

# 基于对象的并发模型

周光明,高继民

(深圳职业技术学院 软件工程系,广东 深圳 518055)

**摘要:**为了研究和发展分布式人工智能,探讨一种新的计算模型,利用面向对象技术提出了一个基于对象的并发模型(OBCM)。讨论了它的两种通讯机制,即同步和异步通讯方式,异步通讯方式又分为过去型消息传递和将来型消息传递,而同步通讯方式则包括现在型消息传递和等待方式。证明了它们可归约为过去型消息传递和等待状态。另外,OBCM为了增强其表达能力,提供了响应目的地机制。在OBCM中,对象的独立自治特点对模拟分布式问题求解尤为适用。最后以一个分布式问题求解的具体实例说明了OBCM的思想。

**关键词:**并发;通讯策略;面向对象程序设计;分布式问题求解

**中图分类号:**TP181

**文献标识码:**A

**文章编号:**1673-629X(2008)12-0147-03

## Object - Based Concurrent Model

ZHOU Guang-ming, GAO Ji-min

(Software Engineering Department, Shenzhen Polytechnic, Shenzhen 518055, China)

**Abstract:** In order to research and develop distributed artificial intelligence and explore a new computational model, a object - based concurrent model (OBCM) is presented by using object - oriented techniques. Its two kinds of communication mechanisms named synchronous communication which includes current type message sending and waiting mode and asynchronous communication which includes past type message sending and future type message sending are discussed. They both can be deduced to past type message sending and waiting mode. Besides, OBCM provides mechanism of destination - responding in order to enhance presentation ability. The object's independent autonomous characteristic is very suitable for simulating distributed problem solving in OBCM. Finally the ideas of OBCM are illustrated by a sample of distributed problem solving.

**Key words:** concurrency; communication strategy; object - oriented programming; distributed problem solving

## 0 引言

分布式问题求解是人工智能进一步发展的必然途径,对于它的研究引起了人们很大的兴趣<sup>[1]</sup>。面向对象程序设计的兴起,又提供了一种研究方法。

对象与动作有点类似,每个对象执行的动作实际是进程的计算。对象带有消息,当它接收消息后立即执行指定的处理,然后再等待后继消息的到来<sup>[2]</sup>。各个对象可以并发执行,而每个对象的执行则通过其它对象将消息发送给它完成。在对象系统中,消息传递是唯一的计算机制。由于消息传递时控制并不返回,因此对象系统具有大量有待开发的固有并行性<sup>[3]</sup>。

利用对象的这些特点,文中提出了一种基于对象的并发模型(OBCM, Object - oriented Concurrence

Model)。

## 1 OBCM模型

OBCM提供了同步和异步两种通讯方式,异步通讯方式又分为过去型消息传递和将来型消息传递,而同步通讯方式则包括现在型消息传递和等待方式。实际上,现在型消息传递及将来型消息传递均可通过过去型消息传递及等待方式实现(详见1.4)。

### 1.1 异步通讯

#### 1.1.1 将来型

将来型消息传递含义是:当对象O发送消息M(将来型)给目标对象T时,如果O并不急需该消息实施的结果,那么,在M发出后,O就不必等待T将结果返回来才能继续下面的计算。当以后的某一时刻需要该结果时,只要检查一下上述将来型消息传递中事先专门设置的用于存放执行结果的某个对象(又称为将来对象)即可。系统运行过程中,可以通过测试它的值

收稿日期:2008-03-10

基金项目:深圳市科技计划项目(07KJcd141)

作者简介:周光明(1964-),男,副教授,硕士,研究方向为人工智能、分布式系统。

来了解将来型消息的执行结果。OBCM 中将来型消息用  $[T \leq M\$x]$  或  $[T < \leq M\$x]$  表示。其中,  $x$  为将来对象,  $T$  为消息接收对象。

### 1.1.2 过去型

OBCM 的过去型消息传递是指当对象  $O$  将过去型消息  $M$  发送出去后就可立即执行紧跟其后的动作序列。过去型消息  $M$  的发送可表示为:

$[T \leq M]$  或  $[T < \leq M]$

## 1.2 同步通讯

### 1.2.1 现在型

现在型消息传递是指:当对象  $O$  发送消息  $M$  给对象  $T$  时,  $O$  不仅要等待接收到  $M$ , 而且要到  $T$  将消息实施结果返回后才能继续执行后继计算。表面上看, 现在型消息传递与常规的函数或过程的语义类似, 但两者的差别是明显的。因为, 后者涉及到控制的返回。现在型消息  $M$  的发送可表示为:

