自适应 Agent 策略描述语言的设计及编译器的实现

郝小雷,董孟高,毛新军,齐治昌
(国防科学技术大学计算机学院,湖南长沙 410073)

摘 要: 当前自适应系统的开发存在自适应逻辑和业务逻辑相互缠绕的问题,使得自适应系统的开发和维护变得极为复杂和困难.论文认为自主性是实现自适应性的基础和前提,提出将自适应逻辑和业务逻辑相分离的思想,设计了一个自适应 Agent 策略描述语言 SADL,用于对系统自适应特征进行描述.自适应 Agent 基于预定义策略,在运行时根据外部环境和内部状态的变化,通过动态绑定、释放、激活或钝化行为规约展示自适应行为.论文介绍了 SADL 语言的语法和语义及其编译器的设计和实现.

关键词: Agent; 动态绑定机制; 自适应策略描述语言
中图分类号: TP301 **文献标识码**: A **文章编号**: 0372-2112 (2009) 4A-065-05

The Design of Self-adaptive Agent Strategy Description Language and the Implementation of SADL Compiler

HAO Xiao-lei, DONG Meng-gao, MAO Xin-jun, QI Zhi-chang
(*Department of Computer Science and Technology, National University of Defense Technology, Changsha, Hunan* 410073, *China*)

Abstract: The self-adaptation logic and enterprise logic of self-adaptive systems are often tangled with together in existing approaches to engineering complex self-adaptive system, which makes it difficult and complicated to develop and maintain self-adaptive systems. We believe autonomy is the basis of self-adaptation and it is necessary to separate the self-adaptation logic and enterprise logic of self-adaptive systems in order to simplify the development of such complex systems. A Self-adaptive Agent strategy Description Language SADL is therefore designed to specify the self-adaptation of systems. Self-adaptive agent can sense the changes of the situated environment and dynamically execute self-adaptive operations such as join, quit, deactivate and activate in order to adapt to the environment changes. The paper introduces the syntax and semantics of SADL and the design and implementation of its compiler.

Key words: agent technology; dynamic binding mechanisms; self-adaptive agent strategy description language

1 引言

Agent 理论研究自上世纪九十年代以来非常活跃,受到了学术界和工业界的广泛关注,取得了许多成果,包括反应式结构、BDI 逻辑模型等.然而,现有的 Agent 技术或者是借于传统的人工智能技术,难以实现与主流技术的集成,或是借鉴于面向对象技术的思想,利用 Agent 类(Agent Class)或 Agent 类型(Agent Type)的概念抽象描述 Agent 的行为、作为生成软件 Agent 的模板[1].如:Gaia 方法采用了 Agent Class 对角色进行封装[2],Tropos 方法则采用了 Agent Type 来封装执行者的能力[3].这样的 Agent 在运行期间其行为规约是固定不变的,难以充分发挥面向 Agent 计算的灵活性和技术潜力[1].

随着互联网技术的发展及其应用的拓展,越来越多的软件系统驻留在动态、开放的环境中,需要适应环境的变化并展示自适应的结构和行为特征.当前自适应系统的开发存在自适应逻辑和业务逻辑相互缠绕的问题,使得自适应系统的开发和维护变得极为复杂和困难.

Agent 的自主性体现为 Agent 具有局部于自身的行为控制机制,能在没有人类或其它 Agent 的直接干涉和指导的情况下运行,并能根据其内部状态和感知到的环境输入决定自身的状态,控制自身的行为[4].自适应的 Agent 能够根据外部环境的变化动态地调整自身的结构和行为.因此,自主性是实现自适应性的基础和前提.

借助于组织学和社会学的思想,我们提出利用动态绑定机制来解释、描述、分析并实现自适应 Agent.我们

收稿日期:2008-10-22;修回日期:2008-12-17
基金项目:国家 863 高技术研究发展计划(No. 2007AA01Z135);国家自然科学基金(No. 60773018)

对现有的面向 Agent 开发平台 JADE(Java Agent DEvelopment framework)进行了扩展,引入了动态绑定机制,通过对行为规约的动态绑定、释放、激活和钝化,为其加入了动态的自适应行为. Agent 在运行过程中可以感知环境变化和 Agent 自身状态的改变,并根据策略文件中定义的策略,动态地加入或退出规约,以及将行为规约置为活跃或不活跃状态,从而展现了 Agent 通过执行不同的行为来适应环境的变化. 在这一过程中,如何定义及描述 Agent 的自主行为策略,以支持在运行时根据环境状态变化采取不同的行为来展示自适应行为的特征,成为了一个重要问题.

