TTCN-3 Test Architecture Based on Port-oriented Design and Assembly Language Implementation

Dihong Gong,
Wireless Information Network Lab
University of Science and Technology of China
Hefei, China, 230027
shidizi@mail.ustc.edu.cn

Sihai Zhang
Department of Electronic Engineering and Information Science
University of Science and Technology of China
Hefei, China, 230027
shzhang@ustc.edu.cn

***Abstract*—The TTCN-3(Testing and Test Control Notation version 3)[1-2] based test systems are widely used for protocol testing in various technical system, but fall to be inefficient in both compiling and running, since the implementation are primarily based on high-level programming language and interpretation-execution mechanism. Viewing the test system as ports state system, this paper proposes a port-oriented, and compiling running mechanism based TTCN-3 test system, which is implemented primarily by assembly programming language. The test process is divided into the implementation of a series of two phases: the first test phase is environment computation which is implemented by assembly language necessary for the following port operations and the second phase is corresponding port operations which can be implemented by higher-level programming language. The assembly language implementation for the TTCN-3 abstract test suit allows much more computational optimizations at a lower level, and the compiling-running mechanism outperforms the performance of interpretation-execution mechanism by saving the efforts for compiling whenever the TS is executed and CPU's sources for interpretation during runtime.**

Keywords-TTCN-3,Test System, Protocol conformance testing

I. INTRODUCTION

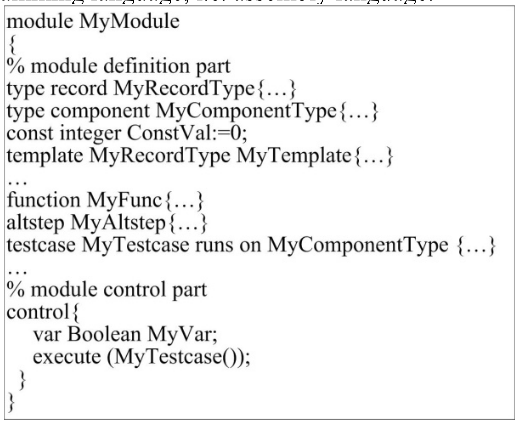
PROTOCOL conformance testing[3] is an experimental method aiming at checking the correctness of some implementation and its conformance to some standard protocols. In the area of protocol engineering, conformance testing is an essential step to determine what extent a single implementation of a particular standard conforms to the requirements of that standard. Conformance testing can increase the confidence of implementation complying with the protocol specification, and enhance the probability of inter-operability between different implementations.

The design for the architecture of protocol conformance testing has gained much attention during the last several decades [4-6]. TTCN-3 is a strongly typed test scripting language used in conformance testing of communicating systems and a specification of test infrastructure interfaces that glue abstract test scripts with concrete communication environments. TTCN-3 has been developed by ETSI, with the predecessor of TTCN-2. Although TTCN-3 originates from TTCN-2, great changes have been taken place in grammars by introducing many new grammatical features such as dynamic testing configurations, communication mechanisms, template mechanisms, and so on. These new features will greatly adapt TTCN-3 to new software structures as well as the next generation Internet protocols. However, such new features also demand a new testing environment for TTCN-3 since the original environment for TTCN-2 has already been out-of-date.

There are currently two proposals for the design of TTCN-3 based test system. The first one inherits the idea of TTCN-2 by compiling the TTCN-3 Abstract Test Suit (ATS) directly into corresponding high-level programming languages, i.e. C++ and Java, and then linking with necessary libraries to get an executable platform-dependent Test System (TS). This design has its merits of high efficiency in running and easy transplant among different platforms. However, it fails to map all TTCN-3 grammars to corresponding high-level programming languages and as a result, limits the application of TTCN-3. In addition, it is not easily to for user to locate runtime errors because the testing process is almost invisible. To improve to performance of the first proposal, another approach based interpretation-execution mechanism was proposed [7]. The primary idea is to compile the ATS to C++ objects in the memory, and then interpret these objects into corresponding machine instructions for execution by an executor, namely TTCN3Runner. The second approach is more powerful in mapping because it has an executor to interpret the C++ objects. Moreover, the executor can manage the process of testing, which makes it easier to locate possible errors. However, this approach is not without its shortcomings. First of all, we need to compile the ATS to C++ objects whenever we run the test system. In addition, it costs extra CPU sources to interpret C++ objects into corresponding machine instructions. Finally, the high-level programming language representation for ATS allows little optimization compared to low-level programming language representation.

