第 35 卷 第 8 期 计 算 机 工 程 2009 年 4 月
Vol.35 No.8 Computer Engineering April 2009
·软件技术与数据库· 文章编号：1000—3428(2009)08—0059—03 文献标识码：A 中图分类号：TP311

AADL 模型的代码自动生成及集成技术

陶 勇[1,2]，桂盛霖[1,2]，马 亮[1,2]，尹立孟[2]
(1. 电子科技大学计算机科学与工程学院，成都 610054；2. 北京科银京成技术有限公司成都研发中心，成都 610051)

摘 要：体系结构分析设计语言(AADL)是一种基于模型驱动体系结构的建模语言，针对如何将 AADL 模型自动生成框架代码的问题，提出 AADL 模型元素同 C 语言元素间的转换规则，设计 AADL 模型转换为 C 语言框架代码的自动代码生成器 Generator。实例证明了 AADL 模型自动转换为可执行 C 代码的有效性。

关键词：嵌入式软件；模型驱动体系结构；转换规则；代码自动生成

Code Automatic Generation and Integration Technology of AADL Model

TAO Yong[1,2], GUI Sheng-lin[1,2], MA Liang[1,2], YIN Li-meng[2]
(1. School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 610054;
2. Chengdu Research & Development Center, Bejing Coretek Technology Co., Ltd., Chengdu 610051)

【Abstract】Architecture Analysis and Design Language(AADL) is a modeling language based on Model Driven Architecture(MDA). Aiming at how to automatically generate executive framework code of AADL models, this paper presents conversion rules from AADL model elements to C language elements and designs a tool of automatic code generator "Generator" that converts AADL models into C language framework code. The example proves the effectiveness of converting AADL model to C code.

【Key words】embedded software; Model Driven Architecture(MDA); conversion rules; code automatic generation

1 背景介绍

嵌入式软件的规模及复杂性的不断增长导致开发时间和开发费用急速增长，如何快速有效地开发嵌入式软件成为目前急待解决的问题。模型驱动体系结构(Model Driven Architecture, MDA)是一种具有生命力和应用前景的开发方法。使用 MDA 方法的软件开发过程如图 1 所示，其中，模型是研究的中心。在嵌入式软件产业中已有许多面向功能的建模工具，如 Simulink, SCADE[1], UML 等。

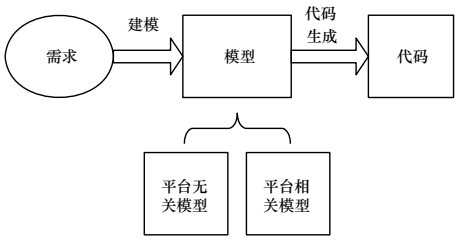

图 1 使用 MDA 方法的软件开发过程

在传统的嵌入式软件开发过程中，缺乏对整个系统体系结构的精确预算，虽然单个功能模块的非功能属性相对容易实现，但在系统集成后如何满足整个系统的非功能属性对于开发人员是一个巨大挑战。要解决这些问题，可以采用 MDA 方法在系统实现前建立模型，在模型级对整个系统的体系结构[2]进行非功能属性规约和验证，消除可能出现的问题，降低开发成本。体系结构分析设计语言(Architecture Analysis and Design Language, AADL)[3]是一种基于 MDA 方法的体系结构建模语言。AADL 模型不关心具体的功能实现，描述的仅仅是系统框架，从而在体系级对系统的非功能属性进行规约，这样系统设计者可以使用各种分析工具对系统的可调度性、可靠性、安全性进行分析，通过分析，可以评估体系结构的平衡和变化，将 AADL 模型转换为针对特定应用的实际语言框架代码，再与实现具体功能的源文本相集成就可以形成符合性能关键属性的可执行代码。