$[T \leq = M]$  或  $[T < \leq = M]$

### 1.2.2 等待方式

为了描述激活方式的对象在其执行过程中可能需要停止正在进行的活动而等待某种模式的消息到来, 引入了等待方式。此时, 只有当所等待的模式的消息发来后, 该对象才能重新被激活。对象转入等待方式是在激活方式的执行过程中遇到了如下结构引起的:

```
(wait
  (= > < 消息模式 1 > where < 限定 1 > < 动作序列 1 >)
  ...
  (= > < 消息模式 n > where < 限定 n > < 动作序列 n >))
```

当对象接收到上述结构中 < 模式  $i$  > 匹配的消息而且满足 < 限定  $i$  > 时, 就立即执行相应的 < 动作序列  $i$  >; 否则, 一直等到与 < 模式  $i$  > 匹配的消息出现为止。

## 1.3 OBCM 中的其它机制

OBCM 为了增强其表达能力, 提供了响应目的地机制。即: 当某个对象将消息  $M$  发送给目标对象  $T$  请求完成处理时, 如果源对象  $O$  希望  $T$  将  $M$  执行后的结果送到某个指定的地点  $C$ , 那么只要在  $M$  发送时附带  $C$  的名字即可。此外, 还假定: OBCM 中的消息传递时发送者对象的名字是一同被传送给目标对象的。显然, 这种对象访问权的控制是十分有用的。因此, 接收者对象在收到对方发来的消息时就可确定出发送方的名字并可依此确定能否接收该消息。为了对对象系统中的对象实行互斥控制, OBCM 还提供互斥控制语句来控制其它对象对该对象中的某个操作程序的不同

实施<sup>[4]</sup>。

OBCM 中, 发送者对象的名字可用如下符号访问: &sender 表示包含它的那个最内层的消息的发出者对象。另外, 还引进了伪变量  $Me$ , 它指称定义中包含有“ $Me$ ”的最内层的那个对象, 而响应目的地则用“< @ < 响应目的地 > ”来描述。因此, 如下结构片段

$(= > < 模式 > \dots! < 表达式 >)$  可等价地表示为:

$(= > < 模式 > @ < 响应目的地 > \dots[ < 响应目的地 > : = < 表达式 > ])$

当消息  $M$  以过去型传递给目标对象  $T$  时, 如果指定响应目的地为  $R$ , 则可显式表示为:

$[T \leq < 消息 > @R]$

但是, 对现在型及将来型消息传递来说, 响应目的地的指定是隐式的(详见 1.4), 不能显式规定。

## 1.4 最小计算模型

异步和同步通讯方式中的现在型消息传递和将来型消息传递是可以通过过去型消息传递及等待方式实现的。

### 1.4.1 归约现在型

假设对象  $A$  的某个操作程序(method)中包含有现在型消息  $M$  并将它发送给目标对象  $T$ ,  $T$  将响应结果或应答信息返回给  $A$ 。

对象  $A$  及  $T$  中可定义消息如下(这里, < 模式 2 > 可与  $M$  匹配):

$(= > < 模式 1 > \dots[T \leq = M] \dots)$  及  
 $(= > < 模式 2 > @R \dots[R \leq < 表达式 >] \dots)$

其中, 后一表达也可表示为:

$(= > < 模式 2 > \dots! < 表达式 > \dots)$

现在, 定义一个新对象(New-object), 它将接收到的所有消息全部传递给  $A$ 。此外, 还引入一个等待结构, 它只接收由“New-object”发来的消息。那么,  $A$  就可以去掉现在型消息传递, 重新描述为(下面的 temporary 为 method 内的局部变量宣称):

```
[object A
  (script
    ...
    (= > < 模式 1 > (temporary [New-object: =
      [object (script (= > any[A < = any]]))]))
    ...
    [T < = M@New-object]
    (wait(= > value where(= &sender New-object)
      ...value...))...))]
```

这样, 一旦  $A$  在将消息  $M$  以过去型传递(而且响应目的地为“New-object”)给  $T$  时, 就会立即处于等

待方式,期待“New-object”发来消息“value”(假定:响应目的地对象“New-object”中一旦装有消息实施结果,就会向自身发出“value”消息)。

