中图分类号: TP 391. 9 文献标识码: A

一种基于 Modelica 语言的混合系统 DEVS 模型架构

陈 彬,张 鹏,刘晓铖

(国防科技大学 机电工程与自动化学院,湖南 长沙 410073)

摘 要: Modelica 语言采用微分方程描述系统,此外它还具备面向对象编程语言的特性,因此它不仅适用于连续系统的建模,还支持离散系统的模型架构。因此,可以将 Modelica 作为一种混合系统的建模语言。提出了一个 Modelica 语言描述的 DEVS(Discrete EVent System specification 离散事件系统规范)模型架构,并通过对模型的编译过程产生 C + + 代码,获取了同时描述连续系统和离散系统建模的能力。最后给出了用 Modelica 语言描述的一个飞机导航控制连续-离散仿真系统的例子。

关键词: Modelica; 离散事件系统规范; 量化积分器; 离散事件系统规范-Modelica 编译器

A Modelica Based Hybrid DEVS Model

CHEN Bin, ZHANG Peng, LIU Xiaocheng

(Institute of Electromechanical Engineering and Automation, National University of Defense Technology, Changsha 410073, China)

Abstract: Modelica describes the system by equations, which is good to modeling for continuous system. In addition, as the feature of object-orientation, Modelica can well support the discrete system modeling architecture. As a result, Modelica is the best choice for hybrid system modeling. The paper introduces the features and specification of Modelica first. Then the DEVS (Discrete EVent System specification) model architecture in Modelica is proposed. Through the compiling process, the C + + code is generated to support the continuous and discrete system. In the end, the example of flight navigation and control system is given.

Key words: Modelica; Discrete EVent System specification; quantified integrator; Discrete EVent System specification-modelica compiler

1 引 言

Modelica 是一种面向对象的描述语言,它提供了一种结构化,计算机支持的方法进行数学建模(基于方程)。Modelica 作为一种标准和成熟的通用建模语言,具有良好的面向对象特性,支持连续离散混合模型,可以用于作战模拟中的多领域混合系统建模与仿真。DEVS 是一种通用仿真形式化方法[1],虽然是为离散模型设计的,但由于其定义的精确性和灵活性,也可以支持连续模型的仿真[2-3]。不过由于 DEVS 的数学描述形式不足以支撑其仿真实现的标准化,因此需要用一种建模语言支持 DEVS 模型的标准化和编译过程,从语法和语义两个层次辅助检测和可执行体的生成过程。

利用 Modelica 语言描述 DEVS 模型可以为其

基金项目: 国家"863"高技术研究发展计划资助项目(2008AA1212021); 国家自然科学基金资助项目(91024030)

提供良好的语法约束和编译检查手段。文献[4]和[5]已经讨论了在Modelica中描述DEVS模型的方法及其在CD++中的应用。但是在转换过程中,需要首先转换为键结图(Bond Graph),这一变换手续,增加了模型变换的复杂度。

本文先简要介绍Modelica规范,讨论用Modelica表示DEVS规范的意义;其次在Modelica语法元素的基础上提出如何在Modelica中表示DEVS,给出Modelica-DEVS架构表示离散模型和连续模型的方法;最后,详细介绍了Modelica-DEVS编译器的结构和实现,并结合飞行导航控制系统的例子,给出了用Modelica-DEVS架构实现的混合仿真系统。

2 Modelica语言简介

Modelica的主要目标是能够容易地交换模型及使用模型库。Modelica是基于微分代数方程形式化(DAE)建立的,同时也包括一些离散事件特性,以解决不连续性和对系统的采样。因此,Modelica是一种多形式化、多领域和通用的建模语言[3]。

Modelica对于面向对象的视角和一般面向对象语言不同。因为Modelica关注于结构化的数学模型,面向对象特性被视作是一种结构化的概念,以描述复杂大系统。在Modelica中,交互由连接完成。基本思想是每个组件都有连接器(connector),通过连接器输出和接收数据。不同组件的连接器可以连接在一起以组成连接关系。因此,一个组件的输出可以连接到一个组件的输入。连接关系会自动地由基于模型方程的编译器演绎表达出来。Modelica是一种声明性语言,系统的动态特性都由数学方程声明确定。下面简要介绍Modelica的类。