为了简化复杂自适应系统的开发和维护,需要将自适应系统的自适应逻辑和业务逻辑相分离. 为此,我们提出了一个自适应 Agent 策略描述语言 SADL(Self-adaptive Agent strategy Description Language),用于对系统的自适应特征进行描述,并设计和实现了 SADL 语言的编译器,将用 SADL 语言编写的程序转化为目标平台上的 Java 代码,从而来支持自适应系统的运行.

2 基于动态绑定机制的 SADL 语言

动态绑定机制是指 Agent 在其生命周期内可以动态绑定或释放某个行为规约(Role),所绑定的 Role 可以处于激活状态或非激活状态[5]. 一个 Agent 可以在同一时刻绑定或激活多个 Role. 在 Role 中定义了 Agent 需要执行的动作. 在不同的环境和状态下,Agent 可以通过绑定或激活不同的 Role,执行 Role 中对应的行为以适应环境和状态的改变. Agent 与 Role 之间的动态绑定关系如图 1 所示.

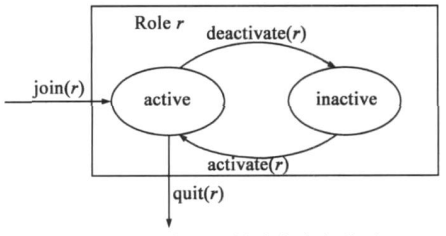


图1 Agent与Role的动态绑定关系

我们对 Agent 开发平台 JADE 进行了扩展,开发了自适应 Agent 的运行支撑平台 SADE(Self-adaptive Agent Development Environment). SADE 支持动态绑定机制,可以对 SADL 语言进行处理和执行,实现 Agent 的自适应性特征.

通过 SADL,用户可以预定义 Agent 的自适应策略,每个自适应策略包含一组自适应规则. 自适应规则描述了当运行环境和自身状态发生变化时,自适应 Agent 如何通过执行不同的自适应操作来适应环境的变化,包括加入一个角色、退出一个角色、挂起一个角色、恢复一个角色. 图 2 展示了 SADL 语言主要部分的 BNF 语法定义.

```
<AdaptiveStrategy> ::=
    "package" <Identifier>
    "strategy" <Identifier>
    "{"
       (<AdaptiveRule>)+
    "}" <EOF>
<AdaptiveRule> ::=
    "when" "(" <EnvironmentChangingEvent> ")"
    "if" "(" <AgentStatesChange> ")"
    "{"
      ( <AdaptiveAction>";")+
    "}"
<AdaptiveAction> ::=
    ("join"|"quit"|"activate"|"deactivate")
    "(" <Identifier> ("," <Identifier>)* ")"
```

图2 SADL语言主要部分BNF语法定义

SADL 语言中的语句可以划分为三种:when 语句、if 语句及自适应操作语句. 通过以上三种语句,SADL 语言对 Agent 的自适应规则进行定义和描述,自适应规则的形一般式如图 3 所示.

```
when( Environment Changing Event )
if( Agent States Change )
{
  AdaptiveAction;
}
```

图3 SADL描述的自适应规则的一般形式

其中,when 语句描述 Agent 外部环境的变化,if 语句描述 Agent 自身状态的变化. 在 Agent 的外部环境和自身状态发生变化以后,它们将变化后的事件和属性与括号中的条件语句进行匹配,选择并执行相应的自适应动作序列. 多个事件和状态的改变可以使用与操作符(&&)和或操作符(‖)连接,通过与、或操作符,用户可以将多个事件进行组合. 同时,为了方便用户进行模糊查找,可以在事件或状态里使用通配符(*),代表任意多个字符. 用户可以根据已知的部分字符串内容,通过搭配通配符方便的进行模糊查找.