#### 1.4.2 归约将来型

假设对象 A 的某个操作程序中含有将来型消息 M 传递给目标对象 T,该消息传递中指明 x 为将来对象,那么,可先令 x 为由创建将来对象(CreateFuture-Object)所创建的对象。

即 A 的定义可由:

```
[object A
  (state... /* A 的状态变量宣称 */
  (future x...) /* 将来变量宣称 */
  (script
  ...
  (= > < 模式 >
  ...[T <= M $ x]...
  ... (ready ? x) ... (next - value x) ... (all - values x)
  ...)))]
```

改为没有将来型消息传递的结构如下:

```
[object A
  (state ... [x: = [CreateFutureObject <= [: new
  Me]]]...) /* Me 指称 A, new 为创建消息 */
  ...
  (= > < 模式 > ... [T <= M @ x] ... [x <= = [:
  ready?]] ...
  [x <= = [: next - value]] ... [x <= = [: all - val-
  ues]] ...)))]
```

因此,将来型消息传递已被带响应目的地为 x 的过去型消息传递所取代。将来对象的行为与一个队列对象相似,它接收四种消息:[empty?],[enqueue...],[dequeue]及[all-elements]。

这里,“CreateFutureObject”对象的详细定义略去。当它接收到消息[:new<对象>]后,立即回送一个能接收[:ready?],[next-value],[all-values]及 returned-value 四种消息的对象 O。其中,只有当上述这四种消息的发出者是<对象>时才能被 O 接收。

## 2 实例

这里以分布式问题求解的一个具体实例来阐明 OBCM 的思想。

假定:要求一个决策者在规定时限内领导一个项目小组解决一个难题。项目小组由一负责人和几个组员组成,每个小组成员都有自己的问题求解策略,他们接到了各自的任务后,进行独立求解。当得出求解结果后,立即向项目负责人报告。如果有几个小组成员

(包括项目负责人)同时解决了该难题,则项目负责人就从中选取最佳答案向决策者报告。同时通知小组成员停止求解。如果规定时间内没人得出解,项目负责人就请求决策者延长时限。当延长后的时限内仍无人解决该难题,项目负责人就向决策者报告失败。这个问题可用 OBCM 表示为:

```
[object ProjectLeader
  (state [team-members:=nil][bestSolution:=nil]
  [time-keeper:= [Create Alarm Clock <= = [:
  new Me]])]
  (future Solutions)
  (script
  (= > [:add-a-team-member M]
  [team-members:= (cons M team-members)])
  (= > [:solve SPEC :by TIME] /* SPEC 为问题
  描述,TIME 为规定时限 */
  (temporary[mySolution:=nil] /* method 内的局
  部变量 */
  [time-keeper<= [:wake-me-at TIME]]
  [team-members<= [:solve SPEC] $ Solutions]/
  * 将来型广播发送 */
  (while (and (not (ready? Solutions)) (null mySolu-
  tion)))
  do...try to solve the problem by his own strategy
  and store his solution in mySolution...
  (atomic
  [bestSolution:= (choose - bestSolution (all - values
  Solutions))])
  [Manager<= [:found best Solution]]
  [team-members<= [:stop-your-task]])
  (= > > [:time-is-up] where(= &sender time-
  keeper)
  (temporary new-deadline)
  (if (null bestSolution)
  then [new-deadline:= [Manager<= = [can-
  extend-deadline?]]]
  else [time-keeper<= [:wake-me-at new-
  deadline]])))]
```

这里,“atomic”所括起来的动作序列是不会被快车方式的消息中断或挂起的。对象 Project Leader 首先创建一个闹钟对象(以后某一时刻唤醒他)并赋值给 time-keeper。假定闹钟现在开始计数(其定义略),当项目负责人(ProjectLeader)接收到消息[:solve...]后,他先请闹钟对象在 TIME 时刻唤醒他,然后向小组成

(下转第 173 页)

最后通过添加 .latencyPlotPort、.packetsPlotPort、.packetsPlotPort2 三个事件端口把统计结果输出到 Plot 上显示。

