role 一同经过 Java 编译器编译生成目标代码.在开发包的支持下,运行平台负责加载和运行应用系统.

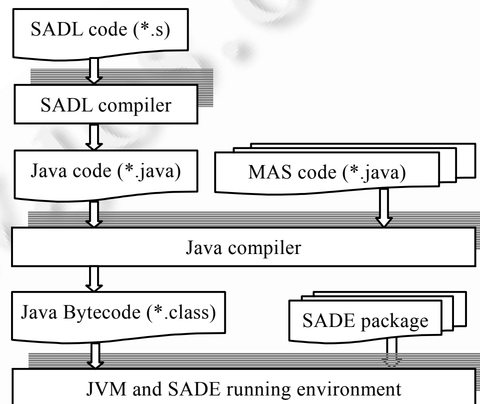

Fig.8 Process of compiling and executing SADL program
图 8 SADL 程序的编译和执行

2.3 自适应系统的开发和运行支撑环境SADE

为了支持利用动态绑定机制和 SADL 语言分析、设计和实现自适应系统,在 JADE[9]的基础上,我们设计并实现了 SADE 原型系统.SADE 的总体框架如图 9 所示,分为 3 个层次:基础层、运行层和开发层.基础层借助 JADE 的基础设施实现了不同节点、不同平台 Agent 之间的通信,为分布式 MAS 的交互提供了支持.运行层提供了支持 SA 加载和运行等功能部件.其中,事件服务是 Agent 感知环境变化的基础,动态绑定机制为 Agent 提供了基本的自适应能力,策略的编译器和驱动引擎为自适应策略描述文件的编译以及编译后的策略类的加载提供支持.为了实现在线更改策略,我们通过对 Java 的类加载器的更改实现了策略类加载器,从而支持策略的动态适配.冲突检查器和软件运行监控器保证了系统运行的正确性.开发层为自适应软件的开发提供支持.其中,SADL 提供了定义自适应策略的语言规范,自适应 Agent 体系结构和软件开发包为 SA,Role 以及自适应策略的开发提供了基本的编程规范和基类支持.

© 中国科学院软件研究所 http://www.jos.org.cn

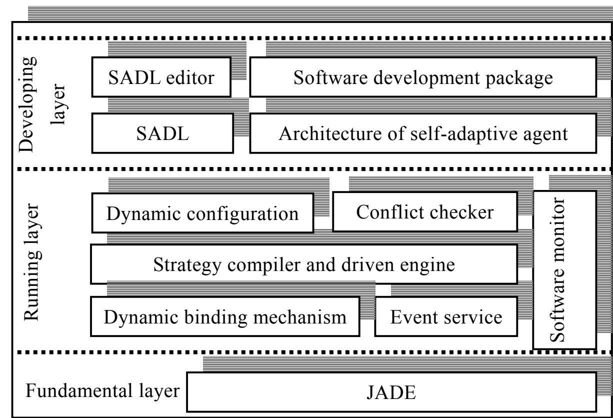

Fig.9 Supported environment SADE for developing and running self-adaptive system

图 9 自适应系统的开发和运行支撑环境 SADE

3 案例分析

为了展示SADL语言以及SADE平台对自适应系统开发和运行的支持,我们开发了一个简单的网上交易系统TradingSystem,该系统用来模拟用户在网上进行交易的行为.系统中有4类角色:浏览者(visitor)、买者(buyer)、卖者(seller)和管理者(manager).浏览者可以浏览和查看系统中的商品信息,但是他不能进行商品交易;买者拥有资金,他不仅可以浏览和查询商品信息,还可以与卖者进行交互,购买自己所需的商品;卖者拥有商品,他既可以查询商品,也可以与买者交互,出售自己的商品;管理者对加入系统的交易者进行管理,并负责界面显示和信息更新.系统处于开放、动态的环境下,用户可以自由地加入或退出系统,系统中的服务也会动态变化.下面两个典型场景(1)、场景(2)用来展示处于动态、开放环境下的 Agent 的自适应行为,场景(3)展示出为了实现 Agent 的目标进行的策略动态调整.

