TITAN, TTCN-3 test execution environment

JÁNOS ZOLTÁN SZABÓ, TIBOR CSÖNDES

Test Competence Center, Ericsson Hungary Ltd.
{janos.zoltan.szabo, tibor.csondes}@ericsson.com

Keywords: testing, TTCN-3, test system implementation

This paper presents the TTCN-3 test execution environment of Ericsson called TITAN. We show the internal details and the operation of the toolset. Unique TITAN features and differences from other commercial TTCN-3 tools are discussed. As a result of our development TTCN-3 and TITAN became a widely used test solution within Ericsson and we contributed to the TTCN-3 standardization work within ETSI.
(In: 2006/9, pp.29–33.)

1. Introduction

During the 80's and 90's the 2nd version of TTCN (Tree and Tabular Combined Notation) was used for testing telecommunication systems based on the OSI Basic Reference Model [1]. The TTCN-2 language had two properties that prevented it from wide acceptance: its difficult tabular format and the limited application area. Therefore, at the end of 90's, ETSI started the redesign of the language and standardization of TTCN version 3.

In the planning phase it was considered that besides the recent conformance testing methods the new language should be applicable for new types of tests, like: interoperability testing, performance testing, robustness testing and system testing.

ETSI gave up the close relationship to OSI BRM and made it possible to test internet based protocols and Application Programming Interfaces (API).

1.1. TTCN-3 Language

The first version of TTCN-3 (Testing and Test Control Notation) was published as a set of ETSI standards in 2000. Since then several minor enhancements and corrections have been made on the language. The latest, third edition of the standard documents were issued in 2005 [2]. The creators of TTCN-3 tried to get rid of the clumsy structures and thus prevent the bad reputation of the TTCN language, which was widespread among telecom experts since TTCN-2. This was worth the efforts since the result has been a language that is easy to understand and helps testing considerably. The textual representation and basic control statements of TTCN-3 is quite similar to programming languages like C/C++ so many potential users can understand the basic language constructions without deeper TTCN-3 knowledge.

A detailed TTCN-3 introduction can be found in [3].

1.2. History of TITAN

The development of TITAN was started as an M.Sc. thesis work in the beginning of 2000. The main goal of the project was to create an efficient, protocol and application independent test environment, which is also capable of running performance tests. TTCN-3, which was still under development in ETSI at that time, was a perfect choice as the input language of the new test tool.

The first prototype of the system was ready in less than one year. This version supported only a subset of TTCN-3 language features, but its architecture was very similar to the current state [4]. Since the first release TITAN has been under continuous development: it follows the changes of the language specification and has more and more convenience functions. TITAN supports almost all constructs of TTCN-3 and numerous non-standard language extensions.

The main milestones of the development were the following:

2000: first prototype
2001: parallel and distributed test execution
2002–2003: support of ASN.1 [5]
and built-in encoders/decoders
2004: graphical user interface
2005: full TTCN-3 and ASN.1 semantic analysis

2. Structure of TITAN

TITAN uses C++ as intermediate language for realization of the TTCN-3 test system. The block diagram of TITAN test execution environment can be seen in *Figure 1* (next page).

TITAN consists of the following parts:

2.1. TTCN-3 and ASN.1 Compiler

The TTCN-3 and ASN.1 compiler is the largest and most complex part of TITAN. Its tasks include the parsing and analysis of the test suite and reporting the syntax and semantic errors that are present in the input. If all modules of the test suite are found to be correct the compiler generates C++ program modules that will be parts of the Executable Test Suite (ETS).

