Test Automation with TTCN-3 - State of the Art and a Future Perspective

Ina Schieferdecker

TU Berlin/Fraunhofer FOKUS, Kaiserin-Augusta-Allee 31, 10589 Berlin, Germany
ina.schieferdecker@fokus.fraunhofer.de

Abstract. Test automation encompasses all activities to automate various steps in the overall testing process including automation of test management, test generation, or test execution. The standardized Testing and Test Control Notation (TTCN-3) addresses selected challenges by defining a test specification language and a test system architecture that enables the implementation and execution of TTCN-3 test suites. Over the years, the standard has continuously being maintained and evolved. For example, concepts for static test configurations or for advanced parameterization and typing have been defined. The paper reviews the history and current status of TTCN-3 and concludes by giving an overview on recent extensions of TTCN-3 and future plans.

Keywords: test automation, test specification, test framework, test execution, test management, TTCN-3.

1 Introduction

With the Testing and Test Control Notation (TTCN-3) the testing community obtained a generic technology, standardized by ETSI and ITU, which allows the development and design of systematic tests and their reuse independently from technologies, product lines and manufacturers. TTCN-3 is a living, widely established and continuously maintained testing technology, available in a version 4.2.1 from July 2010.

However, the access to this testing technology is not always easy; here, the prejudices of the testing automation are pairing with the acceptance threshold of a new technology. Thereby, research done by Motorola shows that the application of this technology is not only increasing the quality of tests, but also the efficiency of its development and the grade of its reutilization [1]. Moreover, studies in China [2] show that this technology is immediately applicable by the testing personnel and can be actively used by people being familiar with the needs, methods and solutions of testing already after a 1-2 day instruction.

TTCN-3 is a powerful testing technology – powerful enough to solve a big variety of testing problems – and with it at first glance also rather complex. However, please allow me to relate a mind game from a discussion with Mark Harman from Brunel University at a Dagstuhl seminar in 2009: Please imagine that the total loss of your system or the corresponding testing system is imminent. You will only be able to reduce the damage if you will decide or to save the system or the testing system. Whereas some years ago in its majority the system would have been saved (because this is the goal), nowadays the voices demanding the salvation of the testing system are increasing, since it is not only preserving the knowledge on the system to be developed, but also the knowledge about its safeguarding. The testing system allows us to reconstruct the system at any time and simultaneously to save the system in its evolution.

This more-of-knowledge is corresponding with the efforts of developing of every system: there is an increasing number of products where the same – if not more – efforts are being invested into the testing system as into the system itself. This shifting of efforts towards the testing system has been observed e.g. with Microsoft during the development of Windows XP, which was beneficial for all of us in terms of the stability of the operating system. At the same time, this additional investment is also reflected in technologies used for the development oft the testing systems. Who is not familiar with the huddle of testing tools, where every tool is already complex, but only the combination allows an adequate analysis and validation of a system. The main purpose of TTCN-3 is to thin out this huddle of tools – if not to abolish it. Thus, it is not amazing that this testing technology has been designed as powerful as it needs to be to consolidate the testing concepts, in the result achieving a simplification. And there is no end to be seen: continuously we are being asked to integrate additional concepts into TTCN-3. But we will recur to this later. In the following, the paper gives outline of the history, state of the art and possible future of TTCN-3.

2 TTCN-3 Yesterday: A Short Retrospection on the TTCN-3 History

TTCN-3 was worked out by ETSI as a new edition of the TTCN (Tree and Tabular Combined Notation), mainly developed for conformance, interoperability and performance tests of communication-based systems including protocols, services, interfaces, etc. TTCN-3 is a modern language for test specification and implementation, by means of which tests can be developed and specified textually or graphically, implemented and executed automatically.