(1) 用户绑定的角色随着环境的改变而变化:Tom 进入系统后,扮演 Visitor 角色浏览商品.当他感知到系统中出售他所需要的商品的服务出现时,就扮演 Buyer 角色,准备购买商品.

(2) 用户暂时离开某个角色,然后又回到该角色:在 Tom 扮演 Buyer 角色准备购买需要的商品时,发现商品的售价大于它拥有的资金,如果此时Tom有要出售的商品,则扮演Seller角色,通过销售商品获得更多的资金,而后再次扮演 Buyer 角色来购买商品.

(3) 用户策略的动态调整:在Tom扮演Buyer角色准备购买需要的商品时,发现商品的售价大于它拥有的资金,此时 Tom 并没有需要出售的商品,为了购买到需要的商品,Tom 要卸载通过销售商品获得更多资金的策略(TradeStrategy_1 策略),加载通过转换角色重新等待低价销售服务的策略(TradeStrategy_2 策略),在更低价格销售服务出现后再次扮演 Buyer 角色来购买商品.

自适应系统设计和实现的步骤如下:

首先,识别和抽象出构成系统的各个角色,并通过继承角色抽象类对角色加以实现.针对该案例,我们设计了 4 个角色类,如图 10 所示.Visitor 角色类中定义了 SERVICE 类型的环境事件,加入该角色的自适应 Agent 可以感知提供书籍销售的服务,图 11 展示了 Visitor 角色类代码.为了感知系统中扮演不同角色的自适应 Agent 的变化,Manager 角色类中定义了 3 个不同 ROLE 类型的环境事件;

其次,识别和分析系统中所需要的自适应 Agent,定义自适应 Agent 的属性,并进行初始化设置(指定自适应 Agent 要绑定的角色类的路径、策略配置文件和初始策略名).本系统中有销售商品的 SellerAgent、购买商品的 BuyerAgent 以及对系统管理的 ManagerAgent,也有能够根据需要不断调整当前扮演的角色的 PowerfulAgent.系统的 Agent 类图如图 12 所示,图 13 展示了 PowerfulAgent 的类代码;

最后,使用 SADL 定义自适应 Agent 的自适应策略.在本系统中,SellerAgent,BuyerAgent 和 SystemAgent 这

© 中国科学院软件研究所 http://www.jos.org.cn

3 类 Agent 没有加载策略,它们在初始创建时刻分别加入 Seller,Buyer 和 Manager 角色类,此后,它们扮演的角色不再发生改变.而 PowerfulAgent 具有自适应策略,需要对环境以及 Agent 自身状态的变化进行判断,并定义在不同场景下绑定不同的角色类的操作.PowerfulAgent 初始加载的策略如图 14 所示,该策略对案例描述的前两个场景进行了定义.图 15 展示了 PowerfulAgent 为了实现场景(3)而动态加载的策略.