class:Modelica的最基本结构元素也是类。但Modelica的类与其他语言中类的结构不同,最显著的区别就是Modelica用类描述模型行为的方式。尽管Modelica中的方程与其他语言一样,用赋值的方式表达,但是Modelica的方程仅仅表示变量之间的等价关系,在方程变量之间不存在赋值关系。在Modelica中,方程'a = b + c'表达变量a,b和c之间的关系,和方程'b + c = a'的意义一样。Restricted Classes类是Modelica中的基本结构元素。Modelica中的类可以用关键字class定义。但是在特定条件下,关键字class可以被换成六种其他的、意义更加确定的关键字:model,connector,record,block,type和function。

model:model唯一的限制就是,这种受限类不能应用在连接中,其语义和Modelica中的通用类结构是一样的。

record:record类用来描述结构性的数据,在对其组件的定义中方程是不存在的。

type:一个type就是一种存在类的拓展。一个受限type类只可能是预定义类型的拓展,enumeration,record或array的type。

connector:connector类的限制与record类相同,除了connector类专门是设计用于建立连接关系的。

block:block受限类用于对因果(input/ouput)块框图进行建模。在Modelica中,input和output这两个关键词用在组件前缀上,从而确定数据流的方向。block中所有声明的变量都有前缀input或output。

function:function是数学模型中自然的组成部分。让用户可以定义数学函数。一个Modelica function的整体就是一个算法,确定当函数调用时候执行的行为。function的参数由关键字input确定,且结果保存在由关键字output标志的变量中。在Modelica的function中有一些限制:首先是在函数体中只允许一条algorithm语句。不容许存在equations和initial algorithms;其次,调用函数需要一条算法或外部函数的接口;第三,不能调用Modelica的内建操作符,der,initial,terminal,sample,pre,edge,change,reinit。

3 利用Modelica语言对混合系统描述

3.1 DEVS形式化基本描述元素

用Modelica语言描述的DEVS模型架构见图1。

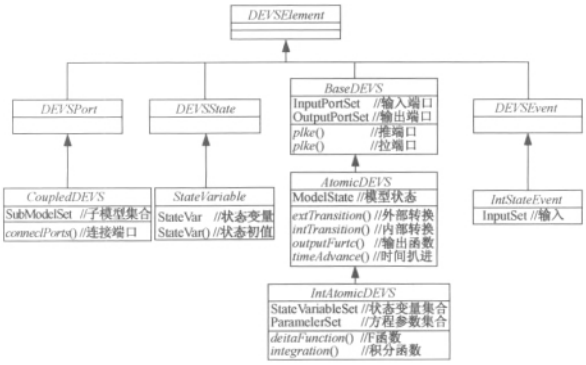

图 1　用 Modelica 语言描述的 DEVS 模型架构

Fig. 1 DEVS model structure described by modelica

如图 1 所示,利用 Modelica 语言描述 DEVS 形式化的基本描述元素包括 DEVSElement, DEVSPort,DEVSState,DEVSEvent 和 BaseDEVS。

(1) DEVSElement: 这是在 Modelica 语言环境中定义 DEVS 架构的最基本元素,其他 DEVS 类全都是由其派生出来的。利用预定义类的目的有: 首先,标识 DEVS 的架构 Modelica 描述都是从相应的预定义类拓展而来的; 其次,可以设计 Modelica 的数组结构,方便对数据集合的操作。利用基类定义函数的参数类型可以方便挂接子类的对象。

(2) DEVSPort: 拓展 Modelica 语言本身连接器 Connector 类的能力,附加上端口的唯一标识,并确定其可以传递的数据类型为 DEVSEvent。

(3) DEVSState: DEVS 规范没有给出状态的定义。此处给出 DEVSState 类,表示原子 DEVS 模型的状态变量。每个 DEVS 原子模型有一个状态变量,表示其模型状态变量以及顺序状态。

(4) DEVSEvent: 同样,DEVS 规范也没有给出事件的定义。此处给出 DEVSEvent,表示 DEVS 模型之间传递的事件。事件传递的是模型的输出或者状态。

(5) BaseDEVS: 作为原子 DEVS 和耦合 DEVS 模型的父类,BaseDEVS 给出了模型的公共操作和端口集合。

3.2　对离散模型建立 DEVS 描述

Modelica 对离散模型的 DEVS 描述按照原子模型和耦合模型的规范进行定义。其中 AtomicDEVS 原子模型描述包括参数、顺序状态、模型状态、输入和输出端口以及行为函数。在参数声明部分,参数用于模型的初始化,顺序状态用 Modelica 的枚举类型来表示。模型状态由一般的 Modelica 类实例来表示。输入和输出端口由 input 和 output 关键字以及预定义类 DEVSPort 来表示。所有的函数都用普通 Modelica 函数来定义。时间推进函数需要一个输出参数 timespan,通过该函数可以确定达到下一个顺序状态的时间间隔。AtomicDEVS 原子模型定义如表 1 所示。

