基于 C++语言转换的 TTCN-3 测试系统的设计与实现

Design and Implementation of TTCN-3 Test System Using C++ Language Translation

张 辉 蒋 凡 (中国科学技术大学计算机科学与技术系 合肥 230027)

摘要:TTCN-3 是一种用于协议与软件测试的标准化测试语言,文章提出了一种实现 TTCN-3 测试系统的设计方法,将用 TTCN-3 书编写的抽象测试套先翻译为 C++语言,进而编译生成可执行的测试套,然后调度执行。并用这种方法实现了 TTCN-3 测试平台 TTPlatForm,运行结果表明,这种基于语言转换的 TTCN-3 测试系统在性能、可移植性和灵活性均比采用编译生成成内存对象并调度内存对象执行的方式有了较大的提高。

关键词:TTCN-3 语言转换 C++ 测试系统 TTPlatForm

TTCN-3(Testing and Test Control Notation Version 3)是一种具有广阔应用前景的标准化测试语言。TTCN-3 核心语言是一种类似常用高级语言的测试语言,它除了具有高级语言所共有的特性外,还包括许多测试专用的语言结构和语言对象,提供强大的模板及模板匹配功能[3]。

TTCN-3 测试系统的已有实现方案主要有两种。一种是将 TTCN-3 语言转换到 C 语言或者 Java 语言,再用 C 编译器转换为可执行代码或者用 Java 虚拟机执行;另一种是先将 TTCN-3 语言编译成内存对象,然后再执行的方案。编译执行的方案[1]容易实现调试执行,但是由于不能生成目标代码,每次执行都需要重新编译链接,故使用起来不是很方便,功能也有一些局限。

我们提出的方案是使用 C++作为中间语言。即先将 TTCN-3 抽象测试套转换为 C++语言,然后再生成可执行的代码。采用这种方案构建的测试系统的具有很好的移植性和灵活性,与转换到 C 语言相比具有更好的可读性,相比于 Java 语言则不需要虚拟机支持,因而使用更方便。同时,基于该方案,实现了 TTCN-3 测试系统 TTPlatForm。

本文内容安排如下:第一段讨论了 TTCN-3 抽象测试套转换为 C++语言的翻译转换的方案设计;第二段讨论 TTPlatForm 测试系统的总体架构;第三、四段分别讨论了编译子系统和执行子系统的实现,最后是总结和展望。

1 TTCN-3 到 C++的翻译转换方案

TTCN-3 是一种强类型的语言,包含和 C++很多相似的特性,这使得 TTCN-3 与 C++的语言之间互相转换成为可能,文[2]将 C++语言映射到了 TTCN-3,提出了使用 TTCN-3 测试 C++软件的方法。但 C++到 TTCN-3 和从 TTCN-3 到 C++是两个不同的过程,故需要重新设计从 TTCN-3 到 C++的翻译转换方案。

分析 TTCN-3 的核心语言可以发现,TTCN-3 的很多简单类型比如 integer、float 等都可以直接对应到 C++的简单类型,但是对用户自定义类型,则不能简单的映射。

```
type integer str3 (4..infinity,77);
```

图 1 构造子类型举例

例如,图一的类型申明定义了整形子类型 st3,该子类型要求类型具有值域为 4 到无穷大和 77。显然,这个类型不能简单的映射到 C++的某个简单类型,而必须用类来表示这种复杂类型。在类中,必须有个方法来获得其类型名,同样需要某个成员来描述其类型限制(在此是值域范围)。同时由于 TTCN-3 支持 import 机制[3],需要在模块之间传递类型信息,因此,需要某个成员来表示该类型所属的模块,考虑到赋值

和拷贝的问题，还需要设计拷贝构造函数和 clone 函数，最后还需要实现用于类型相容性检查的方法。

基于以上分析，为了完整传递类型和值信息，我们设计了"类型－值"这样的表示结构，"类型"描述运行时值的类型信息，且可以创建一个值；"值"完成具体的运行操作，并维护其对应的类型信息，用于运行时类型检查。

在具体实现中，由两个公共基类 CType 和 CVaule 表示对应的抽象类型和抽象值。在 CType 和 CValue 下，封装了所需要的各种类型和值操作，比如类型检查，取值，赋值等操作。所有的类型都从 CType 继承，类型的值从 CValue 继承。

