面向设计流图的代码支撑工具*)

戴清涵　李宣东　赵建华　郑国梁
(南京大学 计算机科学与技术系　南京 210093)

摘　要　在通信网络软件中,代码的可靠性格外重要,编码与测试在很大程度上决定着代码的可靠性。如果让机器自动生成代码,将会减少人工编码出错的概率,而且,也为代码的跟踪、测试提供了方便。为此,本文结合 UML 的相关理论,提出了设计流图的概念并描述基于设计流图的代码生成的原理及其应用。本文首先描述设计流图在软件开发过程中的作用;然后,给出设计流图的形式化定义;第 3 部分给出代码自动生成算法;第 4 部分给出基于设计流图的跟踪、测试方法,最后一部分对工具作了简要的介绍。本文详细描述了如何根据设计流图生成代码,并简要介绍实现基于设计流图的跟踪与测试。本文还实现了一个集成开发环境 AutoCodeGen,在该环境中可以编辑设计流图、对设计流图进行简单的检查、编译设计流图(生成代码)、编译代码、执行设计流图(执行编译后的代码)、基于设计流图的跟踪与测试(动态显示执行路径、当前执行点、路径覆盖情况、异常点位置等)。在实践中,利用该工具实现了 TCAP(Transaction Capabilities Application Part)协议中部分编码与解码。

关键词　设计流图,任务节点,转换边

An Supporting Tool Based on Design Flowchart

DAI Qing-Han　LI Xuan-Dong　ZHAO Jian-Hua　ZHENG Guo-Liang
(Department of Computer Science and Technology, NanJing University, Nanjing 210093)

Abstract　The stability of telecom system is especially important. Coding and testing affect the stability of code. Machine-generated code will contain less bugs than manual-typed code, and the former provides convenience for tracing and testing. For the sake of stability, this paper puts forward the concept of Design Flowchart through referencing UML related theory. This paper also describes the theory and application of auto generating code based on Design Flowchart. The first part of this paper is about the advantage of Design Flowchart in software development. The second part is about the concept of Design Flowchart. The auto-generated-code algorithm is written in the third part. The forth part describes the tracing and testing. The last part introduces the relative tool. This paper describes how to auto generate code from Design Flowchart in detail and how to trace and test the generated code in simple. This paper also realizes an IDE named AutoCodeGen. Some work can be done in the IDE, such as editing flowchart, checking flowchart, compiling flowchart, compiling code, executing flowchart (executing the compiled code), tracing and testing based on flowchart. In practice, AutoCodeGen is used to generate part of code of TCAP protocol.

Keywords　Design flowchart, Task node, Transmit edge

1　简介

在软件开发过程中,详细设计、编码、测试是软件开发周期中重要阶段,有很多因素会导致在编码过程中修改设计或在编码完成后修改设计,如果在编码过程中修改设计,那么需要维护设计文档与代码的一致性,如果是编码完成后修改设计,会大大增加编码的工作量。编码完成后需要进行测试工作,传统的测试是基于代码的,是对代码的测试,如果能对设计结果直接进行测试,那么,测试的层次将得到提高。在某些领域中,代码的正确性与安全性的要求远远高于对其效率的要求,虽然机器生成的代码的效率一般不及手工编码,但是,机器自动生成代码可以避免手工编码的很多缺陷,更重要的是,如果将开发人员与代码彻底隔离,那么,开发的效率将会得到提高,就像使用 3GL 的开发人员不需要了解由 3GL 编译器生成的比 3GL 低级的代码一样。

