Journal of System Simulation

Volume 28 | Issue 7 Article 7

6-4-2020

Implementation and Application of Complex System Modeling and Simulation Language Compiler

Zhou Wen
1. Beihang University, Beijing 100191, China;;

Chi Peng
2. Beijing Simulation Center, Beijing 100854, China;

Bohu Li
2. Beijing Simulation Center, Beijing 100854, China;

Song Xiao
1. Beihang University, Beijing 100191, China;;

Follow this and additional works at: https://dc-china-simulation.researchcommons.org/journal

Part of the Artificial Intelligence and Robotics Commons, Computer Engineering Commons, Numerical Analysis and Scientific Computing Commons, Operations Research, Systems Engineering and Industrial Engineering Commons, and the Systems Science Commons

This Paper is brought to you for free and open access by Journal of System Simulation. It has been accepted for inclusion in Journal of System Simulation by an authorized editor of Journal of System Simulation.

Implementation and Application of Complex System Modeling and Simulation Language Compiler

Abstract

Abstract: Simulation language is a key technology of integrated simulation based design software environment. A difficulty of developing simulation language is the compiler that automatically transfers simulation language Semantics to programming language such as C++. The contents and features of CMSL were introduced. And a compiler which could analyze simulation text written in CMSL was implemented. *A decouple method for Discrete System was proposed and integrated in the compiler.* The features object-oriented and components oriented of CMSL were verified by an example with its ability to describe discrete system.

Keywords

compiler, simulation language, lexical, grammar, decouple

Recommended Citation

Zhou Wen, Chi Peng, Li Bohu, Song Xiao. Implementation and Application of Complex System Modeling and Simulation Language Compiler[J]. Journal of System Simulation, 2016, 28(7): 1528-1538.

This paper is available in Journal of System Simulation: https://dc-china-simulation.researchcommons.org/journal/vol28/iss7/7

第 28 卷第 7 期
2016 年 7 月
系统仿真学报©
Journal of System Simulation
Vol. 28 No. 7
Jul., 2016

复杂系统建模仿真语言编译器的实现与应用

周文[1]，迟鹏[2]，李伯虎[2]，宋晓[1]
（1.北京航空航天大学，北京 100191；2.北京仿真中心，北京 100854）

摘要：仿真语言是实现一体化仿真设计环境的关键技术之一。仿真语言实现的难点是实现其编译器，即自动识别仿真语言语义并将其转换为 C++等编程语言。介绍了复杂系统仿真语言(CMSL)的构成及其特性，*研究了编译器的结构、工作机理，实现了一个面向 CMSL 文本的编译器。重点针对离散事件仿真的多层复合模型，提出并实现了一种内嵌于编译器的扁平化方法。*利用一个离散系统案例对 CMSL 的描述能力及编译器的编译功能进行验证，结果表明，CMSL 语言具有面向对象和面向组件的特性，其编译器能够编译基于 CMSL 的仿真文本，并将复合模型扁平化。

关键词：编译器；仿真语言；词法；语法；扁平化
中图分类号：TP391.1 文献标识码：A 文章编号：1004-731X (2016) 07-1528-11

Implementation and Application of Complex System Modeling and Simulation Language Compiler

Zhou Wen[1], *Chi Peng*[2], *Li Bohu*[2], *Song Xiao*[1]
(1. Beihang University, Beijing 100191, China; 2. Beijing Simulation Center, Beijing 100854, China)

Abstract: Simulation language is a key technology of integrated simulation based design software environment. A difficulty of developing simulation language is the compiler that automatically transfers simulation language Semantics to programming language such as C++. The contents and features of CMSL were introduced. And a compiler which could analyze simulation text written in CMSL was implemented. *A decouple method for Discrete System was proposed and integrated in the compiler.* The features object-oriented and components oriented of CMSL were verified by an example with its ability to describe discrete system.