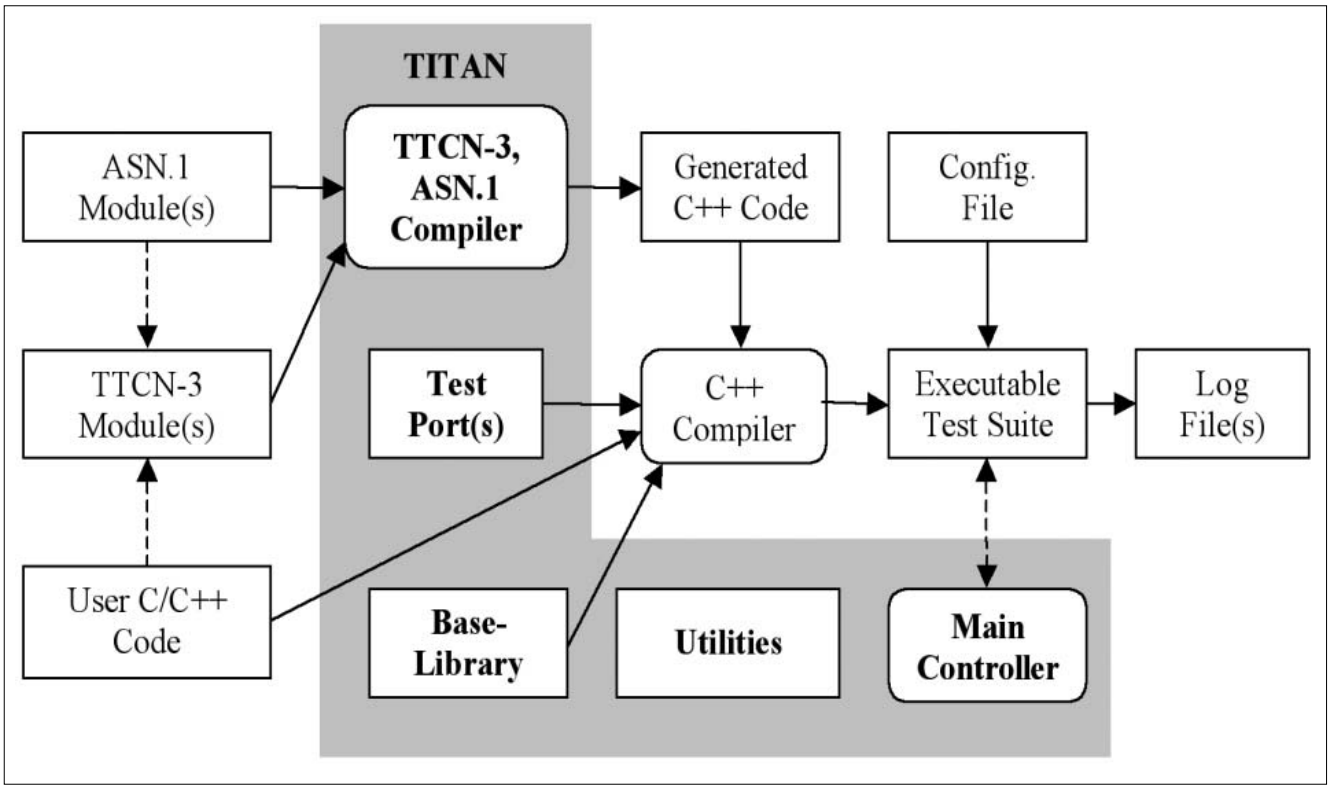


Figure 1. TITAN block diagram

2.2. Base Library

The base library contains those common and static parts of the ETS that are independent of the actual test suite. It consists of manually written C++ code, which is compiled into binary form in advance. The base library comprises the C++ classes representing the basic TTCN-3 data types and the functions implementing the built-in operations of the language (such as timer, port, test component and verdict handling). Other auxiliary functions that are necessary to run the ETS (e.g. the logging and configuration file processing routines) are also parts of the base library.

2.3. Test Ports

TTCN-3 models all external communication between the test system and the outside world using abstract messages sent and received through communication ports. The purpose of test ports is to bridge the gap between the Executable Test Suite and the System Under Test (SUT). In fact, the test ports are the C++ realizations of TTCN-3 communication ports in the ETS.

