基于FLAME的通用ABMS模型表示方法改进

严一实 李 波
(西安交通大学电子与信息工程学院 西安 710049)

摘 要 模型表示方法是Agent建模过程中需要解决的最重要问题之一。通用ABMS模型表示方法的研究能够降低Agent建模工具的使用门槛,从而使其在跨学科领域的研究中发挥巨大作用。FLAME作为现有平台中的佼佼者,不仅完成度高,而且在模型表示、代码生成及可视化显示等方面具有突出的优势。在这样的前提下,选取FLAME平台与XML作为通用模型表示方法的基础,在继承其简洁、完备、并行化等优点的基础之上,通过挖掘其潜在的技术弱点与改进空间,提出了针对性的解决方案。最后,通过一个Agent模型实例验证了改进方案的可行性。

关键词 模型表示方法,ABMS,FLAME,XML
中图法分类号 TP391.9 **文献标识码** A

Improvement of General ABMS Model Representation Based on FLAME

YAN Yi-shi LI Bo
(School of Electronic and Information Engineering, Xi'an Jiaotong University, Xi'an 710049, China)

Abstract Model representation is one of the most important issues during Agent-based modeling. Research on general ABMS model representation can reduce the threshold of using ABMS tools, which plays a significant role in the interdisciplinary research. As the leader of existing ABMS platforms, FLAME is not only with a high completeness, but also outstanding advantages in representation, code generation, visualization, etc. In this context, FLAME and XML were selected as the basis of general model representation. With inheriting their advantages of conciseness, completeness and parallelization, a deep research was achieved to find out the potential weakness and improvement possibilities, with targeted solutions as well. Finally, an Agent model example was tested and verified for the feasibility of solutions.

Keywords Model representation, ABMS, FLAME, XML

1 ABMS建模方法综述

基于主体的建模与仿真(Agent-Based Modeling and Simulation,ABMS)是用于对一组独立或集合形式的自治主体(Agent)的行为进行仿真,并从宏观的角度对它们的行为进行观察和评估的一种模型计算方法[1]。Agent受到若干规则的约束,并在这些规则的基础上与系统进行自发的互动。大量Agent在系统中反复地演化,从而使系统得以重现或预测一些现实中的复杂现象。这是一种从较低级的微观层面设计问题,然后从较高级的宏观层面得出结论的建模与仿真方法。ABMS与多主体系统(Multi-Agent System,MAS)的主要区别在于,前者从一个特定的生态环境中观察并做出解释,而后者则侧重于设计Agent的形式来解决一些实际的复杂问题。在ABMS的一部分应用场合中,可以将其视为一种具有高级智能行为的元胞自动机,两者的研究意图与所采取的方法有异曲同工之妙。

现有的Agent建模与表示方法大体上分为如下几种类别。从系统的语义上可分为面向过程、面向对象、基于有限状态机等,语法上可分为基于用户代码、基于流程图、基于建模语言(如UML)等。在这些方法中,描述模型的自由度依次减弱,但相应地可扩展性与复杂度则依次增强。在早期的ABMS模型中,用户执行代码本身就是模型的描述;而随着方法的逐渐演化,模型的结构渐渐从具体的实现中独立出来,并作为单独的部分存放及处理。存储的格式也是多种多样的,既有使用自行约定的描述语言,也有使用通用标记语言进行表示的方法,譬如下文提到的可扩展标记语言XML。

通用的ABMS模型表示方法一直是Agent建模领域的研究热点。FLAME作为一个优秀的ABMS平台,在简洁、完备、并行化等方面具有比其它平台更高的完成度,因而适宜作为通用表示方法的基础。我们在深入理解FLAME建模原理的基础上,从数学模型、并行性挖掘及语法表示等多个方面提出了针对FLAME的数个改进方案,相比于原有方法具有更强的表达能力、更好的执行效率与更佳的表示完备性。

2 XML及XML Schema语言综述

可扩展标记语言(Extensible Markup Language,XML)是一种用于格式化存储及传输数据的标记语言。XML的标签具有自我描述性,对于用户和计算机而言都具有良好的可读性。FLAME的模型表示方法中应用了XML作为其表达与存储规范,不仅维持了其可读性的优点,还使得模型具有良好的可扩展性。