通信网络系统中,很多软件都还是用 C 语言来编写的,通信网络对代码的正确性、安全性、可维护性的要求相对较高,而对代码的效率没有太高的要求。模型的可维护性一般比代码的可维护性要好,可读性也好。所以,在该领域中,利用机器来自动生成代码是个可行的方案。通信网络的底层由一族协议组成,称之为信令系统(Signaling system),整个信令系统的程序运行在实时系统上(如 iRmx, VxWorks),信令系统中的协议主要是完成编码与解码,每层协议一般由一个或多个进程完成,每个进程负责处理不同的事务,进程间唯一的通信途径是消息,当进程的消息队列不空时,相应的进程就会处理其队列中的消息,进程对其消息队列中的消息的处理是串行化的,即,一个进程一次只能处理一个消息,处理完了后才能继续处理下一个消息。进程内部也不存在并发的执行路径。在协议的实现过程中,单元测试是一个重要的工作,可视化测试过程对测试将带来很大的方便。本文在自动生成非并发代码的同时,实现了对生成的代码的执行路径的跟踪,这对信令系统的协议的实现将带来很大的方便。本文实现了一个集成开发环境 AutoCodeGen,在该环境中可以编辑设计流图、对设计流图进行简单的检查、编译设计流图(生成代码)、编译代码、执行设计流图(执行编译后的代码)、基于设计流图的跟踪与测试(动态显示执行路径、当前执行点、路径覆盖情况、异常点位置等)。在实践中,利用该工具实现了 TCAP(Transaction Capabilities Application Part)协议中部分编码与解码。

*)项目基金:国家自然科学基金(6027036,602339291),863 计划(2002AA116090),省自然科学基金(BK2002079)。戴清涵　研究生,主要研究 MDA,xUML。李宣东　教授,博士生导师,主要研究方向包括面向对象技术和形式化方法,特别是实时和混成系统的模型验证算法和工具。赵建华　副教授,硕导,研究方向主要是模型检验、形式化方法。郑国梁　教授,博士生导师,研究方向主要是软件工程、软件开发环境。

本文包括如下后续内容：设计流图的形式化描述；基于设计流图的代码自动生成算法；基于设计流图的跟踪与测试；AutoCodeGen工具介绍。

2 设计流图的形式化描述

考虑到通信网络软件的特性，通过参考UML的相关模型的概念，本文提出了设计流图的概念。设计流图由任务节点与转换边组成。每个节点包含一组动作，当系统进入该节点时要执行该节点对应的动作。边由两个节点和一个条件 C 组成，两个节点分别是头节点 H 与尾节点 T，当系统处于 T 节点时，当执行完 T 节点中的动作后，如果条件 C 满足，那么系统将从节点 T 转到节点 H。设计流图中有两个特殊的节点：起始节点和终止结点。起始节点只能是尾节点，终止节点只能是头节点。

图1是一个简单的设计流图，图中使用了与UML的状态图相同的起始节点与终止节点的画法，除起始节点与终止节点外，还包含两个任务节点：TaskNode1，TaskNode2。起始节点到两个任务节点间各有一条边：Condiction1和else边；假设当前节点是起始节点，Condiction1边包含一个条件，当该条件满足时，当前节点将从起始节点切换到TaskNode1节点，并执行TaskNode1节点中的任务，如果Condiction1边的条件不满足，当前节点将从起始节点通过else边切换到TaskNode2节点，并执行TaskNode2中的任务。TaskNode1和TaskNode2到终止节点间也各有一条边，这两条边上的条件永为真。

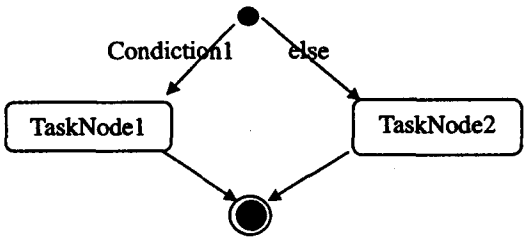

图1

设计流图的形式化描述如下：

一个设计流图是个多元组 $SC=\{S, T, C, A, sf, se, anull\}$；$S=\{s_1, s_2, \cdots, s_n\}$是个有限节点集；边集 T 是$(S\times C\times S)$的子集；$C=\{c_1, c_2, \cdots, c_m\}$是有限的条件集；$A=\{A_1, A_2, \cdots, A_t\}$是一个有限动作集合；$sf\in S$ 是起始节点；$se\in S$ 是终止节点；$anull\in A$，是一个空操作。