表 1　原子模型的 Modelica 定义

Tab. 1　Definition of atomic model in Modelica

```
class modelname
    extends AtomicDEVS;
    parameter declarations { parameter Integer i; ..... }
    sequential states declaration { type SeqStates = enumeration( "
    sq1", "sq2", ..... ); }
    model state declaration { ModelState state( ); }
    input port declarations { input DEVSPort p_in1( ); ..... }
    output port declarations { output DEVSPort p_out1( ); ..... }
    function extTransition //外部状态转移函数
    end extTransition;
    function intTransition; //内部状态转移函数
    end intTransition;
    function outputFunction //输出函数
    end outputFunction;
    function timeAdvance //时间推进函数
    output Real timespan;
    end timeAdvance;
end modelname;
```

CoupledDEVS 耦合模型的定义拥有参数、输入和输出端口、子模型端口连接关系等部分。端口和参数的声明和原子模型中一致。子模型被声明为普通的 Modelical 类实例。端口连接由 Modelica 的 connect 函数定义。原本 connect 函数是用于连接两个 Modelica 的 connector,但是这里连接两个 DEVS 端口。CoupledDEVS 耦合模型定义如表 2 所示。

表 2　耦合模型的 Modelica 定义

Tab. 2　Definition of coupled model in Modelica

```
class modelname
    extends CoupledDEVS;
    parameter declarations
    input port declarations
    output port declarations
    sub - model declarations
    equation
      port connections { connect( m1. p_out, m2. p_in); ..... }
end modelname;
```

3.3 对连续模型建立 DEVS 描述

Modelica 利用一阶微分方程对连续模型进行描述，根据文献[7]，连续系统 DESS 可以经由 DTSS 描述误差任意小的变换为 DEVS 系统。所以在 Modelica 中描述连续系统，首先要按 DESS 规范进行描述，即将 equation 中的方程组，按照 DESS 的要求表述。值得注意的是，equation 正是按照一阶微分方程组的形式对模型进行描述的。因此，equation 自然地可以描述 DESS 模型的微分方程部分，而其仿真是利用积分器来实现的，利用计算机求解，必须对系统进行量化，变换为 DTSS 系统。因此仿真的关键内容就是构造量化积分器，量化的过程会根据时间步长的变化产生一定的误差。DTSS 可以无误差变换为 DEVS 模型。所以描述连续系统的核心工作就是构造 DEVS 量化积分器。

基于前述对离散 DEVS 模型的描述，拓展延伸一种 DEVS 量化积分器的描述。也即，将 Modelica 原有的 equation 的描述方法，变换为利用 DEVS 积分器的描述。因此，根据微分方程组的特点，在积分器中要声明微分方程的状态变量集合，保存变量的积分状态的变量集合及其积分方程。量化积分器的定义如表 3 所示。

表 3 DEVS 量化积分器的 Modelica 定义

Tab. 3 Definition of DEVS quantification integrator in Modelica

```
class modelname
    extends IntAtomicDEVS;
    parameter declarations { parameter Real p11; ... p1m;
                ... pn1; ... pnn;}
    state variables declaration { StateVariable v1; v2; ... vn;
                f1; f2; ... fn;}
    input port declarations { input DEVSPort p_in_u1();
                ... p_in _um();}
    output port declarations { output DEVSPort p_out_v1();
                ... p_out _vn();}
    function deltaFunction //F 函数
    end deltaFunction;
    function integration; //积分函数
    end integration;
end modelname;
```

在离散模型的 DEVS 的基础上定制了参数变量集合和状态变量集合，另外，量化积分器的输出端口和状态变量紧密联系，有多少个状态变量就有多少个输出端口；而输入端口则与输入有关。针对状态变量和输入事件的需求，从 DEVSState 和 DEVSEvent 衍生出了 StateVariable 和 IntDEVSEvent。StateVariable 定义状态变量的标识符和初值，而 IntDEVSEvent 则封装了量化积分器的输入集合。

利用参数集合和状态变量集合在 deltaFunction 中构造 F_1 至 F_n，再利用 integration 进行积分计算。值得注意的是，量化积分器的时间推进函数返回的是定时间步长值 h。h 决定了 DEVS 系统与原有 DESS 微分方程系统之间的误差大小。

4 利用 Modelica-DEVS 编译器实现混合系统 DEVS 仿真

4.1 编译器的结构

Modelica 本身是模型描述语言，只有将其模型描述翻译为程序语言，才能在特定语言(Language Specific)的 DEVS 仿真器中运行。这就是 Modelica 模型编译器需要完成的任务。这里利用 C++作为目标语言来将 Modelica-DEVS 模型编译为 C++表述的 DEVS 模型。