On the balance of the foregoing two ideas, and taking the features of TTCN-3 language into consideration, we propose a new design for the TTCN-3 based test system. The main idea of the proposal lies in the fact that the testing process involves the interaction between TS and System under Test (SUT), in the process of which only ports in TS are visible to

SUT. Consequently, the testing process can be considered as communication between TS and SUT via ports that driven by a series of ports operations, each of which can be divided into two phases: Environment Computation (EC) for necessary information to carry out port operations and the corresponding Port Operation (PO) for the interaction between TS and SUT or within TS. As result, if we schedule these POs reasonably and carry out the corresponding environment computations, a test system comes into being. On the analysis of TTCN-3 grammars, we find that the only two challenges for the implementation of TS are communication mechanism and timer mechanism. The communication problem actually involves POs and can be solved by our proposal by introducing SUT-dependent Application Programming Interface (API), namely SUT Adapter (SA). However, the timer mechanism is platform-dependent and we so need another API to adapt the TS to any platform, namely Platform Adapter (PA). Getting rid of the two challenges, the EC part in ATS can be compiled easily into low-level programming language representations. Summarily, a TS described by TTCN-3 ATS can be implemented by scheduling POs reasonably and carry out corresponding EC which can be implemented by low-level programming language, i.e. assembly language.



```
module MyModule
{
% module definition part
type record MyRecordType{…}
type component MyComponentType{…}
const integer ConstVal:=0;
template MyRecordType MyTemplate{…}
…
function MyFunc{…}
altstep MyAltstep{…}
testcase MyTestcase runs on MyComponentType {…}
…
% module control part
control{
   var Boolean MyVar;
   execute (MyTestcase());
  }
}
```

Figure 1. A Segment of TTCN-3 Program

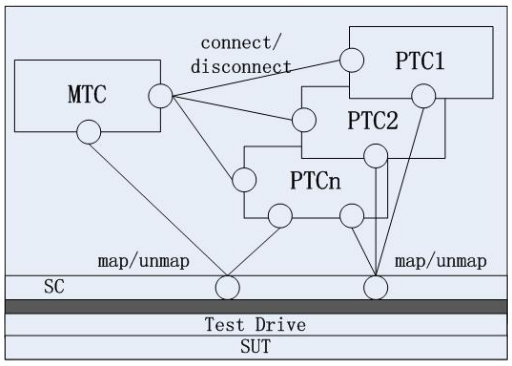


Figure 2. TTCN-3 Testing Configuration

The rest of this paper is organized as follows: First in section II gives a general introduction to the grammar features of TTCN-3. Then the section III introduces the design for the framework of TTCN-3 based TS and the following section IV illustrates the detailed implementations of the proposed design in section III. Finally, section V concludes the advantages of our proposed design and points out necessary efforts in future.

II. General Introduction to The TTCN-3 Language

A. *The Framework of TTCN-3 programs*

TTCN-3 is a modularized language with the top entity of module which can be parameterized to adapt to different testing environments and testing options. The module includes lower-level entities of module definitions, optional control part and attributes declarations. The module definition part is used to declare data types, constant values, module parameters, component types, functions, altsteps, testcase and so on. In addition, TTCN-3 allows importing definitions from other modules, which amalgamates TTCN-3 with the languages of ASN.1, SDL, etc. The control part defines the execution of testcases. Local variables, constant values, timers and so on. Fig.1 shows a segment of TTCN-3 program.