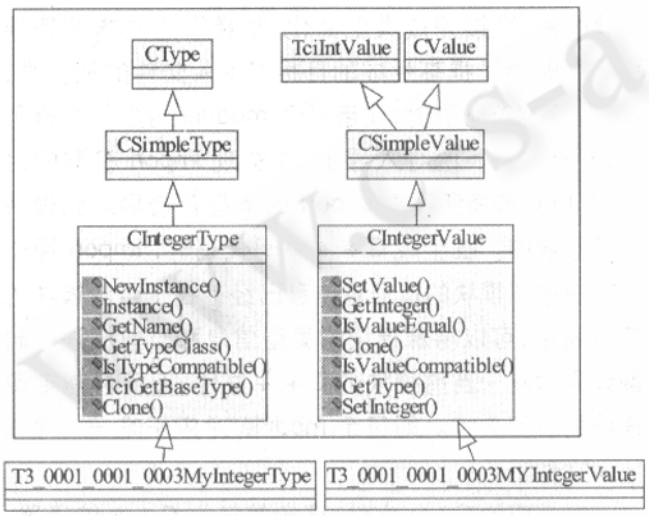

图 2　integer 类型设计

对上面的例子，其 UML 视图见图 2。对于复杂的结构类型，继承方式与简单类型有所不同，这是由于结构类型一般都需要存储多个数值，且类型上的操作也不同，比如 record 需要支持域操作。故对于结构类型设计了不同的基类：CStructedType 和 CStructedValue，分别从 CType 和 CValue 继承，里面的各 field 函数用来处理和域相关的各种类型操作和值操作。其 UML 视图如图 3。

对于 TTCN－3 其他中特有的定义比如 module、control、testcase、altstep 等，其翻译与一般的类型翻译不同，以 testcase 为例，由于 testcase 具有类型信息，它包含 system、runs on 子句，可以传递参数，同时又含有语句块，里面可以包含变量声明，语句等，翻译的时候为了避免类的信息过于复杂，把 testcase 翻译设计为三部分，一个类型类，CTestCase，处理 system 和 runs on 信息，同时记录该 testcase 所在的模块和参数信息；一个 body 函数，具体实现和每个 testcase 相关的各种操作，比如变量赋值，语句执行，日志输出等等；再就是一个 instance 声明，声明和 testcase 相对应的实例。

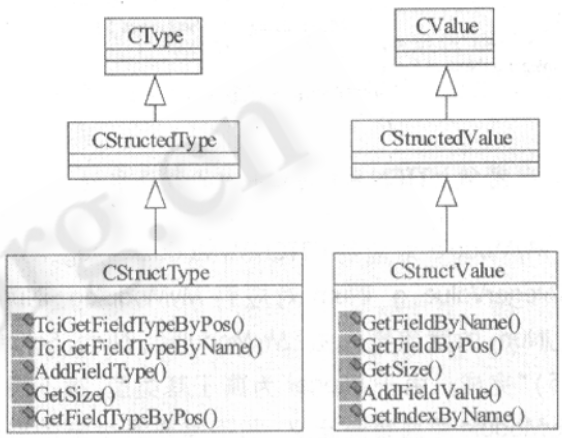

图 3　结构类型设计

module 模块也被封装成了一个类，里面所包含的所有信息，诸如类型声明、函数声明、testcase 声明、control 部分均翻译成了该 module 类的嵌套子类。这样，翻译后的文件在层次上是和原始的 TTCN－3 文件是一一对应的。

为简化翻译层次，具体的翻译方案分为两部分，一部分基类型系统，定义了所有的基础类型信息，这一部分与具体的 ATS 无关，在图一中的矩形框中包含的部分即是 integer 的基类型系统，另一部分是每个 ATS 在翻译时需要实现的部分，实现与 ATS 本身相关的各种派生类定义。以前面的 str3 声明为例，在翻译文件中只需要实现两个相关的派生类：T3_0001_0001_0003str3Type 和 T3_0001_0001_0003str3Value，而不需要实现其基类如 CIntegerType 或 CIntegerValue 等。

为了避免命名冲突，增强标识符描述的语义信息，从 TTCN－3 到 C＋＋的命名转换，主要采用加前缀或者加后缀的方式或者同时加前缀和后缀的方式。比如，对简单类型，采用命名转换规则："C" ＋ Name ＋ Value，这样，integer、float、hexstring 分别被翻译成 CIntegerValue、CFloatValue、CHexstringValue。而图三中的 T3_0001_0001_0003str3Type 则是"模块名＋类型名＋type"的模式。

