TTCN (Testing and Test Control Notation)
A standard test specification and implementation language

Kunal Shanishchara
(kunalhs@umd.edu)

Contents

- History
- Elements and capabilities of TTCN
- Brief overview of the TTCN language syntax.
- TTCN notations
- Applications of TTCN
- Brief overview of SDL
- Automated Test case generation using SDL and TTCN
- Advantages of TTCN
- Disadvantages of TTCN
- Conclusions

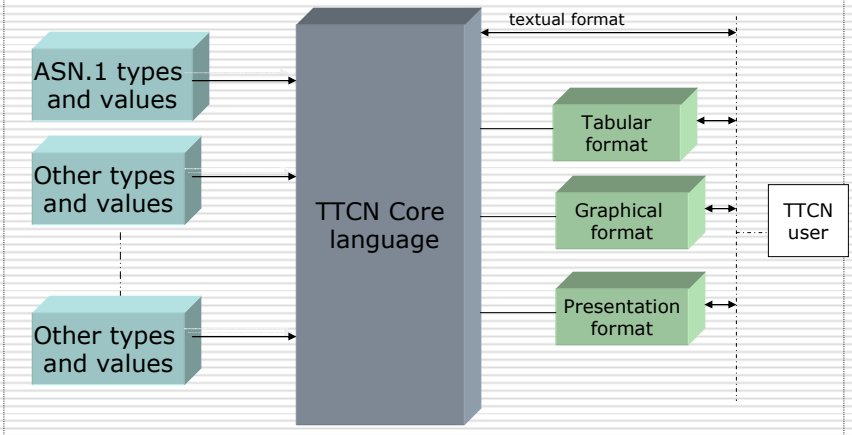
History

- TTCN now referred to as the Testing and Test Control Notation(ver 3) was previously referred to as Tree and Tabular Combined Notation(ver 1 & 2).
- TTCN was developed with the purpose of conformance testing. i.e. to verify that a given system conforms to the standards/specifications.
- TTCN versions 1 & 2 were developed by ISO (International Standards Organization)
- TTCN version 3, the current version, was developed by ETSI (European Telecommunications Standard Institute) and was standardized in 2002. The purpose was to decouple the notation from conformance testing and have a wider application.

TTCN

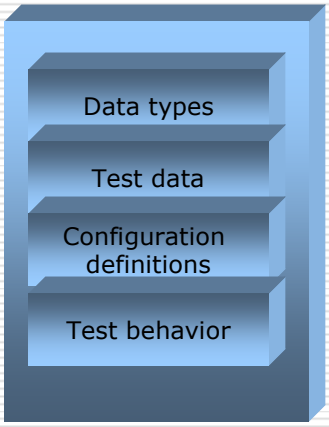
- TTCN allows the specification of test cases in various formats.
- The core language is the text format.
- The core language format can be used to obtain the other two notations. Tabular and Graphical.
- TTCN is compatible with other languages such as ASN.1 and SDL
- TTCN test cases are used for black box testing.

TTCN



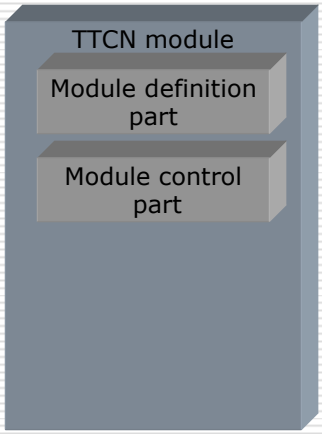
Elements of TTCN

- Built in and user defined data types. e.g. of built in types – integer, float, string, bitstring, verdict, port etc.
- Actual test data sent or received during testing.
- Test configuration definitions.
- Description of dynamic test behavior.



Modules and test cases

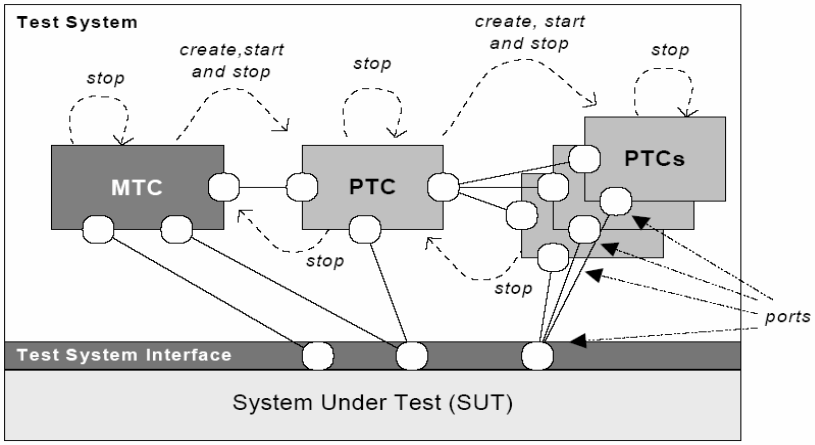
- Modules are the principal building blocks of TTCN.
- Each module has a module definition part and a module control part and is self contained.
- Module definition part defines the top level definitions of the module and the test cases in the module.
- Module control part invokes the test cases defined in the module definition part.