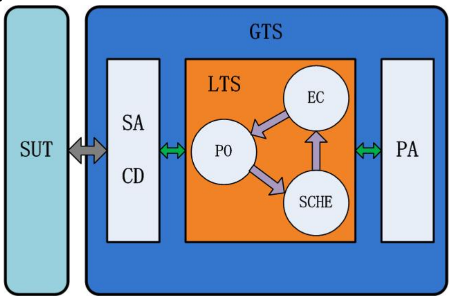


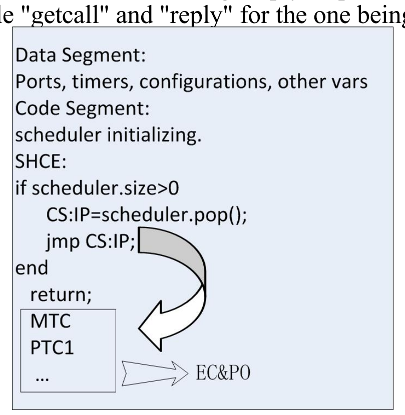
Figure 3. The Framework of TS

B. *Dynamic Testing Configuration*

TTCN-3 supports the dynamic definition of testing configurations of TS which consist of one Main Testing Component (MTC), one System Component (SC) and several optional Parallel Testing Components (PTCs). Fig.2 shows a classical TTCN-3 testing configuration. The MTC is automatically created by TS when testcase is executed and exists throughout the testing process, while PTCs need to be created, started or stopped by corresponding operations, visibly. MTC, PTCs, and SC interact with each other through their ports which can be connected, disconnected, mapped and unmapped dynamically during the testing process.

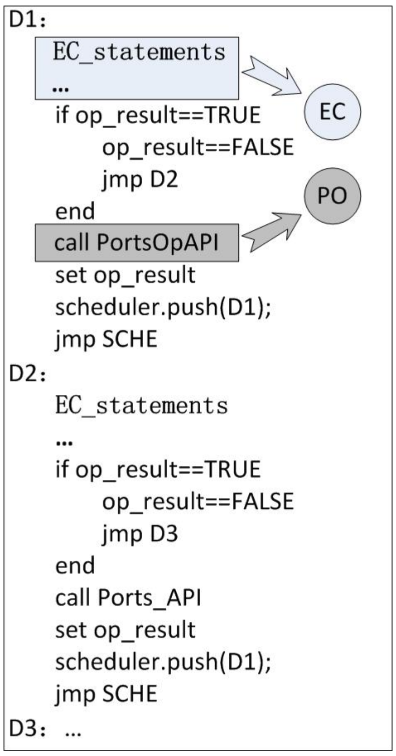
C. *Communication Mechanism*

TTCN-3 supports two kinds of communications: message based communication and procedure based communication. In terms of message based communication, TTCN-3 provides "send" and "receive" operations. The "send" operation sends one object to the message queue of the receiver and objects can be constant or variable values, templates and so on, while the "receive" operation gets one object from corresponding message queue. If template matching mechanism is used in "receive" operation, the object will stay in the queue until it hits the matching template. In terms of procedure based communication, TTCN-3 provides "call" and "getreply" operations for the caller, while "getcall" and "reply" for the one being called.



```
Data Segment:
Ports, timers, configurations, other vars
Code Segment:
scheduler initializing.
SHCE:
if scheduler.size>0
    CS:IP=scheduler.pop();
    jmp CS:IP;
end
  return;
  MTC
  PTC1
  ...
```

Figure 4. The Program Framework of LTS



```
D1:
    EC_statements
    ...
    if op_result==TRUE
        op_result==FALSE
        jmp D2
    end
    call PortsOpAPI
    set op_result
    scheduler.push(D1);
    jmp SCHE
D2:
    EC_statements
    ...
    if op_result==TRUE
        op_result==FALSE
        jmp D3
    end
    call Ports_API
    set op_result
    scheduler.push(D1);
    jmp SCHE
D3: ...
```