XML Schema是用于定义XML文档的合法构建模块的

严一实(1991—),男,硕士生,主要研究方向为基于Agent的建模和仿真,E-mail:yan.yishi@hotmail.com;李 波(1980—),男,硕士,副教授,主要研究方向为Web工程及语义式Web服务发现、时空定性理论及时空知识表示与推理、基于Agent的建模和仿真等。

记法规范，它是在 XML 环境下的文档类型定义（Document Type Definition，DTD）替代者，通常以 XSD（XML Schema Definition）文件的形式被引用。在 FLAME 所使用的 XML Schema[2] 中，模型文件的每一个标签的语法语义都进行了详细的约定。这些约定使得 FLAME 的模型表述充分清晰，同时也将自身的模型表示能力限定在了相应的范围之内。

3 FLAME 建模工具及其模型表示方法综述

弹性的大规模主体建模环境（Flexible Large-scale Agent Model Environment，FLAME）是一组用于在高性能计算机群上创建模型及运行仿真的 ABMS 工具。该工具通过由 XML Schema 描述的文法创建模型并存储为 XML 文件。这一设计在 ABMS 的模型表达方法中具有举足轻重的地位，XML 越来越多地成为 Agent 建模工具首选的表达方式，并成为通用 ABMS 模型表示方法的基础。

FLAME 使用 CSXM（Communicating Stream X-Machine），一种由 FLAME 定义的扩展有限状态机，来描述模型中的主体行为[3]。每个 Agent 都由一组状态（S）与状态间的转移函数（F）所构成的非循环状态机表示。转移函数可以对所属 Agent 的存储区域进行读写，也可以通过“消息板”机制与其它 Agent 进行通信。模型通过 Agent 不断的状态转移来推进运行和演化。当模型中的所有 Agent 都完成了某个时间点的状态转移，则称为模型完成了一个迭代周期的运行。

FLAME 的 XSD 架构文档是整个工具用于进行模型表示的核心特性，5 个全局简单类型、7 个全局复杂类型、4 个全局模型组及 13 个全局元素构成了 FLAME 的模型表示空间，下面分别进行简要说明。

3.1 全局简单类型

全局简单类型包含了用于描述. c 源文件路径字符串的 CSourceFilename、用于描述多种双目运算符的 opString、用于描述集合排序升降性的 sortOrderString、用于描述布尔型的 trueFalseString，以及用于描述. xml 模型文件路径字符串的 XMLFilename。这些简单类型都是为了简化描述元素字段或复杂类型定义而引进的。

3.2 全局复杂类型

全局复杂类型包含了用于描述 Agent 内存变量的 agentMemoryVariableType 类型、用于描述双目算符操作数的 conditionSegmentType、用于描述时间触发条件的 conditionTimeType、将前两者（双目运算符条件、时间触发条件）整合定义的 conditionType、用于支持自定义结构类型的 datatypeType、用于判断 Agent 对转移函数的响应条件的 filterSegmentType，以及用于在多种场合下保存变量的 variableType。这些复杂类型或是为了约束元素字段的语法语义，或是为了以嵌套或递归的方式定义一个多层次的条件表达式而引进。

3.3 全局模型组

全局模型组包含了用于将时间触发条件与表达式触发条件整合的 conditionChoiceGroup、用于约束触发条件表达式形式的 conditionParamGroup、用于约束转移函数响应条件表达式的 filterParamGroup，以及在前者基础上增加了空间范围响应条件 filterParamGroupNew。这些模型组都是为了在模型中以一种简单表述来实现复杂行为的模版。其中调用了一些复杂类型的定义，其自身也被复杂类型定义所调用，从而形成了具有可扩展性的递归型定义。