TTCN-3 has first been presented in public in September 2000, and with the version v2.2.1 gained the necessary stability and maturity for tool development and its implementation in industry. The following versions have been amplified by new concepts, but designed with a backwards compatibility with former versions. The new versions for example included importable types defined in IDL or XML, dynamic templates and a logging-interface. In version v3.3.1, the addressing within the testing system has been improved. In version v3.3.2, a template-restriction was made possible, and in v3.4.1 user-defined attributes for testing judgment (verdicts) have been introduced. Recently for v4.1.1, a package concept has been created in order to allow the definition of specific concepts (e.g. for real-time) which are needed in dedicated domains or for selected applications of TTCN-3, but which would overload the language if being defined for the core language. These packages are made optional and can be chosen depending on specific requirements of a given test solution. All the changes to TTCN-3 have been made transparent for everybody and can be tracked since 2005 by the change request (CR) procedure for TTCN-3.

The parts that currently constitute TTCN-3 are shown in Fig. 1. A user typically works with TTCN-3 via the textual format (the core language), the graphical format (based on Message Sequence Charts) or the tabular format (resembling the old style TTCN tables), although the textual format is used in the majority of cases. The operational semantics and the execution interfaces are most often relevant for tool vendors, but help users also to understand TTCN-3 and to apply it precisely. Language mappings for ASN.1, IDL and XML schemata help to use TTCN-3 for systems under tests using these languages for their data, interfaces, services, or protocols. Specific language concepts (most often extending not only the core language and its operational semantics, but also the execution interfaces) are defined in extension packages for optional use in specific usage contexts or for specific test applications or target systems under tests.

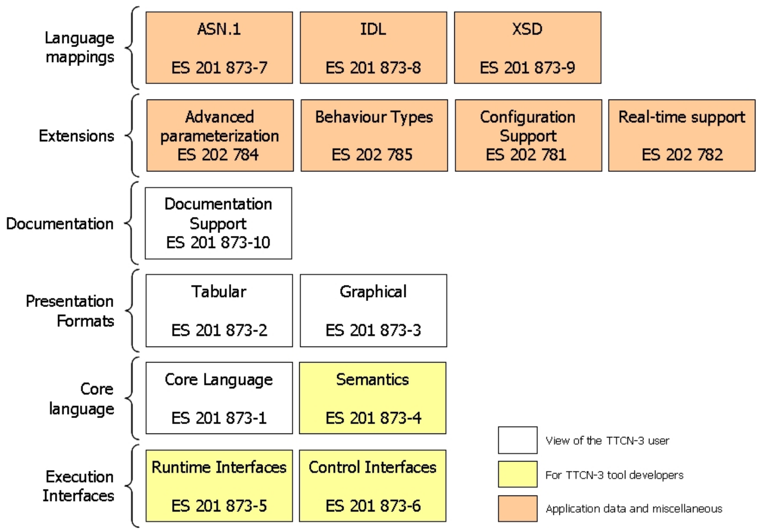

Language mappings
ASN.1 ES 201 873-7
IDL ES 201 873-8
XSD ES 201 873-9
Extensions
Advanced parameterization ES 202 784
Behaviour Types ES 202 785
Configuration Support ES 202 781
Real-time support ES 202 782
Documentation
Documentation Support ES 201 873-10
Presentation Formats
Tabular ES 201 873-2
Graphical ES 201 873-3
Core language
Core Language ES 201 873-1
Semantics ES 201 873-4
Execution Interfaces
Runtime Interfaces ES 201 873-5
Control Interfaces ES 201 873-6
View of the TTCN-3 user
For TTCN-3 tool developers
Application data and miscellaneous

Fig. 1. The TTCN-3 Test Specification Language

In contrast to numerous testing and modeling languages, TTCN-3 does not only comprise a language for the specification of tests, but also a test system architecture and runtime and execution interfaces for TTCN-3 based testing systems (see Fig. 2). In the test system architecture, within the test engine (TE) the tests specified in TTCN-3 are executed – the TE is the runtime environment for the compiled TTCN-3 code of a set of TTCN-3 modules defining a test suite. The TE is wrapped by a number of adaptors that mediate between the TE and the system under test (SUT), the test system, the test management, the logging, the encoding and decoding, and last but not least the (potentially distributed) handling of test components. The functionality of these adaptor components are defined via a number of operations specified in the Interface Definition Language (IDL) and mapped to programming languages for the test devices, i.e. to C. C++, C#, and Java.