Test system

- Test case is executed by the test system.
- TTCN allows definition of dynamic and concurrent test systems.
- Test system consists of various components which are interconnected by well defined ports.
- Test system interface defines the interface to the System Under Test by means of well defined ports.

Test system



Evaluation of verdicts

- TTCN provides a built in type 'verdict' to define the result of the execution of the test case.
- The predefined values for verdict are *pass*, *fail*, *inconc* (inconclusive), *error*.
- TTCN evaluates the verdict of the test case based on the verdicts of each of the component. Each component maintains its local verdict.
- The local verdicts are combined to decide the verdict of the test case.

TTCN language syntax

- The TTCN language syntax is very much similar to C++/Java.
- Provides all normal programming language constructs like functions, if-else statements, for statement, while statement, do-while statement, goto statement.
- TTCN has constructs specifically for test case specification and evaluating the result of the execution of test cases on the SUT.
  - alt – allows alternates for the execution of test case.
  - template – allows defining of template to verify the contents of the data of the messages received.
  - ports – to specify the communication points between components as well as the SUT.
  - send and receive – send and receive operations on ports to send and receive messages on the port.
  - setverdict – setverdict sets the verdict for the scope in which it is present.

The Core language (text) notation

```
function PO49901(integer FL) runs on MyMTC
{
    L0.send(A_RL3(FL,CREF1,16));
    TAC.start;
    alt {
        [] L0.receive(A_RC1((FL+1) mod 2)) {
                TAC.cancel;
                 verdict.set(pass)
           }
        [] TAC.timeout {
                verdict.set(inconc)
           }
        [] any.receive {
                 verdict.set(fail)
           }
    }
    END_PTC1()   // postamble as function call
}
```

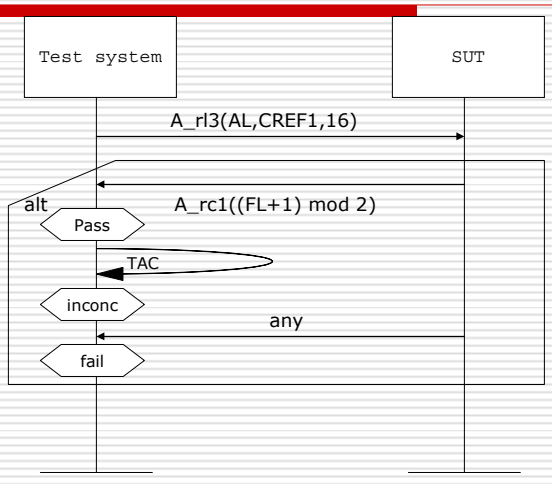
Tabular notation

Test Case Definition			
Name : MyTestcase			
Group :			
Purpose : Example Testcase			
System I/f :			
MTC Type : MyComponentType			
Comments :			
Name	**Type**	**Initial Value**	**Comments**
MyVar	INTEGER	0	
Behaviour Definition			**Comments**

`alt`
`{ [ ] MyPort.receive(Msg);`
`[ ]`
`:`
`}`			

DetailedComments:			

Graphical notation



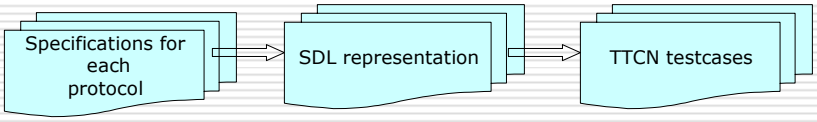
Testing applications of TTCN

- Conformance
- Performance
- Functional
- Unit
- Stress
- Load
- Interoperability
- Product
- System
- Sub-system
- Scalability
- Regression

Test case generation from SDL to TTCN

- SDL (Specification and Description language) was standardized by ITU.
- SDL is a general purpose description language for communicating systems.
- The processes in communicating systems represent Extended State Machine behavior.
- An Extended state machine consists of a number of states and a number of transitions connecting the states.
- Communication is represented by signals

Test case generation from SDL to TTCN



Advantages of TTCN

- Formalized notation for description of test cases.
- Standardized test technology
- Implementation independent
- Distributed test execution environment
- Allows the use of Off-the-shelf test suites
- Test case reusability
- Permits dynamic configuration of test suites.
- Ease of use due to its syntax being similar to standard programming languages
- Supports ASN.1 and SDL representations
- Applicable to testing of almost all protocols.
- Reduces time to market

Disadvantages of TTCN

- Testers need to learn TTCN language to prepare test cases.
- The compilers which translate from TTCN to Java/C++ are hard to develop.
- Suites blackbox testing.

Conclusions

- TTCN finds applicability for testing wide range of communication systems.
- TTCN is a prime example of formal test case description.

References

- TTCN Specifications.
- An Introduction into the Testing and Test Control Notation (TTCN-3)

  Jens Grabowskia, Dieter Hogrefeb, Gy□orgy Rethyc, Ina Schieferdeckerd, Anthony Wilese and Colin Willcockf
- Testing real-Time Systems,*R. Castanet,P. Laurençot*
- Test Generation with Autolink and TestComposer,*M. Schmitt, M. Ebner, J. Grabowski*