基于 MDA 的并行仿真可视化组件建模范式

姚益平 刘 刚
(国防科学技术大学计算机学院 长沙 410073)

摘 要 并行仿真是提高分析仿真运行效率的有效途径.当前国内外并行仿真由于缺乏高效统一的可视化组件建模范式的指导,往往需要采用从上到下的集中式开发模式,开发门槛高、效率低、二次建模困难,且模型间的耦合度大、协调难,模型与仿真平台紧密绑定,难以实现模型的分步开发、封装、快速组装及平台间模型的重用.作者结合组件建模及 MDA 技术的优势,针对当前广受关注的 PDEVS 规范所存在的不支持原子模型可视化建模及事件处理逻辑过于集中、无法挖掘处理逻辑内部并行性的缺点,提出了基于 MDA 的并行仿真可视化组件建模范式——EDEVS,并证明了该范式的描述能力不弱于 PDEVS;在此基础上,设计实现了基于 EDEVS 的可视化建模框架及 EDEVS 模型到 YHSUPE 并行仿真平台模型的代码转换插件,将组件化引入 YHSUPE,使其能够适应大规模并行仿真模型分步开发及快速组装的应用需求,验证了 EDEVS 范式的可行性.

关键词 并行离散事件仿真;组件建模;MDA;DEVS;范式
中图法分类号 TP311 **DOI 号**: 10.3724/SP.J.1016.2011.01488

MDA-Based Visual Component Formalism for Parallel Simulations

YAO Yi-Ping LIU Gang
(*Department of Computer Science, National University of Defense Technology, Changsha* 410073)

Abstract Parallel simulation is an effective way to enhance the running performance of analytic simulations. There are many parallel simulation platforms designed for analytic simulation, which have achieved great running performance. But the modeling paradigms of most of these platforms are difficult to program with and usually require a top-to-bottom developing mode. They are obstacles to develop large scale parallel simulations, because they fail to provide an easy way to facilitate model development, encapsulation and reuse, which is fundamental in developing large scale analytic simulations. To deal with this issue, this paper presents a MDA based Visual Component formalism, which can support hierarchical decomposition of large models and facilitate model reuse. It extends the DEVS (Discrete Event simulation specification) and proposes a component-based formalism, called EDEVS (Event-Scheduling Discrete Event simulation Specification) for the existing parallel simulation platforms. This paper also presents a visual editor to facilitate the development of EDEVS models, and provides a tool to map EDEVS model to YHSUPE model, validating EDEVS's feasibility.

Keywords parallel simulations; component-based modeling; MDA; DEVS; formalism

1 引 言

大规模生态环境仿真、计算系统生物学仿真、复杂环境下的大规模模拟、危机预测预警与处置决策等复杂系统分析仿真不断增长的计算需求,使得基于高性能计算机的并行仿真正成为这类仿真发展的重要趋势.

收稿日期:2010-02-02;最终修改稿收到日期:2011-04-28.本课题得到国家博士点基金(200899980004)和国家自然科学基金(60773019)资助.
姚益平,男,1963 年生,博士,教授,博士生导师,目前重点研究领域为高性能并行仿真支撑技术.E-mail: ypyao@nudt.edu.cn. **刘 刚**,男,1981 年生,博士研究生,研究方向为并行离散事件系统建模与仿真.E-mail: liugang@nudt.edu.cn; john.liu.fujian@gmail.com.

由于这类仿真往往包含大量的实体，实体内又包含多个模型，而且随着仿真应用的不断深入，其实体规模越来越大，实体间以及模型间存在错综复杂的交互[1]，且这些实体模型往往涉及多个领域的专业知识，需要不同领域的专家分别开发，特别是由于知识产权保护等原因，这些模型往往需要以封装好的执行码形式提交给使用方，这就使得模型的组件化开发及封装标准和组装技术变得尤为重要，目前的仿真组件化建模理论和方法要么没有考虑并行仿真事件驱动等特征，只适合于 HLA 等分布式交互仿真系统；要么过于复杂，难以使用，从而使得并行仿真往往需要采用从上到下的集中式开发模式，存在开发门槛高、开发效率低、二次建模困难，且模型间的耦合度大、协调难，模型与仿真平台紧密绑定、难以实现模型的分布高效开发、封装和快速组装及平台间模型的重用等问题[2-3]，从而使得其难以适应大规模并行仿真应用发展的需要.

因此，为满足大规模并行仿真模型分布开发、组件化组装的需求，提高应用的建模开发效率，亟待研究基于组件的并行仿真可视化建模理论与技术.

归纳起来，大规模并行仿真对建模理论和技术提出了以下要求：

(1) 能把大而复杂的模型递归地分解为小的简单的模型，分步建模. 大规模并行仿真中往往包含极其复杂的模型(例如环境模型、C4I 武器系统模型等)，且模型还往往涉及多个领域的专业知识，直接对其进行建模往往非常困难，应当能够将之递归地分解为涉及领域知识较单一的小模型，由地理上分布的各领域专家分别建模[4].

(2) 能够方便地利用已有模型，从小模型构造大模型，从下到上构建仿真应用. 对于大规模仿真中的复杂模型，每次都从头开始建模开发往往工作量巨大，且模型的正确性难以保证. 而利用已有的模型构造符合要求的大模型，并层次化地组装成仿真应用，则是快速、高效、灵活地构造仿真应用的最佳途径[5-7].

(3) 保证模型的自包含特性，即能独立封装、发布，而不依赖于其它模型. 大规模并行仿真中的不同模型往往需要不同领域的专家分别开发，为保证模型开发、调试的独立性及相对的保密性(很多模型涉及版权问题)，就必须要求模型具有自包含特性，使得各单位的模型能独立封装，并易于组合[8-9].

(4) 采用与仿真平台无关的可视化建模方式，并提供高效的代码转换机制. 目前国内外并行仿真平台正处于不断完善和发展之中，尚未形成统一的标准. 因而，有必要在一定程度上保持模型的平台无关特性，而由高效的代码转换机制将平台无关模型转换并封装成平台相关模型，使大规模并行仿真应用能够快捷地使用并行仿真领域的最新成果，推进自身发展[10].

基于组件的建模技术可有效降低组件模型间的耦合度，改善模型的重用性，有利于实现仿真模型开发与使用的分离，由领域专家设计组件化的仿真模型，应用开发人员使用已有的组件模型快速组装仿真应用，从而能够大大提高仿真应用的开发效率；而 MDA(Model Driven Architecture)[11] 技术能使模型与模型的运行平台相分离，从而有效地支持模型在运行平台间的重用.

本文立足大规模并行仿真对建模范式的要求，结合组件建模及 MDA 技术的优势，提出了一种基于 MDA 的并行仿真可视化组件建模范式 EDEVS(Event-scheduling Discrete EVent simulation Specification)[12]，并设计实现了基于 EDEVS 范式的可视化建模框架，为建模人员提供可视化的建模图元，使建模过程直观简便，充分支持快速、分步建模及模型的可视化组装. 在此基础上，本文实现了 EDEVS 模型到 YHSUPE(YinHe Simulation Utilities for Parallel Environment)[1] 仿真平台模型代码的映射，将组件化引入 YHSUPE，提高了其建模开发效率，验证了 EDEVS 的可行性.

本文第 2 节介绍国内外相关工作；第 3 节介绍 EDEVS 原子组件模型范式、耦合组件模型范式及建模实例，并证明该范式描述能力；第 4 节简要介绍 EDEVS 可视化建模环境 VisiCom；第 5 节对本文所做的工作进行总结.

2　相关工作

国内外对基于并行仿真的组件化建模技术的研究主要有 Flames、CCSE 和 JMASS 的组件化建模技术及 BOM(Base Object Model)和 DEVS(Discrete Event System Specifications)组件化建模规范. 美国 Ternion 公司的产品 Flames 是当前著名的仿真集成环境之一，应用于大型仿真. 它对装备进行分类，定义相应的组件化接口，并在此基础上对每类装备实现一个基本模型类，让模型设计者能继承这些模型类而派生出自己的模型. Flames 支持模型参数的可视化配置，但不支持原子模型的可视化建模，

