文章编号:1001 - 9081(2007)04 - 0868 - 04

Agent 应用系统模板语言研究

王家昉,冯志勇
(天津大学 计算机科学与技术学院,天津 300072)
(javacppnimbel@ hotmail. com)

摘 要:为了有效地构造可靠、便于维护的多 Agent 系统(MAS),根据 Grammarware 的工程化思想与产生式编程技术,以基于黑板的单 Agent 结构为基础,提出一种 Agent 模板语言(APL)。通过一个企业业务过程管理(BPM)场景,说明在应用中如何使用 APL 描述 Agent 中针对具体应用的数据类型、知识源组件等,并由解析器将 APL 转换为 JADE 上的执行代码,进而实现多 Agent 系统的快速构建,实现 MAS 自顶向下的开发,从而提高系统开发的效率,增强系统的灵活性与可复用性。

关键词:Agent 模板语言; 多 Agent 系统; 产生式编程; Grammarware; 黑板; JADE
中图分类号: TP311.52 **文献标识码**:A

Research of pattern language for Agent-based application

WANG Jia-fang, FENG Zhi-yong
(*School of Computer Science and Technology, Tianjin University, Tianjin* 300072, *China*)

Abstract: To develop robust, maintainable Multi-agent System (MAS), an Agent Pattern Language (APL) has been proposed in the light of Generative programming and engineering disciplines for Grammarware, in company with a single Agent architecture based on blackboard. With a Business Process Management (BPM) application scenario, how to use the APL language to describe the data type and components like knowledge sources and so on was introduced, as well as a parser for APL parsed APL files and generated executable code for Agent which ran on JADE. In this way, systems based on Agent can be developed fast and MAS can be constructed in a top-down manner. Then not only the productivity for the development agent-base system is improved, but flexibility and reusability of the system are also enhanced.

Key words: Agent Pattern Language (APL); Multi-agent System (MAS); generative programming; Grammarware; blackboard; JADE

0 引言

基于 Agent 的计算被认为是一种新的软件开发的范式,在软件工程、分布式系统开发和智能系统开发上都能够发挥重要的作用。为了构建可靠、有效的基于 Agent 和多 Agent 系统的应用,研究人员提出了很多方法学与建模方法。例如在方法学方面的 AAII 方法学和 Gaia 方法学等;在结构方面包括慎思、反应以及混合等一系列结构。

尽管研究人员提出了很多 Agent 的结构和一些方法学来构造面向 Agent 的系统,但是这些方法多基于传统的面相对象分析/设计(OOA/D),有些采用静态、非分布软件结构的建模,不符合 Agent 所处的动态分布环境;有些结构较为复杂,且一般集中于对单个系统的设计,缺乏针对不同应用的灵活性,使得在 Agent 开发中针对不同应用进行了大量重复工作,降低了开发效率。

借鉴 Grammarware 的工程化[1]与产生式编程[2]思想,基于以黑板系统为基础的单 Agent 结构,提出了一种称为 Agent Pattern Language (APL)的模板语言[3],用来在较高的层次上抽象描述 Agent 中针对具体应用的数据类型、知识源组件等,并由 Agent 解析、转换为执行代码,通过这种方式,实现对于 Agent 与 MAS 自顶向下的构建,从而提高系统开发效率,增强系统灵活性与可复用性。

1 基于黑板系统的 Agent 结构

APL 语言主要针对基于黑板的 Agent 结构[3],由黑板存放 Agent 的运行信息,与知识源相互协作完成特定的功能,整个 Agent 负责由 APL 文件生成可执行代码,完成 APL 说明的目标。根据文献[1]中作者对于 Grammarware 进行工程化开发的原则,以 APL 定义的语法作为 base-line grammar,进而通过定制 APL 文档,描述 Agent 知识源等组件;同时基于黑板的 Agent 结构借鉴产生式编程的思想,解析 APL 文档,最终实现为针对具体应用的系统。同时可根据不同的知识源运行策略来实现反应型或慎思型 Agent 的特点,从而增强 Agent 的灵活性和复用性,提高针对不同应用的 Agent 的开发效率。图 1 是基于黑板的 Agent 结构。

