基于SDL语言代码自动生成技术研究

吴 琦 熊光泽
（电子科技大学计算机科学与工程学院 成都610054）

Research on Automatic Code Generation Based on SDL

WU Qi XIONG Guang-Ze
(College of Computer Science and Engineering, UEST, Chengdu 610054)

Abstract As one of the key technology of CASE tools, code auto-generation has a wide application future. However, at present, some of problems limit its application in the practical project, such as executive efficiency of code generation, the combination with the hardware and software and etc. In this paper, the main factors of code auto-generation are introduced in details. The main parts of the code auto-generation based on SDL and the main factors which will effect the ultimately code performance are analyzed. The improved methods aiming at the different software and hardware platform and application performance are presented.

Keywords SDL, Code generation, Embedded system

1. 引言

目前嵌入式实时系统的复杂度越来越高，自动代码生成技术已成为缩短嵌入式系统开发时间和降低开发成本的主要方法之一，也是近十年软件开发工具研究的热门课题[1~6]。所谓自动代码生成，即把形式化描述的系统需求转化为特定软硬件平台上基于某一目标语言的系统实现。代码自动生成技术研究虽然已经取得一定成果，也被应用到一些商业工具中，但生成代码的执行效率、与实际软硬件平台的结合等问题限制了其在实际工程中的使用。根据转化程度不同自动代码生成又分为完全的自动代码生成和部分的自动代码生成，本文就基于SDL的完全自动代码生成的关键技术及其对最终代码的影响进行初步探讨。

2. 自动代码生成技术的难点分析

自动代码生成技术一般由以下基本要素组成：

1）形式化描述语言：是描述系统需求时使用的一种抽象的系统需求分析语言，其优点就是进行系统需求设计时不必考虑系统目标平台的具体细节。用于嵌入式系统的有SDL[6]、UML[7]、Z[8]、SCR[9]等等。其中，通用建模语言(UML)虽使用广泛，但目前尚只能支持自动生成代码框架；SDL语言可以与UML配合使用，弥补了UML的这一缺点，因此目前利用SDL语言进行通信系统需求描述已相当普遍，并逐渐使用于其他嵌入式实时系统。

2）嵌入式系统硬件平台：从体系结构总体上又可分为多机系统和单机系统。多机系统，即由多台可独立运行的嵌入式计算机构成的嵌入式系统，多机系统中的嵌入式计算机可以是同构的也可以是异构的，它们之间通过嵌入式通信网络相联；单机系统，即由一台独立运行的嵌入式计算机构成的嵌入式系统，单机系统又可分为多CPU和单CPU系统。

3）嵌入式系统软件平台：是指嵌入式实时系统上的系统软件，一般包括板级支持包(BSP)、嵌入式实时操作系统

```
    name CDATA #REQUIRED
    nameID ID #IMPLIED
    pattern CDATA #IMPLIED
    beginsWhen CDATA #IMPLIED
    endsWhen CDATA #IMPLIED
    preCondition CDATA #IMPLIED
    postCondition CDATA #IMPLIED
    timeToPerform CDATA #IMPLIED
)
<! ELEMENT BusinessTransaction ( Documentation *,
RequestingBusinessActivity, RespondingBusinessActivity))
<!ATTLIST BusinessTransaction
    name CDATA #REQUIRED
    nameID ID #IMPLIED
        ⋮  ⋮  ⋮
)
<! ELEMENT BusinessDocument ( ConditionExpression?,
Documentation * ))
<!ATTLIST BusinessDocument
    name CDATA #REQUIRED
    nameID ID #IMPLIED
    specificationLocation CDATA #IMPLIED
    specificationElement CDATA #IMPLIED
)
⋮  ⋮  ⋮
```

总结 本文讨论了B2B电子商务应用领域中商务协作的问题，提出了一种B2B电子商务协作模型EBCM，它通过一个商务知识库来协调协作各方的关系，其中本体库解决了异构系统间概念、术语和域知识统一与共享问题，商务库实现商务实体及其商务协作能力的注册、发现以及协作关系建立机制。同时还给出了商务协作流程层次化的定义方法，并以DTD形式进行了描述，为商务协作流程的实现提供了参考。该模型不仅是一种理论上的思想，同时还具有较好的实用性。