其建模方式与其仿真平台相绑定，且不支持集群式并行仿真，限制了其仿真应用的规模[13]；美国联合仿真三大项目之一的 JSIMS 系统提供了一系列并行仿真支持工具，其中，公共组件仿真引擎 CCSE 为 JSIMS 的综合环境提供核心支撑服务，CCSE 借鉴了组件式建模思想，提供了实体和组件两种基类，供用户开发自己的组件模型，在实体里对组件进行组装，然后应用到 CCSE 的仿真框架中去. 但 CCSE 建模框架不支持可视化建模，且其公布订购式的模型解耦方式存在效率问题，其建模方式也与其仿真平台绑定，不支持平台间的模型重用[8]；JMASS 也是美国联合仿真三大项目之一，目标是促进组件级和平台级系统的模型重用和互操作，与 Flames 一样，JMASS 中的重用仅限于既定的几类模型，没有统一的建模规范，不支持可视化建模，不支持模型的递归分解，重用粒度过粗，也不支持平台间模型重用[14-15]；BOM 是由 SISO 提出的面向分布式仿真的组件化建模框架，但 BOM 定义的是概念模型及该模型到 HLA 规范的映射方式，不是一个完整的并行离散事件仿真建模规范[17-18]；DEVS 是亚利桑那州立大学 Zeigler 教授课题组在 20 世纪 70 年代末提出的并行离散事件系统规范，并于 2000 年初发展出其并行化版本 PDEVS(Parallel DEVS)，且基于此规范实现了 DEVSJAVA、COSMOS 等仿真平台. PDEVS 包括原子模型和耦合模型两个范式. 原子模型范式规定如何从无到有构造模型，而耦合模型范式规定如何利用小模型构造大模型. PDEVS 原子模型可以用一个八元组表示：

$$M=\langle X_m, Y_m, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta\rangle ,$$

其中：$X_m=\{(p,v)\mid p\in IPorts, v\in X_p\}$ 是输入端口与其参数值的二元组集合；

$Y_m=\{(p,v)\mid p\in OPorts, v\in Y_p\}$ 是输出端口与其参数值的二元组集合；

S 是状态变量的集合；

X_m^b 是输入包，可以包含 X_m 中的一个或多个元素，也可以为空；

Y_m^b 是输出包，可以包含 Y_m 中的一个或多个元素，也可以为空；

$\delta_{ext}: Q\times X_m^b\to S$ 是外部状态转换函数，用于响应外部事件；

$\delta_{int}: S\to S$ 是内部状态转换函数；

$\delta_{con}: Q\times X_m^b\to S$ 是冲突消解函数，当外部状态转换与内部状态转换同时发生时被调用；

$\lambda: S\to Y_m^b$ 是输出函数；

$ta: S\to R_{0,\infty}^+$ 是时间推进函数；

$Q=\{(s,e)\mid s\in S, 0\le e\le ta(s)\}$ 是全状态变量，包括状态变量集 S 与 e，e 记录从上次发生状态转换到现在的时间间隔.

任意时刻模型处于某个确定的状态 $s\in S$，如果没有外部输入出现，这个状态将持续 $ta(s)$ 时间，其中 $ta(s)$ 是一个非负实数. 当 $ta(s)$ 为 0 时，称 s 为一个暂态，此时模型不受外部输入的干扰，相反 $ta(s)$ 为 ∞ 时，称 s 为一被动态，只要没有外部输入出现，这个状态将一直持续下去，而 $ta(s)$ 为其它值时，在没有外部输入发生的情况下，经过 $ta(s)$ 时间之后，模型将进入由内部状态转换函数 δ_{int} 所决定的下一个状态 $s'=\delta_{int}(s)$，并产生输出，需要注意的是模型只在此种情况下产生输出；若在 $ta(s)$ 时间之内，有外部输入 $x\in X_m^b$ 产生，则模型将进入由外部状态转换函数 δ_{ext} 所决定的下一个状态 $s'=\delta_{ext}(s,e,x)$. 如果外部输入恰好出现在 $ta(s)$ 时刻，那么模型的下一个状态由冲突消解函数 δ_{con} 决定. 将已有的原子模型或耦合模型的输入输出端口连接起来就可以构成更大的耦合模型. 形式上，耦合模型可以表示成：

$$N=\langle X, Y, D, \{M_d\mid d\in D\}, EIC, EOC, IC\rangle ,$$

其中：X 是输入端口及其参数值的二元组集合；

Y 是输出端口及其参数值的二元组集合；

D 是构建耦合模型的子模型名称集合；

M_d 表示名称为 d 的子模型；

EIC 是该耦合模型输入端口与其子模型的输入端口的连接关系集合；

EOC 是子模型的输出端口与该耦合模型的输出端口的连接关系集合；

IC 是子模型之间的输入、输出端口连接关系集合.

模型的耦合是通过指定端口之间的连接关系来实现的，这样就可以由已有的模型层次化地构成更大的模型[19]. PDEVS 规范还规定了与建模范式相对应的仿真引擎实现规范. 但由于 PDEVS 不支持原子模型的可视化建模，模型开发难度大，且原子模型的事件处理逻辑过于集中，时间同步开销大，运行效率不理想，因此到目前为止未被业界广泛接受.

总之，目前国内外并行仿真组件建模理论与技术正处于研究发展之中，现有的理论与技术要么不支持可视化建模、时间同步开销大或者建模方式与仿真平台绑定，要么不支持复杂模型的递归分解、使用难度大，要么不支持并行仿真，难以满足大规模并行仿真应用在模型递归分解、分布式可视化开发、封

装和快速组装及平台间模型的重用等方面的需求，亟待研究高效的并行仿真组件建模规范及模型可视化建模技术，以促进复杂系统并行仿真应用的发展.

3　EDEVS 范式

EDEVS 范式在保证 PDEVS 范式的递归性、自包含特性及建模表达能力的基础上，针对 PDEVS 不支持原子模型可视化建模及事件处理逻辑过于集中、时间同步开销大的缺点，通过引入内部端口，将 PDEVS 原子模型的外部状态转换函数和内部状态转换函数划分为与端口相对应的端口处理函数，充分开发状态转换函数潜在的并行性，在不违反因果关系的前提下实现更细粒度的并行. EDEVS 包括原子模型与耦合模型两个范式，以下将分别对两个范式进行介绍.

3.1　EDEVS 原子模型

EDEVS 原子模型可以用一个十元组来表示：

$M=\langle X_m, Y_m, K_m, S, Init_{ie}, \delta, \delta_{con}, \lambda, f_{eid}, ea\rangle$，

其中：$X_m=\{(p,v)\mid p\in IPorts, v\in X_p\}$ 是输入端口及其参数值的二元关系集合；

$Y_m=\{(p,v)\mid p\in OPorts, v\in Y_p\}$ 是输出端口及其参数值的二元关系集合；

$InnerPorts=\{\text{"}cancelOP\text{"}\}\cup OtherPorts$ 为内部端口；

$K_m=\{(p,v)\mid p=\text{"}cancelOP\text{"}, v\in int \text{ or } p\in OtherPorts, v\in K_p\}$ 是内部端口及其参数值的二元关系集合，X_m、Y_m 和 K_m 也称为事件集；

$f_{eid}=(OPorts\cup OtherPorts)\rightarrow(int)$ 是一个 ID 生成函数，该函数为每个被调度的事件（目的端口“*cancelOP*”除外）生成一个全局唯一的标识，在所调度的事件未被执行之前，以此标识作为参数调度一个目的端口为“*cancelOP*”的事件，可将其撤销；

$Init_{ie}=\{(p,t)\mid p\in K_m, 0\leq t<\infty\}$ 是目的端口为内部端口的初始事件集，其中“(p,t)”代表 p 事件在 t 时刻执行（“执行”即执行其相应的端口处理函数）；

$\delta:(X_m\cup K_m)\times S\rightarrow S$ 是端口处理函数；

$\delta_{con}:(X_m\cup K_m)^b\times S\rightarrow S$ 是优先级排列函数，当同一组件中的多个事件被调度在同一时刻执行时，将按优先级先后顺序触发相应的端口处理函数；

$\lambda:(X_m\cup K_m)\times S\rightarrow(K_m\cup Y_m)^b$ 为输出函数；

$ea:(K_m\cup Y_m)\times S\rightarrow R^+_{0,\infty}$ 为时间推进函数，决定所调度的事件的执行时间；

$IPorts\cap InnerPorts=\varnothing$，$IPorts\cap OPorts=\varnothing$，$OPorts\cap InnerPorts=\varnothing$.

与 PDEVS 原子模型不同的是，EDEVS 原子模型有三类端口：输入端口、内部端口及输出端口，其中内部端口只对模型自身可见. 输入端口接收由其它模型触发的事件，而内部端口和输出端口则接收模型内部逻辑触发的事件，每个端口都有相应的处理函数来处理事件（事件处理函数为 δ 的子集）. 事件处理基本流程为：接收事件传入到本端口的数据，修改模型的状态变量，调度其它事件（其中，输出端口仅将接收到的事件转发到与该端口相连的其它模型的输入端口）. 就模型本身而言，称输入端口接收到的事件为**外部事件**，内部端口接收到的事件为**内部事件**；相应地，若模型内部逻辑触发了一个目的端口为内部端口的事件，则称**调度了一个内部事件**，若模型内部逻辑触发了一个目的端口为输出端口的事件则称**调度了一个外部事件**. 在任意时刻模型处于状态 s，如果有外部事件或内部事件发生，那么模型进入下一个由端口处理函数 δ 决定的下一个状态 $s'=\delta(s,x)$，而后调度一系列由输出函数 $\lambda(x,s')$ 所决定的内部事件或外部事件，如果这些事件没有被取消，那么经过由时间推进函数 ea 所确定的延时之后，将触发相应的端口处理函数[12].