TITAN provides a well-defined programming interface, the so-called Test Port API for handling incoming and outgoing messages. Typical test port implementations communicate with the SUT using IP-based protocols accessed through the socket API of the operating system. The responsibility of test ports includes the encoding and decoding of structured, abstract TTCN-3 and ASN.1 messages, that is, to transform them to and from the transfer coding (i.e. binary octet streams used on the underlying communication channels).

2.4. Utility Programs

TITAN contains several small utility programs that make test suite development, compilation, test execution and result analysis easier. There are scripts for automated test suite launching, tools for test log post-processing (utilities for merging, formatting and filtering log files) and so on.

2.5. Main Controller

When a test suite requires more than one TTCN-3 test components to be run in parallel, their operation must be coordinated. This task is done by a dedicated application called Main Controller (MC), which belongs to TITAN and is independent of the actual test suite. The MC has direct connections for supervising all other components of the TTCN-3 test system. It contributes to the creation and termination of test components as well as the establishment and break-down of port connections, among others. When running performance tests the test system can be distributed over several networked computers to generate high traffic load against the SUT. In this case the load balancing between the participating tester computers is also done by the MC. To avoid bottlenecks in the test system the MC only has central coordinating tasks; it does not take part in elementary test operations. Further details about TITAN's distributed test architecture can be read in [6].

Besides, the MC has the user interface for interactive test execution, which can be either command line or graphical. The current state of the test system can be continuously monitored and the user has the possibility for intervention in test run by stopping the current test case or starting a new one.

3. The Operation of TITAN

3.1. Syntax and Semantic Analysis

The lexical analysis and syntax checking of input modules are done by parsers generated with conventional tools GNU *flex* and *bison*. During parsing the compiler builds up special memory structures called abstract syntax tree (AST). The integrated TTCN-3 and ASN.1 compiler has the advantage that the different front-ends for the two languages transform the definitions into the unified structures of one common AST. This allows the direct usage of data types and values of protocols with ASN.1 descriptions from TTCN-3 test suites.

The purpose of semantic analysis is to detect errors like invalid references, forbidden operations or type clashes and perform some transformations on the AST for code generation (e.g. calculating and reducing constant arithmetic expressions). TITAN compiler has a one-pass semantic analyzer, which means the algorithm walks through the AST only once. However, the order in which the AST nodes are visited is influenced by the references between the definitions. The most challenging task of semantic analyzer development was to properly identify and handle the ambiguous language constructs of TTCN-3 and ASN.1 that cannot be classified during the syntax check (e.g. the *start* operation of TTCN-3 can refer to a port, timer or test component with the same syntax).

When the compiler detects a syntax or semantic error it does not stop after printing the first error message, but keeps on analyzing to discover more errors in the test suite. Special error recovery techniques are employed in order to prohibit error messages whenever references point to existing, but erroneous definitions. Otherwise one simple fault could launch an avalanche of error messages.

3.2. Code Generation

C++ code generation is based on the AST, on which the semantic checker has made some simplifications. The generated code of TITAN uses static typing, which means that every TTCN-3 and ASN.1 data type is mapped to distinct C++ classes. The main benefits of the statically typed run-time environment are the high execution speed and modest memory usage, since the C++ compiler can arrange the optimal memory structures for TTCN-3 data values and their fields. A further advantage of static typing is that the type correctness of TTCN-3 operations is checked by the C++ compiler as well. This is an extra verification step on the entire ETS without executing it.

The latter property of the generated code was exploited in earlier versions of TITAN where the compiler lacked semantic analyzer. Instead of building AST the equivalent pieces of C++ code were created immediately while parsing the TTCN-3 input. The internal interfaces of the run-time environment were designed in such way that TTCN-3 semantic errors were mapped to similar kinds of faults in the output, which were caught and reported by the C++ compiler.

The significant drawback of static typing is the large size of the generated C++ class hierarchy. The output generated from the data types of complex protocols has long compilation time. The large code of data types is compensated by mapping other TTCN-3 definitions, such as values, data templates and behavior descriptions (functions, test cases, etc.) to very compact C++ code.