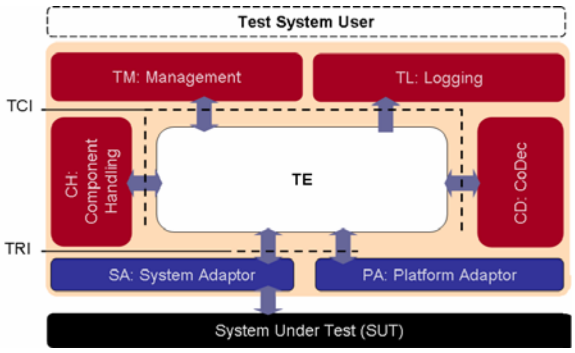
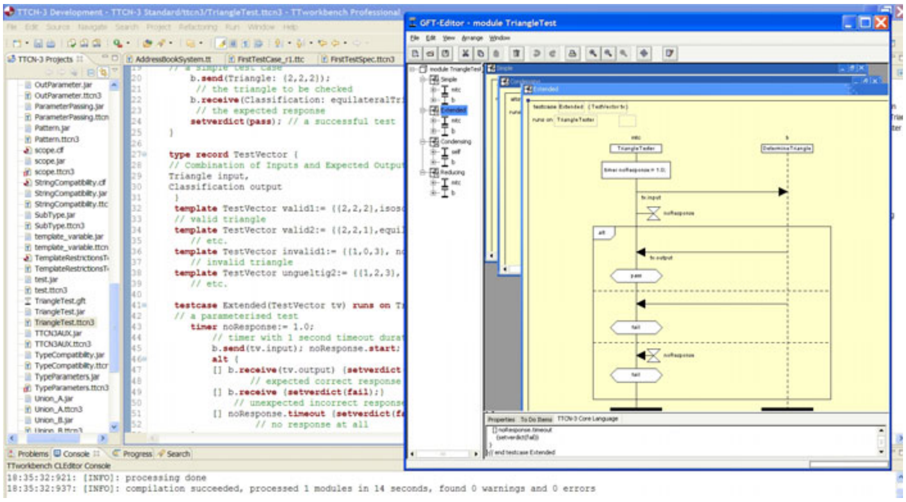


Fig. 2. The TTCN-3 Test Implementation Architecture

In the following, a small example for TTCN-3 is shown (see also Fig. 3) in order to give an insight into this technology: in a well known testing example of G.J. Meyers [3] for the classification of triangles (a triangle is defined by its side lengths und has to be classified), the question of typing is always posed. In TTCN-3, you are explicitly defining it:

```
type integer Triangle[3] (0..infinity);
// whole numbers to characterize a triangle
type enumerated Classification {
// properties of a triangle
   syntacticallyIncorrect,
   noTriangle,
   scaleneTriangle,
   isoscelesTriangle,
   equilateralTriangle
}
type port DetermineTriangle message
 { out Triangle; in Classification }
// Interface to SUT
type component TriangleTester
 { port DetermineTriangle b }
// Test Component
testcase Simple() runs on TriangleTester {
// a simple test case
    b.send(Triangle: {2,2,2});
    // the triangle to be checked
    b.receive(Classification: equilateralTriangle);
    // the expected response
    setverdict(pass); // a successful test
}
```



```
        b.send(Triangle: {2,2,2});
         // the triangle to be checked
        b.receive(Classification: equilateralTr
         // the expected response
        setverdict(pass); // a successful test
    }

    type record TestVector {
    // Combination of Inputs and Expected Output
    Triangle input,
    Classification output
     }
     template TestVector valid1:= {{2,2,2},isosc
     // valid triangle
     template TestVector valid2:= {{2,2,1},equil
        // etc.
     template TestVector invalid1:= {{1,0,3}, no
        // invalid triangle
     template TestVector ungueltig2:= {{1,2,3},
        // etc.

     testcase Extended(TestVector tv) runs on Tr
     // a parameterised test
        timer noResponse:= 1.0;
            // timer with 1 second timeout durat
            b.send(tv.input); noResponse.start;
            alt {
            [] b.receive(tv.output) {setverdict
                   // expected correct response
            [] b.receive {setverdict(fail);}
                // unexpected incorrect response
            [] noResponse.timeout {setverdict(fa
                    // no response at all
```

```
18:35:32:921: [INFO]: processing done
18:35:32:937: [INFO]: compilation succeeded, processed 1 modules in 14 seconds, found 0 warnings and 0 errors
```