AADL 中的组件是通过类型(type)和实现(implementation)来定义的。组件类型规定了组件的功能接口。组件实现规定了组件的内部结构。AADL 中有 3 类组件：应用软件组件，执行平台组件和复合组件。应用软件组件包括线程(thread)、进程(process)、数据(data)、子程序(subprogram)等。执行平台组件包括处理器、内存、外设、总线。复合组件包含系统、系统对软件组件和执行平台组件进行结构化建模。软件工程师协会开发的开源 AADL 工具环境(Open Source AADL Tool Environment, OSATE)是一个基于 eclipse 的 AADL 语言编辑工具，该工具支持对 AADL 语言进行编辑、语法检查、安全检查等，还可以通过添加插件的方式增加该工具的功能。

2 AADL 到 C 语言的元素转换规则

本文主要讨论 AADL 模型的代码自动生成方法，通过对

基金项目：国家自然科学基金资助项目(90718019)；国家“863”计划基金资助重点项目(2007AA010304)
作者简介：陶 勇(1982－)，男，硕士研究生，主研方向：嵌入式模型开发；桂盛霖，博士研究生；马 亮，硕士研究生；尹立孟，硕士
收稿日期：2008-09-06 **E-mail：**super2219@163.com

UML[4], Simulink[5]等模型语言到代码的转换方法研究以及MDA 代码生成技术[6]制定了 AADL 语言到 C 语言的元素转换规则。具体规则如下:

(1)规则 1: AADL 中的组件类型和实现对应转换为<组件类型名.h>和<组件类型名_组件实现名.h>以及<组件类型名_组件实现名.c>。当该组件被实例化时，对应生成<组件实例名.h>和<组件实例名.c>。

(2)规则 2: AADL 语言中的 data 组件类型与 C 中具体变量类型相对应，将模型中出现的数据类型保存在 data.h 中，通过引用该文件，整个工程都可以使用其中的数据类型。当该数据组件作为线程或进程的子组件，在进程或线程被实例化以后，在对应的<组件实例名.c>中实例化一个该数据类型的变量。

(3)规则 3: AADL 语言 features 中的事件数据端口在 C 中对应为包含数据和事件 2 个子类型的结构体。其中，数据子类型是一个端口类型的指针；事件子类型是一个指向事件的字符指针。该端口在对应的<组件类型实例名.c>中实例化。

(4)规则 4:AADL 中的 subprogram 在 C 中对应调用函数，将函数的声明定义在.h 文件中，函数的实现定义在.c 文件中，该函数的实现部分调用由开发人员根据具体的功能需求编写的功能函数。子程序的特征则作为函数的参数。

(5)规则 5: AADL 中的端口关联对应组件端口或参数之间的地址传递。

(6)规则 6: AADL 中 thread 组件转换为 C 代码中与平台结合的 task 的入口点函数。该函数为进程创建线程后的执行函数。代码如下:

```
data 数据类型
properties
    Language_Support::Data_Format=>int;
end 数据类型;
subprogram 子程序类型名 1
  features
    参数名 1: in parameter 数据类型;
end 子程序类型名 1;
subprogram 子程序类型名 2
  features
    参数名 2: in parameter 数据类型;
end 子程序类型名 2;
thread 线程类型名
  features
    端口名 1: in data port 数据类型;
end 线程类型名;
thread implementation 线程类型名.线程实现名
  calls
    {
子程序调用名 1: subprogram 子程序类型名 1;
子程序调用名 2: subprogram 子程序类型名 2;
    };
connections
  parameter 端口名 1 ->子程序调用名 1.参数名 1;
  parameter 端口名 1 ->子程序调用名 2.参数名 2;
end 线程类型名.线程实现名;
```