同一般编译器的结构一样，Modelica-DEVS 编译器也包括解析器、符号表构建器、类型检查器以及代码生成器。如图 2 所示。

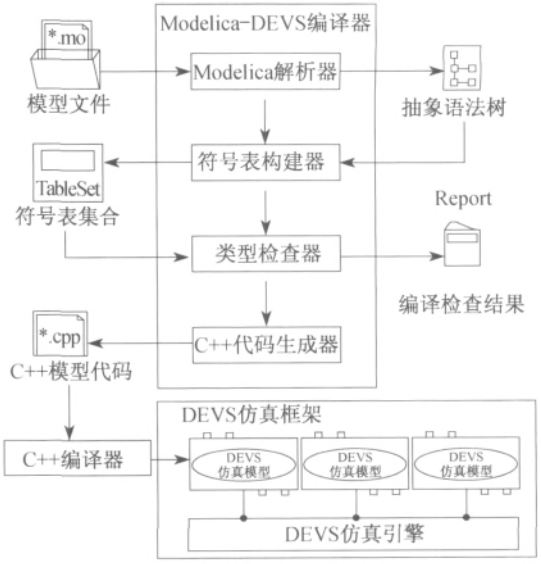

图 2 Modelica-DEVS 编译器的结构

Fig. 2 Structure of Modelica-DEVS compiler

(1) 解析器读入用 Modelica 语言描述的模型文本,构建抽象语法树(Abstract Syntax Tree) 。

(2) 符号表构建器在抽象语法树的基础上建立符号表组。

(3) 类型检查器根据 Modelica 语法对语法树和符号表进行检查,整理出错误,形成编译结果。

(4) C + + 代码生成器则是遍历语法树,生成用 C + + 代码表述的 DEVS 仿真模型。

(5) 利用 C + + 的编译器将 DEVS 仿真模型编译为可以挂接到 DEVS 仿真引擎上运行的模型模块。

前三个部分是编译器对 Modelica-DEVS 模型的编译,与对普通的 Modelica 模型进行编译没有区别。Modelica-DEVS 编译器的关键就是将模型编译为 DEVS 仿真模型。这里采用的离散事件仿真引擎是用 C + + 实现的,所以要求编译器生成 DEVSpp,嵌入到仿真引擎。与 Modelica-DEVS 架构类似,DEVSpp 是一种编程语言级的仿真模型,它也包括几种 DEVS 基本组件: Port, DEVS, Atomic 和 Coupled。

下面详细介绍如何将 Modelica-DEVS 模型映射为 DEVSpp 模型。代码生成的映射分成三个部分: 第一是 DEVS 原子模型,包括普通原子模型和积分器原子模型的映射; 第二是 DEVS 耦合模型的映射; 第三是其他 DEVS 架构类的映射,如 DEVSEvent 和 DEVSState。

如前所述, Modelica 描述的原子模型 AtomicDEVS 包括参数、端口、状态和转换函数。原子模型的映射就是将模型的各个部件映射为 DEVSpp 中 Atomic 的派生类。并且,针对 C + + 类的特点,将部件映射为类的声明和定义两个部分。对于参数,直接将其变换为类中基本数据类型的声明; 对于端口,变换成 Port 给出声明,然后利用 DEVS 基类的添加端口函数产生实例; 对于状态,利用变换过来的 DEVSState 类进行声明。而对于转换函数的描述,Modelica 和 C + + 有所区别。在 DEVSpp 中,需要先将函数及其形参在类中进行声明,然后再给出详细的定义。需要指出的是针对积分器原子模型的映射,需要对其参数和状态进行初始化工作,所以就要在 Atomic 的派生类中增加一个初始化函数 Init( ) ,对端口的添加,给参数集合和状态集合赋初值等。另外,还要变换构造出积分函数 deltaFunction( ) 和 integration( ) ,并将这两个函数分别嵌入到外部和内部状态转换函数中。调用 deltaFunction( ) 后,时间推进函数返回值为零,表示马上要进行积分操作; 调用 integration( ) 后,时间推进函数返回值为时间步长 h。进入下一个积分步骤。

Modelica 描述的耦合模型 CoupledDEVS 拥有参数,端口模型实例(子模型) 以及连接。映射将耦合模型变换为 DEVSpp 中由 Coupled 派生出的类。参数和端口的变换和原子模型相同,增加 Init( ) 完成初始化工作。在实例化子模型时除了参数以外还需要给每个子模型一个名字作标识,利用 DEVS 基类中的 addSubModel( ) 函数添加。对于端口连接函数也替换为 DEVS 中的 connectPorts( ) 函数。