Keywords: compiler; simulation language; lexical; grammar; decouple

引言

仿真语言及其支撑环境是建模仿真技术的重要组成部分，用于描述仿真模型及其连接关系(比如连续时间点上的数据和离散的事件)、定义仿真的运行参数，进而封装形成仿真组件并提供分布式仿真运行时引擎。它的优点是避免工程人员直接编程实现仿真模型、组件和引擎，减少工程人员的工作量，提高仿真模型的参数化，增强其可重用性。

回顾其发展历程，仿真语言首先以单领域建模软件的形式出现，如机械领域的 ADAMS、电子领域的 VHDL 和 Verilog、控制领域的 Matlab\Simulink[1]。单领域建模仿真软件在涉及学科单一的产品或产品部件研发时得到广泛应用。然而，在产品设计日益复杂的情况下，多领域协同设计成为新的产品设计模式，进而也涌现出多领域仿真语言，如数模电混合仿真的 VHDL-AMS、连续系统仿真语言 CSSL

收稿日期：2014-10-12 修回日期：2015-03-31；
基金项目：国家自然科学基金(61473013)；
作者简介：周文(1991-)，男，湖南，硕士生，研究方向为离散事件仿真系统设计与实现；迟鹏(1985-)，男，天津，工程师，研究方向为系统仿真；李伯虎(1938-)，男，上海，院士，研究方向为计算机仿真与集成制造。

及后来的 ASSL、离散系统仿真语言 Arena、连续离散混合仿真语言 Modelica [2]。随着多领域协同设计在工程应用的不断深化，仿真语言正逐步成为一体化仿真设计环境的重要工具之一。

仿真语言能够降低仿真工作者的编程门槛，但它不能直接编译执行，需要编译器将其转化为 C++ 等高级语言才能执行，因此研发出与仿真语言对应的编译器是将仿真语言集成到一体化建模仿真环境中的关键。目前已有大量相关的仿真语言一体化环境，如 DEVSim++[3]，Modelica/Dymola[2]，MSL[4] 等，然而描述仿真语言编译器如何实现的相关文献较少。

为了提高仿真语言的整体技术水平，本文主要研究了如何利用 flex 和 bison 工具[5]开发 CMSL 仿真语言的编译器，并设计了案例对编译器进行功能验证。

1 CMSL 语言介绍

1.1 CMSL 语言的构成

CMSL 语言是一种面向组件的模块化复杂系统建模仿真语言，既可以用于描述连续系统和离散系统，也可以用于描述连续离散混合系统。该语言以模块为单元对系统进行描述，主要包括 3 种模块：函数、模型及外部引用声明。函数模块用于描述系统方法，模型模块用于描述系统行为，外部引用声明则实现模型的重用。

1.1.1 函数模块

CMSL 的函数有 2 种定义方式：C++代码方式和 CMSL 方式。C++代码方式中，用户将利用 C++ 语言定义函数，如例 1 所示。CMSL 方式中，用户将利用更接近仿真语言的方式进行函数定义，如例 2 所示。

例 1：C++方式定义的函数

```
function real max(real a, real b)
{double c=a>b?a:b;
return c;}
end max
```

例 2：CMSL 方式定义的函数

```
function max
initial section
input real a, b;
output real c;
experiment section
c=a>b?a:b;
end max
```

C++代码风格要求用户在声明时用仿真语言的类型进行声明，在实现时用 C++的类型进行实现，用户在函数定义区能完全的使用 C++代码进行函数实现。

1.1.2 模型模块

模型模块包括初始段、模型段、实验段。初始段用于声明与模型描述及模型属性相关的变量，模型段用于描述模型的动态行为，实验段用于设置分析模型行为的方法。模型模块按元素组成分为复合模型模块和元素模型模块两种，复合模型模块中包括对已声明元素模型或复合模型的引用，元素模型则不包含任何模型引用。按求解方法分为离散模型、连续模型和混合模型。典型的模型模块如例 3：

例 3：模型模块

```
discrete element model container
initial section
int curNum=0;
parameter int maxNum;
input event ET_put;
output event ET_get;
state empty, idle, full;
model section
initial state empty
…
end state
state idle
….
```


https://dc-china-simulation.researchcommons.org/journal/vol28/iss7/7 2

第 28 卷第 7 期 系统仿真学报 Vol. 28 No. 7
2016 年 7 月 Journal of System Simulation Jul., 2016

```
end state
state full
…
end state
experiment section
…
end container
```

1.1.3 外部引用模块

外部引用模块用于实现跨文件引用模型和函数，从而提高模块的重用性。典型的外部引用如例 4：

例 4：外部引用

extern model container in container.cmsl;

例 4 外部引用语句描述了对例 3 中模型 container 的引用。

1.2 CMSL 语言的语法特点

一个语言的语法使用包含 3 个方面：符号的声明、符号的引用、语义的表述。CMSL 语言的语法在符号的引用和声明方面保留了 C++语法的灵活，在语义表述方面则吸收仿真语言的自然特性，使得仿真用户能快速的进行建模与仿真。

➢在变量声明方面：

CMSL 的变量声明方式在 C++的基础上增加了变量的使用限定声明，如：

input event ET_exp1，ET_exp2;