参考文献

1 Specification of the XML 1.0. http://www.w3.org/TR/REC-xml. 2001
2 The Common Object Request Broker: Architecture and Specification. http://www.omg.org/ 2002
3 Uschold M, Gruninger M. Ontologies: Principles, Methods and Applications. The KnowledgeEngineering Review, 1996, 11(2): 93~115
4 ebXML Technical Architecture Specification. Version 1.0.4. http://www.ebxml.org/specs, 2001
5 ebXML Business Process Specification Schema. Version 1.0. http://www.w3.org/TR/REC-xml 2001
6 ebXML Collaboration-Protocol Profile and Agreement Specification. Version 1.0. http://www.w3.org/TR/REC-xml 2001
7 Guarino N. Formal Ontology and Information Systems. In: Proc. of FOLS'98, 1998. 3~15

(RTOS)和嵌入式系统组件(Component),具体因系统而异。

4)嵌入式系统目标语言:嵌入式应用软件的实现语言,常用的有 C/C++、Ada、JAVA 和汇编语言,灵巧和功能强大的 C/C++语言目前使用最广。

形式化描述的系统需求一般是屏蔽了系统目标平台的具体细节,从较高层次对系统功能、行为和性能等方面进行描述定义,而最终的系统实现又必须是基于一定的目标平台和一定的目标语言。可见自动代码生成技术的难点和核心就是如何将抽象的高层的形式化系统需求描述映射到具体的底层的系统目标平台上,即特定目标平台上形式化描述语言到目标语言的映射。

3. SDL 到 C 语言的映射

SDL 语言使用的对象类型主要有:系统、功能块、进程、过程、数据、信号、定时器,其中前四个主要用于描述嵌入式系统体系结构,数据和信号用于描述和处理系统数据信息。对应 SDL 语言到 C 语言的映射包括系统体系结构的映射、有限状态机的映射、信号的映射、定时器的映射以及预定义数据类型的映射。

3.1 系统体系结构的映射

这一层次的映射主要包括系统、功能块、进程和过程的映射。其中系统和过程的映射比较简单,系统可映射为系统的整个 C 语言实现代码,过程可映射为 C 语言函数;关键和难点是功能块和进程的映射。

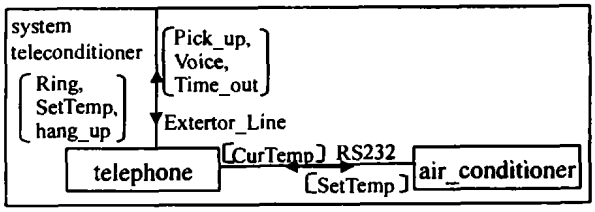

图1 一个嵌入式系统实例

·功能块的映射 功能块是一个层次性概念,在实际系统中不同的功能块、不同层次的子功能块都可能作不同的映射;对于同一功能块也有几种可能映射,且不同映射的执行效率差别很大。目前常用的处理方法是通过 SDL 扩展功能由设计人员在需求描述时对功能块映射进行限制,自动地选择优化方法尚待进一步深入研究。

图1描述了一个使用电话远程控制空调的嵌入式系统,系统包括电话功能块和空调功能块。不难看出该系统体系结构的映射关系:电话功能块映射到电话机节点,空调功能块映射到空调控制器节点。

·进程的映射 SDL 进程实际上是一个可并发运行的任务,不断处理和响应外部信号,与操作系统任务不同,它仅能在某一个执行单元上运行,不能在不同执行单元间切换。图2描述了 Telephone 功能块内部 SDL 进程及相互关系,以下对 SDL 进程映射方法的分析将以此为例。

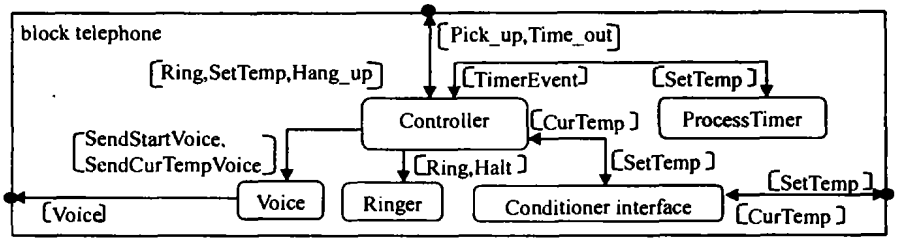

图2 Telephone 功能块的内部进程

·映射为操作系统任务(方法 A)