表 1 when 语句事件类型及对应 SADL 关键字

事件类型名称	SADL 关键字
生命周期变化事件	LIFECYCLE 类型
自适应行为事件	ADAPTIVE 类型
Agent 状态变化事件	STATECHANGE 类型
Agent 操作执行事件	ACTIONDONE 类型
系统环境变化事件	TOPIC 类型
系统服务提供事件	SERVICE 类型
用户自定义事件	EVENT 类型

when 语句中的环境变化可分为七种类型(如表 1 所示). 在综合了 when 语句和 if 语句的判断结果后,执行 AdaptiveAction 中对 Role 的操作序列. 在 if 语句中,不仅可以对 Agent 的状态进行匹配,还可以通过 in() 语句的形式检索某个 Role 是否被激活、绑定等. AdaptiveAction 中的操作包括:join、quit、activate、deactivate,分别代表了绑定、释放、激活和钝化 Role 的操作.

本文将在第 4 节通过一个案例来演示 SADL 语言的实际运用.

3　SADL 程序的编译

SADL 语言所描述的 Agent 执行策略将被保存为单独的策略描述文件(*.s) ,通过由语法分析工具 JavaCC 构造的 SADL 编译器(SADLCompiler) ,将其编译为符合标准 Java 语法规范的策略文件(*.java) . 最后,通过 Java 语言编译器编译为自适应策略的目标代码(*.class) 后. 该代码可与 Agent 一同执行,实现 Agent 自适应特征. 图 4 描述了 SADL 程序的编译过程.

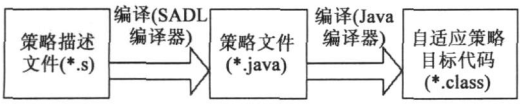


图4　自适应策略的编译过程

JavaCC 是由 Sun 公司开发的一个语法解析器的生成器,与 Java 语言集成. 它可以通过给定的语法描述文件(*.jj 或 *.jjt) 生成相应语法解析器的 Java 类和语法节点树类. SADL 编译器编译策略描述文件的流程如图 5 所示.