对于 DEVSEvent 和 DEVSState 这种 DEVS 的数据信息类,只具有属性,所以可以直接将其变换为 C + + 中类的定义。对于 Modelica 属性和参数中应用的基本数据类型: Real, Integer, Boolean 和 String,将它们直接变换为 C + + 的内建数据类型 float, int, bool 和 string。

4.2　编译器的实现

编译器的实现建立在 μModelica [8] 编译器的基础上,μModelica 是一种 Modelica 模型编译器,它支持 Modelica 规范 1.6 中规定的所有 Modelica 特性。对文本模式的 Modelica 模型解析后生成可用于积分计算的 Octave 文件。这里借用了 μModelica 的前端(Front End) 和后端(Back End) ,将代码生成部分改造成 DEVSpp 仿真模型生成器。下面详细阐述实现方法。

利用 μModelica 解析可以获得 Modelica-DEVS 模型的抽象语法树,这个语法树是基于访问者模式(Visitor Design Pattern) 生成的。所以利用编译器生成代码的过程就是创建一个访问者(visitor) 遍历整个树的过程。访问者要在访问语法树的不同节点时,实现前述映射过程。

图3就是抽象语法树AST的主要组成元素，包括ClassFile，RegularClassDefinition，ElementList，EquationPart，AlgorithmPart和ComponentClause。

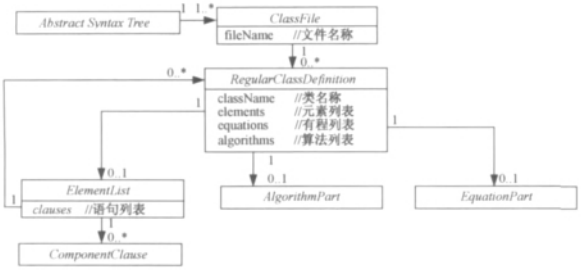

图3 抽象语法树的结构

Fig. 3 Structure of abstract syntax tree

ComponentClause：是一个通用的术语，它可以是变量声明、参数声明等所有Modelica声明。

ElementList：是一个队列，可以有零个或多个ComponentClauses，或者零个或多个Regular ClassDefition。

EquationPart和AlgorithmPart：是Modelica中对模型行为的定义。Equations构建方程，确定连续模型中状态变量之间的等价关联关系；Algorithm描述模型的编程语言特性。

RegularClassDefinitions：是Modelica中一种通常意义下对类的定义，可以是一个package，class，function或是一个restricted class的定义。

ClassFile：与Modelica源文件有一一确定关系，当编译器编译多个文件或源文件要从其他源文件导入类时，一棵抽象语法树可能有多个源文件，所以AST也可能有多个ClassFile。一个ClassFile由0个或多个RegularClassDefinitions组成。

在访问上述节点的过程中根据架构类的映射产生对应C++类，最终生成DEVSpp仿真模型。代码的生成由一系列以make为前缀的函数实现，包括makeIncludes()，makeParameters()，makeFunctionInputs()，makeClassInputs()和makeClassOutputs()。下面依次介绍这些函数：

makeIncludes()：生成模型的模型都是由DEVSpp预定义的类派生而来，所以在代码中要包含预定义类的声明文件。因此，默认要产生对DEVS，Atomic，Port和Coupled声明文件的包含语句。另外，如果还要处理Modelica-DEVS中对其他模型Package的包含。例如模型A的Modelica中引入了模型B，在makeIncludes中就要增加对变换为C++类的B的声明文件的包含。

makeParameters()：在Modelica中，类参数的声明和类变量的声明一致，唯一的区别是参数声明之前有parameter关键字。在DEVSpp中，需要在类声明中对参数给出声明，并在函数init()中进行初始化。如表4中例子。

表4 Modelica参数的C++代码实现

Tab. 4 Implementation of Modelica parameters in C++

Modelica Class:	C++ Class Declaration:	C++ function Definition:
class *A*	**class** *A*: public Atomic	**void** A. init()
parameter Integer *p*1;	int *m_para_p*1;	{
parameter Real *p*2;	float *m_para_ p*2;	*m_para_p*1 = 0;
…	…	m_para_p2 = 0;
end *A*;	**end** *A*;	}

makeFunctionInputsOutputs()：在Modelica中，函数也是类，一个Modelica函数的输入参数由一个拥有关键字input变量的声明确定。在C++中，函数参数的确定和类不一样。所以要将一个Modelica函数的输入参数变为对应Python函数的输入参数。如表5中例子。