该声明限定 event 类型的变量 ET_exp1，ET_exp2 将作为模型的输入。

此外，用户可以利用模型模块定义自己的数据类型进行数据存储和交互。

➢在符号引用方面：

CMSL 要求在引用符号之前必须对符号进行声明，符号的引用方式与 C++一致，如调用例 1 中声明的 max 函数。

int a=3,b=4,c; c=max(a,b);

➢在语义表述上：

一方面，CMSL 保留了 C++传统的 if、while、for 等传统逻辑语句，但在表述方式上，CMSL 有所调整，如 if 语法调整为 if condition then action end if；另一方面，CMSL 增加了一些与仿真相关的语法，如 transition 语法、when 语法等。when 语法在仿真中用于描述与时间相关的条件处理，transition 语法用于离散事件系统建模中状态转移的描述。如

```
when time at beginServingTime+servingTime then transition deparking;
end when
```

表示当仿真时间到达 begin- ServingTime +servingTime 时执行 transition 语法表示的状态转移动作。

2 CMSL 编译器的结构及实现

一个语言编译器依赖于语言的词法和语法，其分析工作往往包含词法分析和语法分析两个部分[5]。

CMSL 编译器由词法分析器、语法分析器和解释器三者构成。词法分析器由 flex 工具编译生成，源文件包括l文件和定义辅助函数的 c 文件，经 flex 编译之后得到词法分析器源文件和头文件。语法分析器由 bison 工具编译生成，源文件包括 y 文件和定义辅助函数的 c 文件，经 bison 编译后得到语法分析器的源文件和头文件。解释器利用 c 语言进行开发，其源文件为 c 文件，同时词法分析器和语法分析器所需的辅助函数也可在该 c 文件中进行定义。之后利用 c 语言编译器 gcc 对词法分析器和语法分析器的源文件及头文件，解释器的源文件和头文件进行编译得到编译器的可执行文件 cmsl.exe。

Flex 和 bison 均为开源工具，在 linux 环境中可输入下面的指令下载和安装 flex 和 bison：

apt-get install flex bison

在 windows 环境中可在网上下载 mingw 和 msys 的压缩包，解压到一个文件夹中，之后在系统环境变量 path 中加入 mingw 及 msys 中 bin 的路径。随后可以在 cmd 中使用 flex 和 bison 进行词法

分析器和语法分析器的编译。此外，运行 msys 目录下的 msys.bat 文件可进行虚拟的 linux 环境，同样可以利用 flex 和 bison 工具。本文中的编译器在 windows 环境下实现。

2.1 基于 Flex 的词法分析器实现

词法分析器完成编译器的第一步工作，即将仿真文本分割成单个的语义元素。每一个语义元素都是由若干个语义字符组成，字符又按人们的使用习惯分为数字字符、字母及其他字符。

若干个语义字符组合在一起依据人们的习惯又被分为常值和符号。由于行为描述的需要，符号又进一步被划分为操作符和指示符。对于每一种语言，符号字符集中都会有一部分操作符和指示符被单独识别与常值一起构成整个语言系统的基元集合，一般而言，操作符集不为空。语言使用者则基于这两个集合来描述事物及其行为，从而衍生出相应于该语言的语义空间，一个好的描述语言总是力求具有较好的完备性和简洁性，即精简而完善的操作符集和指示符集。完备性意味着利用该语言总是可以区别性地描述事物及其行为，简洁性意味着语言系统的保留集合总是线性无关的。

CMSL 语言也有自己的保留字符集，而词法分析器的开发就是将 CMSL 语言的保留字符集用正则表达式表示出来，同时在匹配成功后，执行相应的辅助函数，并在最后返回恰当的记号给语法分析器，以便于语法分析器进行进一步分析。

Flex 开发的词法分析程序，其后缀名为 1，1 文件经 flex 编译后则生成 C 代码。词法分析程序包含 3 个部分，各部分之间用仅有%%的语句行来隔开。

第一部分包含声明和选项设置，该部分用于声明词法分析程序用到的相关变量，同时设置 flex 的编译特性。声明一般在%{和%}之间用 c 代码编写。第二部分则是正则表达式描述的字符组合及对应的动作。第三部分是会被原样拷贝的 c 代码。完成这三部分的代码编写即完成了词法分析程序的开发。其中，第二部分是最重要的，包含程序的词法描述；第三部分的 c 代码也可以放到解释器中编写。

