基于SystemC的AADL软构件仿真代码转换技术

马春燕[1] 董云卫[2] 陆 伟[1] 朱晓燕[3]
(西北工业大学软件与微电子学院 西安 710072)[1] (西北工业大学计算机学院 西安 710072)[2]
(北京航空航天大学计算机学院 北京 100191)[3]

摘 要 目前,AADL在任务关键和安全关键嵌入式领域有着良好的应用。如何在设计阶段对AADL模型进行仿真,并根据仿真结果迭代构造和精化设计模型,以尽早发现设计模型中存在的问题,保障设计模型的质量,进而减少系统开发的代价,是目前急需解决的技术挑战。SystemC是一种软硬件协同仿真的系统描述语言,由此提出了AADL软构件到SystemC仿真代码的转换技术上,设计和实现了转换工具,并以航行控制系统为例,阐释了转换技术和基于SystemC的线程调度仿真。通过本研究成果,用户可以实现基于SystemC的AADL软构件的仿真,包括软构件之间交互、执行时间和线程调度的仿真等,用户也可以将研究成果与基于SystemC的AADL执行平台构件仿真相结合,对软硬件进行协同仿真。

关键词 AADL,SystemC,转换技术,仿真

SystemC-based Simulation Code Generation for AADL Software Component

MA Chun-yan[1] DONG Yun-wei[2] LU Wei[1] ZHU Xiao-yan[3]
(School of Software and Microelectronics, Northwestern Polytechnical University, Xi'an 710072, China)[1]
(School of Computer Science, Northwestern Polytechnical University, Xi'an 710072, China)[2]
(School of Computer Science, Beijing University of Aeronautics and Astronautics, Beijing 100191, China)[3]

Abstract Currently, AADL in mission-critical and safety critical embedded field has a good application. In the design phase, how to simulate AADL model is urgent. Model simulation can iteratively construct and refine design model for early detection of problems, assuring the quality of the design model, thereby reducing the cost of system development. SystemC is a system description language of hardware and software co-simulation. The paper proposed conversion technology and implemented conversion tool for AADL software component to SystemC simulation code. At last, the paper used cruise control system as an example to explain the conversion technology and SystemC-based thread scheduling simulation. Through the research results, users can obtain SystemC-based simulation of AADL software components including the interaction, execution time and sequence between software components, and the thread scheduling simulation, and can implement software and hardware co-simulation through combining AADL hardware component simulation platform.

Keywords AADL, SystemC, Conversion technology, Simulation

1 引言

结构化分析与设计语言(Architecture Analysis and Design Language,AADL)是一种设计软硬件结构的嵌入式实时系统建模语言,它通过定义构件和构件之间的交互对嵌入式系统进行建模。AADL具有语法简单、功能强大、可扩展的优点,其在任务关键和安全关键嵌入式领域具有广阔的应用前景。目前,它得到了欧美工业界,特别是航空、航天领域的支持,同时受到国际知名大学和研究机构的青睐,对AADL的深入研究和扩展相继展开[1]。

AADL模型的质量决定了嵌入式系统的质量。如何保障AADL模型的质量,是目前的研究热点。模型的仿真有助于迭代构造高质量的可信模型,所以在设计阶段提供一个仿真执行环境,以仿真模型的设计约束是否满足需求是必要的。SystemC是一种软硬件协同仿真的系统描述语言,运用SystemC可以在不同抽象层次上实现系统的仿真和测试,这是SystemC特有的优势。本文焦点集中在AADL软构件到SystemC仿真代码的转换技术,并开发了辅助工具,可以自动解析AADL模型,生成SystemC的仿真代码。在本文研究成果的基础上,结合硬构件的转换,可以实现软硬件联合仿真。

到稿日期:2010-09-09 返修日期:2010-12-30 本文受国家高技术研究发展计划(863计划)(2009AA01Z147),陕西省自然科学基础研究计划项目(2009JM8003-5),西工大科技创新基金(2008KJ02045),西北工业大学引进高层次人才科研启动费资助项目及西北工业大学软件与微电子学院"R"孵化基金(2010R005)资助。

马春燕(1978—),女,博士,讲师,主要研究方向为软件测试、任务关键软件的建模和仿真;董云卫(1968—),男,教授,主要研究方向为嵌入式软件设计与验证。

de las Heras 等人[2]提出了 AADL 模型到 SystemC 映射的语义，以支持软硬件联合仿真，但作者没有给出 SystemC 代码生成的具体规则，缺乏实验和辅助工具的研制。本文提出 AADL 软构件到 SystemC 代码的自动转换技术，在此基础上研制仿真代码生成的辅助工具。