对应生成如下代码:

```
<线程类型名.h>
<线程类型名_线程实现名.h>:
#ifndef 线程类型名_线程实现名_H
#define 线程类型名_线程实现名_H
#include "线程类型名.h"
void 线程类型名_线程实现名(void *arg );
void 线程类型名_线程实现名_call(void );
#endif
<线程类型名_线程实现名.c>:
#include "线程类型名_线程实现名.h"
#include "子程序调用名 1.h"
#include "子程序调用名 2.h"
Parameters arg1 = {&(子程序调用名 1_feature .子程序 in 参数名 1), &(子程序调用名 1_feature .子程序 out 参数名 2 )};
Parameters arg2 = {&(子程序调用名 2_feature．子程序 in 参数名 1), &(子程序调用名 2_feature．子程序 out 参数名 2), };
void 线程类型名_线程实现名(void *arg )
{
  Parameters *threadparams = (Parameters *)arg;
   arg1.P1 = threadparams->P1;//关联
   arg2.P1 = threadparams->P1;//关联
线程类型名_线程实现名_call( );
}
void 线程类型名_线程实现名_call (void )
{
    子程序调用名 1 (&arg1);
    子程序调用名 2 (&arg2);
}
```

当线程实例化时，生成<线程实例名. h>:

```
#ifndef INCLUDE_线程实例名
#define INCLUDE_线程实例名
#include "线程类型名_线程实现名.h"
void 线程实例名(void *arg );
#endif
<线程实例名. c>:
#include "线程实例名.h"
线程类型名_feature 线程实例名_feature={0,0};
void 线程实例名( void *arg)
 {
    Parameters *params = (Parameters *)arg;
    线程实例名_feature .端口名= params.P1;
    Parameters threadinspars={线程实例名_feature .端口名};
    while (1) {
    线程类型名_线程实现名(&threadinspars);
  }
}
```

(7)规则 7: AADL 进程对应 C 中的 *main* 函数，进程负责线程的初始化，在 Windows 平台中，线程的创建函数为 *CreateThread*()。

AADL 中的进程组件代码如下:

```
process 进程类型名
end 进程类型名;
process implementation 进程类型名.进程实现名
  subcomponents
    线程实例名: thread 线程类型名;
end 进程类型名.进程实现名;
```

对应生成代码如下：

<进程类型名.h>

<进程类型名_进程实现名.h>:

```
#ifndef INCLUDE_进程类型名_进程实现名
#define INCLUDE_进程类型名_进程实现名
#include “进程类型名.h”
#include “线程类型实例名.h”
#endif
```

<进程实例名.h>:

```
#ifndef INCLUDE_进程实例名
#define INCLUDE_进程实例名
#include "进程类型名_进程实现名.h"
#include "AADL.h"
#endif
void 进程实例名(void*arg)
```

<进程实例名.c>:

```
#include "进程实例名.h"
void 进程实例名(void*arg)
{
    Parameters *process_parms= (Parameters *)arg;
    Create_Thread(线程实例名，Parms);
    while (1) {
}
}
```

3 集成编译

基于转换规则，本文通过在 OSATE 工具中增加插件的方法实现了 AADL 模型到可执行框架代码的自动生成功能。由于 AADL 运行环境位于操作系统上层，所生成的代码中关于线程的创建、数据的互斥访问等必须通过调用操作系统的相关的函数接口来实现，因此 AADL 必须与具体的操作系统平台紧密结合。中间层预定义函数均定义在 AADL.h 和 AADL.c 中。本文基于 OSATE 实现了自动代码生成器 Generator，提供了不同的 API 接口选项，可与不同的操作系统集成编译。

4 实例分析

本文以哲学家问题为例来验证 Generator 的有效性。哲学家问题的图形模型如图 2 所示，初始化了 5 个哲学家线程，系统和进程之间、进程和线程之间通过事件数据端口传递参数，线程和子程序之间通过参数传递参数，每个线程中包含实现具体功能的子程序。