图 4 是一个 TTCN－3 测试脚本的转换实例，在模块中，只有一个常量声明：

```
Module MyModule
{
const integer g_iNum := 5;
}
```

```
CTTCNMyModule*g_CTTCNMyModule= NULL;
class CTTCNMyModule public CModule
{
 CIntegerValue MyModuleg_iNum;
public:
 CTTCNMyModule() :CModule( "CTTCNMyModule" )
 {
     MyModuleg_iNum.SetInteger( 5 );
     ……
 }
 ~CTTCNMyModule(){}
};
```

图 4　TTCN－3 到 C＋＋的翻译举例

MyModule 对应到 CTTCNMyModule，integer 对应到 CintegerValue，g_iNum 对应到 MyModuleg_iNum，对 g_iNum 的赋值操作由"MyModuleg_iNum. SetInteger(5)"完成。由于 integer 为属于基类型，故不需要在 MyModule 里面重新定义，即不需要派生新的嵌套子类，而是直接对应于 CintegerValue。

2　TTPlatForm 测试系统的总体结构

依据 TTCN－3 标准，测试系统由以下六个模块组成：测试管理模块（TM）、编译模块（TC）、执行模块（TE）、编解码模块（CD）、系统适配（SA）和平台适配模块（PA）[3]。采用语言转换方式实现测试系统，TC 需要将 TTCN 的抽象测试套（ATS）转换为 C＋＋的文件格式，然后调用第三方编译器将其编译为对应的可执行测试套（ETS），在我们实现 TTPlatForm 的中，采用了编译生成 DLL 的方式。TTPlatForm 具体的框架图如图 5。

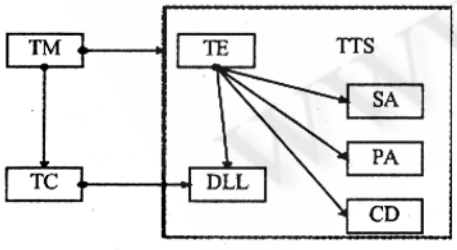

图 5　TTPlatForm 测试系统总体结构图

在具体实现中，系统主要划分为了三块：测试管理子系统（TTM）、编译子系统（TTC）、测试执行子系统（TTS），TTM 对应于标准中的 TM 模块，负责测试套的输入，测试工程的管理，编译器和执行器的装配以及模块参数等的配置；TTC 对应于 TC 模块，编译测试例，依据翻译方案翻译为相应的 C＋＋文件，然后调用第三方 C＋＋编译器编译成对应的 DLL 文件；TTS 完成测试环境配置、测试执行、结果输出等功能。

3　编译子系统

编译子系统将 TTCN－3 核心语言格式的 ATS 转换成基于以上翻译转换规则的 C＋＋测试例。编译器的实现采用两次扫描：第一遍扫描 TTCN－3 脚本，语法分析，建立语法树，处理模块间的导入导出，并处理所有的引用；第二遍扫描语法树，静态语义检查，并生成 C＋＋中间格式。

为了简化翻译操作，提高 TTS 的执行效率，在静态编译阶段，直接完成类型简化、常量表达式求值等操作。同时为了提高生成的目标 C＋＋文件的独立性，TTC 在实现符号表时，使用一个 module 对应一个符号表，支持符号表的导入导出，在实现 import 机制的时候，可以直接将所需要 import 的类型直接导入到模块的符号表中。在生成 C＋＋中间代码时，import 操作所对应的源模块的符号表信息已经存在于当前模块的符号表中，可以将源模块的类型信息和当前模块一起翻译到对应于当前模块的 C＋＋文件，而不再需要源模块的类型支持。即每个 module 对应于唯一一个 C＋＋文件。

对生成的 C＋＋文件，调用第三方 C＋＋编译器，联合执行器模块的预定义头文件和支持库，编译生成与 module 对应的 DLL 文件，由于每一个 module 对应唯一一个 C＋＋文件，编译之后，也将对应到唯一的 DLL 文件。

4　执行子系统

经过编译的 DLL 测试套已经包含了原始 ATS 相关的各种执行信息，但是为了执行这些 DLL，还需要执行系统的支持，比如加载与 DLL 对应的 SA、PA、CD，实现时钟功能，数据发送、接收、匹配操作，启动 PTC，等等，都需要在执行系统中提供相应的实现[3]。