CMSL 关键字包括 and, discrete, model,then 等。编译器文件的部分代码如下：

```
/* 操作符 */
"++"{return T_AA;}
"--"{return T_DD;}
/*常值*/
EXP([Ee][-+]?[0-9]+)
[0-9]+"."[0-9]*{EXP}? |
"."?[0-9]+{EXP}?{yylval->c=strdup(yytext);return T_NUMBER;}
\"[^\"\n]*\"    {yylval->c =strdup(yytext);return T_STRING;}
```

上述代码展示了如何利用正则表达式对操作符，关键字，常值进行描述。此外，1 文件有特殊的段落格式，具体格式见参考文献[1]。利用 flex 工具对完整的1文件进行编译即可得到词法分析器的 C 语言源代码。

2.2 基于 Bison 的语法分析器实现

词法分析器对仿真文本进行初步分析后，得到的是一个基于保留字符集的记号空间。语法分析器分析这个记号空间，并依据既定的语法规则对记号之间的组合关系进行匹配，在匹配失败时报出语法错误，成功时则执行规则之后对应的动作，一般为建立一个语法结构树节点。多个语法结构树节点构成了描述仿真文本结构的语法树。

CMSL 语言有自己的语法规则，将 CMSL 语言的语法规则用 bison 语法分析程序表述出来，完成辅助函数开发和 bison 设置，即完成了 CMSL 语法分析器的开发。Bison 程序后缀名为 y，y 文件由 bison 编译得到 c 代码。语法分析程序与词法分析程序具有相同的结构，不同的是第二部分是用巴克斯范式描述的规则及对应的动作。同样的，第二部分是语法分析程序最为重要的部分。为了使词法分析程序与语法分析程序协同工作，语法分析程序用

第 28 卷第 7 期 系统仿真学报 Vol. 28 No. 7
2016 年 7 月 Journal of System Simulation Jul., 2016

词法分析程序返回的记号编写规则，在第一部分声明这些记号及其值类型以及规则及其值类型。

CMSL 语法分析器 y 文件的部分代码如下：

```
//连接声明
conct_decl: T_BIND exp_list T_TO T_PORT T_BY exp_list ';'{$$=newbind(Bind_d,$4,$2,$6);}
| T_CONCT variable T_TO variable T_BY T_OIDENT ';'
{$$=newconnect(Conct_d,$2,$4,$6);};
```

上述代码为 CMSL 中 bind 语法的 bison 规则表述，‘：’左边为规则名称，规则值用$$指代，规则值类型在 y 文件的声明段进行声明。‘：’右边为规则描述，每一条规则均由其他规则及词法分析器返回的记号前后组合而成，如 T_BIND 为词法分析器返回的记号，而 exp_list 为一条规则。规则匹配成功后将执行规则后由‘{}’包括的 c 语言代码，此处将创建一个绑定类型的语法结构树节点。Y 文件同样有既定的段落格式，见参考文献[1]。利用 bison 对完整的 y 文件进行编译即可得到语法分析器的 C 语言源代码。

2.3 面向组件的解释器实现

解释器由 C 语言编写，解释器由一个主要的 C 函数和一系列辅助子函数构成。

词法分析器和语法分析器对仿真文本分析完成后将创建一个描述文本结构的语法树，解释器的作用就是对该语法树进行解释最终生成目标 C++代码文件及其他辅助文件。由此可见，解释器主函数以语法树类型为输入，同时由于语法树的结构特点，解释器主函数一般具有递归性。针对不同的编译目标和不同的语法环境，解释器的具体内容不同。

CMSL 语言的解释器，最终要求对元素模型[6]进行编译生成 C++代码文本，对复合模型[6]进行编译生成 CMSL 语言描述的扁平化模型，顶层复合模型还要求生成初始化参数文件和连接关系文件。针对 CMSL 语言编译器，解释器函数的流程图如图 1。首先解释器主函数接收语法分析程序创建并传入的语法树指针，然后判断该指针对应的语法树的节点类型，由于语法分析程序的顶层规则是元素模型及复合模型，因此初次传入的语法树其节点必然是元素模型或者复合模型。若是元素模型，则对该结构树的子树进行递归分析，然后输出目标 C++代码；若是复合模型，同样对其子树进行递归分析，然后判断该复合模型是否扁平化，最终输出扁平化后的复合模型信息，即元素模型实例及其连接关系。在第 4 节中进一步阐述了复合模型分级及其扁平化的原理。