3.2　EDEVS 耦合模型

EDEVS 耦合模型的定义与 PDEVS 的耦合模型定义类似，皆通过指定端口连接关系来从原子模型层次化地构造更大的模型[12].

3.3　EDEVS 原子模型建模实例

本节将介绍两个 EDEVS 原子模型建模实例. 其中，第 1 个模型实现了一个积分器，该积分器在给定的积分区间和步长内对 X^2 积分；第 2 个模型实现了一个加法器，对输入该加法器的两个数求和，在下节的耦合模型实例中本文将会把两个积分器的输出端口分别与加法器的两个输入端口相连，构建一个耦合模型.

3.3.1　积分器模型

图 1 为积分器模型示例. 该模型对 X^2 进行定步长积分. 其中 $Step$，$Integral$，$Begin$，End 4 个状态变量分别描述“积分步长”、“积分值”、“积分的起始点”及“积分的结束点”. $Integral$ 的初始值为 0.0，$Step$，$Begin$ 及 End 可以由模型的使用者自行设定，$Step$ 越小，积分值越精确. 模型有一个内部端口和一个输出端口，没有输入端口. 内部端口的参数集为空，其端口处理函数对 $(Begin+t)\times(Begin+t)\times Step$ 即 $X\times dt$ 的值进行累加；输出端口的参数集中只有一个参数，即积分结果，用于向外输出积分值.

该模型的处理流程为:在逻辑时间 0 时刻,模型调度一个目的端口为"*IntegralOperation*"的初始事件,触发积分累加操作 $Integral=Integral+(Begin+t)\times(Begin+t)\times Step$. 此后,若积分未完成(即 $Begin+t<End$),则"*IntegralOperation*"的端口处理函数在完成状态转换后,皆会调度一个目的端口为"*IntegralOperation*",执行时间为当前时间加上"*Step*"个逻辑时间的内部事件,继续积分累加操作,否则调度一个延时为 0、目的端口为"*Ans*"的外部事件,将积分值"*Integral*"输出.

$$
\begin{aligned}
&IPorts=\{\}\\
&InnerPorts=\{\text{"}IntegralOperation\text{"}\}\\
&OPort=\{\text{"}Ans\text{"}\}\\
&K_m=\{(p,v)\mid p\in InnerPorts, v=\varnothing\}\\
&Y_m=\{(p,v)\mid p\in OPorts, v\in\Re\}\\
&S=\{Step,Integral,Begin,End\mid Step,Integral,Begin,End\in\Re\},\\
&S_0=\{Integral=0, Step=configurable, Begin=configurable,\\
&\quad End=configurable\}\\
&Init_{ie}=\{(\text{"}IntegralOperation\text{"},0)\},\delta_{con}=\text{first come first service}\\
&\delta(\langle Begin,End,Step,Integral\rangle,(\text{"}Integral\text{"},\varnothing))=\\
&\quad\begin{cases}\langle Begin,End,Step,Integral+(Begin+t)\times\\ \quad(Begin+t)\times Step\rangle, & \text{if } (Begin+t<End)\\ \langle Begin,End,Step,Integral\rangle, & \text{else}\end{cases}\\
&\lambda(\langle Step,Integral,Begin,End\rangle,(\text{"}Integral\text{"},\varnothing))=\\
&\quad\begin{cases}(\text{"}Integral\text{"},\varnothing\rangle, & \text{if } Begin+t<End\\ (\text{"}Ans\text{"},Integral\rangle, & \text{else}\end{cases}\\
&ea(\text{"}IntegralOperation\text{"},\langle Step,Integral,Begin,End\rangle)=Step\\
&ea(\text{"}Ans\text{"},\langle Step,Integral,Begin,End\rangle)=0
\end{aligned}
$$

图 1 X^2 定步长积分 EDEVS 模型

3.3.2 加法器模型

图 2 为加法器模型示例. 该模型的两个输入端口"*InA*"及"*InB*" 接收两个加数,而后送入两个内部端口"*ans*1"及"*ans*2"进行累加操作:$ans=ans+v$,并记录已累加的操作数的个数:$count=count+1$.

$$
\begin{aligned}
&IPorts=\{\text{"}InA\text{"},\text{"}InB\text{"}\}\\
&InnerPorts=\{\text{"}ans1\text{"},\text{"}ans2\text{"}\}\\
&OPort=\{\text{"}Output\text{"}\}\\
&X_m=\{(p,v)\mid p\in IPorts, v\in\Re\}\\
&K_m=\{(p,v)\mid p\in InnerPorts, v\in\Re\}\\
&Y_m=\{(p,v)\mid p\in OPorts, v\in\Re\}\\
&S=\{ans,count\mid ans,count\in\Re\},S_0=\{ans=0,count=0\}\\
&Init_{ie}=\{\varnothing\},\delta_{con}=\text{first come first service}\\
&\delta(\langle ans,count\rangle,(\text{"}InA\text{"},v))=\langle ans,count\rangle\\
&\delta(\langle ans,count\rangle,(\text{"}InB\text{"},v))=\langle ans,count\rangle\\
&\delta(\langle ans,count\rangle,(\text{"}ans1\text{"},v))=\langle ans+v,count+1\rangle\\
&\delta(\langle ans,count\rangle,(\text{"}ans2\text{"},v))=\langle ans+v,count+1\rangle\\
&\lambda(\langle ans,count\rangle,(\text{"}InA\text{"},v))=(\text{"}ans1\text{"},v)\\
&\lambda(\langle ans,count\rangle,(\text{"}InB\text{"},v))=(\text{"}ans2\text{"},v)\\
&\lambda(\langle ans,count\rangle,(\text{"}ans1\text{"},v))=\begin{cases}(\text{"}Output\text{"},ans\rangle & \text{if}(count==2)\\ \varnothing & \text{else}\end{cases}\\
&\lambda(\langle ans,count\rangle,(\text{"}ans2\text{"},v))=\begin{cases}(\text{"}Output\text{"},ans\rangle & \text{if}(count==2)\\ \varnothing & \text{else}\end{cases}\\
&ea(\text{"}ans1\text{"},\langle ans,count\rangle)=0\\
&ea(\text{"}ans2\text{"},\langle ans,count\rangle)=0\\
&ea(\text{"}Output\text{"},\langle ans,count\rangle)=0
\end{aligned}
$$

图 2 加法器模型

当 $count==2$ 时累加操作完成,端口"*Output*"将向外输出最终结果 *ans*.

3.4 EDEVS 耦合模型建模实例

图 3 为两个积分器与一个加法器耦合而成的模型,该模型计算两个积分器的积分之和. 该耦合模型没有输入端口和内部端口,只有一个用于输出积分之和的输出端口"*CoupledModelOut*". *D* 中定义了耦合组件中的子组件的名称为 *IntergralUnit*1, *IntergralUnit*2 以及 *AdderUnit*1,其中前两个为积分器模型,第三个为加法器模型,*EOC* 中将加法器的"*Output*"输出端口连接至耦合组件的"*CoupledModelOut*"输出端口,*IC* 中将两个积分器的"*Ans*"输出端口分别连接至加法器的"*InA*"及"*InB*"输入端口,由此构成计算积分和的耦合模型.

$$
\begin{aligned}
&OPort=\{\text{"}CoupledModelOut\text{"}\}\\
&X_m=\varnothing\\
&K_m=\varnothing\\
&Y_m=\{(p,v)\mid p\in OPorts, v\in\Re\}\\
&D=\{IntergralUnit1, IntergralUnit2, AdderUnit1\}\\
&M_{IntergralUnit1}=Intergrator,\\
&M_{IntergralUnit2}=Intergrator, M_{AdderUnit1}=Adder\\
&EIC=\varnothing\\
&EOC=\{((AdderUnit1,\text{"}Output\text{"}),(N,\text{"}CoupledModelOut\text{"}))\}\\
&IC=\{((IntergralUnit1,\text{"}Ans\text{"}),(AdderUnit1,\text{"}InA\text{"})),\\
&\quad((IntergralUnit2,\text{"}Ans\text{"}),(AdderUnit1,\text{"}InB\text{"}))\}
\end{aligned}
$$

图 3 积分器与加法器耦合模型

3.5 EDEVS 描述能力证明

EDEVS 的描述能力决定了基于它搭建的仿真建模框架的建模能力,也就决定了整个仿真平台的仿真能力. 而 PDEVS 是从控制理论发展而来的,理论较成熟,其建模离散事件系统的能力业已得到形式化的证明. 因此可以通过证明 EDEVS 能描述 PDEVS 中的所有模型来保证 EDEVS 的建模能力. 为此,以下将证明 EDEVS 能描述 PDEVS 所能描述的任何模型.