In case of dynamic typing, which is used by most of commercial TTCN-3 tools on the market, all TTCN-3 data values are constructed from the same generic structure that can carry the values of any type. Because of this every run-time operation must examine whether the given arguments have the correct types. This applies to, for example, the built-in elementary operation of integer addition, which has to first ensure that the generic structures of arguments contain numbers rather than strings or something else. The extra tasks of dynamic typing can result in 10 or 100 times slower execution speed compared to TITAN.

3.3. Executable Test Suite Derivation

The entire compilation process including the translation of the test suite to C++, the compilation of generated code and test ports to binary form and the final linking of the ETS is managed by UNIX *make*. A special file called *Makefile*, which is interpreted by make, describes the different steps of compilation and the dependencies between them. TITAN can create a Makefile based on the list of TTCN-3 and ASN.1 modules and test ports needed by the test suite.

Typical TTCN-3 test suites contain a lot of modules. It can be observed that the majority of modules change very seldom during the test development process. For example, the modules that define the message types of protocols will never change in normal cases. The changes between two compilations and test runs are usually limited to a few modules and a few lines of TTCN-3 code within them (mostly the behavior statements that describe the test cases). Since a full re-compilation can take several minutes or hours in case of complex test suites this should be avoided after minor changes.

TITAN compiler together with the *make* utility supports incremental compilation. This means that the results of previous compilation are reused as much as possible and only the updated modules are translated to C++ and subsequently to binary code. Identification of the modules that require re-compilation is not an easy task in some cases.

If the definitions of a module are imported by another than any change of the imported definitions will affect the importing definitions, which can be imported into a third module, and so on. So a single change in one module can cause several modules to be recompiled in order to maintain the consistency of the ETS.

Despite the above difficulties practical examples show that the incremental build system of TITAN can significantly decrease the compilation times and improve the efficiency of TTCN-3 test suite development.

3.4. Encoding and Decoding

During test run the abstract messages to be sent to or received from SUT must be encoded or decoded. To make this task easier TITAN contains several built-in codecs, which are accessible through a special C++ API. Using the built-in codec the encoding or decoding of a message can be done in a few lines of C++ code regardless the complexity of the data type. The encoding and decoding functionality can be placed in test ports or external functions, which are written entirely in C++, but can be invoked from TTCN-3.

ASN.1 data types can be encoded according to the standardized Basic Encoding Rules (BER). TITAN supports two different encoding schemes for TTCN-3 data types: a table-based bit-oriented (RAW) and a text-based (TEXT) one. The exact coding rules of TTCN-3 types, which can be quite complex in case of some protocols, are specified using the attributes of the respective type definitions.