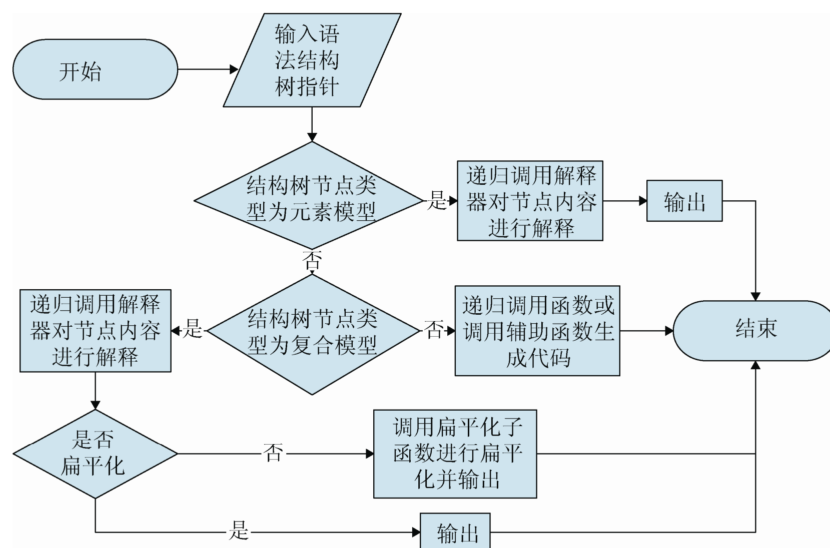

图 1 解释器主函数流程图

2.4 编译器的工作机理

编译器在接到一个编译任务时，首先利用词法分析器和语法分析器扫描输入文本缓冲区，词法分析函数和语法分析函数以及它们的协同工作由 flex 和 bison 共同定义，扫描完成后建立一个语法结构树，再经由解释器对该语法结构树进行处理，编译器工作流程用如图 2 所示。

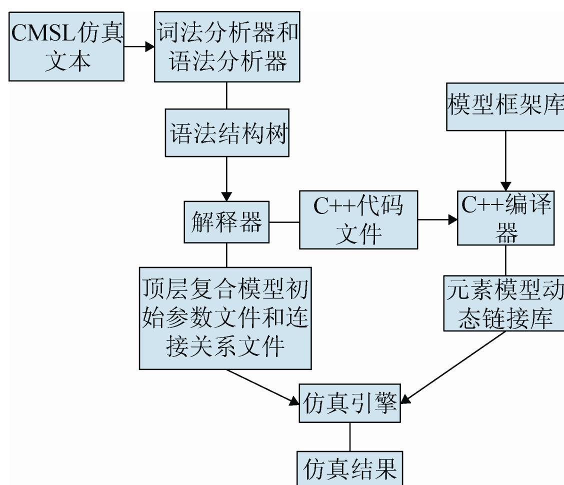

图 2 编译器工作机理框图

词法分析器每次成功匹配一个词法分析文件定义的正则表达式就执行表达式后的 c 代码，代码最后会返回一个记号给语法分析器。语法分析器将词法分析器返回的记号压入堆栈，每次利用多个记号成功匹配语法分析文件中定义的规则，则执行规则之后的动作，动作中一般包含建立与该语法规则对应的语法结构树节点。这样多次匹配直至词法分析器匹配到缓冲区结束标志即文件结束符，语法分析器中有一条规则与此对应，从而建立包含多个节点的语法结构树，该结构树包含输入缓冲区仿真文件的语法结构信息。

解释器则以该结构树为输入，按照特定的编译目标进行递归处理。处理完成后生产目标文件并输出代码。

为了保证编译器的 3 个组件能协同工作，往往需声明用于存储编译信息的全局变量。针对 CMSL 语言编译器的开发，我们声明的全局变量用如下结构体描述：

```
struct cmplinf{
yyscan_t scaninfo;/* scanner context */
struct modspace *curtable;//当前符号表
struct modspace *curvalid;   //当前有效域
struct usertype *maintype;//用户主类型
struct usertype *curhdtype;//当前处理类型
struct ast *a;/* an AST for   */
struct stateprocess *curstate;
char *curdecltype;//当前声明类型
char *file;//当前编译文件
char *flatfile;//当前扁平化输出文件
int typeflag; //内置类型返回标志：0 为元素模型返回 T_VTYPE；1 为复合模型，返回 T_VTYPEC;2 为函数，返回 T_VTYPEF;
};
```