本文第 2 节提出了 AADL 软构件及构件交互到 SystemC 的转换技术，并以航行控制系统 AADL 模型为例阐释了研究成果；第 3 节提出 AADL 软构件到 SystemC 转换工具的设计与实现；第 4 节分析了航行控制系统 AADL 模型的转换结果和线程调度的仿真实验；最后对全文进行总结和展望。

2 AADL 模型到 SystemC 代码的转换技术

每个 AADL 模型的构成都是一个构件树，树根表示整个系统，由上而下依次有子系统、进程、线程作为树的中间层，有子程序或线程作为树的叶子节点。各软构件都可以包含数据访问、端口和构件间通讯的连接，2.1－2.2 节首先给出数据访问、端口和连接到 SystemC 的代码转换技术。然后，依据构件的包含关系，2.3－2.5 节从构成系统构件树的底层（子程序）到树的中间层（线程、进程）逐步阐述构件到 SytemC 代码的转换技术。在 2.1－2.5 节的基础上，2.6 节给出了子系统和系统构件的转换技术。

2.1 数据构件的转换

数据构件代表了 AADL 模型中声明的数据类型，它包含子程序和数据访问。数据构件可以转换为 SystemC 中的基本数据类型或用户自定义的数据类型。对于用户自定义的数据类型，可以采用枚举类型、结构体类型或对象类型，其中可以包含一些函数的定义，这些函数对应于数据构件中的子程序。每个数据构件对应一个包含数据构件定义的文件〈data_数据构件名.h〉。图 1 所示的是航行控制系统[3]中数据构件转换的示例，图的左边为 AADL 中数据构件的说明，右边是转换文件 data_Personnnel_record.h 的代码。

```
data Personnel_record
features
update_address: subprogram update_address;
end Personnel_record;
data implementation Personnel_record.others
subcomponents
Name : data basic::string;
Home_address : data aeii::aadl::relief::Address;
end Personnel_record.others;
subprogram update_address
features
person: in out parameter Personnel_record;
street :in parameter basic::string;
city: in parameter basic::string;
end update_address;
```

```
该例是用户自定义的record类型Personnel_record，它转换成SystemC代码的定义如下：
struct Personnel_record {
    string name;
    Address Home_address;
public:
Personnel_record update_address(Personnel_record person, string street, string city);
};
```

图 1 数据构件转换示例

2.2 构件端口和连接的转换

2.2.1 数据端口和连接

1）数据端口的转换

构件的数据端口是构件状态数据传输的连接点，不支持队列。AADL 的输入（in）、输出（out）和输入输出（inout）类型的数据端口分别转换为 SystemC 基本端口类型，即上述类型的端口分别转换为 sc_in〈T〉，sc_out〈T〉和 sc_inout〈T〉类型的端口变量，其中，T 是端口传输数据的数据类型。

2）数据端口连接的转换

由于系统、子系统和进程转换为 SystemC 的模块（sc_module），因此系统和进程的数据端口连接转换为 SystemC 通过以下通讯方式实现：

（1）子模块的端口可以直接关联到其所属父/子模块的端口进行通讯；

（2）一个模块的端口通过绑定到信号（sc_signal）与其他同级模块的端口相连进行通讯。

对于线程的数据端口连接可以直接转换为对应的信号变量，即 sc_signal 类型的变量，进程内线程间通讯时直接使用信号即可，这样可以节省仿真时间、提高仿真速度。

2.2.2 事件端口

1）事件端口的转换

AADL 中的事件端口和事件数据端口定义为带有队列的消息传输接口。AADL 中的事件端口映射为 SystemC 中先进先出队列通道（FIFO 通道）sc_fifo〈T〉类型的变量。由于事件端口传输的是一个事件，因此可以把所有的事件定义为一个枚举类型，对于通道类型 sc_fifo〈T〉，T 就代表了这个枚举类型。例如：