3.4 全局元素

该部分是 FLAME 模型表达系统的核心框架。

xmodel 是整个 XML 模型文件的根元素，其子节点包括 models、environment、agents、messages 及模型的相关信息等内容。

models 是一组 model 元素的列表，每个 model 元素都是模型组中的一个子模型。FLAME 允许将一个较大规模的模型拆分成若干个子模型，单独设计、存储，并且由一个 enabled 字段控制其是否参加模型的运行过程。

environment 是将整个模型中各个 Agent 的共用内容集中存储及表示的元素，其中的 constants 元素用于存储配置模型的常量，functionFiles 元素用于存储转移函数的. c 源文件 URL，dataTypes 元素用于存储用户定义的数据类型，timeUnits 元素则用于存储用户定义的时间单位与运行迭代次数间的对应规则。

agents 是一组 xagent 元素的列表，每个 xagent 元素都代表一个 Agent 个体。FLAME 允许每一个 Agent 个体拥有自己独立的名字、描述、变量存储区域及转移函数列表。转移函数列表是由一组 function 元素组成的，每一个 function 元素对应当前 Agent 的一种状态转移方式，其字段中记录着当前状态、目标状态、转移条件、输入消息、输出消息等信息，同时还允许对 Agent 的输入消息响应进行随机化、筛选和排序。

messages 是一组 message 元素的列表，其中记录的是在 Agent 的状态转移过程中用到的全部消息。每个消息记录在一个 message 元素中，并通过其中的 name 字段和状态转移函数中的各个消息进行绑定。

综上所述，FLAME 模型编辑器通过清晰的层次设计及结构划分，使得模型的各个设计部分都具有良好的可读性及可扩展性；许多字段中都提供了完善和详细的功能设置，这允许模型的设计者以较大的自由度来设计一个基于 Agent 的复杂模型。

4 FLAME 模型表示方法的评价及改进

如前所述，FLAME 的模型表示方法是一个结构清晰、描述能力强、具有较高设计自由度的解决方案。该方案在许多大型项目中得到了应用与验证，如 EURACE 经济学模型、SUMO2 细胞结构模型等[4]。此外，FLAME 平台本身也是一个成熟的 ABMS 解决方案，它支持大规模的并行计算仿真，具有解决复杂 Agent 建模问题的能力，同时提供了图形用户界面进行模型设计。然而，FLAME 模型表示方法仍在多个方面具有改进空间。首先，FLAME 所使用的状态机模型较为简单，不能完全地反映出 Agent 模型的智能特性；其次，FLAME 在并行代码生成方面，未能通过数据依赖性分析挖掘出更多潜在并行性，同时数据颗粒度过大也影响着并行效率；此外，通信机制及语法表示上的不足之处也有改进空间。本文针对状态机模型、数据依赖性以及语法表示不足这 3 个方面提出了改进方案，对于数据颗粒度与通信机制的问题则暂时还没有好的解决思路。下面将分别对几个改进方案进行阐述。

4.1 CSXM 状态机的改进

在 FLAME 中，所有的 Agent 都被描述为一个 CSXM，其

实质是具有通信功能的扩展有限状态机(Enhanced Finite State Machine,EFSM)。对于一个行为简单的系统而言,EFSM是能够满足其描述能力的。然而在具有复杂与智能行为的Agent模型中,EFSM在多个方面有着关键性的不足。其一,EFSM的状态转换仅取决于输入与变迁条件,而与Agent停留在当前状态的保持时间无关。这使得EFSM不得不将状态的每一个保持时间都视为一个独立的状态,从而导致状态空间的急剧膨胀。其二,Agent会因收到外部事件而发生变化,也会在没有收到外部事件时自发地进化,这正是Agent智能性的体现。然而EFSM无法区分二者,在Agent自发进化的过程中仍会无意义地遍历输入事件集。这不仅是Agent建模语义上的缺陷,而且在仿真规模变大的情况下,将成为不可忽视的效率损失。

有鉴于此,提出如下的解决方案。在保留FLAME现有通信机制"消息板"的前提下,将CSXM扩展为具有原子DEVS特性的eCSXM。

离散事件系统规范(Discrete Event System Specifications,DEVS)是由美国学者B. P. Zeigler提出的一种形式化描述体系[8]。其最大的特点是将系统中的每个模块都视为相互独立的组件,每个模块的内部行为都是自治的,且模块之间通过输入-输出接口的方式进行耦合。这样的特性恰好与FLAME所描述的Agent间的关系完全一致。因而将DEVS的原子模型部分移植到Agent的CSXM中,能够保持Agent的现有的形态基本不变,同时扩充其内部状态机的描述能力。