该结构体用于存储编译的即时信息，词法分析器、语法分析器及解释器都依据该结构体存储的信息进行相应的信息处理。如词法分析器依据 typeflag 决定匹配到内置类型时，应返回何种记号，curdecltype 决定词法分析器为当前匹配到的新符号分配何种类型。file 指明当前编译的文件，用于在语法分析器匹配失败时，指出语法错误的文件出处。

3 基于 CMSL 的连续/离散事件系统建模

从时间流机制的角度看，仿真模型在广义上可以分为连续、离散和离散连续混合 3 类。在连续仿真模型中，系统状态被看作在时间上连续改变，系统行为由一组微分方程描述，微分方程描述了作为仿真时间函数的系统状态如何改变[7]。在离散仿真模型中，则系统状态值在离散的仿真时间点上由事件触发相应的改变。CMSL 支持对这 3 类仿真模型的描述和编译。

第 28 卷第 7 期 系统仿真学报 Vol. 28 No. 7
2016 年 7 月 Journal of System Simulation Jul., 2016

3.1 连续系统建模

连续模型的一般用微分方程描述。CMSL 对一个偏微分方程的描述如下：

```
model PDEProblem extends PDE2D
initial section //初始块
parameter Real c=1;
parameter Real a=0;
Point2D p1(px=-1,py=1), p2(px=-1,py=-1);
Point2D p3(px=1,py=-1), p4(px=1,py=1);
Curve2D c1(p1,p2,x+1=0);
Curve2D c2(p2,p3,y+1=0);
Curve2D c3(p3,p4,x-1=0);
Curve2D c4(p4,p1,y-1=0);
Boundary2D regBoundary (c1,c2,c3,c4);
set "hyperbolicPDESolver" as solver
model section //模型块
if onBoundary2D (regBoundary) then u=(x^2+y^2)*time;
  end if
when time = 0 then
u=0; der(u,t)=x^2+y^2;
end    whender(der(u,t),t)-div(c*grad(u))+a*u=(x+y)*time;
experiment section //实验块
start_simulation;
plot_result;
end PDEProblem
```

该模型描述了一个双曲型偏微分方程，方程的一般形式是：

$$\frac{\partial^2 u}{\partial t^2} = a^2\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) + f(x, y, t)$$

在文本中，a=0，f(x,y,t)=(x+y)*t。边界条件是 u=(x2+y2)*t，初值条件是 u(0)=0,du/dt=x2+y2。编译后生成的目标代码(C++)为：

```
double funf(double x,double y,double time)
{ return ((x+y)*time);   }
double fung(double x,double y) {   return 0; }
double fung1(double x,double y)
{ return (pow(x,2)+pow(y,2)); }
double funbd(double x, y, double time)
{ return ((pow(x,2)+pow(y,2))*time); }
int main(void)
{double c=1;
double a=0;
Point p1((-1),1),p2((-1),(-1));
Point p3(1,(-1)),p4(1,1);
double coef1[2]={1,1};
double coef2[2]={1,1};
double coef3[2]={-1,1};
double coef4[2]={-1,1};
MathFunction fun1(1,0,coef1);
MathFunction fun2(0,1,coef2);
MathFunction fun3(1,0,coef3);
MathFunction fun4(0,1,coef4);
Curve c1(p1,p2,fun1,true);
Curve c2(p2,p3,fun2,true);
Curve c3(p3,p4,fun3,true);
Curve c4(p4,p1,fun4,true);
Curve BdCurve[4]={c1,c2,c3,c4};
ConvexReg regBoundary(4,BdCurve);
double (*p_funbd[4])(double,double,double);
p_funbd[0]=funbd; p_funbd[1]=funbd;
p_funbd[2]=funbd; p_funbd[3]=funbd;
hyperPDE2 test(regBoundary,h,t,T);
test.SetEquationProp(c,funf);
test.SetZeroBdCondition(fung,fung1,p_funbd);
test.Solving();
system("pde_plot.exe 100.txt 0.05");
return 1; }
```

上述代码为仿真文本对应的目标代码主要是求解边界声明，初边值设置，框架代码放在其他源文件中。

3.2 离散事件系统建模

以典型的离散事件仿真系统复合队列(Queue)模型(包含元素组件、复合组件)为例。假设这个队列模型是学生报到，需要分别办理 2 个业务，入学手续和住宿手续。办理入学手续需分别进入缴纳学费和提取入学资料 2 个队列进行办理，办理住宿手续需分别进入缴纳住宿费和提取住宿资料 2 个队列办理，入学手续必须先于住宿手续办理，缴费必须先于提取资料办理。其队列结构如图 3。办理 4 个业务所需的时间不等，现要仿真办理 4 个业务的工作流程。