enum events {event1，event2，event3}；

sc_fifo〈events〉 *Event1*，*Event2*，*Event3*；

2）事件端口连接的转换

参与事件端口连接的两个构件转换后分别绑定到事件端口对应的通道变量，通过调用通道的 read()和 write()方法向通道读或写事件，来仿真构件之间的事件通讯。

2.2.3 事件数据端口

1）事件数据端口的转换

事件数据端口映射为 SystemC 中的自定义端口 sc_port〈InterfaceType〉类型的变量，InterfaceType 是端口所要连接通道的接口类型。由于事件数据端口定义为带有队列的消息传输接口，因此端口 InterfaceType 所要连接通道映射为 sc_fifo_in_if〈T〉、sc_fifo_out_if〈T〉或 sc_fifo_inout_if〈T〉，其中 sc_fifo_in_if〈T〉和 sc_fifo_out_if〈T〉是 SystemC 提供的接口，sc_fifo_inout_if〈T〉是继承 sc_fifo_in_if〈T〉和 sc_fifo_out_if〈T〉的用户自定义接口。

输入事件数据端口在 SystemC 中对应的变量声明如下：

sc_port〈sc_fifo_in_if〈T〉〉 *eventdataport1*

输出事件数据端口在 SystemC 中对应的变量声明如下：

sc_port〈sc_fifo_out_if〈T〉〉 *eventdataport2*

输入输出事件数据端口在 SystemC 中对应的变量声明如下：

sc_port〈sc_fifo_inout_if〈T〉〉 *eventdataport3*

其中 T 为通过端口传输的数据或事件类型。

2）事件数据端口连接的转换

首先声明一个 sc_fifo〈T〉通道类型的变量，参与事件端口连接的两个构件转换后，分别将其事件数据端口变量与该通道变量绑定，通过调用通道的 read()和 write()方法向通道读或写事件，来仿真构件之间的事件数据通讯。

2.3 子程序构件的转换

AADL 子程序转换为 SystemC 中的 C++函数。由于子程序是由线程或其他子程序调用的，而线程又作为进程的子构件，因此子程序转换后的函数声明在相应进程转换的文件〈P_进程名.h〉中。对于带参数的子程序，in 类型的参数作为函数的参数传递到参数内部使用，采用值传递的方式。而 out 或 in out 类型的参数作为函数的参数传递到参数内部使用，采用引用传递的方式，即函数的返回类型为 void。函数声明语法如下：

void 子程序名 1(in 或 inout 参数数据类型 参数名 1,……);

在相应文件〈P_进程名. cpp〉中实现函数,语法如下:

```
void 进程名::子程序名 1(in 或 in out 型参数类型 参数名 1,……){
    //函数实现
    ……
}
```

输出事件端口、输出事件数据端口、端口组以及需要数据访问等子程序的类型特征到 SystemC 的转换代码将作为函数体的局部变量。对于航行控制系统,其子程序的转换见图 3(c)的右方。

2.4 线程构件的转换

线程是程序中顺序执行的可调度单元。线程构件映射为 SC_THREAD(*threadName*),其中 *threadName* 是以线程名命名的函数,该函数的实现对应线程时间语义的仿真行为。由于线程作为某个进程的子构件,因此本文将线程的端口和数据转换后的变量作为其所属进程的全局变量(详见 2.5 节)。设线程所属进程转换的文件为〈P_进程名. h〉和〈P_进程名. cpp〉,则文件〈P_进程名. h〉中,函数 threadName 声明语法如下:

Void threadName();

在文件〈P_进程名. cpp〉中,函数 threadName 实现语法如下:

```
Void 进程名::线程名 2(){
  //具体实现
  ……
}
```

根据 AADL 模型中线程时间属性的语义描述,函数 threadName 仿真代码实现的功能包括:

1)模拟线程的初始化。在 SystemC 中,用 wait(*Init_execution_time*,*SC_MS*)来模拟线程初始化时消耗的时间。Init_execution_time 是定义的一个 int 型的变量,其值为线程的属性 Initialize_Execution_Time 中规定的初始化执行时间最大与最小值之间的一个随机值。

2)初始化完成后,若线程中调用了子程序则调用子程序转换的函数。

3)模拟线程执行阶段。在线程执行阶段,首先消耗一个最小执行时间,用 wait(*compute_min_time*,*SC_MS*)实现,该时间值是 AADL 模型属性 Compute_Execution_Time 中规定的最小值。

4)模拟线程端口与其它构件端口之间的通讯(根据 2.2 节连接转换的规则撰写仿真代码)。当所有的操作执行完后,比较线程的执行时间与线程周期。若还未到周期值,则要保证线程消耗一定的时间,直到该周期结束。其实现通过在线程执行开始和结束分别加一个时间戳,由 sc_time_stamp()获取当前仿真时间,分别存入时间变量 *start_time* 和 *final_time* 中,两时间差与周期 period 比较。若小于周期,则执行语句 wait(*final_time-start_time*)。