对于DEVS的另一部分——DEVS耦合模型,由于它所对应的是FLAME中的通信机制"消息板",而"消息板"是FLAME模型解析器自动生成并行代码的关键技术,在没有充足的理由下盲目移植将会破坏整个系统的稳定性,因此这里没有采用其方案。

DEVS原子模型是由下面的七元组所定义的:

$DEVS_{Atomic}=\langle X,Y,S,ta,\delta_{int},\delta_{ext},\lambda\rangle$

X 是输入事件集合;

Y 是输出事件集合;

S 是系统状态集合;

$ta:S\rightarrow R^{+}_{0,\infty}$ 是时间推进函数,$ta(s)$表示在没有外界输入的情况下状态 s 保持的时间,$ta(s)=0$ 表示瞬态,$ta(s)=+\infty$ 表示稳态,$Q=\{(s,e)\mid s\in S,0\leqslant e\leqslant ta(s)\}$ 为包含了驻留时间计数 e 的系统总状态集合;

$\delta_{int}:S\rightarrow S$ 是内部转移函数,在 s 状态保持 $ta(s)$ 的情况下将状态转移为 $\delta_{int}(s)$,并将 Q 中的计数 e 重置;

$\delta_{ext}:Q\times X\rightarrow S$ 是外部转移函数,在收到外部输入 x 的情况下将状态转移为 $\delta_{ext}(s,e,x)$,并将 Q 中的计数 e 重置;

$\lambda:S\rightarrow Y$ 是输出函数,在系统离开状态 s 时产生输出 $\lambda(s)$。

将FLAME原有的状态机扩充为上述具有原子DEVS特性的eCSXM后,用于描述Agent行为的模型文件也将发生相应变化,如下述XML片段所示:

```
〈agents〉
  〈agentClass〉
    〈name〉person〈/name〉
    〈description〉…〈/description〉
    〈memory〉…〈/memory〉
    〈initFunction〉…〈/initFunction〉
    〈states〉
      〈state〉
        〈name〉A〈/name〉
        〈maintain〉+inf〈/maintain〉
      〈/state〉
      〈state〉
        〈name〉B〈/name〉
        〈maintain〉5〈/maintain〉
      〈/state〉
    〈/states〉
    〈functions〉
      〈function〉
        〈name〉transfer〈/name〉
        〈description〉…〈/description〉
        〈currentState〉A〈/currentState〉
        〈currentStateCount〉10〈/currentStateCount〉
        〈nextState〉B〈/nextState〉
        〈condition〉…〈/condition〉
        〈inputs〉…〈/inputs〉
        〈outputs〉…〈/outputs〉
      〈/function〉
      〈function〉
        〈name〉selfupdate〈/name〉
        〈description〉…〈/description〉
        〈currentState〉B〈/currentState〉
        〈currentStateCount〉5〈/currentStateCount〉
        〈nextState〉A〈/nextState〉
        〈outputs〉…〈/outputs〉
      〈/function〉
    〈/functions〉
  〈/agentClass〉
〈/agents〉
```

该片段定义了一个类型名为person的Agent,它具有两个状态,状态A是一个稳态,状态B的维持时间为5;声明了两个状态转移函数,transfer在状态A维持10个周期且满足条件时转换到B(当currentStateCount标签省略时,任意周期的A都可以发生转换),selfupdate在状态B维持5个周期时转换到A(currentStateCount等于maintain说明该函数是 δ_{int},因而没有condition和inputs)。

通过上述分析与例子可以看出,$ta(s)$与 Q 的出现解决了EFSM无法描述状态驻留的问题,而状态转移函数细化为 σ_{int} 与 σ_{ext} 两个部分则进一步完善了Agent演化行为的语义,同时提升了模型的执行效率。