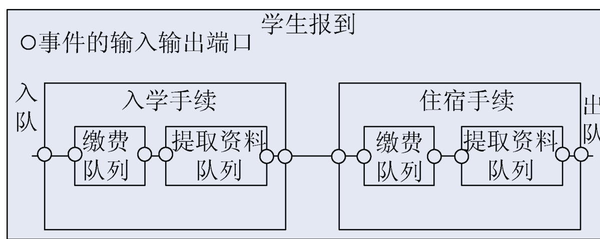

图 3 学生报到队列模型结构图

4 个业务对应 4 个队列模型实例，其中包含的离散事件主要是出入队列事件。基于该元素模型可以参数化、实例化多个队列元素模型。这里介绍队列元素模型的 CMSL 描述，具体如下：

```
//   file Queue.cmsl
discrete element model BinstQueueSample
  initial section//实例参数声明及初始化
inout event EVENT_PORT;
parameter int queueLength = 0;//队列长度
parameter real servingTime = 0.1;//服务时间
parameter real beginServingTime=0.0;//服务开始时间
state idle,serving,deparking;//队列状态
  model section//队列行为描述
when receive EVENT_PORT then
            queueLength++;
end when//接收到出入队事件则队列长度加 1
initial state idle//空闲状态行为
          when queueLength>0 then
              transition serving;
          end when
end state
state serving//服务状态行为
          beginServingTime=time;
  when time at beginServingTime+servingTime then
    transition deparking;
  end when
end state
state deparking//出队状态行为
queueLength--;
      sendcontent(queueLength) by EVENT_PORT;
  if queueLength == 0 then
            transition idle;
  else
            transition serving;
  end if
end state
experiment section//仿真引擎参数设置
set "ConservativeAdvanceSolver" as solver;
end BinstQueueSample
```

该模型在初始段中定义了 3 个成员变量队列长度 queueLength，完成服务所需时间 serverTime，服务开始时间 beginServerTime，1 个影响模型的事件为 EVENT_PORT。2 个模型状态空闲态 idle，服务状态 serving，善后状态 deparking。模型段则描述了该模型的行为，实验块设置改模型用到的仿真求解引擎。

在每个队列各安排一个服务人员，则入学手续和住宿手续各包含两个队列，可描述为包含两队列元素模型的复合模型，其 CMSL 代码如下：

```
//file cqueue.cmsl
extern model BinstQueueSample in "queue.cmsl";
```

第 28 卷第 7 期　　系统仿真学报　　Vol. 28 No. 7
2016 年 7 月　　Journal of System Simulation　　Jul., 2016

```
discrete compound model CompoundQueue
  initial section//初始化进行模型组件声明
parameter int length=40; //长度参数
inout event EVENT_PORT;
BinstQueueSample queue1
(queueLength=length, servingTime=0.3);
BinstQueueSample
queue2(queueLength=0,servingTime=0.2);
//2 个队列成员实例
  model section //模型块进行连接关系声明
bind queue2 to outport by EVENT_PORT;
//端口绑定
bind queue1 to inport by EVENT_PORT;
connect queue1 to queue2 by EVENT_PORT;
//内部连接
end CompoundQueue
```

该模型初始段声明了 1 个成员变量初始队列长度 length 和 2 个成员对象。在模块段定义了模块内对象连接关系，和模块端口的绑定关系。复合模型不参与结算，仅进行模型连接定义，因此省去实验块。

整个新生手续办理的仿真模型的 CMSL 代码如下：

```
//file cqueue.cmsl
discrete compound model QueueSimulation2
  initial section
CompoundQueue
compoundQueue1(length=40);
CompoundQueue compoundQueue2(length=0);
inout event EVENT_PORT;
  model section
    connect compoundQueue1 to
compoundQueue2 by EVENT_PORT;
end QueueSimulation2
```

该模型定义了入学手续和住宿手续两者的连接关系。

3.3 混合系统建模

针对混合系统，对连续系统模型进行离散封装以提供与离散模型进行交互的接口。如：discrete element model BinstEnemyMissle

```
initial section
input   event ET_ask_emy_attr;
output event ET_emy_attr;
input   event ET_blk_suc;
output event ET_att_suc;
//用于与其他组件交互的离散事件
real starttime;
point v;
parameter point emypos;
……
state tracetarg;
model section
initial state tracetarg //离散的状态
starttime = time;
v = (targpos-emypos)/ valueof
(targpos-emypos)*speed;
emypos:=integ(v,derstep)+emypos;
//导弹的运动微分方程描述：连续部分
when receive ET_ask_emy_attr
thensendcontent(emypos,targname) to
ET_ask_emy_attr.instName by ET_emy_attr;
end when
when receive ET_blk_suc then
transition destroyed;
end when
when (valueof(targpos-emypos)<detB) then
        sendcontent() by ET_att_suc;
    transition destroyed;
end when
when time at starttime+maxtime then transition
destroyed;
end when
```