5)模拟 AADL 线程的终止消耗时间,使用 wait(*final_execution_time*,*SC_MS*)实现。*final_execution_time* 是 int 型变量,其值为 AADL 模型 Finalize_Execution_Time 属性中规定的终止执行时间最大与最小值之间的一个随机值。

图 2 左边是航行控制系统中线程 Button_Panel,对于其声明的端口和数据,转换后的变量作为进程(Hci_Process)的全局变量,详见图 3(b)的行 21—34。根据线程 Button_Panel 的属性(properties),实现上述 5 个步骤的仿真功能,函数 Button_Panel()的仿真代码见图 2 下方,其中行 7—8 实现步骤 1)、行 10—15 实现步骤 2)和 3)、行 17—32 实现步骤 4)、行 34—35 实现步骤 5)。

```
thread Button_Panel
  features
    Activate: out event port;
    Cancel: out event port;
    Onnotoff: out data port Bool_Type;
    Incspeed: out data port Bool_Type;
    Decspd: out data port Bool_Type;
end Button_Panel;

thread implementation Button_Panel.Gui
  calls
    {
      subpl_1: subprogram subpl_1.impl;
    };
  properties
    AADL_Properties::Dispatch_Protocol => Periodic;
    AADL_Properties::Period => 20 Ms;
    AADL_Properties::Compute_Entrypoint => "Button_Panel_func";
    AADL_Properties::Source_Text => "./fButton_Panel.cpp";
    UC::POSIX_Scheduling_Policy => SCHED_RR;
    UC::Priority => 20;
    AADL_Properties::Compute_Execution_Time => 1 Ms .. 10 Ms;
    AADL_Properties::Initialize_Entrypoint => "Initialize_Button_Panel";
    AADL_Properties::Initialize_Execution_Time => 6 Ms .. 9 Ms;
    AADL_Properties::Finalize_Entrypoint => "Finalize_Button_Panel";
    AADL_Properties::Finalize_Execution_Time => 6 Ms .. 9 Ms;
end Button_Panel.Gui;
```

```
5  void Hci_Process:: Button_Panel(){
7      int Init_execution_time = (6+rand()%(9-6+1));
8      wait(Init_execution_time, SC_MS);
9      while(true){
10         sc_time period(20, SC_MS);
12         sc_time start_time = sc_time_stamp();
13         subpl_1();
14         int compute_min_time = 1;
15         int timeout = 10; //
16          wait(compute_min_time,SC_MS);
17         //端口通讯, 线程的端口变量作为进程的全局变量
18         Onnotoff.write((short)64);
19         onnotoff_wr_ev.notify();
20         Incspeed.write((short)65);
21         inspd_wr_ev.notify();
22         Decspd.write((short)66);
23         decspd_wr_ev.notify();
24         Activate1.write(activate1_event);
25         active1_ev.notify();
26         Cancel.write(cancel_event);
27         cancel_ev.notify();
28         Activate2.write(activate2_event);
29         active2_ev.notify();
30         sc_time final_time =sc_time_stamp();
31         if((final_time-start_time) < period/1000){
32             wait(final_time-start_time);
33         }
34     }
35     int final_execution_time = (6+rand()%(9-6+1));
36     wait(final_execution_time,SC_MS);
37 }
```

图 2 线程转换示例

2.5 进程构件的转换

进程实现的描述转换为文件〈进程名. cpp〉,该文件内容为其包含线程或子程序的函数实现,转换规则如表 1 所列。进程 Hci_Process 转换的〈Hci_Process. cpp〉文件实现见图 3(c)所示,共包含 4 个线程 Button_Panel,Drivermodelogic,Refspd,Instrumentpanel 对应函数的实现,以及其线程调用的子程序 subpl_1,subpl_2,llamado,call_server 的实现,函数的具体实现参见线程和子程序的仿真代码转换规则(详见 2.3 节和 2.4 节),例如函数 Hci_Process::Button_Panel()的实现见 2.4 节的图 2 下方。