3.5. Graphical User Interface

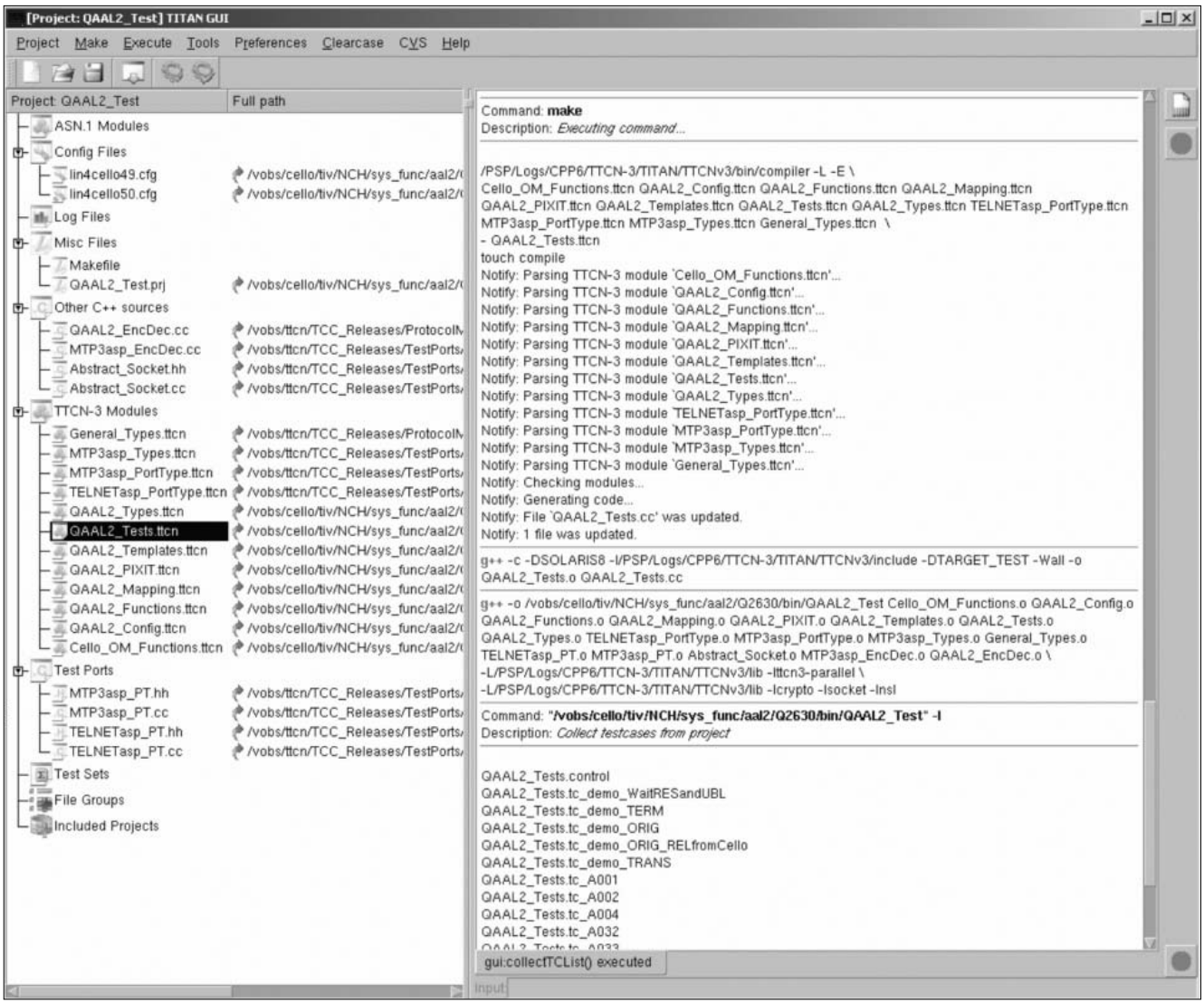
TITAN contains a Graphical User Interface (GUI) built together with the Main Controller, which provides a user-friendly environment both for test development and test execution.

Figure 2 shows a screenshot of the main GUI window. The left part shows the overview of the current project (lists of TTCN-3 modules, test ports, configuration files and other source files) while the right window lets the user follow the compilation process. The example presents the results of an incremental compilation.

4. TTCN-3 Interfaces

A TTCN-3 Executable Test Suite can communicate with the outside world via interfaces. ETSI has defined 6 interfaces in 2 standards (TTCN-3 Runtime Interface, TRI

Figure 2. Graphical User Interface



[7] and TTCN-3 Control Interface, TCI [8]). The two TRI interfaces (SUT Adapter and Platform Adapter) describe the connections between the ETS and the system under test as well as the operating system (e.g. timer handling). TCI contains four interfaces: Test Management (test case execution, test suite parameterization), Component Handling, Coding/Decoding and Logging.

These standards describe the interfaces with programming language independent notation and give canonical C, Java and XML mappings for the data types and procedures. The aim of the interface standardization was to allow users to switch from one TTCN-3 runtime environment to a tool of another vendor without changing the application specific software modules.