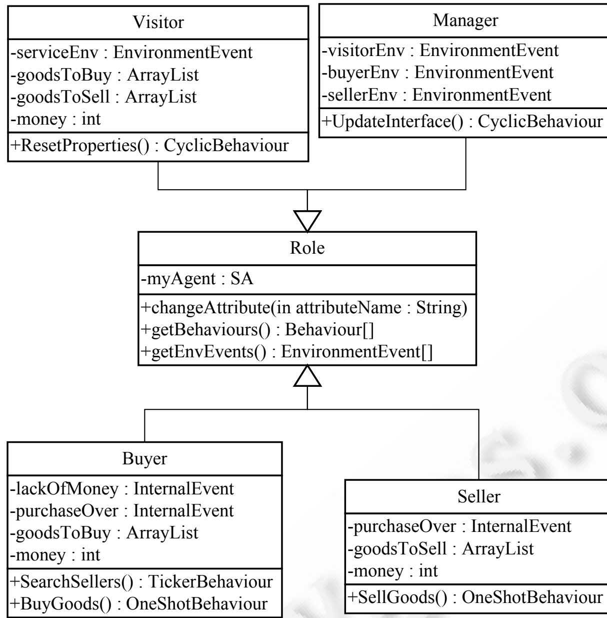

Fig.10 Role class diagram of TradingSystem

图 10 网上交易系统角色类图

```
public class Visitor extends Role {
        EnvironmentEvent serviceEvent=new
                EnvironmentEvent("Service","Sell","Book");
        private int money=0;
        private ArrayList goodsToBuy=new ArrayList();
        private ArrayList goodsToSell=new ArrayList();
        public void setGoodsToBuy(ArrayList goodsToBuy){
                this.goodsToBuy=goodsToBuy;
                changeAttribute("goodsToBuy");
        }
        ...
        public class ResetProperties
                        extends CyclicBehaviour {
                public void action() {...}
        }
        public Behaviour[] getBehaviours() {
                Behaviour[] behaviour=new Behaviour[1];
                behaviour[0]=new ResetProperties();
                return behaviour;
        }
        public EnvironmentEvent[] getEnvEvents() {
                EnvironmentEvent[] envEvent=
                        new EnvironmentEvent[1];
                envEvent[0]=serviceEvent;
                return envEvent;
        }
}
```

Fig.11 Sample of the visitor code

图 11 Visitor 角色类的代码示例

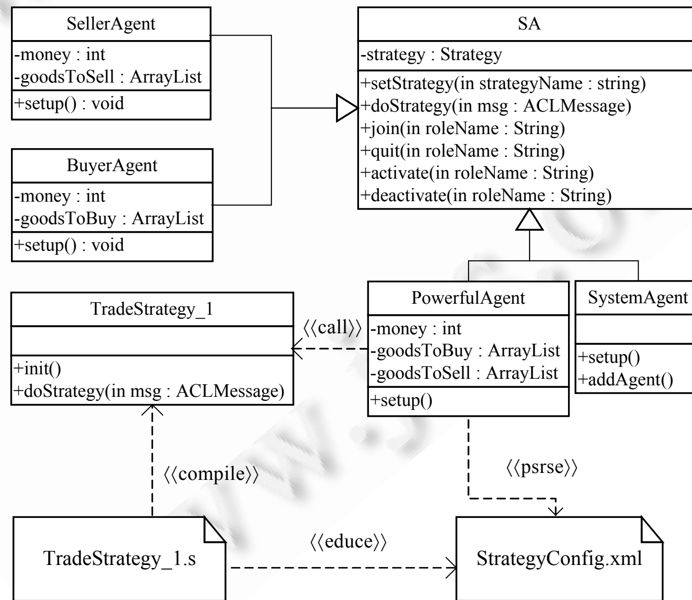

Fig.12 Agent class diagram of trading system

图 12 网上交易系统 Agent 类图

© 中国科学院软件研究所 http://www.jos.org.cn

```
public class PowerfulAgent extends Agent {
        //definition of property of agent
        private ArrayList goodsToBuy=new ArrayList();
        private ArrayList goodsToSell=new ArrayList();
        private int money=0;
        public ArrayList getGoodsToBuy() {
                return goodsToBuy;
        }
        public void setGoodsToBuy(ArrayList goodsToBuy) {
                this.goodsToBuy=goodsToBuy;
                changeAttribute("goodsToBuy");
        }
        …
        public PowerfulAgent() {
                //initialize configuration parameters
        setRolePath("examples.tradingSystem.roles");
                setStrategyProfile("examples"+
                        "\\tradingSystem\\StrategyProfile.xml");
        }
        protected void setup() {
                //set adaptive strategy that agent loads
                setStrategy(this,"TradeStrategy_1");
        }
}
```

Fig.13 Sample of the PowerfulAgent code

图 13 PowerfulAgent 代码示例

```
package examples.tradingSystem.strategies;
import examples.tradingSystem.roles.*;
//PowerfulAgent uses the strategy
use examples.tradingSystem.agents.PowerfulAgent;
program TradeStrategy_1 {
        //Agent joins Visitor initially
        InitStrategy { join(Visitor); }
        strategy Wait {//Agent waits for service occurrence
                when (SERVICE_EVENT(Sell, Book)) {
                        deactivate(Visitor); join(Buyer); } }
        strategy Buy {
                //Agent binds to various role for buying goods
                when (INTERNAL_EVENT(LackOfMoney, Buyer)) {
                        if (self.goodsToSell.length>0
                                && Seller@UNBOUND) {
                                deactivate(Buyer); join(Seller);
                        }
                }
                when (INTERNAL_EVENT(PurchaseOver, Buyer)) {
                        if (self.goodsToBuy.length > 0) {
                                deactivate(Buyer); activate(Visitor);
                        } else{ quit(Buyer); quit(Visitor); } }
        }
        strategy Sell {
                //Agent changes role to prepare to buy again
                when (INTERNAL_EVENT(SaleOver, Seller)) {
                        if (Buyer@INACTIVELY-BOUND) {
                                quit(Seller); activate(Buyer); } }
        }
}   //of program TradeStrategy_1
```