Fig. 3. A TTCN-3 Integrated Development Environment based on Eclipse

This test specification is wiring the testing data firmly with the testing activity, which is not very easy to maintain. Another method would be to separate the test data from the test behavior:

```
type record TestVector {
// combination of inputs and expected outputs
   Triangle input,
   Classification output
 }
 template TestVector valid1:= {{2,2,2}, isoscelesTriangle};
 // valid triangle
 template TestVector valid2:= {{2,2,1}, equilateralTriangle};
// etc.
 template TestVector invalid1:= {{1,0,3}, noTriangle };
// invalid triangle
 template TestVector invalid2:= {{1,2,3}, noTriangle };
// etc.

 testcase Extended(TestVector tv) runs on TriangleTester {
 // a parameterised test
   timer noResponse:= 1.0;
   // timer with 1 second timeout duration
   b.send(tv.input); noResponse.start;
   alt {
      []            b.receive(tv.output)           {setverdict(pass);}
         // expected correct response
      [] b.receive {setverdict(fail);}
         // unexpected incorrect response
      [] noResponse.timeout {setverdict(fail);}
         // no response at all
    }
 }
```

In the extended test case given above, we are additionally taking into account that the SUT could also be failing or not answer at all. This could be followed up by you by e.g. starting queries with two parallel testing components to the SUT or by fixing two testing sequences, which first are checking all the valid, and afterwards, in dependence of positive testing results, all the non-valid triangles. Additional variations are possible, but cannot be demonstrated in this paper.

3 TTCN-3 Today: 10 Good Reasons to Use TTCN-3

This is leading us to a description of today's situation with TTCN-3, summarized in several "good reasons" for TTCN-3.

1. One is able to learn and apply TTCN-3 directly: TTCN-3 is a testing specific language, developed from testing persons for testing persons. A couple of key concepts, which one needs typically in the majority of the test cases like test cases, test verdicts or test data (called templates), allow one to design and specify tests directly and easily. But this is not the only advantage of TTCN-3: a great number of additional concepts allow one to develop also test for heterogeneous interfaces, distributed tests, dynamic tests etc. If this is not enough, one has the possibility to integrate further concepts into the language or to directly contribute to the TTCN-3 standards. I would like to draw your attention to the large number of books, magazines or articles dealing with this technology, its application and the respective tools (see also the TTCN-3 bibliography given in Fig. 4).
2. One is able to verify ones testing knowledge and get a certificate with successful attendance. On the basis of the certified Tester Schema of the International Software Testing Qualification Boards (ISTQB), the German Testing Board (GTB) has developed a TTCN-3 training scheme (see Fig. 4). Certified training providers are offering TTCN-3 courses training in the applications of TTCN-3. An inclusion of the TTCN-3 certificate into the Certified Tester Expert Level with ISTQB is in preparation.
3. One is able to contribute to the development of TTCN-3 by oneself. At ETSI, the Change Request (CR) process for TTCN-3 is administered and corrections, amendments and proposals for improvements can be placed (see Fig. 4). Whenever you will see the need of further development in your practical applications, you will be able to write this as CR and track the path to the solution in a transparent procedure. CRs can lead to corrections, clarifications, extensions or amendments. They can also be appointed as copies of an already existing CR or be closed well-founded without changes on the standard.
4. One is able to use a wide range of TTCN-3 tools. As TTCN-3 is a standard, it allows the development of off-the-shelf tools. Hence, it is possible to detach the extensive care and development of proprietary in-house tools by license and maintenance contracts and to concentrate on the design and specification of test cases. The development and maintenance of tool components for the test development, application, reporting or integration will be carried out by a third party. The availability of a large number of TTCN-3 tools requires no dead-end-decisions for one

provider – the investment in TTCN-3 remains secure independently from a concrete tool. The future spread of TTCN-3 is linked to a growing number of tool providers, like e.g. IBM, Elvior or Testing Technologies. A reference to commercial, free and open source tools is given in Fig. 4.