Figure 5. The Implementation of Scheduled EC and PO

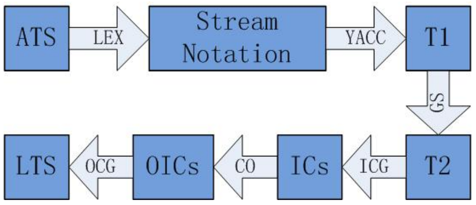


Figure 6. The Framework of TTCN-3 Compiler

III. The Design for The Framework of TTCN-3 Based TS

According to the TTCN-3 description for TS, the testing is a process of interaction between MTC, PTCs and SC and we can easily construct the implementation model of multiple threads for TS. In the multiple threads model, every component is implemented by a thread, with its own message queues and local environments. When the testcase is executed, main thread corresponding to MTC is started, and then other threads are invoked or stopped by corresponding configuration operations. The CPU schedules all the threads in parallel in CPU time slot level and it seems that they are working synchronously to the SUT. However, this obvious construction has many flaws. The first one lies in the fact that the implementation of the multiple threads relies on the Operating System (OS) where TS runs, which makes the TS hard to be transplanted among different platforms. The second one is about the inefficiency of communications. Communications within TS are equal to communications between threads of TS, which requires efforts of the OS for synchronization. Finally, the invoking and suspending of threads will further decrease the execution efficiency by adding extra burdens of saving and restoring local environments of threads on CPU.

Looking from the primitive construction, it is clear that all the drawbacks were brought about by multiple threads which were initially used to imitate the parallel computation of components. However, the fact is that the parallel attributes of TS will only be manifested when POs are involved, and such parallel attributes guarantee corresponding environments for POs have been calculated. Consequently, the parallel computation can be imitated in PO level, rather than CPU time slot level. If POs can be scheduled reasonably (by a scheduler, SCHE), and the corresponding environment can be calculated accordingly (by Environment Computation, EC), the parallel computation of the TS can be imitated. Fig.3 illustrates the basic framework of this idea.

Since the timer mechanism is platform-dependent, a PA is introduced to provide time service, so that the Local Test System (LTS) can be platform-independent. In additional, Code-Decode (CD) module is also introduced to adapt TS to SUT with different coding mechanism. Finally, SA serves as adapter between TS and SUT in different communication environment. Combining the SA, CD, LTS and PA, we get the Global Test System (GTS), a TS on specific platform, corresponding to specific SUT.

IV. The Implementations of TS

In this section, we introduce an assembly language based implementation of LTS with high-level APIs and the combination of SA, CD, LTS and PA to get the GTS, respectively.

A. The Framework of LTS Program

The LTS is a platform and SUT independent module that consists of three parts: SCHE, EC and PO (see Fig.4). The SCHE serves to arrange the computation of EC (by directing the CPU to corresponding CS:IP) and the PO carries out port operations corresponding to the calculated EC. The LTS is represented primarily by assembly language, with APIs involving timer service from PA and communication service between TS and SUT from SA. However, we do not localize different communication ports within LTS. Consequently, communication service within LTS will be implemented simply by a procedure of LTS, which operates the global message queues accordingly. A segment of LTS program indicating the program framework of LTS is showed in Fig.4, and the implementation of each scheduled EC is illustrated in Fig.5.

B. *The Framework of TTCN-3 Compiler*

The assembly language representation of ATS is the target output of our TTCN-3 compiler. The TTCN-3 compiler compiles TTCN-3 ATS to the corresponding LTS, leaving timer or communication service APIs. The structure of the compiler shares many similarities with the classic compiler, i.e. C language compiler. Fig.6 shows the basic structure of our TTCN-3 compiler. The compiler first scans ATS by a lexical analyzer LEX [8], getting the notation stream of ATS. Since the TTCN-3 supports import and forward referencing mechanism, we set up the semantic tree by a semantic analyzer YACC [8] without checking the legality of grammars, getting the first semantic tree, namely T1. Then the T1 is applied to a Grammatical Scanner (GS) to check possible grammatical errors, getting T2 if successfully. Next step the Intermediate Codes (ICs), a three-address notation of ATS, are generated from T2 by an Intermediate Code Generator (ICG). A Code Optimizer (CO) is also applied in this stage to get a more effective notation of ATS, the Optimized ICs (OICs). Finally, an Object Code Generator (OCG) translates the OICs with corresponding 80386 machine instructions, getting the LTS.