定义 1. 给定任意 PDEVS 模型 $M_{pdevs}=\langle X_M, Y_M, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta\rangle$, 定义相应的 EDEVS 模型 $M_{pdevs}=\langle X_M, Y_M, K_m, S_{ed}, Init_{ie}, \delta, \delta_{con}, \lambda_{ed}, f_{eid}, ea\rangle$. 其中(以下提及的时间均为仿真时间):

$InnerPorts=\{k_1,k_2,k_3\}$;

$K_1=\{k_1,\varnothing\}, K_2=\{k_2,\varnothing\}, K_3=\{k_3,\varnothing\}$;

$K_m=\{K_1,K_2,K_3\}$(K_1 的端口处理函数取代 δ_{int}, K_2 用于取消 $NIeid_0$ 所标识的被调度但尚未执行的事件, K_3 用于模拟 δ_{ext} 的行为,本文将用 $Port^{time}$ 来标识目的端口为"Port",执行时刻为"time"的事件);

$S_{ed}=\langle NIeid_0, NIeid_n, tl, tn, IPset, xflag, yflag, S_o, S_n\rangle$；

$NIeid_0$ 用于保存上一个被调度的 K_1 事件的标识；

$NIeid_n$ 用于保存将被调度的 K_1 事件的标识；

tl 是 K_1 或 K_3 事件最近一次被执行的时刻；

tn 是最后被调度的 K_1 事件将要被执行的时刻；

$IPset$ 用来构造 X_m^b；

$xflag, yflag$ 用来保证 K_3 在每个仿真时刻只能被调度一次；

S_o, S_n 分别是 S（M_{pdevs} 模型中的状态变量）的上一个状态和当前状态，由于 EDEVS 模型先执行端口处理函数，后执行输出函数，而 PDEVS 模型正好相反，为了取得与 PDEVS 模型一致的输出，必须用 S_o 保存上一个状态.

M_{edevs} 用于响应外部事件的端口处理函数定义如下：

$\forall p\in IPorts$（以下 if 语句中的“=”为赋值运算符，隐含其值在给定的定义域内，“==”为相等测试运算符）：

$$\delta(X_p,\langle NIeid_o, NIeid_n, tl, tn, IPset, xflag, yflag, S_o, S_n\rangle)=\begin{cases}\langle NIeid_o, NIeid_n, tl, tn, IPset\circ X_p, \text{true}, \text{false}, S_o, S_n\rangle, & \text{if}(xflag==\text{false})\\ \langle NIeid_o, NIeid_n, tl, tn, IPset\circ X_p, \text{true}, \text{true}, S_o, S_n\rangle, & \text{else}\end{cases};$$

$$\lambda_{ed}(X_p,\langle NIeid_o, NIeid_n, tl, tn, IPset, xflag, yflag, S_o, S_n\rangle)=\begin{cases}(K_3), & \text{if}(xflag==\text{ture}\&\&yflag==\text{false})\\ \varnothing, & \text{else}\end{cases};$$

$$ea(K_3,\langle NIeid_o, NIeid_n, tl, tn, IPset, xflag, yflag, S_o, S_n\rangle)=0;$$

其中“$IPset\circ X_p$”表示将 X_p 存入 $IPset$.

M_{edevs} 状态转换函数用于响应内部事件的部分定义如下：

$$\delta(K_1,\langle NIeid_o, NIeid_n, tl, tn, IPset, xflag, yflag, S_o, S_n\rangle)=\begin{cases}\langle NIeid_n, K_1^{t+ta(S_{k1})}, t, t+ta(S_{k1}), IPset, xflag, yflag, S_n, S_{k1}\rangle, & \text{if } (M_{pdevs}:\delta_{int}(S_n)=S_{k1})\\ \langle NIeid_o, NIeid_n, tl, tn, IPset, xflag, yflag, S_o, S_n\rangle, & \text{else}\end{cases};$$

$$\lambda_{ed}(K_1,\langle NIeid_o, NIeid_n, tl, tn, IPset, xflag, yflag, S_o, S_n\rangle)=\begin{cases}(K_1,\lambda(S_o)), & \text{if}(\delta_{int}(S_n)\text{ is defined})\\ \varnothing, & \text{else}\end{cases};$$

$$ea(K_1,S_{ed})=ta(S_n), ea(\lambda(S_o),S_{ed})=0;$$

$$\delta(K_2,\langle NIeid_o, NIeid_n, tl, tn, IPset, xflag, yflag, S_o, S_n\rangle)=\langle NIeid_o, NIeid_n, tl, tn, IPset, xflag, yflag, S_o, S_n\rangle;$$

$$\lambda(K_2,\langle NIeid_o, NIeid_n, tl, tn, IPset, xflag, yflag, S_o, S_n\rangle)=\langle \text{"cancelOP"}, NIeid_o\rangle;$$

$$ea(\langle \text{"cancelOP"}, NIeid_o\rangle,\langle NIeid_o, NIeid_n, tl, tn, IPset, xflag, yflag, S_o, S_n\rangle)=0;$$

$$\delta(K_3,\langle NIeid_o, NIeid_n, tl, tn, IPset, xflag, yflag, S_o, S_n\rangle)=\begin{cases}\langle \varnothing, K_1^{t+ta(S_{k3})}, t, t+ta(S_{k3}), \varnothing, \text{false}, \text{false}, S_n, S_{k3}\rangle, & \\ \text{if}(M_{pdevs}:\delta_{ext}((S_n, t-tl), IPset)=S_{k3}\text{ and } tn==t) & \\ \langle NIeid_n, K_1^{t+ta(S_{k3})}, t, t+ta(S_{k3}), & \\ \quad \varnothing, \text{false}, \text{false}, S_n, S_{k3}\rangle, & \\ \text{else if } (M_{pdevs}:\delta_{ext}((S_n, t-tl), IPset)= & \\ \quad S_{k3}\text{ and } tn!=t) & \\ \langle NIeid_o, NIeid_n, tl, tn, \varnothing, \text{false}, \text{false}, S_o, S_n\rangle & \text{else}\end{cases}$$

$$\lambda_{ed}(K_3,\langle NIeid_o, NIeid_n, tl, tn, IPset, xflag, yflag, S_o, S_n\rangle)=\begin{cases}(K_1,K_2), & \text{if}(NIeid_o\neq\varnothing)\\ K_1, & \text{else}\end{cases}$$

$$ea(K_1,\langle NIeid_o, NIeid_n, tl, tn, IPset, xflag, \lambda flag, S_o, S_n\rangle)=ta(S_n);$$

$$ea(K_2,\langle NIeid_o, NIeid_n, tl, tn, IPset, xflag, \lambda flag, S_o, S_n\rangle)=0.$$

若 M_{pdevs} 中的冲突消解函数定义为 $\delta_{con}(s,ta(s),x)=\delta_{ext}(\delta_{int}(s),0,x))$，则 M_{edevs} 的 δ_{con} 定义为

$$\delta_{con}((x_m\cup k_m)^b,s)=\delta(k_3,\delta(\cdots\delta(x_m^1,\delta(k_1,\delta(k_2,s)\cdots)),$$

即 K_2 的优先级最高，其次是 K_1，然后是外部事件，最后是 K_3；

否则 M_{edevs} 的 δ_{con} 定义为

$$\delta_{con}((x_m\cup k_m)^b,s)=\delta(k_1,\delta(k_3,\delta(\cdots\delta(x_m^1,\delta(k_2,s)\cdots)),$$

即 K_2 的优先级最高，其次是外部事件，然后是 K_3，最后是 K_1.

引理 1.　内部事件 K_2 或 K_3 只能伴随外部事件发生

证明.　从状态转换函数的定义中可以知道，K_2 只由与 K_3 相对应的状态转换函数调度且延时为 0，而 K_3 只由外部事件调度且延时为 0，因而 K_2 及 K_3 只能发生在有外部事件发生的时刻，即伴随外部事件发生.

定理 1.　M_{edevs} 能准确描述 M_{pdevs}.

证明.

只要证明,给定输入串$\{(X_1^{t_1},X_2^{t_2},\cdots,X_n^{t_n})|(0\leq t_1\leq\cdots\leq t_n)\}$及$M_{pdevs}$的初始状态$S_0$,置$M_{edevs}$的初始状态$S_{ed0}=\langle\varnothing,K_1^{ta(S_0)},0,ta(S_0),\varnothing,\varnothing,\text{false},\text{false},S_0,S_0\rangle$,$Init_{ie}=\{\langle K_1,ta(S_0)\rangle\}$(即调度$K_1$事件在$ta(S_0)$时刻执行),那么在任意时间段内,$M_{edevs}$的状态转换序列在$S_n$分量上的投影序列与$M_{pdevs}$的状态转换序列相同,输出序列也相同(参看文献[19]第1章第5节关于模型同构的论述).