At the moment TITAN does not support any of the standard interfaces above. There are several reasons. On one hand, in 2002-2003, when the interface standards were published by ETSI, TITAN was already a mature and complete TTCN-3 test system with its full featured proprietary interfaces. On the other hand, during the development we preferred other technical aspects than ETSI.

Our goal was to provide an effective test system in such a way that the users need to develop the smallest and simplest external program modules possible. Therefore TITAN provides only one programming interface towards SUT, which is the test port interface. TITAN supports the functionalities of other parts of TRI and TCI as efficient built-in modules without public programming interfaces.

TITAN test port interface has more advantages than the SUT Adapter interface of TRI. A test port instance always handles single TTCN-3 communication port and consequently one protocol. Therefore the distribution of messages of different protocols is solved by the interface itself. In contrast, with TRI all the messages toward SUT are processed by the same function of the single adapter.

The separation of different protocols has to be implemented in the adapter by the user. Whenever a new protocol or system interface is introduced in the test system, a new test port, like a building block, can be simply added with TITAN. However, with TRI it will be necessary to redesign the entire adapter. In addition, test ports are more suitable for distributed performance testing since the test ports connect the TTCN-3 parallel test components directly to the SUT eliminating traffic bottlenecks.

Later implementation of the standardized interfaces in TITAN could cause difficulties, because they assume dynamic typing and multi threaded operation. Generally the interfaces of TRI and TCI were not designed for efficient operation so the practical advantages are questionable in TITAN. We think it would not be possible to achieve better performance, simpler structure or more comfortable usage compared to the existing built in functionalities of TITAN.

5. Summary

Thanks to the development and usage of TITAN, Ericsson has joined the TTCN-3 standardization work within ETSI. Ericsson has submitted 196 out of the total 340 Change Requests (CRs), which represents our activity in the field of TTCN-3 standardization. Most of our CRs resolve ambiguous structures and conflicts in the core language; but several extensions have also been proposed, which made the language simpler and more usable.

TITAN is the official TTCN-3 Test Tool within Ericsson since 2003. Since then a department with almost 40 people, the Test Competence Center is working on the development, deployment and support of TITAN and TITAN-based test solutions. Our product portfolio includes Test Ports, TTCN-3 Protocol Modules and complete Test Suites. Test Competence Center has customers from Ericsson units all over the world. Although TITAN has not been sold outside Ericsson the number of its users has been continuously growing during the last years. Almost 50 Test Ports and 100 Protocol modules have been developed for TITAN, which gives a great opportunity for testing a wide spectrum of telecommunication systems.

References

[1] ITU-T, X.200, Information Technology – Open Systems Interconnection – Basic Ref. Model: The Basic Model, 1994.

[2] ETSI ES 201 873-1, v3.1.1 (06/2005) The Testing and Test Control Notation, version 3. Part1: Core Language

[3] Jens Grabowski, Dieter Hogrefe, György Réthy, Ina Schieferdecker, Anthony Wiles, Colin Willcock, “An introduction to the testing and test control notation (TTCN-3)”, Computer Networks, Vol. 42, Issue 3, pp.375–403., Elsevier North-Holland, Inc. 2003.

[4] János Zoltán Szabó, “Experiences of TTCN-3 Test Executor Development”, Testing of Communicating Systems XIV., Application to Internet Technologies and Services, Edited by I. Schieferdecker, H. König and A. Wolisz, Kluwer Academic Publishers, 2002.

[5] ITU-T X.680 (07-2002) Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation.

[6] János Zoltán Szabó, “Performance Testing Architecture for Communication Protocols”, Periodica Polytechnica, Electrical Engineering, Budapest University of Technology and Economics, 2003. 47/1-2.

[7] ETSI ES 201 873-5, v3.1.1 (06/2005) The Testing and Test Control Notation, version 3. Part5: TTCN-3 Runtime Interface (TRI)

[8] ETSI ES 201 873-6, v3.1.1 (06/2005) The Testing and Test Control Notation, version 3. Part6: TTCN-3 Control Interface (TCI)