5. At present, TTCN-3 is used in a major extent. It is applied in the areas of e.g. telecommunication, Internet technologies, for control devices in automotive, avionics, transport and medical technologies, software and Web services of business applications and in eGovernment etc. Different bodies – not only ETSI or ITU – are voting for TTCN-3: 3GPP is using it for UMTS and LTE tests for wireless interfaces and backbones. Also, the WIMAX Forum for certification of hardware, the AUTOSAR consortium for basic software the automotive software middleware and the OMA for functional tests of enablers and mobile services are providing TTCN-3 tests. Thus, one is often be able to resort to existing libraries and TTCN-3 test suites and does not have to develop them right from the start. Further applications are emerging in the context of XÖV (XML-based exchange of data in the public administration), HL7 (communication stacks in integrated medical environments [4]) or MOST (entertainment application and components in the automobile).

6. One has the possibility to become a member of the permanently growing TTCN-3 community, meeting regularly at the annual TTCN-3 User Conference (T3UC, see Fig. 4) in order to share experience on applications, case studies, experiments, tools and new developments of TTCN-3. The 7th T3UC was taking place 2010 in Beijing. At T3UC, one has the chance to meet users, experts and tool manufacturers and to obtain new information about recent developments in the TTCN-3 tool and service area. Moreover, tutorials are held on a wide number of conferences such as SOFTEC, IQNITE, EuroStar or CONQUEST.

7. And most important: with TTCN-3 one has the possibility to tackle the testing tasks in an effective and efficient way. Thereby, TTCN-3 does not require a direct and absolute change of the current testing methods and environments, but allows a step by step migration to the new testing technology. One is able to combine existing solutions with new TTCN-3 testing solutions and thus use the established testing basis until one does not need it any more, when e.g. the tested systems are phased out.

8. Besides different programming languages like C, C++, Java or C#, which are supported by TTCN-3 tools, one gets a wide range of accesses to interface-technologies of the systems to test. This includes accesses for ASN.1, IDL and XML based systems – interfaces and formats which are elements of the TTCN-3 standard series. Additionally, tool manufacturers have been developing accesses for Java, C/C++, Tcl, Python, WSDL, BPEL, etc., which – although not (yet) standardized – are at disposition.

9. One is able to adequately comment the test cases and test suites with TTCN-3 by means of documentation annotations (documentation tags) and thus enhance the legibility, reutilization and maintainability of the tests. Additionally, one has the

possibility to use metric, samples, anti-samples and refactoring for TTCN-3, analyzing and optimizing the test suites.

10. TTCN-3 can be used for different kinds of testing and phases. The strengths of TTCN-3 are lying in the specification and application of tests, using interfaces or other communication means as test-accesses to the SUT. In some cases one has to deal with the heterogeneity of the system or part of the system with regards to interfaces, the used programming languages, the combination of SW and HW components, etc. It is exceptionally useful for integration, system and acceptance testing. Moreover, TTCN-3 can be used not only with functional testing, but also for non-functional testing like e.g. for performance tests (see e.g. [6)). In case one exploits the reuse potential of TTCN-3 test cases in different testing phases and types, one will not only win by a wide use of the testing technology, but also by reducing the input for instructions and training of the employees and the care of testing tools. At the same time, existing TTCN-3 test suites or libraries can be used more often.