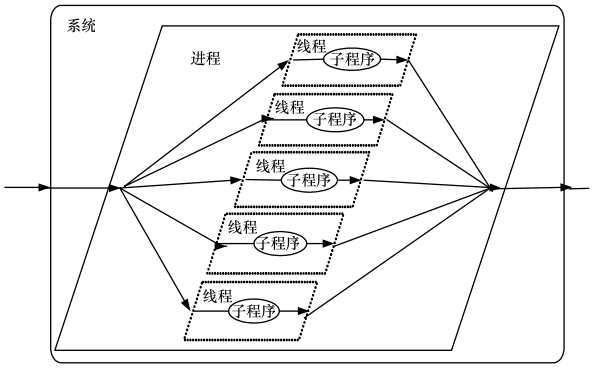

图 2　哲学家问题的图形模型

在 Generator 中输入对应哲学家问题的 AADL 模型代码，选择 Windows 操作系统为下层操作系统。Generator 在创建线程时会调用 Windows API 中的 CreateThread 函数创建线程，自动生成出对应的 C 框架代码。根据子程序 Source_Text 属性(描述功能函数所在文件的路径)和 Source_Name 属性(描述子程序在该文将中所对应的函数名)对模块代码进行调用，也就是与描述哲学家思考、饥饿、吃这 3 个状态迁移以及打印结果的模块代码函数相集成，共同编译后即可生成可执行代码，执行结果见图 3，该结果显示出了 5 个哲学家在思考、饥饿、吃这 3 个状态之间的迁移，且未发生死锁现象。

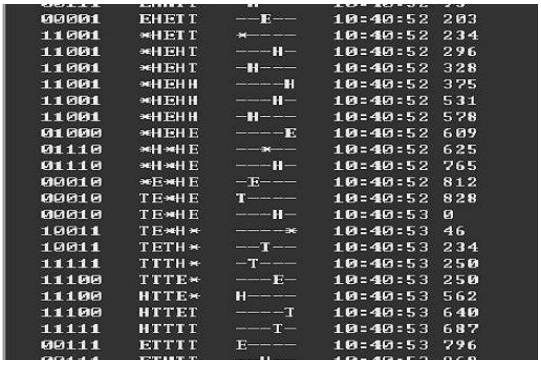

图 3　集成代码运行结果

5 结束语

本文通过对 OSATE 工具添加插件的方式设计了代码自动生成器，实现了 AADL 模型到 C 框架代码的自动转换。通过与模块代码共同编译，形成可执行的 C 代码，实现了基于 AADL 模型的代码自动生成及集成，验证了该方法的有效性。下一步的研究方向是根据 AADL 标准及 annex 对相关元素、错误属性及行为按照应用需求做出扩展，结合具体的操作系统平台应用于航空电子、舰船电子以及汽车电子等领域，同时在本自动代码生成器工具中加入调度性分析、仿真等功能。

参考文献

[1] Caspi P, Curic A, Maignan A, et al. From Simulink to SCADE/ Lustre to TTA: A Layerd Approach for Distributed Embedded Applications[C]//Proc. of LCTES’03. San Diego, California, USA: [s. n.], 2003.

[2] Clements P C, Weiderman N. Report on the 2nd International Workshop on Development and Evolution of Software Architectures for Product Families[R]. Pittsburgh, USA: Carnegie Mellon University, Technique Report: CM U/SEI-98-SR-003, 1998.

[3] SAE. AS 5506-2004 Embedded Computing Systems Committee, Aerospace Avionics Systems Division, Architecture Analysis and Design Language(AADL)[S]. 2004.

[4] 张中宝, 韩同欣, 刘西洋. 从 UML 类图到 Java 代码自动生成技术研究[Z]. (2007-06-27). http://www.paper.edu.cn/paper.php?serial_number=200706-536.

[5] 陈永春. 从 Matlab/Simulink 模型到代码实现[M]. 北京: 清华大学出版社, 2002.

[6] 陈　翔, 王学斌, 吴泉源. 代码生成技术在 MDA 中的实现[J]. 计算机应用研究, 2006, 23(1): 147-150.

编辑　顾姣健