4.2 批处理能力的改进

在FLAME中,被创建的各个Agent间不具有类别的概念,每一个Agent都被独立地描述及创建。这对于一些Agent行为简单但数量庞大的仿真问题而言,其模型的表述中增加了许多重复字段。这既不利于程序的执行效率,也不利于用户阅读理解模型。

针对这种问题,提出下面的改进方案。在FLAME原有的agents字段中,使用以类别和数量描述Agent的agentClass元素,在其中定义初始化的数量及其它相关信息。如下述XML片段所示:

```
〈agents〉
  〈agentClass〉
    〈name〉person〈/name〉
```

```
    〈description〉…〈/description〉
    〈number〉100〈/number〉
    〈startID〉1〈/startID〉
    〈memory〉…〈/memory〉
    〈initFunction〉…〈/initFunction〉
    〈functions〉…〈/functions〉
    〈exception〉
      〈xagent〉
        〈ID〉10〈/ID〉
        〈memory〉…〈/memory〉
        〈initFunction〉…〈/initFunction〉
        〈functions〉…〈/functions〉
      〈/xagent〉
    〈/exception〉
  〈/agentClass〉
〈/agents〉
```

该片段定义了一组数量为 100 的 person 对象，第一个 person 对象的 ID 为 1，其余对象的 ID 默认依次递增；对于个别需要例外处理的 Agent(如代码片段中 ID 为 10 的 Agent)，将其在 exception 字段中标注出来，使它享有单独的 memory、initFunction 及 functions 字段。这种方式弥补了原有 FLAME 方案中存在大量冗余的描述信息的缺陷。批量创建过程充分简洁，同时又不失去处理某些特例 Agent 的能力。

4.3 模型并行化能力的改进

FLAME 是一个具有大规模问题处理能力的建模平台，其模型解析器能够将用户描述的模型文件转换为能够在多机运行环境中执行的并行代码。然而，在其模型表示的约定中，任意一个 Agent 的状态转换函数都有权限访问全局所有的内存变量，其最小的数据单元是一个 Agent 实例，这种特性大大降低了数据的并行划分粒度。解析器由于无法事先知晓每一个转移函数的内存访问请求，从而无法假设他们之间的依赖关系，只能通过用户给出的状态图来分析。在这样的前提下，解析出的模型执行代码的并行度将非常低[5]。针对这种问题，从以下两个方面提出解决方案。

第一，由于 FLAME 允许 Agent 的状态转换函数读写全局中的任意变量，因此会不可避免地迫使执行代码顺序地处理每个 Agent 的读写请求，从而避免临界区冲突。如果在 agentClass 的定义中增加一个 group 字段，用于描述具有分组特性的 Agent，那么对于具有 group 字段约束的 Agent，其转移函数的访问权限将受到限制，即无法访问分组 Agent 以外的内存区域。如下述 XML 片段所示：

```
〈agents〉
  〈agentClass〉
    〈name〉person〈/name〉
    〈description〉…〈/description〉
    〈number〉100〈/number〉
    〈startID〉1〈/startID〉
    〈groupID〉1〈/groupID〉
    〈memory〉…〈/memory〉
    〈initFunction〉…〈/initFunction〉
    〈functions〉…〈/functions〉
    〈exception〉…〈/exception〉
  〈/agentClass〉
  〈agentClass〉
    〈name〉person〈/name〉
    〈number〉50〈/number〉
    〈startID〉101〈/startID〉
    〈groupID〉2〈/groupID〉
    〈exception〉…〈/exception〉
  〈/agentClass〉
〈/agents〉
```

该片段首先定义了一组数量为 100 的 person 对象，其对象 ID 从 1 开始计数；然后又定义了一组数量为 50 的 person 对象，其对象 ID 从 101 开始计数，且该组 person 对象沿用之前 person 类定义中 description、memory、initFunction 及 functions 字段的值。