Fig.14 Adaptive strategy TradeStrategy_1 of PowerfulAgent

图 14 PowerfulAgent 的自适应策略 TradeStrategy_1

© 中国科学院软件研究所 http://www.jos.org.cn

```
package examples.tradingSystem.strategies;
import examples.tradingSystem.roles.*;
//PowerfulAgent uses the strategy
use examples.tradingSystem.agents.PowerfulAgent;
program TradeStrategy_2 {
        //Agent joins Visitor initially
        InitStrategy {
                if (Visitor@UNBOUND) {join(Visitor);} }
        strategy Wait { //Agent waits for service occurrence
                when (SERVICE_EVENT(Sell, Book)) {
                        if (Visitor@ACTIVELY-BOUND) {
                                deactivate(Visitor); }
                        if (Buyer@UNBOUND) {join(Buyer); }
                        else if (Buyer@INACTIVELY-BOUND) {
                                activate(Buyer);} }
        }
        strategy Buy {
                  //Agent binds to various role for buying goods
                  when (INTERNAL_EVENT(LackOfMoney, Buyer)) {
                        if (self.goodsToSell.length==0
                                && Visitor@INACTIVELY-BOUND) {
                                deactivate(Buyer); activate(Visitor); }
                }
                when (INTERNAL_EVENT(PurchaseOver, Buyer)) {
                        if (self.goodsToBuy.length>0) {
                                deactivate(Buyer); activate(Visitor);
                        } else {quit(Buyer); quit(Visitor); }
                }
        }
}   //of program TradeStrategy_2
```

Fig.15 Adaptive strategy TradeStrategy_2 of PowerfulAgent

图 15 PowerfulAgent 的自适应策略 TradeStrategy_2

对策略 TradeStrategy_1 的解释如下:

(1) Agent 执行 Wait 策略,如果当前感知到类型为 SERVICE、内容为 Sell、约束为 Book 的环境事件,说明环境中有 Agent 提供了销售书籍的服务,此时,Agent 钝化 Visitor 角色类,加入 Buyer 角色类准备购买商品;

(2) Agent 执行 Buy 策略,将会对两个内部事件进行处理:第 1 个内部事件名为 LackOfMoney,Agent 感知到该事件,说明当前自身具有的资金不足以购买需要的商品.为了获得更多的资金,Agent 将钝化 Buyer 角色类,加入 Seller 角色类出售自己的商品;第 2 个内部事件名为 PurchaseOver,Agent 感知到该事件时,判断如果有要购买的商品,则钝化 Buyer 角色类,激活 Visitor 角色类,重新等待销售商品服务的出现;否则,退出已绑定的角色类;

(3) Agent 执行 Sell 策略,会对名为 SaleOver 的内部事件进行处理,Agent 感知到该事件时,若 Agent 绑定的 Buyer 角色类处于非活跃状态,则退出 Seller 角色类,激活 Buyer 角色类,开始继续购买商品.

系统运行界面如图 16 所示,其中,左上角的 Agents List 展示了当前系统中的所有应用相关 Agent.下面每一栏从左至右分别展示了 Agent 的名字、当前加入的角色类以及 Agent 参与交易和自适应动作执行的信息,角色图片背景为灰色说明该角色类当前处于非活跃状态.图 16 展示了 PowerfulAgent 加载策略 TradeStrategy_1 运行的场景.