③ 活动节点模型: AliveSensors 是一个组件, 用于仿真 WSNs 中活动节点随时间变化的规律。以此来作为衡量 WSNs 的生存时间的指标之一。

## 2 仿真实验

本节采用改进后的 WSNs 仿真框架对 LEACH 协议进行仿真实验。在  $50\text{m} \times 50\text{m}$  的区域内, 随机生成 100 个传感器对 3 个目标区域进行监测, 仿真时间为 3600 秒, 节点初始能量 0.25 J。

图 4 为仿真 LEACH 协议后得到仿真结果图。(a) 图仿真了 WSNs 中 100 个传感器节点的活动情况; (b) 图仿真 WSNs 全网的能量消耗信息; (c) 图仿真了 WSNs 中 Sink 节点收到的数据量, 其中上方的曲线表示簇首节点对收到的原始数据不经过数据融合直接发往 Sink 节点, 而下方的曲线表示簇首经过数据融合, 再把数据发往 Sink 节点; (d) 图仿真的是簇头到 Sink 节点之间的延迟时间。

## 3 结束语

文中对 J-Sim 下的 WSNs 仿真框架进行了改进使其可以支持能量仿真, 并在此框架上扩展实现了 LEACH 协议。实验表明改进后的 WSNs 仿真框架可以很好地仿真 WSNs 路由协议运行, 获取有价值的数

据。为以后对 WSNs 能量高效利用的路由协议研究提供了新的仿真平台。

### 参考文献:

- [1] Sobeih A, Chen W-P, Hou J C, et al. J-sim: a simulation environment for wireless sensor networks[C]//Proceedings of the 38th IEEE Annual Simulation Symposium (ANSS'05). [s.l.]: IEEE Press, 2005: 175-187.
- [2] Heinzelman W, Chandrakasan A, Balakrishnan H. Energy-efficient communication protocol for wireless microsensor networks[C]//Proceedings of the 33th Hawaii International Conference on System Science. [s.l.]: [s.n.], 2000: 1-10.
- [3] Tyan H-Y, Sobeih A, Hou J C. Towards composable and extensible network simulation[C]//Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05). Washington: Washington IEEE Computer Society, 2005.
- [4] Tyan H-Y. Design, realization and evaluation of a component-based compositional software architecture for network simulation[D]. USA: The Ohio State University, 2002.
- [5] Project V, Fall K, Varadhan K, et al. The ns manual[EB/OL]. [2007-01-05]. <http://www.isi.edu/nsnam/ns/doc/index.html>.
- [6] Park S, Savvides A, Srivastava M B. SensorSim: A simulation framework for sensor networks[C]//The 3rd ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM). Boston, Massachusetts, USA: [s.n.], 2000: 22-27.

(上接第 149 页)

员分派发送消息[:solve SPEC] \$ Solutions](将来型发送)。如果某个成员求解出该难题, 他便将解答发送到将来对象 Solutions 中。而项目负责人在求解过程中, 则周期性地测试该将来对象(这里略去了闹钟对象的创建者(Create Alarm Clock)及决策者、小组成员等对象的详细定义)。

## 3 结束语

通过对 OBCM 的阐述, 可以看到, 基于对象的计算模型具有较高的并发性。过去型消息传递、将来型消息传递、现在型消息传递以及响应目的地机制和等待方式一起为并行系统的描述提供了较强的手段。此外, 在 OBCM 中, 对象的独立自治特点对模拟分布式问题求解尤为适用。笔者认为, 对 OBCM 的进一步探索包括:

(1) 并发对象间的通讯、同步及互斥、访问控制。

(2) 并发对象的调度算法的设计与实现<sup>[5]</sup>。

(3) 并发对象的粒度大小研究及其在分布式问题求解中的应用。

### 参考文献:

- [1] 周光明. 分布式问题求解评述[J]. 高性能计算技术, 2003(6): 9-13.
- [2] Yonezawa A. Object-oriented concurrent programming in ABCL/1[J]. ACM, 1986, 21(11): 258-268.
- [3] 崔文静, 贾智平. 基于网络的分布式实时数据库系统的并发控制[J]. 计算机应用, 2004, 24(2): 132-134.
- [4] 陈庆章, 洪宁, 胡同森, 等. 基于 Web 的知识分布式学习系统的设计和实现[J]. 计算机工程与应用, 2001, 37(18): 80-82.
- [5] Ishiwakawa K. A concurrent object-oriented knowledge representation language orient84/K: Its features and implementation[J]. ACM, 1986, 21(11): 232-241.