表5 Modelica中函数的C++代码实现

Tab. 5 Implementation of Modelica Functions in C++

Modelica Class:	C++ Class Declaration:	C++ function Definition:
function foo	**class** *A*: public Atomic	**int** *A*. foo(int *a*)
input Integer *a*;	…	{
output Integer *b*;	int foo(int *a*) ;	…
…	…	return *b*;
end foo;	**end** *A*;	}

makeClassInputs()和makeClassOutputs()：正如根据前文提到的，将Modelica类的输入和输出端口定义用DEVSpp预定义类的函数代替，这些工作由makeClassInputs()和makeClassOutputs()完成。makeClassInputs()对应将Modelica的输入

端口声明变换为 init( ) 函数中对 DEVSpp 的 addInputPort( ) 的调用,而 makeClassOutputs( ) 则负责输出端口声明的变换,即对 addOutputPort( ) 的调用。表 6 给出了一个例子。

表 6　Modelica 中端口的 C + + 代码实现

Tab. 6　Implementation of Modelica ports in C + +

Modelica Class:	C + + Class Declaration:	C + + function Definition:
class *A*	class *A*: public Atomic	void *A*. init( )
input DEVSPort *p_in*( ) ;	IPort *p_in*;	{
output DEVSPort *p_out*( ) ;	OPort *p_out*;	*p_in* = addInputPort ('p_in')
…	…	*p_out* = addOutputPort ('p_out')
end A;	end *A*;	}

5　飞行导航控制系统用例

本文采用飞行导航控制系统为对象进行基于 Modelica-DEVS 的混合系统仿真。飞行导航控制系统的结构如图 4 所示。系统分为两个部分: 导航系统(离散) 和飞行控制系统(连续) 。它们的工作原理如下:

(1) 导航系统根据任务计划装订飞行航迹,在感知当前位置的前提下制定运动轨迹规划,随后分解规划任务,设定飞行控制系统的工作状态。导航系统的工作过程是离散的,只有飞机抵达规划指定位置时才会工作,向飞行控制系统输出指令信号。

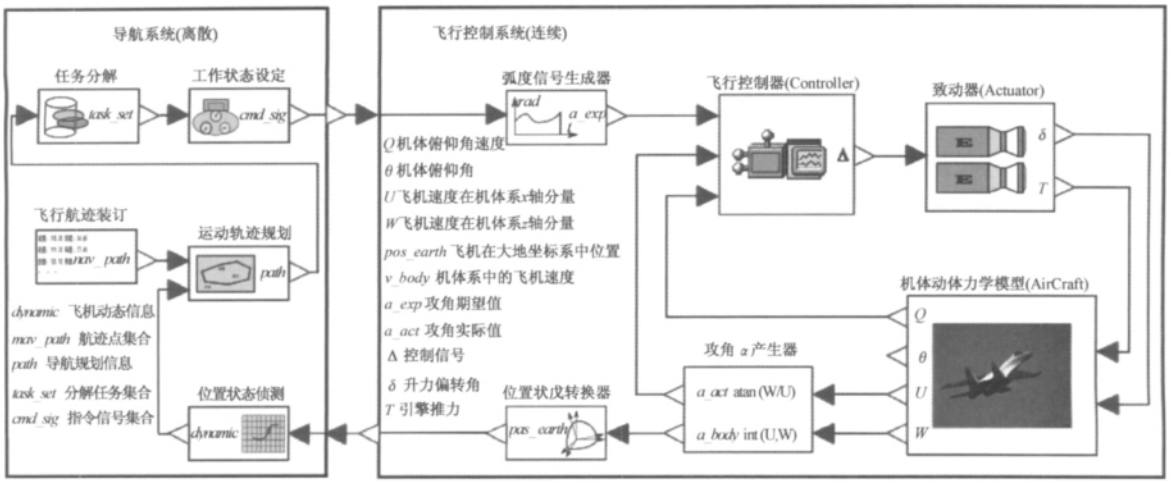

图 4　混合系统仿真用例:飞行导航控制系统

Fig. 4　Case study of hybrid system simulation: navigation and control system of flight

(2) 飞行控制系统是一个连续的控制系统,由弧度信号生成器、飞行控制器、致动器、飞机动力学模型、攻角产生器和位置状态转换器组成。其中飞行控制器是一个典型的连续系统,由微分方程组表示。位置状态转换器则是将飞机体系上的速度积分变换为飞机当前位置,并转换到经纬度坐标系上,它把位置和速度等机体动态信息按时间步长间隔传递回导航系统。