第二，与前面的思路类似，当一个 Agent 的转移行为具有“只读”的特性时，它就不会与其它任何访问产生临界区冲突。例如，一个 Agent 希望观察其周围的数个 Agent 的状态，并以此作为自己行为的决策。在这样的前提下，“观察”这一行为并不会对 Agent 的内存区域产生影响，此时这一行为就是只读的。如下述 XML 片段所示：

```
〈function〉
  〈name〉observe〈/name〉
  〈description〉…〈/description〉
  〈currentState〉A〈/currentState〉
  〈nextState〉B〈/nextState〉
〈readonly〉true〈/readonly〉
  〈condition〉…〈/condition〉
  〈inputs〉…〈/inputs〉
  〈outputs〉…〈/outputs〉
〈/function〉
```

该片段定义了一个名为 observe 的转移函数，在 Agent 由状态 A 转移到状态 B 时发生，并且将 readonly 字段声明为 true。这种声明表示该转移函数不会对内存区域进行写操作，仅使用消息机制与其它 Agent 或者自己进行通信。通过这种方式，可以减少模型并行运行时不必要的同步操作，从而提升执行效率。

上述两个并行化改进实际上都可以归结为一点，即借由用户显式声明的内存访问限制，来给予模型解析器更多的相关性信息，有效地减少同步点的数量；同时亦使得调度程序做出合理的任务划分，比只根据空间或聚类的划分方式更少使用组间通讯。事实上，如果用户能显式声明转移函数中的所有内存访问，解析器就能够将并行性最大化。然而，这样的改进是与“降低用户建模难度与复杂性”的初衷相违背的。所以考虑到执行效率与复杂性的权衡，上述两点改进都在后者基本不变的前提下，为大多数模型中可能出现的两种情景提供改进方案。

5 改进的 FLAME 模型表示方法示例及对比

考虑如下 Agent 模型。在平面直角坐标系 $S=\{(x,y)\mid x\in[-100,100], y\in[-100,100]\}$ 的 4 个象限中，每个象限有 100 个位置随机分布的 Agent，它们的初始状态随机确定。所有的 Agent 都具有这样的行为：在指定的象限中以 1 单位/时钟周期的速度随机向 4 个方向运动，且每 5 个时钟周期进行一次观察，如果相邻象限中所有 Agent 与它的平均距离不足 100，则向远离该象限的方向移动 5 个单位，反之则向靠近方向移动。迭代 300 个周期后观察现象(见图 1)。

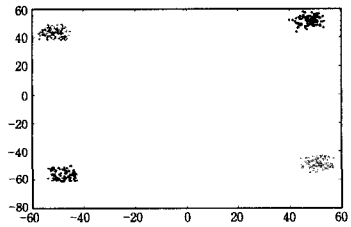

图 1　迭代 300 个周期后的预期现象

对于上述模型，使用改进的表示方法可以容易地写出下面的 XML 模型文件：

```
〈xmodel〉
  〈agents〉
    〈agentClass〉
      〈name〉dot〈/name〉
      〈description〉…〈/description〉
      〈number〉100〈/number〉
      〈startID〉1〈/startID〉
      〈groupID〉1〈/groupID〉
      〈memory〉…〈/memory〉
      〈initFunction〉…〈/initFunction〉
      〈states〉
        〈state〉
          〈name〉roaming〈/name〉
          〈maintain〉5〈/maintain〉
        〈/state〉
        〈state〉
          〈name〉observing〈/name〉
          〈maintain〉0〈/maintain〉
        〈/state〉
        〈state〉
          〈name〉moving〈/name〉
        〈/state〉
      〈/states〉
      〈functions〉
        〈function〉
          〈name〉roam_observe〈/name〉
          〈description〉…〈/description〉
          〈currentState〉roaming〈/currentState〉
          〈currentStateCount〉5〈/currentStateCount〉
          〈nextState〉observing〈/nextState〉
        〈/function〉
        〈function〉
          〈name〉observe_move〈/name〉
          〈description〉…〈/description〉
          〈currentState〉observing〈/currentState〉
          〈currentStateCount〉0〈/currentStateCount〉
          〈nextState〉moving〈/nextState〉
          〈readonly〉true〈/readonly〉
          〈outputs〉
            〈output〉
              〈messageName〉distance〈/messageName〉
            〈/output〉
          〈/outputs〉
        〈/function〉
        〈function〉
          〈name〉move_roam〈/name〉
          〈description〉…〈/description〉
          〈currentState〉moving〈/currentState〉
          〈nextState〉roaming〈/nextState〉
          〈inputs〉
            〈input〉
              〈messageName〉distance〈/messageName〉
            〈/input〉
          〈/inputs〉
        〈/function〉
      〈/functions〉
    〈/agentClass〉
    〈agentClass〉
      〈name〉dot〈/name〉
      〈number〉100〈/number〉
      〈startID〉101〈/startID〉
      〈groupID〉2〈/groupID〉
    〈/agentClass〉
    〈agentClass〉
      〈name〉dot〈/name〉
      〈number〉100〈/number〉
      〈startID〉201〈/startID〉
      〈groupID〉3〈/groupID〉
    〈/agentClass〉
    〈agentClass〉
      〈name〉dot〈/name〉
      〈number〉100〈/number〉
      〈startID〉301〈/startID〉
      〈groupID〉4〈/groupID〉
    〈/agentClass〉
  〈/agents〉
  〈messages〉
    〈message〉
      〈name〉distance〈/name〉
      〈variables〉…〈/variables〉
    〈/message〉
  〈/messages〉
〈/xmodel〉
```