© 中国科学院软件研究所 http://www.jos.org.cn

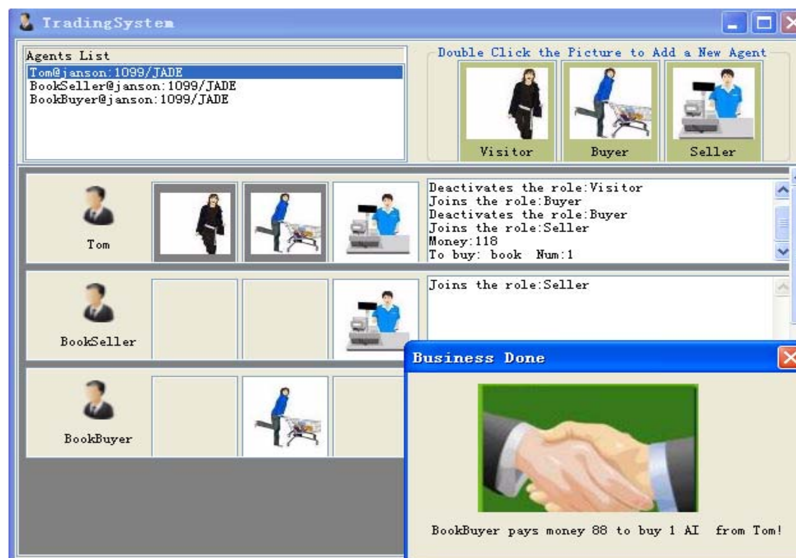

Fig.16 TradingSystem running snapshot

图 16 网上交易系统运行结果

4 相关工作对比分析

近年来,如何支持复杂自适应系统的开发成为热点问题.下面从自适应软件的基础理论、核心机制和构造技术 3 个方面对现有工作进行介绍和对比分析.

在自适应软件的基础理论方面,目前的自适应软件大多以控制论为基础,将自适应系统运行视为一个“运行-收集-反馈-决策-调整-再运行”的循环过程.Kokar 等人将控制理论作为分析和设计自控制和自适应软件的数学基础,并且提出了基于控制理论的软件模型[10].我们借助于社会组织学的抽象和思想,通过动态绑定机制来支持和实现 Agent 的自适应性,并且建立了基于动态绑定机制的自适应理论框架[5],这为在技术层面的研究和实现奠定了理论基础.

在自适应软件的核心机制方面,计算反射技术被视为构建自适应软件系统的基础[11].反射(reflection)[12]是以自述的方式表示系统的状态和行为,对自述所描述的系统状态和行为与系统真实的状态和行为之间建立因果关联(causal connected),从而使得一方的变化影响到另一方.以反射技术作为基础的自适应软件系统将系统分为元层和基层,使得系统设计较为复杂,运行效率较低,耗费内存较大.

在控制学理论和反射机制及技术的支持下,目前自适应软件的具体构造技术有:(1) 基于模型技术,如 Yoder 等人提出了自适应的对象模型 AOM(adaptive object model)[2],用户可以在以元模型描述的抽象层次上应用支撑工具定制系统,并在运行时应用反射技术将描述系统的配置信息映射为对象模型的运行时描述.然而,由于对象固有的静态特性,这种技术并不能很好地适合动态和开放环境下自适应系统的实现;(2) 基于软件体系结构技术[3],这种技术着重关注系统整体的自适应,通过对软件体系结构中部件和连结子的增加、删除、替换实现系统的自适应.基于体系结构的技术主要考虑的是从全局的视点支持软件自适应,但是对构件层次的自适应支持不足;(3) 基于中间件技术,如 Moura 等人提出了使用 CORBA Trading Service 支持对动态组件的选择,并且支持对动态变化的需求的监测[13].北京大学梅宏教授及其研究小组开发了基于反射的 J2EE 应用服务器 PKUAS[14].该服务器通过中间件来支持运行时查看并调整内部状态和行为,以实现反射体系对系统整体的表示和控制.不同于以控制学理论为基础的自适应软件研究技术,文献[15]提出了基于数字演化技术,这种技术受生物自然进化思想的启发,希望程序能够通过类似生物进化的方式(如复制自身)不断演化,从而提高软件的鲁棒性.

另一个研究方向是借助软件 Agent 技术,通过 Agent 在运行期间转换角色或类型实现系统的自适应[16,17].