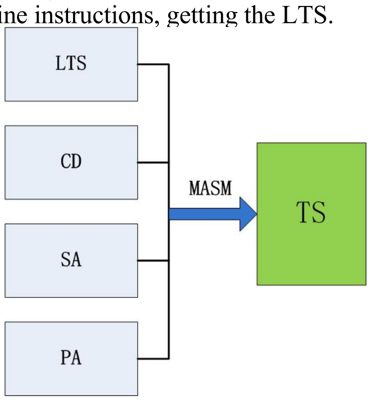


Figure 7. The Generation of GTS

C. *The Implementation of GTS*

The LTS is a platform and SUT independent assembly language representation of ATS, and the GTS is the completed TS adapting to specific platform and SUT. Given specific SUT and platform, we can develop our CD, SA and PA with high-level programming language, i.e. C++, to give the implementations of APIs. This approach is effective in developing because there have been applicable template for APIs, i.e. socket communication mechanism. However, the employment of high-level programming language won't obviously decrease the efficiency of TS since the communication mechanism depends much more on communication environment such as wire speed, compared with programming language. When LTS and all the necessary modules for API are ready, the executable TS can be generated by third-party assembly language compiler, i.e. MASM. Fig.7 demonstrates how GTS is generated.

V. CONCLUSIONS

This paper focus on providing a design for the TTCN-3 based test system, from construction, to system architecture, and finally implementations. Our design presented in the paper has, in principle, advanced in several aspects from the existing designs. First, our proposed configuration is primarily a low-level programming language representation for ATS. The low-level representation allows much more optimizations on machine instructions level, compared with the existing high-level representation. Second, our design is based on compiling-running mechanism, which means that our design is more effective in both compiling and running. The compiling-running mechanism does not need to compile the ATS to pseudo-codes for interpretation by an executor, and at the same time, it saves the CPU's resource by directly running the TS, rather than maintaining an extra executor for interpretation. Finally, the LTS/GTS structure is robust in both platform-SUT independence of LTS and convenience in developing of APIs. In order to minimize the unavoidable platform-SUT dependence of the TS, we construct a platform-SUT independent LTS with necessary APIs, the implementation of which is platform-SUT dependent. When the developed TS is applied to different platform or SUT, we just need to reconstruct the implementation of APIs, without changing the implementation of the TTCN-3 compiler.

In summary, our design is encouraging in the optimized implementation of TTCN-3 based test system. But this paper only represents a start of this challenging task. Many more efforts are required to develop a well-rounded TTCN-3 based TS.

ACKNOWLEDGMENT

This work is supported by the National Ministry of Education of China 2009-2011 National Innovation Experiment Program for University Students with Grant (091035837).

REFERENCES

[1] ETSI, “The Testing and Test Control Notation Version 3”, Methods for Testing and Specification (MTS), Part 1: TTCN-3 Core Language, 2003-02.

[2] Jens Grabowski ,”TTCN-3 - A new Test Specification Language for Black-Box Testing of Distributed Systems”, Proceedings of the 17th International Conference, 2000.

[3] ISOIIEC 9646, Information Technology Open Systems Interconnection-Conformance testing methodology and framework-Part 1, 2004

[4] A Novel Test Architecture for Wireless Communication System, Zhang Sihai, Han Lingjuan,Yu Linhong, Zhou Wuyang, 2011

International Conference on Information and Industrial Electronics, pp367-370, January 14-15, 2011, Chengdu, Sichuan, China.

[5] EI-Fakih, K., N. Yevtushenko and G. Von Bochmann, ”FSM-based incremental conformance testing methods,” IEEE Transactions on Software Engineering, vol. 30, no. 7, pp. 425-436, 2004.

[6] G. Luo, G. v. Bochmann and A. Petrenko, ”Test selection based on communicating nondeterministic finite-state machines using a generalized Wp-method,” IEEE Trans. on Software Eng., Vol. 20, No. 2, pp.149-161, 1994.

[7] Jiang Fan, Ji Xiangdong, Zeng Fanping , ”Design and Implementation of TTCN-3 Test System”, Computer Engineering, Vol.31 No.11, pp.80-81,June 2005.

[8] J R Levine, T Mason, D Brown. Lex & Yacc [M]. Third edition. Sebastopol, USA: O’ Reilly & Associates Inc, 1955.