即将每一个 SDL 进程都映射为一个操作系统任务(或进程)由操作系统内核管理和调度,这种方式比较直观方便,能保证任务的并发执行,但占用系统资源较大,特别是当系统包含的 SDL 进程较多、能并发的进程很少时。以下是 Telephone 功能块采用本方法映射的 C 语言实现代码:

```
main()
{
   ...
   createtask(Controller); //Create controller process
   createtask(Voice); //Create voice process
   createtask(Ringer); //Create ringer process
   createtask(ConditionerInterface); //Create conditioner interface
       process
   createtask(ProcessTimer); //Create processtimer process
   starttask(ProcessTimer); //Start processtimer process
   starttask (ConditionerInterface); // Start conditioner interface
       process
   starttask(Ringer); //Start ringer process
   starttask(Voice); //Start voice process
   starttask(Controller); //Start controller process
   ...
}
```

·映射为函数调用(方法 B)

将 SDL 进程映射成为一个可重入的函数,由负责功能块运行控制的操作系统任务调用。采用本方法,系统需建立一个 SDL 进程控制块表保存每个进程的内部数据和对应函数地址,还需维护一个未处理信号表保存每个未处理 SDL 信号的类型、内容和对应进程。功能块运行控制任务循环检查未处理信号表,对每一个未处理信号调用对应的进程处理函数进行处理。本方法的优点是占用系统资源相对较少,但进程的并发性能将受到影响。功能块运行控制任务其主要 C 语言实现代码如下:

```
main()
{
  while(true) //start the checkup loop
  {
    next_signal := get_next_signal(); //Get the next signal
    if (is_not_empty(next_signal))
    {
      call_next_procedure(next_signal); //call the corresponding
          function
    }
  }
}
```

针对以上方法的进程并行性能差的缺点可作如下改进(方法 C):根据系统状况动态创建上述功能块运行控制任务。如此不同进程的外部信号可以得以并发处理,对于同一进程的信号可采用信号量等机制保证其被对应进程顺序处理。上述几种映射方法的性能比较如图3所示。

3.2 有限状态机的映射

SDL 的核心和理论模型是扩展有限状态机(EFSM)。SDL 进程具有各种不同的状态,外部事件(信号)激发进程的状态变迁,进程在状态变迁时又向其他进程发送信号。SDL