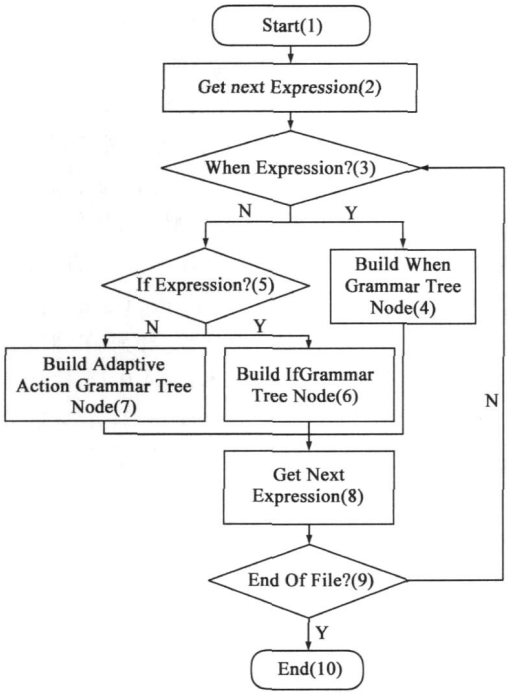


图5　SADL编译器编译策略的算法流程

编译器首先调用 ASTStrategyDeclaration 类的实例(1) ,通过其 process 方法在此构造节点树,进行初始化和实例化工作,随后编译器获取策略描述文件的第一个语句(2) 并进入循环,该循环的终止条件为:到达策略描述文件结尾,即文件中所有语句都已解析完毕. 循环内,首先判断当前语句是否为 when 语句(3) ,若是,则在此通过调用 ASTWhenExpression 类,构造一个 when 节点树(4) ,对 when 语句进行语法分析,检查其是否符合 SADL 的语法规范,并将其编译为 Java 代码,若否,则继续判断其是否为 if 语句并根据结果决定是否通过调用 ASTIfExpression 类来构造 if 节点树(5 ,6) ,若当前语句也不是 if 语句,则直接调用自适应行为的相关节点树构造代码,生成相应的自适应行为节点生成树(7) ,然后编译器获取下一语句并判断是否到达文件结尾(8 ,9) .

```
61 ......
62
63 //打印Strategy实例生成语句
64 if (! printdostrategy && (t.kind == SADLCompilerConstants.LBRACE)) {
65   ostr.println("\n\tpublic void setStrategy(Agent a) {");
66   ostr.println("\t\tagent = ("+agentName+")a;");
67   ostr.println("\t}");
68   ostr.println("\tpublic void init() {");
69   ostr.println("\t}");
70   ostr.println("\tpublic void doStrategy()");
71   ostr.println("\t{");
72   ostr.println("\t\tif (agent == null){");
73   ostr.println("\t\t\treturn;");
74   ostr.println("\t\t}");
75   printdostrategy = true;
76   prefix = "NEW LINE: ";
77 }
78
79 ......
80
```

图6　ASTStrategyDeclaration.java部分代码

为了能够更好地完成语法分析及编译工作,SADL 编译器需要对特定的语句解析,从而生成节点树. 我们一共在以下 4 个部分构造了节点树:算法初始化部分、when 语句、if 语句及 AdaptiveAction 的自适应操作语句,其功能是:完成初始化、实例化工作,检查 when、if、AdaptiveAction 语句是否符合语法规范,并将其内容编译为符合 Java 语法的语句. 这 4 部分的节点树构造代码使用 Java 语言编写,根据 JavaCC 语法规范,这些文件分别被命名为 ASTStrategyDeclaration. java、ASTWhenExpression. java、ASTIfExpression. java 和 ASTAdaptiveAction. java. 在这几个节点树构造代码中,除了 ASTStrategyDeclaration. java 是在算法开始时调用一次并直接生成初始化和实例化的 Java 代码以外,其余语句均在算法判断

```
1 //strategy节点生成树, 完成strategy类初始化及实例化工作
2 void StrategyDeclaration() #StrategyDeclaration :
3 { Token t1, t2;
4   Set varset = new HashSet();
5 }
6 {
7   ( t1 = "strategy" t2 = <IDENTIFIER> ) { jjtThis.m_strName = t2.image;
8   StrategyBody()
9 }
10
11 void StrategyBody() :
12 {}
13 {
14   "{"
15   ( StrategyBodyStatement() )+
16   "}"
17 }
18
19 void StrategyBodyStatement() :
20 {}
21 {
22   ( (WhenExpression() [IfExpression()]) | IfExpression() )
23   "{" (StrategyStatement())+ "}" )+
24 }
25
26 //when语句表达式以及语法生成树
27 void WhenExpression() #WhenExpression :
28 {}
29 {
30     "when" "(" EnvironmentExpression() ")"
31 }
32
33 //if语句表达式
34 void IfExpression() :
35 {}
36 {
37   "if" "(" InternalStateExpression() ")"
38 }
```

StrategyDeclaration 节点数标记

when节点数标记

图7　SADL编译器语法描述文件部分代码

出语句类型后调用相对应的类,对语句中的代码进行SADL语法检查并生成对应Java代码.图6展示了AST-StrategyDeclaration.java的部分代码示例.

SADL编译器的语法以BNF范式的形式在语法描述文件中定义.在通过JavaCC工具编译后,SADLCompiler.jjt文件中的JavaCC代码将会被编译为普通的Java类,并最终被编译为Java的目标代码.在语法描述文件中,可以在特定部分进行标记,以构造节点树.这样,在最终生成的编译器目标代码中,就会在具有标记的地方通过调用指定的类文件,生成节点树,完成特定的初始化、词法转换工作.SADL编译器的语法描述文件(SADLCompiler.jjt)及其语法树标记的部分代码如图7所示.

4 SADL程序的执行

在SADE平台中,Agent的行为执行和外部环境的改变将会产生消息.SADE平台将这些消息发送给Agent.Agent将所有消息以消息队列的形式进行保存,对消息队列中的内容逐一进行解析,获得行为和事件改变,从而与策略描述文件中由when和if语句预定义的事件进行匹配,选择执行相对应的自适应行为.