© 中国科学院软件研究所 http://www.jos.org.cn

软件 Agent 技术被视为开发动态、复杂软件系统的有效泛型[18],Agent 具有环境驻留性和行为自主性,能够在没有人类或外界干扰的情况下自主完成一定的任务,满足设计目标.以 Agent 作为软件实体的系统能够更好地适应环境的改变,实现系统的自适应特征.在过去的几年中,一些研究人员借助于组织学和社会学的思想,将角色的概念引入到 MAS 的分析和设计中来,提出了相关的角色分配机制[19],并将其用于规约、分析动态和自适应的多 Agent 系统.然而,如何提供有效的机制和语言设施来支持自适应 MAS 的开发仍然面临着挑战.

在自适应 Agent 开发语言方面,3APL 是一种面向 Agent 的程序设计语言,该语言支持对角色的 4 种操作[16],如 enact,deact,activate 和 deactivate,能够支持开发具有动态调整角色能力的 Agent.但是它们对 Agent 扮演角色进行了一些限制,如同一时刻 Agent 扮演的角色中只能有一个处于活跃状态,其他角色必须处于非活跃状态,因而,这样做不能很好地解释现实世界中的很多问题(比如同时扮演多个角色).SLABSp[17]是一个面向 Agent 的程序设计语言,该语言基于 catse 机制[20].caste 是 Agent 行为的抽象,是设计和实现 MAS 的基本模块单元.Agent 可以 join(加入)一个 caste,也可以 quit(退出)一个 caste,SLABSp 实现了 join 或 quit 某个 caste 的操作.但 SLABSp 没有提供单独的自适应行为定义的语言设施,自适应行为直接定义在行为规则中,从而造成不同关注点交织在一起,不利于构建复杂的 MAS,也不利于系统的维护和升级以及行为规则的重用.

在策略语言方面,近年来,为了解决不同领域的问题,人们基于不同的技术提出了很多策略语言,如使用 DAML 作为策略表示方式的 KAoS[21]支持对 Web Services、网格计算和 MAS 的策略描述;面向对象的策略描述语言 Ponder[22]能够支持分布式系统和网络的管理.这些策略语言具有语法复杂、功能强大的特征,在各自的领域都得到了很好的应用.但是这些语言并不是针对自适应系统设计的,不能直接支持自适应系统的开发和实现.文献[23]提出了支持自适应系统开发的规则语言,这种规则类似于本文的策略.该规则语言用于基于构件的自适应系统开发,使用 XML 语言制定.但是这种语言的抽象层次较低,开发人员用它设计自适应策略较为困难.

在自适应系统的设计和实现方面,ROAD(role-oriented adaptive design)[24]是一个用来设计和开发分布式自适应应用系统的框架.ROAD 中的角色定义了业务过程,业务过程由角色的 Player(扮演者)驱动执行.在使用 ROAD 进行系统开发时,开发人员可以通过动态改变角色扮演者与角色的关系来适应系统及其环境的动态改变,从而为实现系统的自适应提供了基础.但是,ROAD 没有提供对环境的显式表示和感知,也没有提供在环境变化时软件实体自主执行一定的自适应行为的技术手段.

5 总结和进一步工作

为了适应 Internet 环境的动态、开放和不可预测性,部署于 Internet 之上的应用系统需要具有自适应能力.如何有效支持这类系统的开发、运行和维护已成为计算机软件技术面临的一项重要挑战.本文在支持自适应系统开发和运行的核心机制和语言设施方面展开研究,提出了基于动态绑定的自适应机制.该机制借鉴组织学和社会学的概念、抽象和思想,通过让 Agent 动态绑定不同的角色类来达到适应环境变化的目标.基于动态绑定的自适应机制及其思想能够直观地解释、描述和分析系统的自适应特征,可以有效地指导对面向组织的自适应系统的设计和实现.为了支持自适应逻辑的独立描述,从而将系统的自适应逻辑和业务逻辑分离,本文提出了基于动态绑定机制的自适应策略描述语言 SADL.该语言采用类高级编程语言的语法,结构类似于基于规则的语言形式,使用该语言定义自适应策略具有描述简单、系统模块化程度高、易于重用和支持在线维护等优势.