进程内部状态的变迁可以映射为C语言的CASE语句，不同的状态对应不同CASE分支，不同信号对应不同子CASE分支。

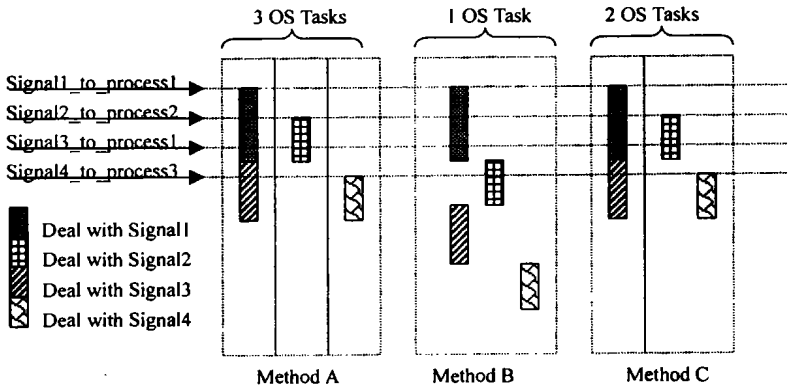

图3 几种映射方法性能比较

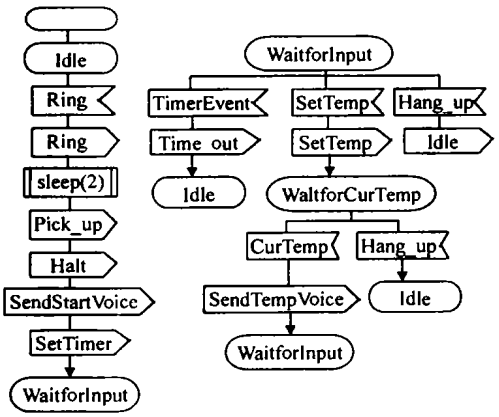

图4 Controller进程描述

图4描述了Telephone功能块的Controller进程内部状态变迁关系，Controller进程有三种状态：Idle、WaitforInput、WaitforCurTemp，接受外部信号有：Ring、SetTemp、CurTemp、TimerEvent、Hang_up。Controller进程映射为C语言的实现代码如下：

```
Controller()
{
start:
    state = Idle;
    signal = get_signal();
    switch(state)
    {
    case Idle: //Idle state
        switch(signal)
        {
        case Ring: //Ring signal
            send_signal(Ring,Ringer);
            sleep(2000);
            send_signal(Pick_up,Environment);
            send_signal(Halt,Ringer);
            send_signal(SendStartVoice,Voice);
            send_signal(SetTimer,ProcessTimer);
            state = WaitforInput;
            break;
        default:
            break;
        }
        break;
    case WaitforInput: //WaitforInput state
        switch(signal)
        {
        case TimerEvent: //TimerEvent signal
            send_signal(Time_out,Environment);
            state = Idle;
            break;
        case SetTemp: //SetTemp signal
            send_signal(SetTemp, ConditionerInterface);
            astate=WaitforCurTemp;
            break;
        case Hang_up: //Hang up signal
            state=Idle;
            break;
        default;
            break;
        }
        break;
    case WaitforCurTemp: //WaitforTemp state
        witch(signal)
        {
        case CurTemp: //CurTemp signal
            send_signal(CurTemp,Voice);
            state-WaitforInput;
            break;
        case Hang_up: //Hang up signal
            state-Idle;
            break;
        default:
            break;
        }
        break;
    default:
        break;
    }
    goto start;
end:
return;
}
```

3.3 信号的映射

信号(Signal)是SDL语言提供了一种异步通信机制，所有进程之间的信息交换都是通过交换Signal来实现。对于不同的嵌入式系统，信号可有多种映射方法。下面着重讨论同一节点上进程之间信号的几种映射方式。

·信箱方式 SDL语言的信号机制与许多嵌入式操作系统的信箱机制非常相似，所以最直观也是最方便的方式就映射为操作系统的信箱。每一个进程一个信箱，发往进程的信号映射为发往对应信箱的邮件。这种方式在进程之间通讯信息量不大时比较适合，当进程之间交换信息量较大，这就意味着大量的数据拷贝操作，必然影响生成代码的执行效率和空间需求。

·改进的信箱方式 针对上述方法的缺点，可以建立一个公共存储区域存放所要传递的数据，邮件中仅包含所要传递数据的描述信息，如地址指针、数据大小等。如此可避免大量的数据拷贝操作，但频繁的指针操作必然带来安全隐患。以上两种方式都必须依赖操作系统信箱机制的支持，当嵌入式操作系统不具有信箱机制时须采取其他方法，如操作系统的信号量机制：系统中建立一个存储信号的公共区域，每一个

SDL 进程 Pi 一个信号量 Si；当一个 SDL 进程 P1要向另一个进程 P2发送信号，先将信号存放在这个公共区域中，释放进程 P2对应的信号量 S2；进程 P2获得信号量 S2即知道有本进程的信号到达，然后到公共区域中取回信号。

3.4 定时器的映射

与 SDL 信号一样，SDL 定时器的映射同样与嵌入式系统软硬件平台相关，对于单机系统具体可采用以下两种映射方式。

·映射为操作系统的定时器　如嵌入式操作系统内核提供定时器支持，可以使用操作系统内核定时器来实现 SDL 定时器，即当一个 SDL 进程创建 SDL 定时器，相应在操作系统内核中创建一个操作系统内核定时器，当时间到达指定时刻，该操作系统内核定时器向对应 SDL 进程发送定时器到时信号。本方法比较直观，实现也比较简单，但操作系统内核定时器存在消耗系统资源多的缺点。

·使用定时器管理进程实现　嵌入式操作系统如没有提供定时器支持，SDL 定时器可以使用专门的 SDL 定时器管理任务来实现，即系统初始化时操作系统内核中创建一个专门的定时器管理任务(ProcessTimer)，当 SDL 进程创建 SDL 定时器，相应向 ProcessTimer 注册一个定时器，ProcessTimer 定时检测其管理的 SDL 定时器，当某一 SDL 定时器到达设定时间，ProcessTimer 则向对应 SDL 进程发送一个定时器到时信号。本方式实质上是在操作系统内核外实现原来的系统定时器机制，其缺点是定时器计时精度低于操作系统内核定时器。

表1　SDL 预定义数据类型

类型	值域	操作
Boolean	True,False	not;and;or;xor;=
Char	Character enclosed by ''	(;(=;);)=;Num; Chr
Integer	0;1;…	−;+;−;*;/;(;(=;);)=;Float; Fix
Real	0;…1;…;107.86;…	−;+;−;*;/;(;(=;);)=
PID	Null	none
Duration	as Real	+;−;);*;/
Time	as Real	−;+;−;(;(=;);)=
Charstring	Characters enclosed by ''	Mkstring; Length; First; List; //; (index); Substring (string; start position; length)