对于设计流图 $SC=\{S,T,C,A,sf,se,anull\}$

定义1 边 $t_i:=s_j\rightarrow c_k\rightarrow s_l$，$CurNode(t_i)=s_j$，$NextNode(t_i)=s_l$，$Condiction(t_i)=c_k$，其中 $s_j\in S, s_l\in S, c_k\in C, t_i\in T$

定义2 节点 s 的前驱节点集 $PS(s)=\{ps \mid ps\in S$ AND $c\in C$ AND $(ps\rightarrow c\rightarrow s)\in T\}$

定义3 节点 s 的后继节点集 $NS(s)=\{ns \mid ns\in S$ AND $c\in C$ AND $(s\rightarrow c\rightarrow ns)\in T\}$

定义4 节点 s 的后继边集 $TS(s)=\{ts \mid ts\in T$ AND $CurNode(ts)=s\}$

定义5 节点 s 的后继边的条件集 $CS(s)=\{cs \mid t\in T$ AND $CurNode(t)=s$ AND $Condiction(t)=c\}$

定义5 节点 s 的操作 $Operation(s)=Ai, Ai\in A$

定义6 如果 $c\in C$，那么 $ValueOf(c)\in\{true, false\}$

定义7 对于边 t，如果 $ValueOf(Condiction(t))=true$，那么称该边有效。

定义8 节点 s 的后继边集为 $TS(s)$，令 $TC(s)=\{c \mid c\in CS(s)$ AND $ValueOf(c)=true\}$，$FC(s)=\{c \mid c\in CS(s)$ AND $ValueOf(c)=false\}$，如果 $|TS(s)|>0$，那么 $|TC(s)|=1$。

定义9 对于起始节点 sf，$PS(sf)=\phi$，$NS(sf)\neq\phi$

定义10 对于终止节点 se，$NS(se)=\phi$，$PS(se)\neq\phi$

定义11 起始节点与终止节点不等，即 $sf\neq se$

3 代码生成算法

3.1 基于设计流图的代码结构

基于设计流图的代码结构是一个基于节点切换的结构，在任何时刻，都有一个当前节点CurrentNode，开始时处于起始点 sf，结束时处于终止节点 se，当某个后继边的条件满足时，当前节点将切换到后继边的头节点。代码结构如下：

```
begin
    CurrentNode:=sf;
    whileCurrentNode <> se do //如果节点不是终止节点，继续
        切换
        Do Operation(CurrentNode); //执行当前节点中的动作
        for each t in TS(CurrentNode)do //判断执行哪个切换
            if ValueOf(Condiction(t))= true then //ValueOf(c)
                对条件c求值
                CurrentNode:=NextNode(t);
                exit for
            endif
        end for
    end while
    Do Operation(se); //执行终止动作
end
```

3.2 具体的代码生成算法

代码生成算法就是根据输入的设计流图生成满足3.1节中代码结构的C代码。假设有设计流图 $SC=\{S, T, C, A, sf, se, anull\}$。在代码生成中需要对 SC 中的各个元素设计数据结构，采用如下数据结构如下：

A:(Code:String)

S:(ID:Integer,Operation:A)

C:(LogicalStatemen:String)

T:(Head:S,Tail:S,Con:C)

$sf.ID$≡0

se.ID≡0xffffffff

$anull$≡null

算法如下：

GenerateCodeFromSC(SC)函数根据输入设计流图SC生成相应的代码。

GenerateCodeByState(s)函数根据指定的节点 s，生成以 s 为尾节点的所有切换代码。

Output "…" 是打印""间的字符

Output statement 是打印表达式 statement 的值

GenerateCodeFromSC生成的目标代码是采用switch-case结构来匹配当前节点，对每个作为当前节点的非终结节点分别生成一个case分支，在case分支中生成从当前节点到所有后继结点的切换相关代码，对到不同后继节点间的切换的代码采用if-else if结构来判断哪个边的条件成立。结构如下：

```
switch(当前节点)
{
    case 节点1:
    {
        if( 边1.Condiction == TRUE )
        {
        }
        else if( 边2.Condiction == TRUE )
        {
        }
            ……
        }
        break;
```

```
        case 节点 2:
        {
            ......
        }
        break;
    }
```