1.1 Agent 结构

1.1.1 黑板与知识源

黑板是 Agent 的全局数据库,主要包括数据对象结构与子句对象结构两部分。

数据对象结构是由一组有限数据对象类型的实例构成的集合。数据对象类型用来描述一类数据对象,包括一个类型名和属性集合,其中属性用来描述数据对象的特征,一个属性

收稿日期:2006 - 10 - 13;**修订日期**:2006 - 12 - 02 **基金项目**:天津市重点科技攻关资助项目(04310891R)
作者简介:王家昉(1982 -),男,甘肃临夏人,硕士研究生,主要研究方向:智能 Agent、知识工程; 冯志勇(1964 -),男,内蒙古武川人,教授,博士生导师,主要研究方向:软件体系结构、知识工程、语义网.

定义包括属性名与属性类型。

子句对象结构是关系实例和函数实例的集合。其中,关系是数据对象类型到真假值的映射,而函数是数据对象到数据对象的映射。将关系表达为 $R(d_1, d_2 \cdots d_n)$,当每个参数被赋予具体的实例时,则形成关系实例 RelationInstance,函数实例类似。

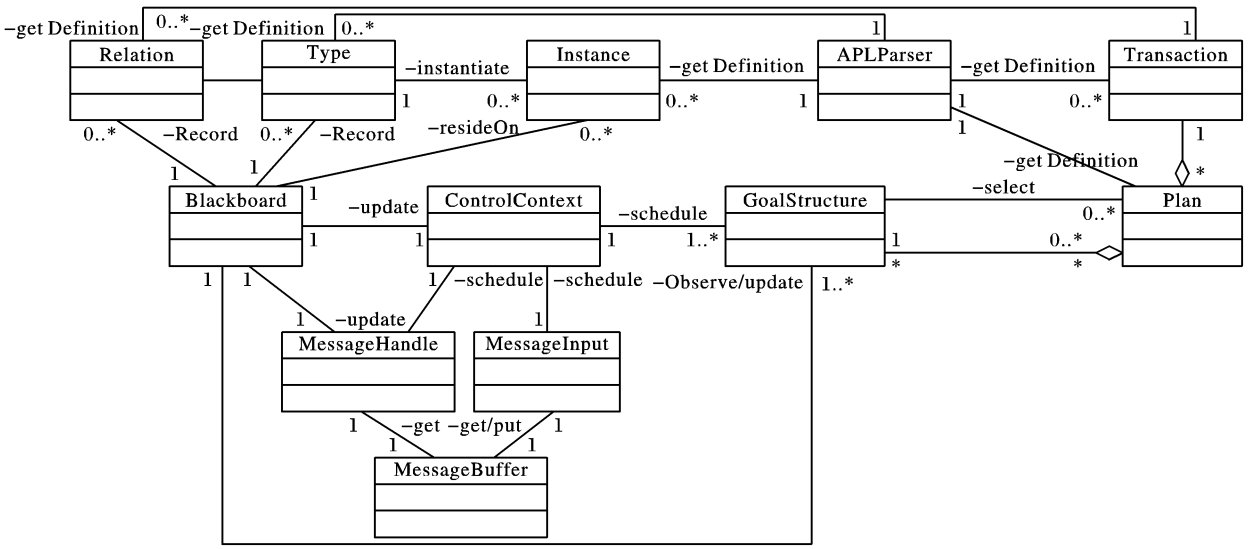

图 1　基于黑板系统的 Agent 结构

知识源(Knowledge Source,KS)作为独立解决问题的模块,由如下部分构成:

1)输入变量集是一个数据对象实例集合的列表,表达为如下形式:

$InputVariable = \{IV_m \mid IV_m = \{IV_{m1}, IV_{m2}, \cdots, IV_{mi}, \cdots, IV_{mk}\}$,$IV_{mi}$是数据对象实例$\}$

2)输入条件集是一个 boolean 对象集合的列表,表达为如下形式:

$InputCondition = \{IC_m \mid IC_m = \{IC_{m1}, IC_{m2}, \cdots, IC_{mi}, \cdots, IC_{mk}\}$,IC_{mi}是 boolean 对象实例$\}$

3) 前提条件 *PreCondition* 是一个布尔表达式,由条件语句以及关系实例构成;