假设M_{pdevs}中$\delta_{con}(s,ta(s),x)=\delta_{ext}(\delta_{int}(s),0,x))$,则

(1)$t=0$时,分两种情形.

(a)若$t_1<ta(S_0)$:

对于M_{pdevs},此时外部输入$X_1^{t_1}$先于内部状态转换发生,因而先执行$\delta_{ext}:Q\times X_m^b\to S$,记$\delta_{ext}(S_0,t_1-0,X_1^{t_1})=S_1$,则$M_{pdevs}$的状态变化序列$ST_{pdevs}=\{S_0^0,S_1^{t_1}\}$,而由于输出只能在内部状态转换函数即将执行时产生,因此此时输出为$\varnothing$,输出序列$YT_{pdevs}=\varnothing$,时间推进到t_1,若没有外部输入干扰,内部状态转换函数将在$t_1+ta(S_1)$时刻执行.

对于M_{edevs}:由于K_1被调度在$ta(S_0)$时刻执行,而由引理知$0\leq t<ta(S_0)$时没有K_2或K_3之间发生,事件队列为$\{X_1^{t_1},K_1^{ta(S_0)}\}$,因此先处理$t_1$时刻的外部事件$X_1^{t_1}$,即执行$\delta:X_m\times S\to S$,对$X_1^{t_1}$的第1个分量$X_{1(1)}^{t_1}$有

$\delta(X_{1(1)}^{t_1},\langle\varnothing,K_1^{ta(S_0)},0,ta(S_0),\varnothing,\text{false},\text{false},S_0,S_0\rangle)=\langle\varnothing,K_1^{ta(S_0)},0,ta(S_0),X_{1(1)}^{t_1},\text{true},\text{false},S_0,S_0\rangle$;

$\lambda_{ed}(X_{1(1)}^{t_1},\langle\varnothing,K_1^{ta(S_0)},0,ta(S_0),X_{1(1)}^{t_1},\text{true},\text{false},S_0,S_0\rangle)=\langle K_3\rangle$;

$ea(K_3,\langle\varnothing,K_1^{ta(S_0)},0,ta(S_0),X_{1(1)}^{t_1},\text{true},\text{false},S_0,S_0\rangle)=0$;即调度$K_3$在$t_1$时刻执行.事件队列为$\{X_1^{t_1}-X_{1(1)}^{t_1},K_3^{t_1},K_1^{ta(S_0)}\}$,但从$\delta_{con}$的定义可知,在$t_1$时刻发生的众多事件当中,$K_3$应在最后执行,因而接下来处理$X_1^{t_1}$的下一个分量.

对$X_1^{t_1}$的第2个分量$X_{1(2)}^{t_1}$有

$\delta(X_{1(2)}^{t_1},\langle\varnothing,K_1^{ta(S_0)},0,ta(S_0),X_{1(1)}^{t_1},\text{true},\text{false},S_0,S_0\rangle)=\langle\varnothing,K_1^{ta(S_0)},0,ta(S_0),X_{1(1)}^{t_1}\circ X_{1(2)}^{t_1},\text{true},\text{true},S_0,S_0\rangle$;

$\lambda_{ed}(X_{1(2)}^{t_1},\langle\varnothing,K_1^{ta(S_0)},0,ta(S_0),X_{1(1)}^{t_1}\circ X_{1(2)}^{t_1},\text{true},\text{true},S_0,S_0\rangle)=\varnothing$.

类似可以推出,对$X_1^{t_1}$的最后一个分量$X_{1(n)}^{t_1}$有

$\delta(X_{1(n)}^{t_1},\langle\varnothing,K_1^{ta(S_0)},0,ta(S_0),X_{1(1)}^{t_1}\circ X_{1(2)}^{t_1}\circ\cdots\circ X_{1(n-1)}^{t_1},\text{true},\text{true},S_0,S_0\rangle)=\langle\varnothing,K_1^{ta(S_0)},0,ta(S_0),X_{1(1)}^{t_1}\circ X_{1(2)}^{t_1}\circ\cdots\circ X_{1(n-1)}^{t_1}\circ X_{1(n)}^{t_1},\text{true},\text{true},S_0,S_0\rangle=\langle\varnothing,K_1^{ta(S_0)},0,ta(S_0),X_1^{t_1},\text{true},\text{true},S_0,S_0\rangle$;

$\lambda_{ed}(X_{1(n)}^{t_1},\langle\varnothing,K_1^{ta(S_0)},0,ta(S_0),X_1^{t_1},\text{true},\text{true},S_0,S_0\rangle)=\varnothing$;

此时事件队列为$\{K_3^{t_1},K_1^{ta(S_0)}\}$,下面执行事件$K_3$:

$\delta(K_3,\langle\varnothing,K_1^{ta(S_0)},0,ta(S_0),X_1^{t_1},\text{true},\text{true},S_0,S_0\rangle)=\langle K_1^{ta(S_0)},K_1^{t1+ta(S_1)},t1,t1+ta(S_1),\varnothing,\text{false},\text{false},S_0,S_1\rangle$;

$\lambda_{ed}(K_3,\langle K_1^{ta(S_0)},K_1^{t1+ta(S_1)},t1,t1+ta(S_1),\varnothing,\text{false},\text{false},S_0,S_1\rangle)=\{K_1,K_2\}$;

$ea(K_1,\langle K_1^{ta(S_0)},K_1^{t1+ta(S_1)},t1,t1+ta(S_1),\varnothing,\text{false},\text{false},S_0,S_1\rangle)=ta(S_1)$;

$ea(K_2,\langle K_1^{ta(S_0)},K_1^{t1+ta(S_1)},t1,t1+ta(S_1),\varnothing,\text{false},\text{false},S_0,S_1\rangle)=0$;

即调度事件K_2在t_1时刻执行,调度K_1事件在$t_1+ta(S_1)$时刻执行.此时事件队列为$\{K_2^{t1},K_1^{ta(S_0)},K_1^{t1+ta(S_1)}\}$,下面处理事件$K_2$:

$\delta(K_2,\langle K_1^{ta(S_0)},K_1^{t1+ta(S_1)},t1,t1+ta(S_1),\varnothing,\text{false},\text{false},S_0,S_1\rangle)=\langle K_1^{ta(S_0)},K_1^{t1+ta(S_1)},t1,t1+ta(S_1),\varnothing,\text{false},\text{false},S_0,S_1\rangle$;

$\lambda_{ed}(K_2,\langle K_1^{ta(S_0)},K_1^{t1+ta(S_1)},t1,t1+ta(S_1),\varnothing,\text{false},\text{false},S_0,S_1\rangle)=\langle\text{"}cancelOP\text{"},NIeid_o\rangle$;

$ea(\langle\text{"}cancelOP\text{"},K_1^{ta(S_0)}\rangle,\langle K_1^{ta(S_0)},K_1^{t1+ta(S_1)},t1,t1+ta(S_1),\varnothing,\text{false},\text{false},S_0,S_1\rangle)=0$;

此时事件$K_1^{ta(S_0)}$被取消,事件队列为$\{K_1^{t1+ta(S_1)}\}$.在下一事件到来前,M_{edevs}的状态稳定在$\langle\varnothing,K_1^{t1+ta(S_1)},t1,t1+ta(S_1),\varnothing,\text{false},\text{false},S_0,S_1\rangle$,$K_1$事件被调度于$t_1+ta(S_1)$时刻发生.$\varphi_{sn}(S_{ed}^{t1})=S_1$,$Y_{ed}^{t1}=\varnothing$,即状态转换序列在$Sn$分量上的投影$\varphi_{sn}(ST_{edevs})=\{S_0^0,S_1^{t1}\}$,输出序列$YT_{edevs}=\varnothing$,可见,$ST_{pdevs}=\varphi_{sn}(ST_{edevs})$,$YT_{pdevs}=YT_{edevs}$,且$M_{edevs}$中$K_1$即将发生的时刻与$M_{pdevs}$中下个内部状态转换时刻相同,都为$t_1+ta(S_1)$.

(b)若$t_1\geq ta(S_0)$:

对于M_{pdevs},由于$\delta_{con}(S,ta(S),x)=\delta_{ext}(\delta_{int}(S),0,x))$,因而先执行$\delta_{int}:S\to S$,假设$\delta_{int}(S_0)=S_1$,$\lambda_{pdevs}=Y_1^{ta(S_0)}$,则$ST_{pdevs}=\{S_0^0,S_1^{ta(S_0)}\}$,$YT_{pdevs}=\{Y_1^{ta(S_0)}\}$,若没有外部输入干扰,内部状态转换函数

将在 $ta(S_0)+ta(S_1)$时刻执行.