为了完成事件和状态的匹配查找工作,我们设计和实现了Strategy类.Strategy类由Java语言实现,其主要功能是:find方法对when语句中的七种事件进行匹配查找,Search方法对if语句中的Role状态进行匹配查找,Agent的各种状态的查找匹配由Agent中的get方法直接完成.编译后的策略类为Strategy类的子类,通过调用Strategy类中的Search、find方法以及Agent类的get方法进行策略定义中的事件和状态匹配.Strategy类的部分代码如图8所示.

对于具体的Agent实例,可以在setup方法中通过

```
34  ------
35  //查找某个角色是否在绑定、激活、钝化的角色序列中
36  public boolean search(String roleID, List roleList) {
37    if (!roleList.isEmpty()) {
38      Iterator i = roleList.iterator();
39      Role r;
40
41      while(i.hasNext()){
42        r = (Role)i.next();
43        String tmp = r.whoAmI();
44        if(tmp.equalsIgnoreCase(roleID)){
45          return true;
46        }
47      }
48    }
49    else {
50      return false;
51    }
52    return false;
53  }
54
55  //查找某一个符合条件的事件消息, 返回值为Boolean
56  public boolean find(Object object, List msgList) {
57    if(msgList.isEmpty()) {
58      return false;
59    }
60    ------
61  }
62  ------
```

图8 Strategy.java部分代码

SpringFramework指定该Agent所使用的策略文件名称,该策略文件通过SADL编译器编译后生成的Java类是Strategy类的子类.在执行策略的时候,Agent类调用Strategy基类的doStrategy方法,由Java虚拟机查找并调用指定的具体策略类中的doStrategy方法,并进一步调用search等方法对事件和状态匹配,从而实现策略文件的执行.其UML类如图9所示.

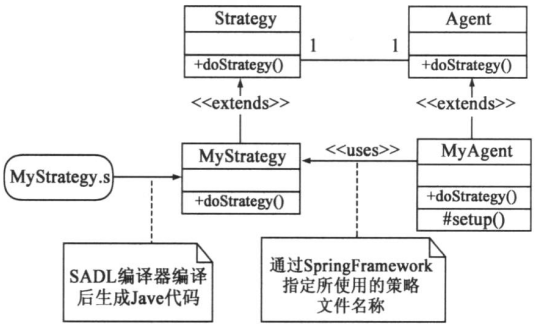


图9 自适应策略执行的UML类图

5 案例分析

为了演示SADL语言的功能及SADL编译器的使用,我们设计了一个网上交易系统的实例TradingAgent,其主要功能是模拟用户在网上交易的具体行为.其场景描述为:系统中的用户类型有3种:浏览者(Visitor),买者(Buyer),卖者(Seller).浏览者可以查看商品信息,但不能进行交易.买者不仅可以浏览商品信息,还可以与卖者交互,购买商品.卖者持有待出售的商品,可以查询商品,也可以与买者交互,出售自己的商品.系统中,Agent会根据它内部状态的情况和感知到的外部环境的变化,动态的绑定或激活不同的Role,执行不同的行为.在此基础上,我们通过SADL语言定义此实例中各Agent的行为策略,并用SADL编译器将策略进行编译.

下面我们通过描述此系统中Agent的具体行为演示SADL的运用和编译:系统启动以后,Agent如发现系统中存在其他Agent出售自己需要购买的货物,将绑定Buyer.此实例中我们假定Agent需要购买的商品为Abook.此过程中,Agent产生的决策行为是:发现有其他Agent在出售其所需的Abook后,绑定并激活Buyer.自主决策行为描述的SADL代码描述如图10所示.

在使用SADL将Agent的决策描述完毕后,将其保

```
1 package TradingAgentStrategy;
2
3 strategy TradingAgentStrategy;
4 {
5   when (SERVICE(Sell, Abook, Existent))
6   if (money > 0 && in(Visitor,activeRoles) && !in(Buyer,boundRoles))
7   {
8     join(Buyer);
9   }
10 }
```

图10 Agent决策的SADL代码描述(TradingAgentStrategy.s文件)

存为单独的策略描述文件(TradingAgentStrategy.s.),并使用 SADL 编译器对其进行编译,保存为 Java 文件(TradingAgentSrategy.java).图 10 中 SADL 代码编译后生成的 Java 代码如图 11 所示.