表 1 进程转换规则

进程构件	转换到 SystemC 代码文件〈P_进程名. h〉的规则
端口或数据访问	转换规则 5.1:依据 2.1 节和 2.2 节数据、端口的转换规则,生成仿真变量
子构件的端口或数据访问	转换规则 5.2:依据 2.1 节和 2.2 节端口的转换规则,生成仿真变量
子构件(线程或数据)	转换规则 5.3:声明线程转换的函数,声明线程调用的子程序对应的 C++函数,以及子程序调用的 C++函数
子构件之间的连接和通讯	体现在子构件的仿真代码中
子构件(线程或数据)	转换规则 5.4:在模块构造函数 SC_CTOR(P_进程名)中,注册该进程包含的线程,并声明各线程的敏感事件

```
process Hci_Process
  features
    Breakpedalpressed: requires data access Bool_Type;
    Clutchpedalpressed: in data port Bool_Type;
    Autospd_Mph: in data port Float_Type;
    Cruiseactive: out data port Bool_Type;
    Refspd_Mph: out event data port Float_Type;
end Hci_Process;
process implementation Hci_Process.Model
  subcomponents
    Button_Panel: thread Button_Panel.Gui;
    Drivermodelogic: thread Drivermodelogic.Simulink;
    Refspd: thread Refspd.Simulink;
    Instrumentpanel: thread Instrumentpanel.Gui;
  connections
    -- Bus
    CBreakpedalpressed: data access Breakpedalpressed -> Drivermodelogic.Breakpedalpressed;
    CClutchpedalpressed: data port Clutchpedalpressed -> Drivermodelogic.Clutchpedalpressed;
    CAutospd_Mph: data port Autospd_Mph -> Instrumentpanel.Autospd_Mph;
    CCruiseactive1: data port Drivermodelogic.Cruiseactive -> Cruiseactive;
    Refspd1: event data port Refspd.Refspd_Mph -> Refspd_Mph;
    -- Internal
    CCruiseactive2: data port Drivermodelogic.Cruiseactive -> Refspd.Cruiseactive;
    event port Button_Panel.Activate -> Drivermodelogic.Activate;
    event port Button_Panel.Cancel -> Drivermodelogic.Cancel;
    COnnotoff: data port Button_Panel.Onnotoff -> Drivermodelogic.Onnotoff;
    event port Button_Panel.Activate -> Refspd.Activate;
    CIncspeed: data port Button_Panel.Incspeed -> Refspd.Incspeed;
    CDecspd: data port Button_Panel.Decspd -> Refspd.Decspd;
    Refspd2: event data port Refspd.Refspd_Mph -> Instrumentpanel.Refspd_Mph;
end Hci_Process.Model;
```

(a) 进程 Hci_Process 构件规格说明示例

```
#include"systemc.h"
SC_MODULE(Hci_Process) {
    //对应转换规则5!
    sc_in<short> Breakpedalpressed;
    sc_in<short> Clutchpedalpressed;
    sc_in<float> Autospd_Mph;
    sc_out<short> Cruiseactive;
    sc_port<sc_fifo_out_if<float> > Refspd_Mph;
    //线程名作为其映射的函数名

    void Button_Panel();
    void Drivermodelogic();
    void Refspd();
    void Instrumentpanel();
    //以子程序调用名作为函数名
    void subp1_1();
    void subp1_2();
    short Hci_Process:: llamado(short in_parameter, short p1);
    void call_server();

    sc_signal<short> Onnotoff;
    sc_signal<short> Incspeed;
    sc_signal<short> Decspd;
    sc_signal<short> Cruiseactive2;
    sc_fifo<float> Refspd_Mph2;
    enum events {activate1_event, activate2_event, cancel_event};
    sc_fifo<events> Activate1, Activate2, Cancel;
    //子构件通信要借助的事件对应变量声明
    sc_event onnotoff_wr_ev;
    sc_event inspd_wr_ev;
    sc_event decspd_wr_ev;
    sc_event cruiseactive2_wr_ev;
    sc_event refspd_ev2;
    sc_event active1_ev, active2_ev, cancel_ev;
    SC_CTOR(Hci_Process) {
        SC_THREAD(Button_Panel);
        sensitive_pos << clk;
        SC_THREAD(Drivermodelogic);
        sensitive << Breakpedalpressed << Clutchpedalpressed;
        SC_THREAD(Refspd);
        SC_THREAD(Instrumentpanel);
        sensitive << Autospd_Mph;
    }
};
```