对于 M_{edevs}，初始事件集为$\{K_1^{ta(S_0)}\}$，时间推进到 $ta(S_0)$，执行事件 K_1：

$\delta(K_1,\langle\varnothing,K_1^{ta(S_0)},0,ta(S_0),\varnothing,\text{false},\text{false},S_0,S_0\rangle)=\langle K_1^{ta(S_0)},K_1^{ta(S_0)+ta(S_1)},ta(S_0),ta(S_0)+ta(S_1),\varnothing,\text{false},\text{false},S_0,S_1\rangle$；

$\lambda_{ed}(K_1,\langle K_1^{ta(S_0)},K_1^{ta(S_0)+ta(S_1)},ta(S_0),ta(S_0)+ta(S_1),\varnothing,\text{false},\text{false},S_0,S_1\rangle)=\{K_1,Y_1\}$；

$ea(K_1,\langle K_1^{ta(S_0)},K_1^{ta(S_0)+ta(S_1)},ta(S_0),ta(S_0)+ta(S_1),\varnothing,\text{false},\text{false},S_0,S_1\rangle)=ta(S_1)$；

$ea(Y_1,\langle K_1^{ta(S_0)},K_1^{ta(S_0)+ta(S_1)},ta(S_0),ta(S_0)+ta(S_1),\varnothing,\text{false},\text{false},S_0,S_1\rangle)=0$.

在下一事件到来前，M_{edevs} 的状态稳定在$\langle K_1^{ta(S_0)},K_1^{ta(S_0)+ta(S_1)},ta(S_0),ta(S_0)+ta(S_1),\varnothing,\text{false},\text{false},S_0,S_1\rangle$，$K_1$ 事件被调度于 $ta(S_0)+ta(S_1)$时刻发生. $\varphi_{sn}(S_{ed}^{ta(S_0)})=S_1$，$Y_{ed}^{ta(S_0)}=Y_1^{ta(S_0)}$，即 $\varphi_{sn}(ST_{edevs})=\{S_0^0,S_1^{ta(S_0)}\}$，输出序列 $YT_{edevs}=\{Y_1^{ta(S_0)}\}$，可见 $ST_{pdevs}=\varphi_{sn}(ST_{edevs})$，$YT_{pdevs}=YT_{edevs}$，且 M_{edevs} 中 K_1 即将发生的时刻与 M_{pdevs} 中下个内部状态转换时刻相同，都为 $ta(S_0)+ta(S_1)$.

(2) 假设 t_i 时刻之前 $ST_{pdevs}=\varphi_{sn}(ST_{edevs})$，$YT_{pdevs}=YT_{edevs}$ 均成立，记 M_{pdevs} 所处状态为 $S_{i(0)}^{t_k}(t_{i-1}\le t_k<t_i)$，若没有外部输入干扰，内部状态转换函数将在 $t_k+ta(S_{i0}^{t_k})$时刻执行，则 M_{edevs} 的内部状态为：$\langle K_1^{t_k-l},K_1^{t_k+ta(S_{i(0)}^{t_k})},t_k,t_k+ta(S_{i(0)}^{t_k}),\varnothing,\text{false},\text{false},S_{i(-1)}^{t_k-m},S_{i(0)}^{t_k}\rangle$，$K_1$ 事件被调度于 $tk+ta(S_{i(0)}^{t_k})$时刻发生，模型的输入为$\{(X_1^{t_{i+1}},X_2^{t_{i+2}},\cdots,X_n^{t_{i+n}})\mid(t_i\le t_{i+1}\le\cdots\le t_{i+n})\}$.

(a) 若 $t_k+ta(S_{i(0)}^{t_k})<t_i$：

证明方法与第(1)步的(a)情况类似；

(b) 若 $t_i\ge t_k+ta(S_{i(0)}^{t_k})$：

证明方法与第(1)步的(b)情况类似；

而 M_{pdevs} 中 $\delta_{con}(s,ta(s),x)=\delta_{int}(\delta_{ext}(s,0,x))$ 时，证法类似，不再累述.

综上可知 M_{edevs} 的状态转换序列在 S_n 分量上的投影序列与 M_{pdevs} 的状态转换序列相同，输出序列也相同，因此命题正确. 由 M_{pdevs} 的任意性可知 EDEVS 能描述 PDEVS 所能描述的任意模型.

证毕.

4　EDEVS 可视化建模框架的实现

EDEVS 吸收了组件建模及 MDA 技术的优点，支持可视化组件建模. 本文基于 EDEVS 范式开发了一个可视化建模框架 VisiCom[20]. 该框架将 EDEVS 范式中的元素图元化，使得用户能够通过拖拽图元的方式直观、快捷地建立 EDEVS 模型. VisiCom 包括可视化模型编辑器及代码转换器两大组成部分. 可视化编辑器提供建模图元，接收用户以图元形式表示的 EDEVS 模型，将之转换并保存为 XML 模型描述文件；代码转换器负责将 XML 模型文件中的平台无关模型转换成特定平台下的模型代码.

4.1　可视化模型编辑器

与 EDEVS 范式相对应，可视化编辑器包括原子模型编辑器及耦合模型编辑器. 原子模型编辑器用来构造 EDEVS 原子模型，耦合模型编辑器利用已建好的原子模型或耦合模型，构造更复杂的模型，实现层次化建模.

4.1.1　原子模型编辑器

原子模型编辑器以图元或图元属性的形式对 EDEVS 原子模型范式中的元素进行可视化. 其中，“输入端口”、“输出端口”、“内部端口”、“状态”、“事件调度”、“取消调度”及“初始化事件集”都有图元与之相对应；“端口处理函数”及“优先级函数”分别作为端口的“伪码”属性和“优先级”属性实现；“事件标识生成函数”及“时间推进函数”分别作为“事件调度”的“名字”属性和“延时”属性实现.

原子模型编辑器的界面如图 4 所示：“内部端口”、“事件调度”、“取消调度”、“状态”图元分布在编辑器左边的 tab 页内，将图元拖入编辑器即生成该图元实例；“输入端口”、“输出端口”图元处于工具条上，单击图元即在模型编辑区的边界出现该图元实例；“初始化事件集”图元实例在每个原子模型内有且仅有一个，内置于编辑区中. 每个图元实例都有相应的属性框，其中，“端口”图元实例的属性框包括“名称”、“参数列表”、“优先级”及“伪码”4 个属性条目，“参数列表”定义此端口的传入参数类型，可以是 int、double、bool 或 string 等类型，“优先级”是一个整型数值，定义以此端口为目的端口的事件在同时刻发生的事件中的处理次序，“伪码”描述“端口处理函数”的处理逻辑，该处理逻辑应遵循以下流程：首先，解析事件传入参数，其次根据参数及状态变量值计算状态变量新值，最后计算事件调度延时和传出参数值，并调度目的端口为“内部端口”或“输出端口”的事件；事件调度图元实例的属性框包括“名称”、“条件”及“延时”3 个条目，“名称”在此模型中唯一，用于与 f_{eid} 相关联. “条件”为真时调度相应的

事件,“延时”表示该事件经过多长时间后到达目的端口,从而触发目的端口处理函数的执行;“取消调度”图元实例用来取消被调度但尚未执行的事件,其属性框包括“名称”、“条件”及“目标”3 个属性条目,“目标”属性保存要取消的“事件调度”的名字,当条件为真时取消之;“状态”图元实例用来描述模型的状态变量,其属性框包括“名称”、“类型”、“可配置”、“取值范围”、“默认值”及“属性描述”6 个属性条目,其中“类型”可以是 int,double,string 等,“可配置”属性表示是否允许模型使用者通过配置接口对此属性进行配置.