若使用原有的 FLAME 表示方法，不仅模型文件的篇幅大到难以在文中呈现，而且无法显式地声明数据的依赖性，从而失去了执行效率上的诸多优化空间。

结束语　上述的解决方案从表达能力、执行效率及完备性等多个方面对 FLAME 原有的表示方法进行了补强，在基本维持了方案原貌的前提下，实现了对原来无法简洁或是精确描述的模型进行描述。除此以外，还提出了数据颗粒度与通信机制上的不足，作为日后进一步的改进方向。然而，与本文改进方案所配套的模型解析器尚未完成全部的开发工作，因而还无法对文中所陈述的并行效率提升进行定量分析，只能从理论上定性地说明其改进原理。未来会在后续相关部署完成时，推进方案的验证与改进工作。此外，FLAME 中的动态任务调度、线程安全及多层次并行等问题同样影响着模型表示方法的进化，这些内容将会作为未来的着眼点，有待进一步的研究。

参 考 文 献

[1] Wikipedia. Agent-based model[EB/OL]. http://en.wikipedia.org/wiki/Agent-based_model

[2] FLAME. XMML Schema[DB/OL]. http://www.flame.ac.uk/schema/xmml_v2.xsd

[3] FLAME. User Manual [EB/OL]. http://www.flame.ac.uk/docs/user_manual.html

[4] FLAME. Projects[EB/OL]. http://www.flame.ac.uk/projects
[5] Coakley S, Gheorghe M, Holcombe M, et al. Exploitation of High Performance Computing in the FLAME Agent-Based Simulation Framework[C]// IEEE Computer Society. 2012, 2012: 538-545
[6] Chin L S, Worth D J, Greenough C, et al. FLAME: an approach to the parallelisation of agent-based applications. RAL-TR-2012-013. [R]. Science and Technology Facilities Council, 2012
[7] Chin L S, Worth D J, Greenough C, et al. FLAME-II: a redesign of the Flexible Large-scale Agent-based Modelling Environment. RAL-TR-2012-019. [R]. Science and Technology Facilities Council, 2012
[8] Park S, Zeigler B P. Distributing Simulation Work Based on Component Activity: A New Approach to Partitioning Hierarchical DEVS Models [C]// IEEE Computer Society CLADE. 2003: 124-131
[9] 伍选,文中华,汪泉,等. 多 agent 规划领域中的观察信息约简[J]. 计算机科学,2014,41(6):176-179,192

(上接第 463 页)

本方差值在用户指定的可接受范围之内,则认为模型是运行时稳定的;否则,若样本方差过大则应检查模型的逻辑合理性、算法或修改模型的参数等,调整模型反复进行测试。

5.2 模型验证

两组数据序列如下(n=8):