(b) 进程转换为〈Hci_Process. h〉文件的示例

```
include"Hci_Process.h"                    //余下子程序函数实现
定义线程实现                               void Hci_Process:: subp1_1(){
oid Hci_Process:: Button_Panel(){          }
                                           void Hci_Process:: subp1_2(){
oid Hci_Process:: Drivermodelogic(){       }
                                           short Hci_Process:: llamado(short in_parameter, short p1){
oid Hci_Process:: Refspd(){                }
                                           void Hci_Process:: call_server(){
oid Hci_Process:: Instrumentpanel(){       }
```

(c) 进程转换为〈Hci_Process. cpp〉文件的示例

图 3

2.6 系统构件的转换

```
system Cruisecontrol
  features
    Breakpedalpressed: requires data access Bool_Type;
    Clutchpedalpressed: in data port Bool_Type;
    Autospd_Mph: in data port Float_Type;
end Cruisecontrol;
system implementation Cruisecontrol.Generic
  subcomponents
    S_Hci: system Hci.Model;
    S_Cruisecontrollaws: system Cruisecontrollaws.Model;
  connections
    data access Breakpedalpressed -> S_Hci.Breakpedalpressed;
    data port Clutchpedalpressed -> S_Hci.Clutchpedalpressed;
    data port Autospd_Mph -> S_Hci.Autospd_Mph;
    data port Autospd_Mph -> S_Cruisecontrollaws.Autospd_Mph;
    data port S_Hci.Cruiseactive -> S_Cruisecontrollaws.Cruiseactive {
      AADL_Properties::Allowed_Connection_Binding => reference lan;
    };
    Refspd1: event data port S_Hci.Refspd_Mph -> S_Cruisecontrollaws.Refspd_Mph {
      AADL_Properties::Allowed_Connection_Binding => reference lan;
    };
```

(a) 系统 Cruisecontrol 构件规格说明示例

```
1  //File:Cruisecontrol.h
2  #include "systemc.h"
3  #include "S_Hci.h"
4  #include "S_Cruisecontrollaws.h"
5  SC_MODULE(S_Cruisecontrol) {
6      sc_in<short> Breakpedalpressed;
7      sc_in<short> Clutchpedalpressed;
8      sc_in<float> Autospd_Mph;
9      S_Hci * subs_hci;
10     S_Cruisecontrollaws * subs_cruisecontrollaws;
11     sc_signal<short> sigCruiseactive;
12     sc_fifo<float> fiforefspd_Mph;
13     SC_CTOR(S_Cruisecontrol){
14         subs_hci = new S_Hci("subs_hci");
15         subs_hci->clk(clk);
16         subs_hci->Breakpedalpressed(Breakpedalpressed) ;
17         subs_hci->Clutchpedalpressed(Clutchpedalpressed);
18         subs_hci->Autospd_Mph(Autospd_Mph);
19         subs_hci->Cruiseactive(sigCruiseactive);
20         subs_hci->Refspd_Mph(fiforefspd_Mph);
21         subs_cruisecontrollaws = new S_Cruisecontrollaws("subs_cruisecontrollaws");
22         subs_cruisecontrollaws->clk(clk);
23         subs_cruisecontrollaws->Autospd_Mph(Autospd_Mph);
24         subs_cruisecontrollaws->Cruiseactive(sigCruiseactive);
25         subs_cruisecontrollaws->Refspd_Mph(fiforefspd_Mph);
26     }
27 };
```

(b)系统构件转换为〈Cruisecontrol. h〉文件的示例

图 4

系统构件转换为包含一个模块 SC_MODULE(S_系统名)声明的文件〈系统名. h〉。转换规则如表 2 所列。图 4(a)是航行控制系统中系统构件 Cruisecontrol 的规格说明。根据提出的转换规则,系统构件转换为文件〈Cruisecontrol. h〉,如图 4(b)所示,行 6-8 对应转换规则 6. 1、行 9—10 对应转换规则6. 2、行 11—25 对应转换规则 6. 3。

表 2 系统转换规则

系统构件	转换到 SystemC 代码的规则
端口或数据访问	转换规则 6. 1:依据 2. 1 节和 2. 2 节节数据、端口的转换规则,生成仿真变量
子构件(进程或系统)	转换规则 6. 2:声明子构件对应模块类型的变量
子构件之间的连接	转换规则 6. 3:依据 2. 2 节端口和连接的转换规则,生成仿真相应的仿真代码