图11　略编译后的Java代码(TradingAgentStrategy.java文件)

图 12 详细描述了策略编译后的 Java 代码,各部分对应的 SADL 代码分别是:(1) SERVICE(Sell,Abook,tom)部分;(2) money > 0 部分;(3) in(Visitor,activeRoles)部分;(4) in(Buyer,boundRoles)部分;(5) join(Buyer)部分.

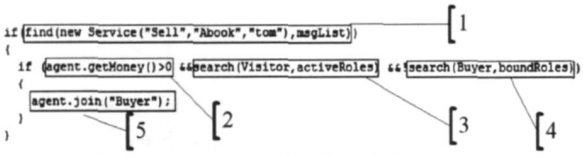


图12　SADL代码编译后相对应的Java代码

6　总结和进一步工作

随着 Internet 的快速发展和普及,软件实体的自适应能力已经成为软件系统的一项重要特征.为了更好的展现 Agent 的自适应特征,实现 Agent 的动态运行,我们提出了动态绑定机制.作为支持 Agent 动态运行的重要基础,如何定义和描述 Agent 的自主行为策略成为了一个重要问题.针对上述问题,本文提出设计描述 Agent 策略语言 SADL,并将策略描述与 Agent 分离,从而实现了 Agent 的自适应行为,并提高了可重用性.同时,为了支持 SADL 的运行和使用,利用 JavaCC 实现了 SADL 编译器.我们需要完成的进一步工作包括:(1) 完善 SADL 语言,使其在能够展现 Agent 自适应特征的基础上,进一步展现 Agent 的自演化特征.(2) 结合现有的 Agent 技术,设计和实现能够支持 SADL 编辑、编译和调试的运行平台.

参考文献:

[1] K H Dam, M Winikoff. Comparing agent-oriented methodologies[A]. Proceedings of AOIS'2003, LNCS 3030[C]. Heidelberg: Springer, 2004. 78 - 93.

[2] F Zambonelli, N Jennings, M Wooldridge. Developing multiAgent systems: the gaia methodology[J]. ACM Transactions on Software Engineering and Methodology, 2003, 12(3): 317 - 370.

[3] Zhu H. SLABS: a formal specification language for agent-based systems[J]. International Journal of Software Engineering and Knowledge Engineering, 2001, 11(5): 529 - 558.

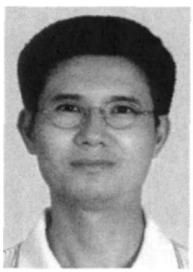
[4] 毛新军. 面向主体的软件开发[M]. 北京:清华大学出版社, 2005.
Mao Xinjun. Agent Oriented Software Development[M]. Beijing: Tsinghua University Press, 2005. (in Chinese)

[5] 常志明,毛新军,齐治昌. 基于 Agent 的网构软件构件模型及其实现[J]. 软件学报, 2008, 19(5): 1113 - 1124.
Chang Zhiming, Mao Xinjun, Qi Zhichang. Component Model and Its Implementation of Internetware Based on Agent[J]. Journal of Software, 2008, 19(5): 1113 - 1124. (in Chinese)

作者简介:

郝小雷　男,1981 年 8 月出生于安徽涡阳.国防科学技术大学计算机学院硕士生,研究方向为软件工程、多 Agent 系统.
E-mail: iambirdiambird@gmail.com

董孟高　男,1979 年 2 月出生于陕西高陵.国防科学技术大学计算机学院博士生,研究方向为软件工程、多 Agent 系统.
E-mail: mgdong@nudt.edu.cn



毛新军　男,1970 年生于浙江江山.博士,教授,博士生导师,中国计算机学会会员,感兴趣的研究方向包括:面向 Agent 软件工程、新颖软件开发方法学、分布计算技术、软件体系结构和模式等.
E-mail: xjmao@nudt.edu.cn

齐治昌　男,1942 年生于北京,现为国防科学技术大学计算机学院教授、博士生导师.主要研究方向为软件工程.