具体生成算法如下:

```
GenerateCodeFromSC(SC)
    begin
      Output "state="+SC. sf. ID+";";
      Output "while(state! ="+ SC. se. ID +"){"
      Output "switch(state){"
      for each s in SC. S do
          if s. ID<>SC. se. ID then
              GenerateCodeByState(s);
          endfor
          Output "}"; //switch
          Output SC. se. Operation. Code; //终止节点动作
          Output "}"; //while
end;
```

GenerateCodeByState 生成当前节点对应的 case 分支,并生成所有后继切换的 if-else if 结构。GenerateCodeByState 通过遍历边集 T 来查找以当前节点为尾节点的边,并生成相应的 if-slse if 结构。

```
GenerateCodeByState(s)
    begin
      EdgeNum := 0;
      Output "case "+s. ID+":";
      Output "{";
      Output s. Operation. Code;
      for each t in T do
      begin
          Tail := t. Tail;
          Head := t. Head;
          if Tail = s then
          begin
              if EdgeNum=0 then
                  begin
                      Output "if(" + t. Con. LogicalStatement+")";
                  end
              else
                  begin
                      Output "else if("+t. Con. LogicalStatement+")";
                  end
              endif
              Output "state="+Head. ID +";";
              EdgeNum:=EdgeNum+1;
          end
      endfor
      Output "}"
      Output "break;"
end;
```

4 基于设计流图的跟踪与测试

如果能够在模型上体现代码的执行过程,那么,这将大大方便测试过程。这将意味着直接对模型进行测试。

4.1 跟踪方法

程序执行路径跟踪对测试有重要意义,一些要求比较高的系统在测试时对路径覆盖率有较高的要求。在设计模型上直接显示路径覆盖比在代码上跟踪要方便得多。通过完善 AutoCodeGen,可以实现路径覆盖率统计与路径执行频率统计。

AutoCodeGen 实现路径跟踪是通过修改生成的代码的基本结构来完成,在生成的代码中加入与 AutoCodeGen 进行交互的代码,AutoCodeGen 通过与生成的代码的执行进程交互,能知道执行进程的当前执行位置,并将其直接对应到 AutoCodeGen 中的模型上。AutoCodeGen 根据这些信息可以做后续分析与统计。

可跟踪的代码结构是在以上基本结构的基础上作如下调整:

```
begin
    CurrentNode:=sf;
    whileCurrentNode <> se do //如果节点不是终止节点,继续
                              切换
        Do Operation(CurrentNode); //执行当前节点中的动作
        for each t in TS(CurrentNode)do //判断执行哪个切换
            if ValueOf(Condiction(t))= true then //ValueOf
                (c)对条件 c 求值
                SendTransMsgToIDE (t);
                WaitForResponseFromIDE();
                CurrentNode:=NextNode(t);
                exit for
            endif
        end for
    end while
    Do Operation(se); //执行终止动作
end
```

SendTransMsgToIDE (t):向 AutoCodeGen 发送当前使用的边的信息,包括头节点、尾节点、边的条件。

WaitForResponseFromIDE():等待 AutoCodeGen 回送响应。测试时,通过该函数,AutoCodeGen 可以在需要的时刻再回送响应。AutoCodeGen 也可以在指定节点设置"断点",当执行进程执行到该点时,只要 AutoCodeGen 不给回送响应,执行进程将停在该处,直到获得响应。

在编译模型(生成代码)时,根据需要生成哪种结构的代码,代码的生成对用户完全是透明的。用户只要在 AutoCodeGen 中设置是要生成测试版本(可跟踪)还是发行版本(不可跟踪)。