3 转换工具 AADLToSimCode 的设计与实现

本文实现了 AADL 软构件到 SystemC 仿真代码的转换工具 AADLToSimCode 的设计,图 5 是该工具的系统框架结构图。转换工具首先解析 AADL 描述文件 MySystem. aaxl 和系统实例 MySystem_Instance. aaxl 文件(这两个文件可以由 AADL 的建模工具 OSATE(open source AADL tool environment)[4])获取,并存储相关数据信息(即连接数据、实例构件数据和属性及其他构件数据),然后根据转换规则,将解析结果映射生成仿真代码。图 6 是工具 AADLToSimCode 实现的类图,其中类 Parser 是工具的执行入口,负责解析 XML 文件;类 ComponentInstance 存储模型数据;类 GenerateSWComponent根据解析结果和转换规则生成并输出软构件仿真代码文件。

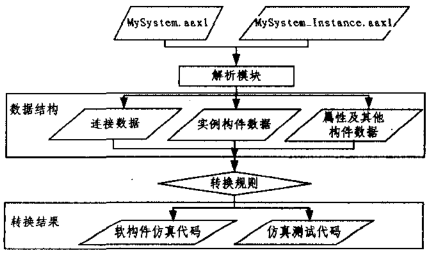

图 5 转换工具 AADLToSimCode 的系统框架结构图

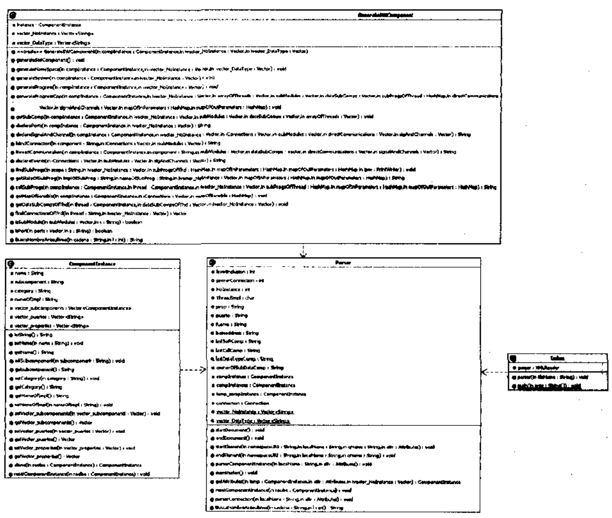

图 6 工具 AADLToSimCode 实现的系统类图

4 航行控制系统案例分析

航行控制系统(Cruisecontrol)包含两个子系统:S_Hci 和

(下转第 196 页)

参 考 文 献

[1] Pawlak Z. Rough Sets [J]. International Journal of Computer and Information Science, 1982, 11(5): 341-356

[2] 徐章艳,刘作鹏,杨炳儒,等. 一个复杂度为 max(O(|C||U|),O(|C|²|U/C|))的快速属性约简算法[J]. 计算机学报,2006,29(3):391-399

[3] 苗夺谦,胡桂荣. 知识约简的一种启发式算法[J]. 计算机研究与发展,1999,36(6):681-684

[4] 王国胤,于洪,杨大春. 基于条件信息熵的决策表约简[J]. 计算机学报,2002,25(7):759-766

[5] Skowron A, Rauszer C. The discernibility matrices and functions in information systems[C] // Słowiński R, ed. Intelligent Decision Support Handbook of Applications and Advances of the Rough set Theory. Kluwer Academic Publishers, Dordrecht, 1992, 311-362

[6] 杨明. 一种基于改进差别矩阵的属性约简增量式更新算法[J]. 计算机学报,2007,30(5):815-822

[7] Miao D Q, Zhao Y, Yao Y Y, et al. Relative reducts in consistent and inconsistent decision tables of the Pawlak rough set model[J]. Information Sciences, 2009, 179: 4140-4150

[8] Qian J, Miao D Q, Zhang Z H, et. al. Hybrid approaches to attribute reduction based on indiscernibility and discernibility relation[J]. Int. J. Approx. Reason, 2010, doi: 10. 1016 /. ijar. 2010. 07. 011

[9] 王立宏,吴耿锋. 基于并行协同进化的属性约简[J]. 计算机学报,2003,26(5):630-635

[10] 肖大伟,王国胤,胡峰. 一种基于粗糙集理论的快速并行属性约简算法[J]. 计算机科学,2009,36(3):208-211

[11] 刘鹏. 云计算[M]. 北京:电子工业出版社,2010

[12] Ghemawat S, Gobioff H, Leung S T. The Google file system[J]. SIGOPS-Operating Systems Review, 2003, 37(5): 29-43