TTS 主要分为三个模块：执行机子模块（TE）、类型和值子模块、行为管理子模块。

执行机子模块，加载并执行 ETS。主要包括 Component、Port 的管理、定时器的操作、执行状态的管理（如 mtc，system，control，global verdict 等）、执行器线

程的管理以及快照语义的支持等等。实现时，通过 component、port、timer 容器来维护所有执行器的对应状态，利用锁机制实现线程的调度执行。

类型和值子模块，TTCN 抽象测试套转化后的 C++测试套中所有的派生类型的基类需要在本模块中实现。例如前面的 CIntergerValue、CIntergerType 和 CStructValue、CStructType 及它们的基类等都需要在该模块中实现编码。本模块中，实现所有的基础类型架构和与类型相关的各种操作，具体组织见前面的翻译方案。

行为管理子模块，TTCN－3 抽象测试套中 control，testcase，function，altstep 都是封装了一定执行行为的行为定义体，此模块负责正确启动管理这些行为。在实现上，采用了多线程调度执行的方式，较[1]中的单线程模拟执行有了很大的进步。为简化实现，使用开源多线程库 Zthread[5] 实现多线程管理和执行，具体的设计类视图见图 6。

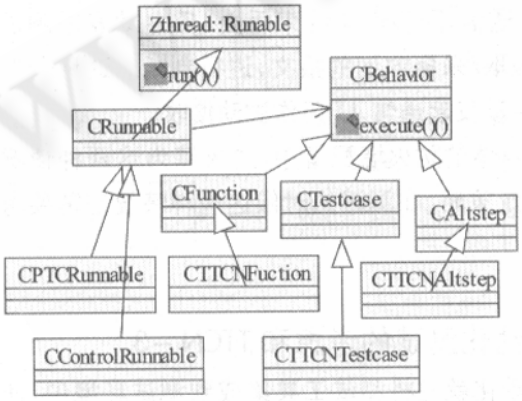

图 6　TTS 行为管理子模块视图

CRunnable 实现多线程的启动操作，CPTCRunnable 和 CControlRunnable 分别对应与 PTC 和 Control 的执行操作。CRunnable 在启动线程时调用 CBehavior 的 execute( ) 方法。CFunction、CTestcase、Caltstep 分别对应与 function、testcase、altstep 的执行行为操作，三者均继承于 CBehavior 类，里面的 execute( ) 方法实现相应的执行语义。比如在某 PTC 中需要执行一个 testcase，则由 CPTCRunnable 的 run( ) 操作调用对应的 CTTCNTestcase 的 execute( ) 操作，从而实现多线程的启动执行。

TTS 还要实现 SA、PA、CD 的配置以及日志功能等。SA、PA、CD 的实现与一般的 TTCN－3 测试系统实现没有区别，在此不详述。

5　总结和展望

文中提出的 TTCN－3 的翻译转换方案，已在设计的测试系统 TTPlatForm 中实现。使用该系统，我们对 linux 系统下的 iptables 防火墙进行了测试，取得了很好的效果。

由于生成了可执行代码，测试人员只需要直接加载执行测试例，这比使用编译内存对象执行的方式灵活得多。由于使用了多线程调度的方式，较[1]的单线程运行方式在性能上也有了一定的提高。整个系统均使用标准 C/C++实现，充分地考虑了平台无关性，具有很好的可移植性。

同时，也要看到使用翻译转换成 C++方案的不足：由于将 TTCN－3 的各种类型都翻译成了 C++类，较之原始的 TTCN－3 的抽象测试套，翻译之后的文件要长很多，一般是 4 到 6 倍，有些达到 10 倍左右，一定程度上影响了从 C++编译生成 DLL 的速度，对执行效率也有一定的影响。进一步的研究翻译转换设计方案，高效的生成中间 C++代码的组织形式将是下一步工作的重点。

参考文献

1　面对对象的 TTCN－3 测试系统的实现 刘小勇，蒋凡 计算机工程 Vol. 32 No. 10 2006. 5.

2　mapping C++ Data Types into a Test Specification Language，Pekka pulkkinen，VTT PUBLICATIONS 542，ESPOO 2004.

3　TSI ES 201 873－1 V3. 1. 1 Methods for Testing and Specification (MTS)；The Testing and Test Control Notation version 3；Part 1：TTCN－3 Core Language，2005. 6.

4　http://www. antlr. org

5　http://sourceforge. net/projects/zthread/