4) 后置条件 *PostCondition* 是一个集合,包含在知识源运行完毕后需要更新黑板上的数据对象及子句对象实例。

根据上面的定义,输入变量集 *InputVariable* 中存放的是该知识源运行所需要的,存放于黑板的数据对象实例。当这些实例被外界到达的消息或其他知识源的运行所修改,就有可能触发该知识源。在 *InputVariable* 的定义中,任何 $IC_m \in InputCondition$ 对应于 $IV_m \in InputVariable$,任给 $IC_{mi} \in IC_m$ 对应 $IV_{mi} \in IV_m$。每一个 IV_{mi} 被更新后,对应的 IC_{mi} 设置为 true。任何 $IC_m \in InputCondition$ 中的所有 *Boolean* 对象都设置为 true 时,该知识源被触发。当该知识源被 *ControlContext* 选中执行时,知识源将被更新的数据对象实例从黑板拷贝到本地,将设置为 true 的 $IC_{mi} \in IC_m$ 设置为 false。

1.1.2　消息相关知识源和规划库

知识源 MessageInput 负责 Agent 与其他 Agent 之间的消息收发,而知识源 MessageHandle 负责处理接收到的消息。此外还有一些其他的模块支持 Agent 的运行。

当 MessageInput 接收到一个外界到达的消息后,将其放在缓冲队列中,并通过事件通知 MessageHandle。MessageHandle 收到消息后,就等待 ControlContext 选择。被选中执行后,MessageHandle 根据运行中的目标结构与规划从缓冲队列中选择消息,根据消息内容更新黑板变量。各目标结构在运行中所需要发送的消息也放在缓冲队列中。MessageInput 监测到缓冲队列中有消息要发送,就等待 ControlContext 选择,选中后将队列中消息发送出去。

为了在实际应用场景中支持 MAS 的开发与运行,基于黑板的 Agent 是在 JADE[4] 平台上实现的。Agent 之间利用 ACL(Agent Communication Language)进行通信,MessageInput、MessageHandle 知识源负责消息的处理。这里采用集中的消息处理是为了避免 GoalStructure 知识源各自进行消息收发可能带来的错误,并且可以根据需要用恰当的消息更新黑板。

此外,知识库 Knowledge 存放 Agent 运行所需的一些全局知识,而规划库 PlanLibrary 中存放用于不同目标结构的规划。规划由名称、objective(规划针对的目标结构名)、PreCondition(规划执行所要满足的条件)、优先级以及规划体组成,规划体采用确定有穷状态自动机 DFA 表示,其中状态之间的转换由一组 Transaction(一组完成一定功能的动作)和 GoalStructure 组成。根据 Agent 运行的上下文,Plan 选择不同的状态转移,最终达到终结状态,完成规划执行。

1.1.3　控制机制

根据上面的介绍,ControlContext 协调不同的知识源协同工作完成特定的目标,其中由议程表 Agenda 变量控制知识源的执行。每个目标结构知识源有一个初始的优先级,而 MessageHandle 和 MessageInput 的优先级最高。当 Agenda 打开时,ControlContext 选择处于 Active 的知识源准备运行,选定后关闭 Agenda。如果 MessageHandle 或 MessageInput 被选中则执行相应的功能。若某个 GoalStructure 被选中,如果该目标结构是首次执行,那么根据 Agent 当前的环境上下文,从 PlanLibrary 选择一个合适的 Plan;如果目标结构已经选定了 Plan 并已经开始执行,那么就根据 Plan 当前的状态与环境上下文选择状态转换(可能是 Transaction 或 GoalStructure)执行一步(如果选中子目标 GoalStructure,则保存当前目标,先执行子目标)。当 MessageInput、MessageHandle 完成处理,或 GoalStructure 执行完一步,打开 Agenda,开始下一轮选择。控制过程参见图 2。

当 ControlContext 连续选中同一个目标执行的时候,其优先级相应增加。而有时候因为环境的动态变化,一个高优先

级的 GoalStructure 会打断当前正在执行的 GoalStructure。可以通过指定不同的策略决定 GoalStructure 之间的竞争策略，使 Agent 在对于环境的反应性和对目标的坚持性上做出平衡。