4.2 测试

在跟踪方法的基础上可以完成测试,对于不同的测试用例,可以在 AutoCodeGen 中的模型上察看其执行路径的变化与覆盖情况。如果在测试中发现错误,定位也更加方便,如:算法错误一般导致执行路径错误;异常代码导致执行进程执行到异常处时退出,当前执行点将停留在 AutoCodeGen 模型上的错误的节点,这样错误的定位将更方便、准确。

4.3 工具介绍

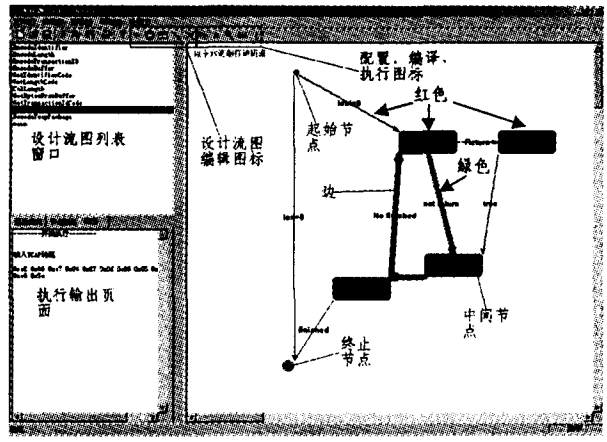

图 2

AutoCodeGen 本文作者实现的工具,该工具实现了本文中的算法,是一个集成开发环境,如图 2,提供设计流图的编辑、编译、错误定位、执行路径跟踪等功能。在编辑功能中,提供了设计流图中的节点的输入,节点中的信息的输入与修改,边的输入,边的信息的输入与修改,设计流图辅助信息的输入与修改,节点与边的删除。在编译功能中,提供了 C 编译器的配置,生成的代码类型的配置(是否是可跟踪的代码),代码自动生成,调用 C 编译器编译生成的 C 代码。在错误定位功能中,如果在编辑过程中输入了非法的信息(如非法的 C 语言表达式),那么在编译生成代码时,这些错误将被带到代码中,当 C 编译器在对生成的代码进行编译时,将会报告错误,并指出错误在生成的代码中的位置,AutoCodeGen 将会把代

码中的对应位置逆向定位到设计流图的节点或边上。在执行路径跟踪功能中，当一个项目包含一个 main 设计流图时，系统生成了一个包含 main 函数的代码，当设计流图被编译后，可以在开发环境中执行，开发环境会跟踪被执行的代码的当前执行的边或节点，并在设计流图中标出执行过的路径。

图 2 显示了一个正在运行系统，红色部分为已覆盖部分，即已经至少运行过一次；边的粗细表示该边有效的次数的多少，越粗表示累计有效次数越多。绿色的边表示当前有效的边。图 2 中即将从 Print 节点切换到 PrintHex 节点。

附录中的 C 代码是 AutoCodeGen 生成的以十六进制打印的代码，每行打印十个十六进制数，打印的十六进制的数的格式为“0xXX”。/ * User Code * /与/ * -User Code- * /间的部分是来自用户编辑设计流图时输入的动作，其余代码大部分自动生成。附录中的代码与图 2 中的设计流图相对应，是一个可跟踪的代码。详细代码参考附录。

结束语 AutoCodeGen 可以用于通信系统中的非并发代码生成。只要用户在 AutoCodeGen 中的输入合法，AutoCodeGen 将保证生成的代码的正确性，同时能管理部分与模型相关的文档，提高了编程的效率，在编程层次上也是一个提高。如果系统用 AutoCodeGen 构建，那么对系统的维护将直接从代码级提升到模型级，AutoCodeGen 能保证模型与代码的同步。

AutoCodeGen 还需要做几点改进。第一，不支持面向对象的概念。第二，缺少对并发的支持。第三，生成的代码是基于节点转换的，当设计流图中的节点很多时，生成的代码很长，代码的效率也随之降低，因此需要做节点与边的简化。第四，缺少对设计流图的正确性的检查。在以后的工作中将结合 UML 的活动图、并利用 xUML 的相关技术对这方面进行改进，使其生成面向对象的代码。