为了支持自适应 MAS 的开发和运行,我们成功开发了支持动态绑定机制和 SADL 语言的 SADE 原型系统.实现了一些具有自适应特征的应用程序,如网上交易系统、垃圾清理机器人、兴趣小组管理系统.这些开发实践和应用案例分析说明了上述开发方法和支撑环境的可行性和有效性、SADL 的简单性和易用性以及所开发的软件系统的易维护性和易演化性.

未来将在以下几个方面进一步展开研究:

- 在机制层,通过对自适应机制在微观层和宏观层的扩充,提出基于行为改变的自适应机制和支持系统整体调整的自适应机制,从而支持更多类型的自适应软件的开发;
- 在开发层,将提供对自适应策略的正确性和一致性进行检查的方法;

© 中国科学院软件研究所 http://www.jos.org.cn

- 在运行层,Agent 绑定角色类过程中可能会出现冲突问题.针对这一问题,将要研究相应的冲突检查机制,并提供运行时冲突检查器;
- 在平台层,将加强语言机制和开发平台的功能,支持多个 Agent 间的协作;
- 在应用层,希望通过对更多的、功能更为丰富的案例的分析和研究来验证我们工作的有效性;
- 在方法层,将为自适应 MAS 的开发提供系统的开发方法,从而指导分析、设计、实现、运行和维护等各个阶段的软件实践活动.

References:

[1] Laddaga R, Robertson P, Shrobe H. Introduction to self-adaptive software: Applications. In: Proc. of the IWSAS 2001. LNCS 2614, Heidelberg: Springer-Verlag, 2003. 275−283. [doi: 10.1007/3-540-36554-0_1]

[2] Yoder J, Johnson R. The adaptive object-model architectural style. In: Proc. of the Working IEEE/IFIP Conf. on Software Architecture (WICSA 2002). Montreal, 2002. 25−31. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.66.3382

[3] Garlan D, Cheng SW, Huang AC. Rainbow: Architecture-Based self-adaptation with reusable infrastructure. Computer, 2004, 37(10):46−54. [doi: 10.1109/MC.2004.175]

[4] McKinley PK, Sadjadi SM, Kasten EP, Cheng BHC. Composing adaptive software. IEEE Computer, 2004,37(7):56−64. [doi: 10.1109/MC.2004.48]

[5] Mao XJ, Shang LJ, Zhu H, Wang J. An adaptive casteship mechanism for developing multi-Agent systems. Int'l Journal of Computer Applications in Technology (IJCAT), 2008,31(1/2):17−34. [doi: 10.1504/IJCAT.2008.017716]

[6] Chang ZM, Mao XJ, Wang J, Qi ZC. Dynamic binding mechanism and its operational semantics of component in multi-agent system. Journal of Computer Research and Development, 2007,44(5):806−814 (in Chinese with English abstract). [doi: 10.3724/SP.J.1001.2008.01113]

[7] Weyns D, Van Dyke Parunak H, Michel F, Holvoet T, Ferber J. Environments for multiagent systems: State-of-the-Art and research challenges. In: Proc. of the Environments for Multiagent Systems. LNCS 3374, Springer-Verlag, 2005. 1−47. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.58.9477

[8] JavaCC. 2009. https://javacc.dev.java.net/

[9] Bellifemine F, Caire G, Poggi A, Rimassa G. JADE: A white paper. Telecom Italia Lab Journal EXP, 2003,3(3):6−19.

[10] Kokar MM, Baclawski K, Eracar YA. Control theory-based foundations of self-controlling software. IEEE Intelligent Systems, 1999,14(3):37−45. [doi: 10.1109/5254.769883]

[11] Andersson J, de Lemos R, Malek S, Weyns D. Reflecting on self-adaptive software systems. In: Proc. of the ICSE 2009 Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS). Vancouver, 2009. 38−47. http://www.computer.org/portal/web/csdl/doi/10.1109/SEAMS.2009.5069072 [doi: 10.1109/SEAMS.2009.5069072]

[12] Costa F. Combining meta-information management and reflection in an architecture for configurable and reconfigurable middleware [Ph.D. Thesis]. Lancaster: Lancaster University, 2001.