Published by Journal of System Simulation, 2016

```
end state
experiment section
   set "ConservativeAdvanceSolver" as solver;
set derstep as discrete time;
end BinstEnemyMissle
```

该模型描述了一个离散事件驱动的导弹模型，模型块中既包含导弹的离散状态描述，又包含导弹连续运动的微分方程描述。

4 利用 CMSL 编译器对连续、离散仿真系统文本进行编译

以上经过编译的仿真模型对应于 DEVS 中提出的元素模型和复合模型[8, 9]。复合模型按其包含模型的级数分为一级模型、二级模型和多级模型。元素模型为零级模型，仅包含元素模型的为一级模型，其他复合模型分级在其所包含的最高级复合模型的级别上加 1。对不同的模型，编译的任务目标不一样，编译时解释器的处理方式不一样。

4.1 零级模型(元素模型)

元素模型编译的任务目标是供仿真引擎调度的动态链接库文件。解释器首先将其编译为过渡文本，然后利用过渡文本编译得到 C++代码，最后调用 C++编译器对生成的代码文本和模型框架文本进行编译得到最终的 DLL 文件。

4.2 一级复合模型

一级复合模型中仅包含元素模型，又称为扁平化模型。若扁平化模型为闭合模型，则利用该模型生成初始化参数文件和连接关系文件。闭合模型指模型行为不依赖外界输入(外界输入简化为常值)，仅与时间及初始条件相关。否则，该一级复合模型仅为过渡模型用于对模型结构进行清晰的描述，在编译过程中该模型将直接输出，其模型数据将存储在数据结构中作为高级模型引用的依据。

4.3 二级及高级复合模型

二级复合模型中引用了其他复合模型，为非扁平化模型。对二级复合模型进行编译需先进行扁平化。扁平化即是将二级复合模型转化为一级复合模型。对二级复合模型扁平化采用的是溯源算法，溯源算法指对二级复合模型所引用的一级复合模型，从模型库中找到一级复合模型利用其数据替换二级复合模型中与该模型相关的声明及连接关系。在转化过程中，每次遇到对一级复合模型的引用，则调用溯源算法进行替换，最终完成二级模型扁平化。二级模型扁平化后，其后续编译过程按一级模型处理。

高级模型扁平化结构关系如图 4，针对高级复合模型，由于 CMSL 引用语法的限定，低级模型总是先于高级模型完成编译过程，因此，在二级模型扁平化完成后，所有的复合模型其复合阶数减 1，即所有的三级模型转化为二级模型从而可以按二级模型的编译过程进行处理。

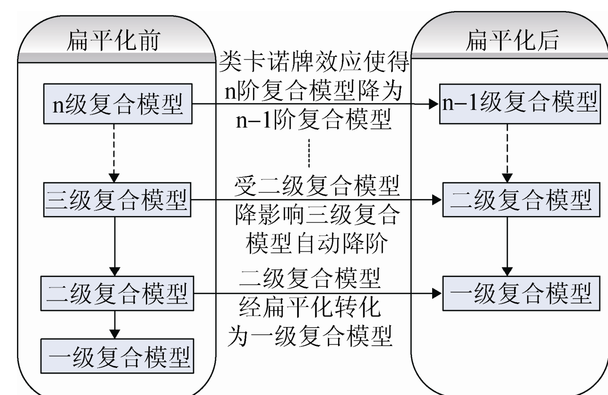

图 4 二级复合模型扁平化前后模型结构图

5 结论

本文介绍了 CMSL 仿真语言的构成及其特点，阐述了如何利用 flex 和 bison 实现一个可编译仿真文本的编译器，最后说明了编译器的工作机理及其对元素模型及复合模型的处理方法。在案例的建模过程中，我们先将整个仿真系统分解得到最小的概念对象，然后对概念对象进行建模得到元素模型，队列是我们抽象出的概念对象，然后依据元素模型复合重塑仿真模型。这体现了 CMSL 语言面向对象和面向组件的特性。

（下转第 1546 页）