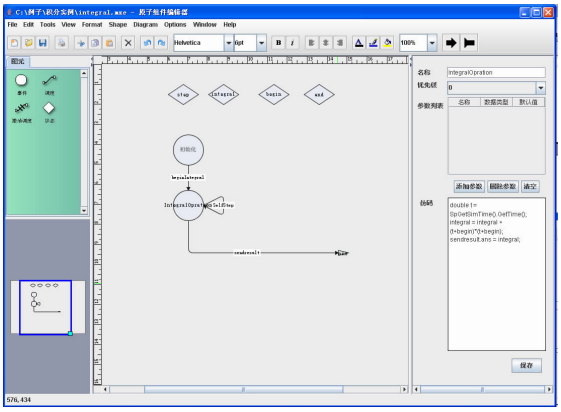

图 4 原子组件编辑器

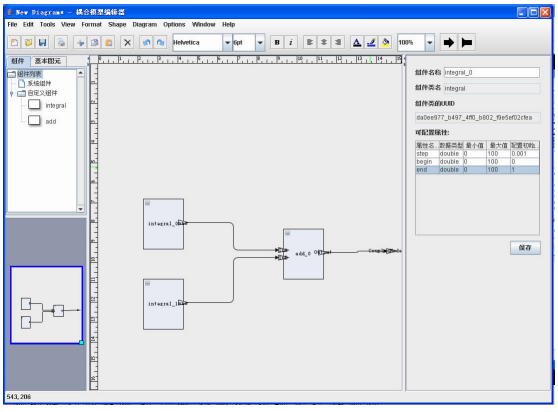

图 5 耦合组件编辑器

图 4 即为 3.3.1 节介绍的定步长积分器的可视化模型,包括 10 个图元实例,其中有 **4 个状态变量图元实例**,状态变量图元实例分别代表 *Step*、*Integral*、*Begin* 及 *End* 4 个状态变量;**1 个初始化事件集图元实例**;**1 个名为 *IntegralOperation* 的内部端口实例**,用来计算积分值,其伪码为 *integral*=*integral*+(*t*+*begin*)×(*t*+*begin*);*sendresult.ans*=*integral*(对名为“*sendresult*”的事件调度的参数集赋值);**1 个名为 Ans 的输出端口实例**,用来向外界输出积分值;**3 个事件调度图元实例**,其中,第 1 个名为 *BeginIntegral*,它连接初始化事件集图元与 *IntegralOperation* 端口,调度延时为 0,表示逻辑时间的 0 时刻调度一个指向 *IntegralOperation* 端口的事件,第 2 个名为 *SelfStep*,它连接 *IntegralOperation* 图元自身,延时为 *Step*,条件为 *Begin*+*t*<*End*,即在计算积分值的过程中此端口的端口处理函数会调度一个目的端口为自身的事件,此事件在 *Step* 时间间隔之后执行,驱动积分的累加过程,最后一个名为 *SendResult*,它从 *IntegralOperation* 指向名为 *Ans* 端口,延时为 0,条件为 !(*Begin*+*t*<*End*),表示积分完成之后立即向 *Ans* 端口输出积分值.

4.1.2 耦合模型编辑器

如图 5 所示,耦合组件编辑器的基本图元是“输入端口”、“输出端口”、“端口连接线”及“组件库中的组件”(表现为带输入、输出端口的方框). 图 5 是一个名为 Couple 的耦合组件可视化模型实例. 该耦合模型实例化两个 4.1.1 节所建立的定长积分器模型,并把它们输出的积分值输入到 3.3.2 节介绍的加法器模型内做加法运算,而后将结果输出. 该耦合组件只有一个输出端口“Answer”,用来输出积分和. 两个 *Integral* 子组件实例的输出端口分别与加法器子组件实例的两个输入端口相连;而加法器子组件实例的输出端口与 Answer 端口相连. 点击子组件实例,可在编辑器右方的属性编辑框中对子组件的可配置属性进行设置.

4.1.3 EDEVS 模型文件

可视化建模工具接收用户输入的图形模型后,将此模型转换并存储为 XML 模型描述文件. 此文件将是代码生成工具生成平台相关模型的输入. 原子模型 XML 描述文件包括 InfoRegion、InitRegion、PortRegion、DeltaRegion 及 StateRegion 五个部分. 其中,InfoRegion 描述组件的基本信息,包括组件名、组件 UUID 以及组件的开发者及版权等信息;InitRegion 描述初始事件集,即在逻辑时间的 0 时刻需要仿真引擎预先调度的事件;PortRegion 描述端口信息,包括输入、内部及输出端口的名称,优先级及参数集,以及端口可能发出的事件调度;DeltaRegion 描述模型的端口处理函数的执行逻辑;StateRegion 描述状态变量信息,包括状态变量的名称、类型、取值范围及可配置与否等信息. 耦合模型 XML 描述文件包括 InfoRegion、InitRegion、SubComConfRegion、PortRegion 及 PortConnection 五个部分. 与原子模型 XML 描述文件一样,耦合模型

XML 描述文件也包含 InfoRegion、InitRegion 及 PortRegion；不同的是，耦合模型没有 DeltaRegion 和 StateRegion，因为其事件处理逻辑及状态变量都分布在所包含的原子模型中，此外其增加了 SubComConfRegion 及 PortConnection 两个区段，其中，SubComConfRegion 描述子模型的 UUID 及其参数配置情况；PortConnection 描述耦合组件与子组件之间以及子组件与子组件之间的端口连接关系.

4.2　代码转换器

Visicom 可视化模型编辑器生成的 EDEVS 组件化模型文件与平台无关，支持平台间的模型重用，只要实现相应的代码转换器即可.

YHSUPE 是作者所在课题组开发的一个并行仿真平台，其在基于银河高性能计算机的仿真应用中取得了较高的执行性能，但其采用集中式开发模式，开发协同程度大、模型间耦合度高，较难重用，且其编程过程较复杂，学习曲线较高、二次建模较困难. 为此，本文实现了 EDEVS 模型到 YHSUPE 模型的代码转换器，将可视化组件建模引入 YHSUPE，以解决以上问题.

为了降低代码转换器的复杂度，本文利用 YHSUPE 所提供的 API 实现了一个 SpBaseComponent 类，此类对 EDEVS 规范中的元素(端口、事件调度、取消调度等)进行了相应的实现，包含三类接口：

(1) 组件配置接口

void ConfigHandler(SpParameterSet&para).

para 是〈*ParaName*，*ParaValue*〉形式的名值对，用来对组件中名为 *ParaName* 的可配置属性配置初始值.

(2) 连接关系配置接口

void AddSubscriber（*char*×*myPort*，*char*×*hisUUID*，*char*×*hisPort*）用来将 *UUID* 为 *hisUUID* 的组件的 *hisPort* 端口加入到此组件的 *myPort* 端口的输出列表中.

(3) 组件业务逻辑接口

① 端口注册接口

(a) *void RegisterInputPort*（*string PortName*，*int pri*=0)；注册名为 *PortName* 优先级为 *pri* 的输入端口.

(b) *void RegisterInnerPort*（*string PortName*，*int pri*=0)；注册名为 *PortName* 优先级为 *pri* 的内部端口.

(c) *void RegisterOutputPort*（*string PortName*，*int pri*=0)；注册名为 *PortName* 优先级为 *pri* 的输出端口.

(d) *void RegisterPortHanler*（*string Port*，*FuncPtr Func*)；将端口处理函数 *Func* 关联到 *Port* 端口.

② 初始事件调度接口

void PrimaryEvent()；此接口由仿真引擎在组件创建时刻调用，用来构造初始事件调度集.

③ 事件调度接口

(a) *void SendMsgToPort*（*char*×*myPort*，*SpParameterSet&pset*，*SpCancelHandle&handle*，*double delay*=1)；用来向本组件的内部端口或输出端口发出事件调度.

(b) *void SendMsgToSubscribers*（*char*×*myPort*，*SpParameterSet&pset*，*double delay*=0)；用来向输出端口的订购者发出事件调度.

④ 取消调度接口

void CancelMessage(*SpCancelHandle&handle*)；用来取消事件调度.

YHSUPE 下的 EDEVS 组件继承 SpBaseComponent 类，使用此类提供的组件接口实现自身的逻辑并与其它组件进行交互.

图 6 和图 7 是利用本文实现的 YHSUPE 代码生成器转换 4.1.1 节以及 4.1.2 节所建立的图形化模型所生成的 YHSUPE 模型的代码片段. 其中图 6 是头文件片段，图 7 是实现文件片段. 尽管各并行仿真平台的 API 函数名称不同，但实现的功能集合都大同小异. 只要提供相应的代码生成器，就能将 Visicom 图形化建模生成的 EDEVS 模型转化为相应并行仿真平台下的实现代码，从而实现仿真平台间的模型重用.

```
class integral :public SpBaseComponent {
//以下代码由模型解析工具自动生成请勿修改
private://组件公共框架代码
        typedef void (integral::*FuncPtr)(SpParameterSet &);
        typedef map<string,FuncPtr,less<string> > FuncMap;
        FuncMap m_FuncMap;
        void RegisterPortHanler( string , FuncPtr );
        void ClearPortMap();
        virtual void MsgHandler(SpParameterSet &in,string & port);
        virtual void ConfigHandler(SpParameterSet &para);
public://组件公共框架代码
        virtual void Init();
        virtual void PrimaryEvent();
        virtual void Terminate(double tend);
        virtual void DiscoverFo(SpFo *fo);
        virtual void RemoveFo(SpFo *fo);
private://组件私有状态变量
        FO_double step;
        FO_double integral;
        FO_double begin;
        FO_double end;
private://端口处理函数
        void PrimaryEvent(SpParameterSet &in);
        void Integral0pration(SpParameterSet &in);
        void ans(SpParameterSet &in);
};
SUPE_DEFINE_COMPONENT(da0ee977_b497_4ff0_b802_f9e5ef02cfea,integral)
#endif
```