致谢 在此，我们向对本文的工作给予支持和建议的导师、同行、同学表示感谢。

附录：

图 2 中的设计流图对应的代码如下：

```
/*
以十六进制打印码流
*/
void
PrintHexCode (
BYTE * pBuf,
WORD len )
{
    UINT uint_State=0;
    int i;
    while(uint_State! =0xffffffff)
    {
        switch(uint_State)
        {
            case 0x0:/ * Start * /
            {
            SendCommand(&GlobalSocket,0x0,0x9,0x0);
                / * User Code * /
                i=0;
                printf("\n");
                / * -User Code- * /
                if( len==0 )
                {
                uint_State=0xffffffff;
            SendCommand(&GlobalSocket,0x2,0x9,0x0);
                }
                else
                {
                    uint_State=0x1;
            SendCommand(&GlobalSocket,0x2,0x9,0x1);
            SendCommand(&GlobalSocket,0x1,0x9,0x1);
                }
            }
            break;
            case 0x1:/ * Print * /
            {
            SendCommand(&GlobalSocket,0x0,0x9,0x1);
                / * User Code * /
                / * -User Code- * /
                if( i&&i%10==0 )
                {
                    uint_State=0x3;
            SendCommand(&GlobalSocket,0x2,0x9,0x3);
                }
                else
                {
                    uint_State=0x2;
            SendCommand(&GlobalSocket,0x2,0x9,0x2);
            SendCommand(&GlobalSocket,0x1,0x9,0x2);
                }
            }
            break;
            case 0x2:/ * PrintHex * /
            {
            SendCommand(&GlobalSocket,0x0,0x9,0x2);
                / * User Code * /
                printf((pBuf[i]<0x10)? "0x0%x ":"0x%x ",
                pBuf[i]);
                flushall();
                / * -User Code- * /
                if( true )
                {
                    uint_State=0x4;
            SendCommand(&GlobalSocket,0x2,0x9,0x6);
                }
            }
            break;
            case 0x3:/ * PrintReturn * /
            {
            SendCommand(&GlobalSocket,0x0,0x9,0x3);
                / * User Code * /
                printf("\n");
                / * -User Code- * /
                if( true )
                {
                    uint_State=0x2;
            SendCommand(&GlobalSocket,0x2,0x9,0x4);
                }
            }
            break;
            case 0x4:/ * Mid * /
            {
            SendCommand(&GlobalSocket,0x0,0x9,0x4);
                / * User Code * /
                i++;
                / * -User Code- * /
                if( i<len )
                {
                    uint_State=0x1;
            SendCommand(&GlobalSocket,0x2,0x9,0x5);
                }
                else
                {
                    uint_State=0xffffffff;
            SendCommand(&GlobalSocket,0x2,0x9,0x7);
            SendCommand(&GlobalSocket,0x1,0x9,0xffffffff);
                }
            }
        break;
    }
    }
    / * User Code * /
    printf("\n");
    / * -User Code- * /
    };
```

以上代码中，SendCommand 同时封装了 SendTransMsgToIDE 和 WaitForResponseFromIDE，SendCommand 的函数体完全是自动生成的。代码编译执行时，是通过 TCP/IP 与开发环境交互的，如果用户的设计流图中包含 main 设计流图，生成的 main 函数将自动包含网络初始化等相关代码，即，所有与开发环境交互的代码对用户都是透明的。

参 考 文 献

1 OMG Unified Modeling Language Specification. March 2003 Version 1.5

2 Mellor S J, Balcer M J. Excutable UML

3 David S. Frankel. Model Driven Architecture

4 严蔚敏，吴伟民. 数据结构(C 语言版). 北京：清华大学出版社，1997

5 Donald Hearn M, Baker P. 计算机图形学. 北京：电子工业出版社，2003