[13] de Moura AL, Ururahy C, Cerqueira R, Rodriguez N. Dynamic support for distributed auto-adaptive applications. In: Proc. of the Aspect Oriented Programming for Distributed Computing Systems. Vienna, 2002. 451−456. http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1030811 [doi: 10.1109/ICDCSW.2002.1030811]

[14] Mei H, Huang G, Zhao HY, Jiao WP. A software architecture centric engineering approach for Internetware. Science in China (Series E), 2006,36(10):1100−1126 (in Chinese with English abstract). [doi: 10.1007/s11432-006-2027-1]

[15] McKinley PK, Cheng BHC, Ofria C, Knoester DB, Beckmann B, Goldsby HJ. Harnessing digital evolution. IEEE Computer, 2008,41(1):54−63. [doi: 10.1109/MC.2008.17]

[16] Dastani M, van Riemsdijk MB, Hulstijn J, Dignum F, Meyer J-J Ch. Enacting and deacting roles in agent programming. In: Odell J, *et al*., eds. Proc. of the Agent-Oriented Software Engineering (AOSE 2004). LNCS 3382, Springer-Verlag, 2005. 189−204. [doi: 10.1007/978-3-540-30578-1_13]

[17] Wang J, Shen R, Zhu H. Towards agent oriented programming language with caste and scenario mechanisms. In: Dignum F, Dignum V, Koenig S, Kraus S, Singh MP, Wooldridge M, eds. Proc. of Int'l Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2005). New York: ACM Press, 2005. 1297−1298. [doi: 10.1145/1082473.1082741]

© 中国科学院软件研究所 http://www.jos.org.cn

[18] 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]

[19] Cabri G, Ferrari L, Leonardi L. Enabling mobile agents to dynamically assume roles. In: Proc. of the 2003 ACM Symp. on Applied Computing (SAC). Melbourne, 2003. 56−60. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.99.9928 [doi: 10.1145/952532.952546]

[20] Zhu H, Lightfoot D. Caste: A step beyond object orientation. In: Proc. of the JMLC 2003. LNCS 2789, Berlin: Springer-Verlag, 2003. 59−62. [doi: 10.1007/978-3-540-45213-3_8]

[21] Uszok A, Bradshaw J, Jeffers R, Suri N, *et al*. KAoS policy and domain services: Toward a description-logic approach to policy representation, deconfliction, and enforcement. In: Proc. of the 4th IEEE Int'l Workshop on Policies for Distributed Systems and Networks (POLICY 2003). IEEE CS Press, 2003. 93−96. http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?isNumber=27164&arNumber=1206963&isnumber=27164&arnumber=1206963 [doi: 10.1109/POLICY.2003.1206963]

[22] Damianou N, Dulay N, Lupu E, Sloman M. The ponder policy specification language. In: Proc. of the Workshop on Policies for Distributed Systems and Networks (Policy 2001). LNCS 1995, Bristol: Springer-Verlag, 2001. 18−39.

[23] Wang QX. Towards a rule model for self-adaptive software. ACM SIGSOFF Software Engineering Notes, 2005,30(1):1−5. [doi: 10.1145/1039174.1039198]

[24] Colman A, Roles H. Players and adaptive organizations. Applied Ontology, 2007,2(2):105−126.

附中文参考文献:

[6] 常志明,毛新军,王戟,齐治昌.多 Agent 系统中软构件的动态绑定机制及其操作语义.计算机研究与发展,2007,44(5):806−814. [doi: 10.3724/SP.J.1001.2008.01113]

[14] 梅宏,黄罡,赵海燕,焦文品.一种以软件体系结构为中心的网构软件开发方法.中国科学(E 辑,信息科学),2006,36(10):1100−1126. [doi: 10.1007/s11432-006-2027-1]

董孟高(1979－),男,陕西高陵人,博士生,主要研究领域为软件工程,多 Agent 系统.

王戟(1969－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为面向 Agent 软件工程,形式化方法.

毛新军(1970－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为 Agent 理论与技术,面向 Agent 软件工程,软件体系结构和模式.

齐治昌(1942－),男,教授,博士生导师,主要研究领域为软件工程.

常志明(1979－),男,博士,主要研究领域为软件体系结构.

© 中国科学院软件研究所 http://www.jos.org.cn