图 6　YHSUPE 组件头文件

```
void integral::ConfigHandler( SpParameterSet &para ){//组件配置响应函数
        //在此添加组件响应配置请求的代码
        double tmpstep;
        if( para.GetDouble("step",tmpstep)==GoodStatus ){
                step = tmpstep;
        }
        double tmpbegin;
        if( para.GetDouble("begin",tmpbegin)==GoodStatus ){
                begin = tmpbegin;
        }
        double tmpend;
        if( para.GetDouble("end",tmpend)==GoodStatus ){
                end = tmpend;
        }
}
void integral::IntegralOpration(SpParameterSet &in){//端口处理函数
        //定义调度事件所用的SpParameterSet变量
        SpParameterSet SelfStepParaSet;
        SpParameterSet sendresultParaSet;
        double sendresult_answer =0;
        //事件处理逻辑
        double t = SpGetSimTime().GetTime();
        integral = integral + (t*begin)*(t*begin);
        sendresult_ans = integral;
        //调度事件IntegralOpration
        if(t*begin<end) {
                SendMsgToPort( "IntegralOpration",SelfStepParaSet,step);
        }
        //调度事件ans
        sendresultParaSet.SetDouble("answer",sendresult_answer);
        if(!(t*begin<end)) {
                SendMsgToPort( "ans",sendresultParaSet,0);
        }
```

图 7 YHSUPE 组件实现文件

5 总 结

由于缺乏高效统一的可视化组件建模范式，国内外并行仿真平台往往存在二次建模困难、编程调试复杂和模型难以在平台间重用的问题. 本文针对当前广受关注的 PDEVS 规范所存在的不支持原子模型可视化建模、事件处理逻辑过于集中、无法挖掘处理逻辑内部并行性的缺点，提出了基于 MDA 的并行仿真可视化组件建模范式——EDEVS，证明了该范式的描述能力不弱于 PDEVS，并实现了基于该范式的可视化建模框架及 EDEVS 建模框架到 YHSUPE 仿真平台建模框架的映射，改善了 YHSUPE 的建模框架，使其能够满足大规模并行仿真分布、快速建模的需要. EDEVS 继承了 PDEVS 的优点，通过基于端口调度的组件化技术实现模型之间的解耦合，提高模型的可重用性；通过模型驱动的建模技术将建模过程与模型的代码实现分开，使建模人员只需关注建模，而无需关注模型在具体平台上的实现，降低了建模复杂度，同时支持模型在仿真平台间的重用，通过在原子模型的定义上引入内部端口，EDEVS 将 PDEVS 的外部处理函数和内部处理函数划分为与外部端口和内部端口相对应的端口处理函数，从而可充分开发各端口处理逻辑的并行性，在不违反因果关系的前提下实现更细粒度的并行，同时提供端口定义及端口之间调度关系的可视化方法，改善了建模效率.

参 考 文 献

[1] Yao Yi-Ping, Zhang Ying-Xing. Solution for analytic simulation based on parallel processing. Journal of System Simulation, 2008, 20(24): 6617-6621(in Chinese)
(姚益平，张颖星. 基于并行处理的分析仿真解决方案. 系统仿真学报，2008，20(24)：6617-6621).

[2] Qing Du-Zheng, Li Bo-Hu, Zhang Liang, Zhou Ming, Zhang Han, Li Zhi-Ping. Research of component-based integrated modeling and simulation environment. Journal of System Simulation, 2008, 20(4): 900-904(in Chinese)
(卿杜政，李伯虎，孙磊，张良，周敏，张晗，李志平. 基于组件的一体化建模仿真环境(CISE)研究. 系统仿真学报，2008，20(4)：900-904)

[3] Li Bo-Hu, Chai Xu-Dong, Zhu Wen-Hai, Di Yan-Qiang, Wan Peng, Shi Guo-Qiang, Tan Juan, Yin Run-Min, Hou Bao-Cun. Some Focusing Points in development of modeling and simulation technology. Journal of System Simulation, 2004, 16(9): 1871-1878(in Chinese)
(李伯虎，柴旭东，朱文海，邸彦强，王鹏，施国强，谭娟，殷润民，侯宝存. 现代建模与仿真技术发展中的几个焦点. 系统仿真学报，2004，16(9)：1871-1878)

[4] Steinman J, et al. A proposed open system architecture for modeling and simulation//Proceedings of the Fall 2007 Simulation Interoperability Workshop. Ligure, Italy, 2007, 07F-SIW-044

[5] Steinman Jeff. Scalable distributed military simulations using the SPEEEDS object-oriented simulations framework//Proceedings of the Object-Oriented Simulation Conference (OOS'98). California, USA, 1998: 3-23

[6] Wang Cheng-Jun. Architecture driven component development for top-down software reuse//Proceedings of the International Conference on Computer Science and Software Engineering. Wuhan, China, 2008, 5: 1349-1352

[7] Ismail, Suryani, Wan-Kadir, Wan M. N, Saman, Yazid M, Mohd-Hashim, Siti Z. A review on the component evaluation approaches to support software reuse. International Symposium on Information Technology, Kuala-lumpur, Malaysia, 2008, 4: 1-6

[8] Jin Wei-Xin. Large-Scale Simulation System. Beijing: Electronic Industry Press, 2004(in Chinese)
(金伟新. 大型仿真系统. 北京：电子工业出版社，2004)

[9] Ahmet Zengin, Hessam Sarjoughian. Devs-suite simulator: A tool teaching network protocols//Proceedings of the 2010 Winter Simulation Conference. Maryland, USA, 2010: 2947-2957

[10] Li Qun. Simulation Model Portability Specification and Application. Beijing: Electronic Industry Press, 2009(in Chinese)
(李群. 仿真模型可移植性规范及其应用. 北京：电子工业出版社，2009)

[11] Anneke Kleppe. Mda Explained, the Model Driven Architecture: Practice and Promise. Massachusetts, USA: Addison-Wesley, 2003

[12] Liu Gang, Yao Yi-Ping, Peng Shao-Liang. Edevs: A scalable devs formalism for event-scheduling based parallel and

distributed simulations//Proceedings of the 13th IEEE/ACM Symposium on Distributed Simulation and Real-Time Applications. Singapore, 2009: 239-242

[13] Chen Xin. Research of U. S Military Modeling and Simulation Object Model Framework. Beiing: Military Science Press, 2008(in Chinese)
(陈欣. 美军建模仿真对象模型体系框架研究. 北京：军事科学出版社，2008)

[14] Maria Valinski, Jonathan Driscoll. Providing a parallel and distributed capability for JMASS using SPEEDES. Enabling Technologies for Simulation Science VI, Proceedings of SPIE, Orlando FRANCE, 2002, 4716: 150-159

[15] Zhang Wei-Hua. Research on the key technologies of joint modeling and simulation for both engineering and engagement-level. Changsha: National University of Defense Technology, 2006(in Chinese)
(张卫华. 工程级与交战级联合建模与仿真关键技术研究. 长沙：国防科技大学，2006)

[16] Di Hua. Joint modeling and simulation system overview. Jiangsu: Information Command Control System and Simulation Technology, 2001(in Chinese)
(棣华. 联合建模与仿真系统概述. 江苏：情报指挥控制系统与仿真技术，2001)

[17] Simulation Interoperability Standards Organization. Base object model (BOM) template specification volume I—Interface BOM. SISO-STD-003.1-TRIAL-USE-V0.9, 2004. http://www.boms.info/spec/BOM_VolI_v0_9_trialuse.pdf

[18] SISO Base Object Model Product Development Group. Guide for base object model (BOM) use and implementation. SISO-STD-003-2006, 2006

[19] Zeigler B P, Praehofer H, Kim T G. Theory of modeling and simulation. San Diego, USA: Academic Press, 2000

[20] Liu Gang, Yao Yi-Ping, Liu Bu-Quan. VISICOM: A component-based parallel discrete event modeling framework//Proceedings of the 2nd International Conference on Advances in System Simulation (SIMUL 2010). Nice, France, 2010: 158-163

YAO Yi-Ping, born in 1963, Ph. D., professor. His research interests include supporting technologies for high performance parallel simulation.

LIU Gang, born in 1981, Ph. D. candidate. His research interests include parallel discrete event system modeling and simulation.

Background

Historically two classes of simulation applications have received the most attention: analytic simulations and virtual environments. Analytic simulations usually attempt to capture detailed quantitative data concerning the system being simulated, typically include limited or no interaction with human participants or physical devices during the execution of the simulation program, and usually execute "as-fast-as-possible". The application area of analytic simulation includes war gaming simulations, telecommunication network simulations, digital logic circuits and computer system simulations, and protein folding simulations etc. Parallel simulation is an effective way to achieve running performance, as the scale of the simulation grows huge. Building models for large scale parallel simulations start from scratch every time is inefficient. How to provide a modeling method which can provide an easy way to create and encapsulate models and support model reuse is a hot-spot in parallel simulation field. This paper extends the DEVS (Discrete Event simulation specification) and proposes a MDA based Visual Component formalism, called EDEVS (Event-Scheduling Discrete Event simulation Specification) for the existing parallel simulation platforms. This formalism takes its advantage in supporting hierarchical decomposition of large models, and providing a visual way to develop models.

This work has been funded by the Foundation of Doctoral Scholarship in China (No. 200899980004), the National Natural Science Foundation of China (No. 60773019).