模拟数据序列{X_1}:0.994570、0.989843、0.992003、0.996662、0.987338、0.97718、0.987935、0.996443;

实际数据序列{X_2}:0.995421、0.985820、0.995916、0.988551、0.986710、0.985244、0.978873、0.981318。

5.2.1 定性分析

(1)数据分布图

如图 4 所示,分布图中除了一个异常点以外,其他的数据偏差不大。

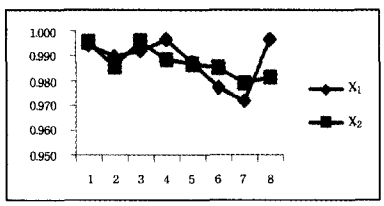

图 4 数据分布图

(2)箱线图

模型数据范围包括实际数据范围,且 model 的中位线高于 system 的第一分位线,仿真模型无效,如图 5 所示。这与总体结论相悖,原因是数据太少,箱线图要求数据至少在10—30 个或者更多,这样才能合理表现出统计特性。

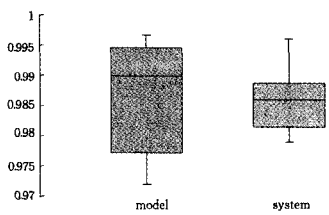

图 5 箱线图

(3)样本直方图

如图 6 所示,除了最后一组数据外,其他各组数据的差别不大,在 0.015 的范围内,从而得出数据吻合得较好。

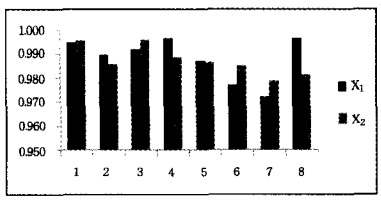

图 6 样本直方图

(4)样本特征值分析

从表 2 可看出两组数据吻合得较好。

表 2

	{X_1}	{X_2}
样本容量 n	8	8
最大值 X_{max}	0.996662	0.995916
最小值 X_{min}	0.971935	0.978873
样本均值 $\bar{X}$	0.988239	0.987232
样本标准差	0.008554	0.005648

5.2.2 定量分析——置信区间估计

$\overline{x_1}(n_1)$=0.988247

$\overline{x_2}(n_2)$=8.34E−5

S_1^2=0.987232

S_2^2=3.65E-5

ν=12.14,取 ν=12,查表得:$t_{\alpha/2,\nu}$=2.68099。

根据式(4)计算得置信度为 95% 的置信区间为(−0.00936,0.011394),分析置信区间可得出如下两点结论性意见:

a)由于该置信区间包含零,则可判断在 95%置信度下两组数据性能无差异。

b)假定该模型的可接受精度范围 ARA 为(−0.1,0.1),由于该置信区间满足 ARA 要求,则可判断该模型是有效的,且模型的可信度 Pa 在 95%以上。

结束语 本文提出了一个基于 Agent 仿真模型的校核与验证框架,包括具体的表面验证技术、灵敏度分析、模型校准与运行时验证,并从定性和定量两个角度进行运行时验证,适用于一般的 ABMS,其缺点是没有恰当的例证。基于 Agent 模型的校核与验证是 Agent 仿真开发中关键的环节,目前甚至在将来的一段时期内仍是 ABMS 领域的一大挑战。

参 考 文 献

[1] 宋承龄. 关于仿真模型验证[J]. 计算机仿真,2000,17(4):8-11
[2] 唐见兵,查亚兵,李革. 仿真 VV&A 研究综述[J]. 计算仿真,2006,23(11):82-98
[3] 廖守亿,陈坚,陆宏伟,等. 基于 Agent 的建模与仿真概述[J]. 计算机仿真,2008,25(12):1-7
[4] Sargent R G. Verification and validation and simulation models [C]// Proceedings of the 2010 Winter Simulation Conference. 2010: 166-183
[5] Janssen M A, Ostrom l. Empirically based, gent-based models [J]. Ecology and Society, 2006, 11(2)
[6] Ngo A, See L, A J. Heppenstall et al. Agent-based models of geographical systems[D]. Vietnam: Hanoi University of Agriculture, 2012