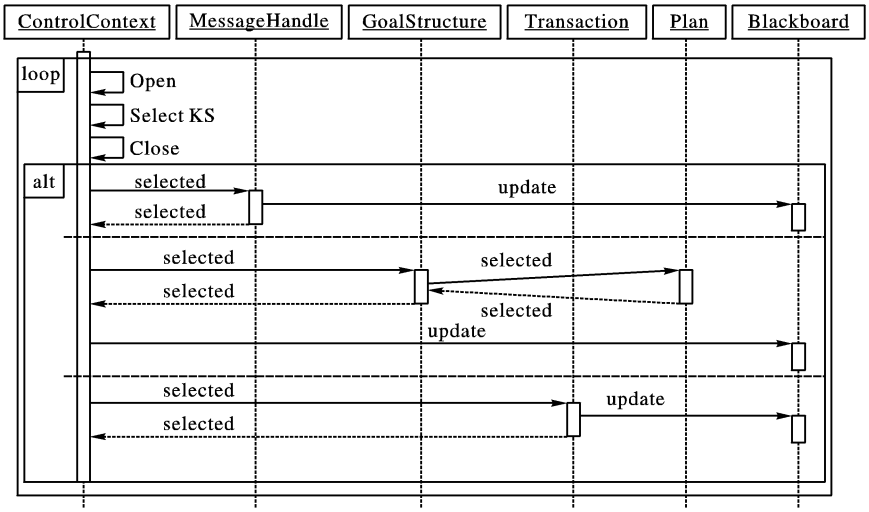

图 2 ControlContext 的控制过程

2 APL 与场景示例

2.1 Agent 模板语言 APL

前面介绍的 Agent 结构，在针对不同的应用时，需要不同的配置信息，如针对于特定应用的数据类型、规划、不同目标结构的调度策略等，这些信息可以通过 APL 语言来描述。目前 APL 主要描述应用场景中的三方面内容：1）表达数据对象类型，子句对象类型以及它们的实例；2）描述目标结构知识源，包括目标名，输入变量集，前提条件，优先级等内容；3）描述针对不同目标结构的规划的相关信息，包括规划名，Objective，规划体，前提条件等。这里采用 XML 进行表达，因为 XML 具有较强的适应性，并且可以简化编译/解释器的构造。

从图 1 可以看出，Agent 中有一个 APL Parser 解析器部件。Agent 在运行中与具体应用相关的知识，包括各种数据对象实例、子句对象实例、GoalStructure 的实例、规划 Plan 的实例等，都编制在 APL 文件中，在 Agent 初始化时由 APL Parser 解析 APL 文件，并生成各种实例的对象，而后 Agent 按照上面介绍的 Agent 各个部件的运行机制，完成指定的目标。

2.2 场景描述

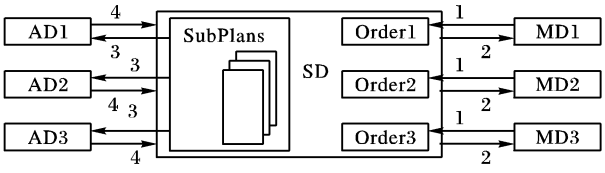

图 3 场景描述

这里采用一个企业业务过程管理的场景来说明基于黑板的 Agent 结构与 APL 结合如何实现特定的目标。利用 MAS 进行业务过程管理，将业务过程委托于不同的 Agent。Agent 通过对环境的观测和与环境的交互，根据环境和业务策略，可以对业务过程管理中动态的变化做出实时、甚至预动的反应，展现出目标导向的特点[5]。

考虑如下场景。某汽车制造商由若干部门组成：几个生产车间（用 AD 表示）；几个市场部门（MD），其中每个市场部门负责处理一个地区的订单；一个调度部门（SD），调度部门主要负责为生产车间制定生产计划。当有新订单到达时，SD 要根据到达的订单以及当前已经制定的生产计划生成新的生产计划，并发送给各 AD。此外，AD 可能因为故障等原因，不能按期完成计划，将这个情况发送给 SD 后，SD 要修改当前生产计划，并将修改后的计划发送回各 AD。场景描述以及消息往来示例参见图 3。