对于多机系统，定时器的映射还将面临两种选择，即每个节点都创建一个定时器管理进程还是整个系统仅创建一个定时器管理进程。前者，定时器到时信号传输不存在网络延迟，但不同节点的时钟存在误差。而后者，虽然消除了不同节点的时钟误差，但不可避免定时器信号在网络上传输存在延迟。如何选择，这是硬实时的分布式系统研究的难点。

3.5 预定义数据类型的映射

表1给出了 SDL 语言的预定义数据类型及其可能取值和运算操作。它们与 C 语言的基本数据类型相似，映射转换比较简单，与嵌入式系统软硬件平台基本无关，限于篇幅本文不作详细描述。

结束语　本文简单介绍了自动代码生成技术的基本概念，详细分析了基于 SDL 语言自动代码生成技术的主要环节，剖析了影响最终代码性能的主要因素：进程的并行性、数据拷贝、定时器精度等，并提出了针对不同软硬件平台和应用特性的改进办法。

参考文献

1 Manas J A, de Miguel-More T. From LOTOS to C. In: K. J. Turner, ed. Formal Description Techniques. North-Holland, 1989. 79～84

2 van Bokhoven L J, Voeten J P M, Geilen M C W. Software Synthesis for System Level Design Using Process Execution Trees. In: Proc. of the 25th EUROMICRO Conf. Sept. 8-10, Milan, Italy, Vol. I, 1999. 463～467

3 Gotzhein R, et al. Improving the efficiency of automated protocol implementation using Estelle. Computer Communications, 1996, 19 (14): 1226～1235

4 Jia X, Skevoulis S. Code Synthesis based on Object-Oriented Design models and Formal Specifications. In: Proc. of the Twenty-Second Annual Intl. Computer Software and Applications Conf. 1998. 393～398

5 Passerone C, et al. Modeling Reactive Systems in Java. In: Proc. of the Sixth Intl. Workshop on Hardware/Software Codesign, Seattle, Washington, USA, 1998

6 Hsiung P A. Formal Synthesis and Code Generation of Embedded Real-Time Software. In: Proc. of the ACM/IEEE 9th Intl. Symposium on Hardware/Software Codesign, New York, USA, 2001

7 ITU-T Recommendation Z.100 Annex F: SDL Formal Semantics Definition. International Telecommunication Union, Geneva, 2000

8 Fernandes, Joo Machado R, Santos H. Modeling Industrial Embedded Systems with UML. 8th IEEE/IFIP/ACM Int. Workshop on Hardware/Software Co-Design, E. U. A., May, 2000

9 Spivey J M. The Z Notation: A Reference Manual. Prentice Hall, 2nd edition, 1992

10 Heitmeyer, Constance L. Software Cost Reduction. Encyclopedia of Software Engineering, Vol. I, J. J. Marciniak, ed. Jan. 2002

(上接第94页)

空间数据库的整体存储体系设计的基础上，是其在纯关系数据库管理系统(RDBMS)上实现的空间数据库解决方案。两者对应关系如下：空间数据库对应于一种源关系数据库；在 RDBMS 存储中，空间数据库整体设计时的 Datasets、Database Info 和 Dataset 仅仅是个概念，而不存在具体实现；Geometry、Dataset Info、Metadata 等存储单元由 RDBMS's Table 实现。同一 Dataset 的各相关表遵循命名规则：表名加"_"加 Dataset 名。图6给出了基于 RDBMS 的空间数据库存储体系与 GeoDB 空间数据库整体存储体系的对应关系。

总结　空间数据库是一种面向地理信息系统的工程数据库，主要负责存储和管理空间信息。本文以空间数据库原型系统 GeoDB 为例，详细阐述了空间数据库的存储体系结构原理、设计和实现；讨论了空间数据库的基本存储单元——数据集；针对空间数据库的海量数据存储问题，本文提出了矢量空间数据的对象级压缩存储机制及基于 RDBMS 的空间数据库存储体系的设计实现。

参考文献

1 柏延臣，李新，冯学智. 空间数据分析与空间模型. 地理研究，1999 (2)

2 吴信才. 地理信息系统的基本技术与发展动态. 地球科学，1998(4)

3 OpenGIS Consortium, The OpenGIS Specifcation Mode Topic 1: Feature Geometry Version 3, March, 1998

4 Gtlting R H. An Introduction to Spatial Database Systems. Spatial Database Systems of the VLDB Journal, 1994, 3(4)