该系统是一个典型的混合系统,需要应用混合仿真体制对其进行建模和仿真。下面给出位置状态侦测和机体动力学模型子模块的 Modelica-DEVS 描述。

位置状态侦测子模块的功能是监测飞机的动态信息,在满足设置条件的情况下上报飞机动态信息并激活导航系统进行下一个周期的任务规划。这是一个典型的离散事件模型,在 Modelica-DEVS 模型架构下建立的模型描述如表 7 所示。

机体动力学模型是一个连续模型,其微分方程和参数描述如下所示:

表7 位置状态侦测模型的 Modelica 描述

Tab.7 Model of Position Monitor in Modelica

```
class DynamicMonitor
    extends AtomicDEVS;
    parameter Real m_para_lon; m_para_lat; m_para_hei; //当前位置
    parameter Real m_para_dlon; m_para_dlat; m_para_dhei; //指定位置
    parameter Dynamic m_para_dynamic; //动态信息
    …
    type SeqStates = enumeration("Monitor","Report"); //模型状态
    model state declaration {ModelState state();}
    input DEVSPort p_in_position(); //位置输入端口
    input DEVSPort p_ in_velocity(); //速度输入端口
    input DEVSPort p_ in_posture(); //姿态输入端口
    output DEVSPort p_out_dynamic(); //动态信息输出端口
    function extTransition
      …
      if m_para_lon = = m_para_dlon; //判断是否达到指定位置
      …
      state. SeqState = 'Report'; //重置状态
      timespan = 0;
    end extTransition;
    function intTransition;
      state. SeqState = 'Monitor';
      timespan = ∞;
    end intTransition;
    function outputFunction
      m_para_dynamic. lon = m_para_lon; //融合动态信息
      …
      poke(m_para_dynamic, p_out_dynamic); //输出动态信息
    end outputFunction;
    function timeAdvance
    output Real timespan;
    end timeAdvance;
end DynamicMonitor;
```

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix}U\\W\\Q\\\theta\end{bmatrix}=\begin{bmatrix}-WQ\\UQ\\0\\Q\end{bmatrix}+\begin{bmatrix}-g\sin(\theta)\\g\cos(\theta)\\0\\0\end{bmatrix}+\begin{bmatrix}T/m\\0\\0\\0\end{bmatrix}+$$

$$0.5\rho v^2 S\begin{bmatrix}(C_x+C_{x,\delta}\delta)\alpha/m\\(C_z+C_{z,\delta}\delta)\alpha/m\\\bar{c}(C_M+C_{M,\delta}\delta)\alpha/m\\0\end{bmatrix}$$

式中:g 为重力加速度;T 为发动机推力(控制输入,kN);m 为飞机质量(常量,kg);ρ 为空气质量(假设也为常量);v 为飞机速度(m/s);α 为攻角 $\arctan(W/U)$;δ 为水平舵与机体平面偏转角(控制输入);$\bar{c}$ 平均空气动力弦角;S 机翼面积;C_x,C_z,C_M 为水平舵与机体平面水平时($\delta=0$)的攻角系数(由空气动力函数简化);$C_{x,\delta}$,$C_{z,\delta}$,$C_{M,\delta}$ 为水平舵与机体平面不水平时的攻角系数。

这是一个典型的连续系统模型。所以在构建 Modelica-DEVS 描述时,需要利用 DEVS 架构中的量化积分器 IntAtomicDEVS 来描述。考虑相对飞机飞行的快变化过程,机体某些参数属于慢变化,所以将其视为常量,可以把方程组化简。简化后的模型有四个状态变量 U,W,Q 和 θ,以及两个输入 T(推力)和 δ(偏转角)。引入参数的值,可以构建机体动力学的 Modelica-DEVS 描述如表8。

表8 体动力学模型的 Modelica 描述

Tab.8 Model of body dynamic in modelica

```
class AirCraft
    extends IntAtomicDEVS;
    parameter Real p14; p15; p17;
    parameter Real p24; p26; p27;
    parameter Real p37;
    parameter Real p43;
    StateVariable U; W; Q; θ; Uf; Wf; Qf; θf;
    StateVariable T; δ;
    input DEVSPort p_in_T(); p_in _δ();
    output DEVSPort p_out_U(); p_out_W(); p_out_Q(); p_out _θ();
    function deltaFunction
      Uf = p14 · sin(θ) + p15 · T + p17 · (W · Q)
      Wf = p24 · cos(θ) + p26 · δ + p27 · (U · Q)
      Qf = p37 · δ
      θf = p43 · θ
    end deltaFunction;
    function integration;
      U = U + Uf · h
      W = W + Wf · h
      Q = Q + Qf · h
      θ = θ + θf · h
    end integration;
    function intTransition
      …
      integration();
      timespan = h;
    end intTransition;
    function extTransition
      …
      deltaFunction();
      timespan = h;
    end extTransition;
    function timeAdvance
    output Real timespan;
    end timeAdvance;
end AirCraft;
```