2.3 场景实现

这里从调度部门的角度进行分析。假定每个部门或生产车间都有一个 Agent。生产部门的 Agent 负责接收生产计划与异常情况的发送，市场部门的 Agent 负责向 SD 发送订单，并接收对订单的确认。这些 Agent 构成一个 MAS，Agent 之间相互通信，通过交换数据与控制信息来协同生产过程。

首先，要用 APL 表达与生产过程控制相关的关键信息。对于 SD，应该在黑板上记录如下信息：MD 发送过来的订单信息；为每个 AD 制定的生产计划信息；AD 发送过来的异常信息。根据前面的形式化定义，APL 要表达出这些信息涉及的数据对象类型，每个类型的实例，所用到的关系以及关系的实例等。下面的代码展示了订单类型的定义。

```
<type>
  <name>Order</name>
  <attribute>
    <name>region</name>
    <type>string</type>
  </attribute>
  <attribute>
    <name>quantity</name>
    <type>integer</type>
  </attribute>
  …
</type>
```

如代码所示，通过 type、name、attribute 等标签，Order 类型用 region（所面向的地区），quantity（订购商品数量）这些属性进行表达。而 Order type 的实例可以通过如下代码来表达：

```
<instance>
  <name> order2 </name>
  <type> Order </type>
  <attr_value>orderId: SH001</attr_value>
  <attr_value>region: Beijing</attr_value>
  <attr_value> quantity: 800 </attr_value>
  …
</instance>
```

同样地，生产计划和异常信息也可以通过类似的方式表达。在表达数据结构、子句对象结构的同时，Agent 所要完成的目标结构，以及用以完成这些目标结构的规划，也要用 APL 代码以表达。在这个场景中，SD 的 Agent 主要有三个目标：根据到达的订单制定新的生产计划，向 AD 发送生产计划，处理 AD 发送过来的异常。如下代码是处理订单目标的目标结构 APL 代码说明（省略部分标签）。

```
<goal>
```

```
name: processOrderGoal
<input_sets>
  <set> order1 </set> <set> order2 </set>
  <set> order3 </set>
  </input_sets>
precondition: Null Priority: 7
</goal>
```

注意这个目标结构的输入变量集 <input_sets> 分为三组,每组的成员分别针对不同地区的订单,当有一个订单到达,按照 1.1.1 中介绍的方式触发该知识源。在 ControlContext 允许该目标结构知识源的对象开始执行时,该对象在 PlanLibrary 中查找可以满足该目标结构的规划。下面的代码是用 APL 表达的规划 Plan 的示例:

```
<plan>
  name: Plan1
  objective: processOrderGoal
  <body>
    init_state: S1 fianl_state: S3
    <rules> S1 -(r1) ->S2 ->(r2) ->S3 </rules>
    <tr> r1: processOrder(Transaction) </tr>
    <tr> r2: DealOrder (Sub-goal) </tr>
  </body>
</plan>
```

这个规划有三个状态,通过两个状态转换(<tr>标签)顺序连接起来。第一个状态转换是 Transaction processOrder,判断到达的订单是否可以接受。第二个状态转换是一个子目标,根据前面的 Transaction 计算得到的结果,如果接受订单就计算新的生产计划,并放入发送队列;如果是拒绝订单,就生成拒绝订单的消息,并放入发送队列中。

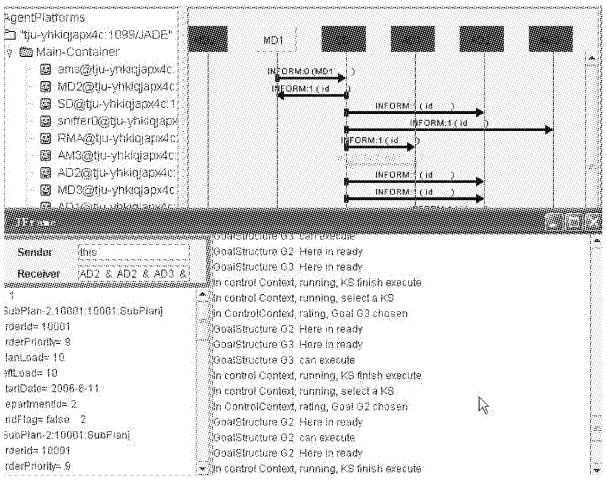

图 4　SD、AD、MD 间消息的交换与 SD Agent 运行控制信息

此外,场景中还需要一个目标结构以及对应规划来实现向 AD 发送生产计划的目标,和用于处理 AD 发送过来的异常信息的目标结构及对应规划。根据上面的设计,图 4 显示了场景的一个实现结果,由 JADE Sniffer Agent 显示了 SD、AD 和 MD Agent 之间的消息往来,以及 SD Agent 中 GoalStructure 与 Plan 等的执行情况。