[13] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107-113

[14] Chu C T, Kim S, Lin Y A, et al. MapReduce for machine learning on multicore[C] // Proceedings of NIPS. vol. 19, 2006

[15] Zhao Wei-zhong, Ma Hui-fang, He Qing. Parallel k-Means clustering based on MapReduce[C] // Jaatun M G, Zhao G, Rong C, eds. CloudCom 2009, LNCS 5931. 2009: 674-679

[16] Hadoop[EB/OL]. http://lucene. apache. org/hadoop

(上接第 164 页)

S_Cruisecontrollaws,图 7 是其软构件结构图。子系统 S_Hci 中有两个进程:一个进程有 4 个线程,另一个进程包含 1 个线程;子系统 S_Cruisecontrollaws 有一个进程,该进程包含 2 个线程。

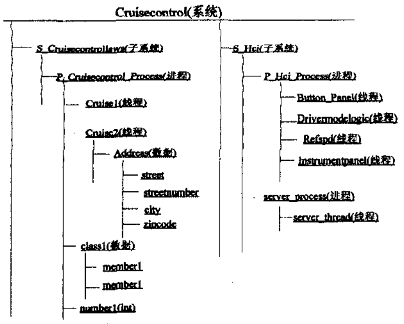

图 7 AADL 模型航行控制系统结构

4.1 航行控制系统的转换

将文件 cruise. aaxl 和 cruise_Instance. aaxl[3] 作为工具 AADLToSimCode 的输入,该工具自动生成的仿真代码文件为 Curisecontrol _ Process. cpp、Hci _ Process. cpp、Hci _ Process. h、S _ tb. cpp、Server _ Process. cpp、Curisecontrol _ Process. h、Cruisecontrol. h、Cruisecontrollaws. h、Hci. h、S _ tb. h 和 Server _ Process. h,其中 Cruisecontrol. h、Hci _ Process. cpp 和 Hci_Process. h 的部分代码已在第 2 节展示。

4.2 调度仿真

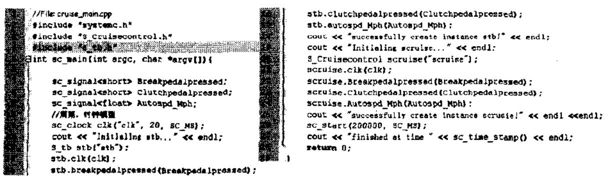

图 8 仿真驱动程序

在 4.1 节生成的仿真代码的基础上,项目组撰写了仿真启动和激励产生文件 Curise_main. cpp,如图 8 所示。SystemC 仿真内核支持仿真进程的调度执行,直接控制并发进程的调度策略。图 9 是基于 SystemC 仿真内核对 AADL 线程调度仿真结果的部分截图,显示了线程 Button_Panel 调度执行的起始时间、终止时间以及线程与其它线程的数据和事件的通讯过程。

图 9 基于 SystemC 仿真内核的调度仿真

结束语 通过本文的研究成果,用户可以实现基于 SystemC AADL 软构件的仿真,包括软构件之间交互、执行时间和以及线程调度的仿真等。用户也可以将本文的研究成果与基于 SystemC 的 AADL 执行平台构件仿真相结合,对软硬件进行协同仿真,根据仿真结果迭代构造和精化设计模型,以尽早发现设计模型中存在的问题,保障设计模型的质量,进而保证任务和安全关键领域软件系统的质量。

目前由于 SystemC 仿真内核不支持线程的占先调度策略,项目组正在研制基于 SystemC 的占先调度器。另外,项目组在本文基础上,已实现了基于 SystemC 的实时性仿真和流延迟仿真,正在撰写基于 SystemC 的实时性仿真和流延迟仿真相关论文。

参 考 文 献

[1] 杨志斌,皮磊,等. 复杂嵌入式实时系统体系结构设计与分析语言:AADL[J]. Journal of Software, 2010, 21(5): 899-915

[2] de las Heras E, Villar E. Specification for SystemC-AADL Interoperability, Intelligent Solutions in Embedded Systems[C] // 2007 Fifth Workshop. 2007: 76-86

[3] http://www. aadl. info/aadl/currentsite/examplemodel. html

[4] The SEI AADL Team. An extensible open source AADL tool environment (OSATE)[EB/OL]. http://www. aadl. info/aadl/downloads/osate13, 2006