将飞行导航控制系统的 Modelica 模型输入 Modelica-DEVS 编译器,生成 DEVSpp 模型。将这些模型加载到 DEVS 仿真引擎上,就可以进行仿真。图5是由仿真系统输出的飞机的期望攻角 α_exp 和实际攻角 α 在某一时间段内变化的情况。

可以看出,导航系统给出的设置信号由飞行

控制器执行,将飞机的攻角 α 在一定的时间内调整到期望的大小。

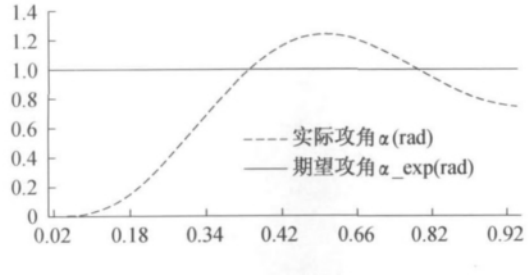

图 5　期望攻角 α_exp 和实际攻角 α 的变化情况

Fig. 5　Comparison of Expected value of attack angle and the real value of attack angle

6　总结和展望

本文从作战模拟的仿真需求出发,提出了用 Modelica 语言解决混合仿真中模型体制不同的问题。针对以往必须先将 Modelica 模型变换为键结图,再变换为 DEVS 模型的方法的不足,直接构造 Modelica-DEVS 模型架构描述的连续和离散的 DEVS 模型,并且给出了编译器的结构组成和用例实现。

在多范式建模与仿真的背景下,丰富多样的形式化方法都必须利用 DEVS 来描述。所以还需要继续丰富 Modelica-DEVS 架构,使其不仅仅能够适应连续模型,而是要能适应军事模型领域所涉及的所有建模形式化方法。这需要对 Modelica 的特性进一步研究,在其规范的指导下,提取基本元素,使 Modelica-DEVS 具有更加广泛的表达能力。另一方面,Modelica-DEVS 编译器仅仅有解释的功能是不够的,还需要根据特定领域的特点研究如何对生成的代码进行优化,从而提高仿真效率。总之,Modelica 为多范式建模提供了模型与仿真代码的桥梁,为进一步模型检测和性能优化工作提供了高层次语言描述平台。

参考文献:

[1] Vangheluwe Hans, Juan de Lara, Mosterman Pieter J. An introduction to multi-paradigm modelling and simulation [C] // In Proceedings of AIS. [s. l.]: AIS, 2002: 9 - 20.

[2] Vangheluwe Hans L M. DEVS as a common denominator for multi-formalism hybrid systems modeling [C] // Proceedings of the 2000 IEEE International Symposium on Computer-Aided Control System Design. [s. l.]: IEEE, 2000: 129 - 134.

[3] Kofman Ernesto, Lee J S, Zeigler B P. DEVS representation of differential equation systems: review of recent advances [C] // Proceedings of ESS'01. [s. l.]: ESS, 2001: 591 - 595.

[4] D'Abreu Mariana C, Wainer Gabriel A, M/CD + +: modeling continuous systems using Modelica and DEVS [C] // MASCOTS, 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems. [s. l.]: IEEE, 2005: 229 - 238.

[5] D'Abreu Mariana C, Wainer Gabriel A. Experimental results on the implementation of Modelica using DEVS modeling and simulation [J]. simulation, 2006, 38(1): 114 - 124.

[6] WU Yizhong, JIANG Zhansi, CHEN Liping. Design optimization of multi-domain model based on modelica [J]. Journal of System Simulation, 2009, 21(12): 3748 - 3752.

[7] Zeigler Bernard P. DEVS theory of quantized systems [R]. [s. l.]: DARPA, 1998.

[8] XU Weigao. The design and implementation of the μmodelica compiler [D]. [s. l.]: McGill University, 2005.

陈　彬　男(1981 -),安徽无为人,讲师,博士,主要研究方向为分布式仿真、基于 DEVS 的建模与仿真。

张　鹏　男(1987 -),重庆人,博士生,主要研究方向为分布式仿真系统的设计、面向对象仿真理论、知识工程。