2.4　与相关研究的比较

上面的场景是一个较典型的模型,显示了在生产过程中存在的协作与竞争关系。整个过程需要各个部门(SD、MD、AD)互相交流,互相协调才能完成。而协作中也存在竞争关系,如不同目标由于优先级的不同而带来的竞争等等。要处理好这些竞争,需要设计比较完善的规划、调度算法,融入到 Agent 的运行中。

在文献[6]中提出采用基于黑板的 Agent 结构处理企业中过程计划与生产调度的生成。这里知识源处理生产调度/规划中出现的问题利用,以此提供灵活、用户导向的处理方式。但在面对不同企业不同的业务需求时,其知识源的编制需要根据具体的业务场景来编制,可能带来大量的重复工作,缺乏 APL 与黑板 Agent 结构结合带来的灵活性。

在由汉堡大学开发的 JADEX 中[7],采用了 Agent 定义语言(Agent Definition Language,ADL)描述 Agent 运行时需要的内容,包括所需的对象和变量等。通过这种方式能够给针对不同应用场景的 Agent 的开发带来一定灵活性。但是其描述文件的抽象层次比较低,文件定制不够方便,而 APL 可以借助一些工具,使系统设计人员能够方便的生成 APL 文件,提升开发效率。

3　结语

在基于 Agent 与 MAS 的系统中,利用模板语言 APL 从应用场景中抽象出数据类型和组件等的描述,自顶向下完成 MAS 的开发。一方面,增强了 Agent 针对不同应用场景的适应性,以及 Agent 的可维护性与健壮性,另一方面也提高了 Agent 开发人员的开发效率。在给出的场景中,它使得部门更加适应内部组织以及来自市场的变化,为业务过程管理提供更大的灵活性。在进一步的工作中,可以借鉴 Web Service 研究中的成果,扩充 APL 的定义,包含更加复杂有效的业务流程描述;在 Agent 中引入较为复杂的控制机制,提升 Agent 的性能;也可以采用本体等方式来丰富黑板上所记录信息的语意内涵。

致谢　感谢赵飞、王辉、张卓博的协助,特别感谢李铁,他的工作是我们的工作的基础。

参考文献:

[1] KLINT P, LÄMMEL R, VERHOEF C. Toward an engineering discipline for Grammarware[J]. ACM Transactions on Software Engineering and Methodology, 2005, 14(3): 331-380.

[2] CZARNECKI K, EISENECKER UW. Components and generative programming[A]. Proceedings of the 7th European software engineering conference held jointly with the 7th ACM SIGSOFT international symposium on Foundations of software engineering[C]. Toulouse, France: Springer-Verlag, 1999: 2-19.

[3] 李铁. 面向过程集成的多 Agent 系统研究[D]. 天津: 天津大学, 2004.

[4] RIMASSA G. JADE A White Paper[EB\OL]. http://jade.tilab.com/papers/2003/WhitePaperJADEEXP.pdf, 2003.

[5] WANG MH, WANG HQ. Intelligent Agent Supported Business Process Management [A]. The 38th Annual Hawaii International Conference on System Sciences (HICSS'05)[C]. Washington, DC, USA: IEEE Computer Society, 2005.

[6] HILDUM D, SADEH-KONIECPOL N, LALIBERTY TJ, *et al*. Blackboard agents for mixed-initiative management of integrated process-planning/production-scheduling solutions across the supply chain. [A] Proceedings of the Ninth Conference on Innovative Applications of Artificial Intelligence (IAAI-97)[C]. 1997. 1000-1005.

[7] POKAHR A, BRAUBACH L, LAMERSDORF W. Jadex: Implementing a BDI-Infrastructure for JADE Agents[J]. EXP-In Search of Innovation (Special Issue on JADE), Telecom Italia Lab, Turin, Italy, 2003, 3(3): 76-85.