Home Page:	www.ttcn-3.org
Quick Reference Guide:	http://www.blukaktus.com/
Bibliography:	www.ttcn-3.de
Mailing List:	ttcn3@list.etsi.org
Change Requests:	http://www.ttcn-3.org/ChangeRequest.htm
TTCN-3 Certificate:	http://www.german-testing-board.info/de/ttcn3_certificate.shtm
TTCN-3 User Conferences:	www.ttcn-3.org → Events
Tools:	www.ttcn-3.org → Tools
Tutorials:	www.ttcn-3.org → Courses and Tutorials

Fig. 4. TTCN-3 Sources

4 The Current TTCN-3 Version: TTCN-3 v4.2.1

In 2009, version 4.1.1 was prepared and completed in version 4.2.1 summer 2010. Here, the focus is on concepts for a more efficient application of testing configurations, on a better assistance of libraries and a further dynamization of testing specifications, e.g. by type-parameterization. Furthermore, concepts for the testing of real-time systems and for performance testing have been considered. In fact, such tests can already be formulated and applied by former TTCN-3 versions, but the objectives were to achieve a more efficient application of TTCN-3 by a more direct support of decided concepts, like those used with automotive engineering, automation, aeronautical and medical engineering.

Let us first however shortly review version v3.1.1 as several major extensions have been introduced. The following is not an exhaustive list of extensions but rather name the major ones only. The first group of extensions dealt with test components: test components can be **alive test components** meaning that several test behaviors can be executed on such a test component without termination of test component after termination of the test behavior. Hence, the test component state, i.e. the values of its port queues, local timers and variables, can be passed and reused between the tests behaviors started on an alive component. Furthermore, **inheritance for test component**

types has been introduced, so that test component types can extend other test component types – for example, to share a common set of ports.

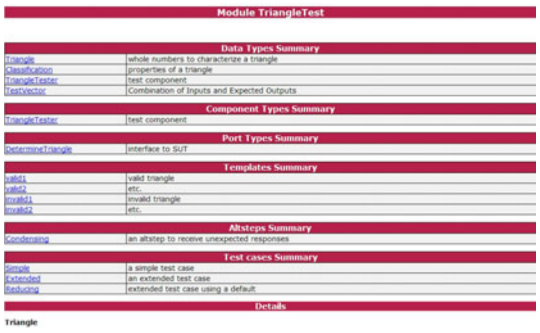
The second group of extensions addressed the communication means. So far, unicast communication could be used only. In addition, **multicast communication**, i.e. communicating to or from a set of test components, and **broadcast communication**, i.e. communicating to or from all test components connected to the communicating test component, have been defined.

The third group of new concepts provided means for handling templates, i.e. test data, in TTCN-3 dynamically. So far, templates could be predefined in the module scope only. Parameters allowed to pass values into templates, however, neither the parameterization of templates with templates was not possible nor the construction of templates at runtime – in particular the dynamic combination of matching mechanisms according to the responses from the SUT. Therefore, v3.1.1 added the concepts of **template variables**, **template parameters**, **template returning functions**, and **local templates**, so that templates became as flexible as values.

A fourth group of extension added **nested type definitions**: so far, every type used within a structured type – for example the type of a record field – had to be explicitly defined before being referenced in the structured type definition. Now, type definitions can be made at the place where they are used – without giving them an explicit type name. This eases the mapping of external type systems like e.g. XML to TTCN-3.

Furthermore, the **logging** has been extended. The log statement has been enabled to log the “status” of any object in TTCN-3, i.e. of values and expressions, test components, timers, ports, and defaults. Test components can be named in the create operation, which eases the reading of test executions logs. Furthermore, the TCI was extended with an explicit logging interface – the TTCN-3 logging interface TLI [10] via which a test execution can be traced in various details – providing the various events during test execution with its timing and additional information.

Last but not least, **documentation support** has been added as a new part [14]: documentation comments with predefined documentation tags such as “@author”, “@remark”, “@desc”, etc. can be used to document TTCN-3 modules on the basis of which documentation artifacts like html-pages can be generated. An example documentation for the triangle tests presented above is given in Fig. 5.



Module TriangleTest

Data Types Summary	
Triangle	whole numbers to characterize a triangle
Classification	properties of a triangle
TriangleTester	test component
TestVector	Combination of Inputs and Expected Outputs

Component Types Summary	
TriangleTester	test component

Port Types Summary	
DetermineTriangle	interface to SUT

Templates Summary	
valid1	valid triangle
valid2	etc.
invalid1	invalid triangle
invalid2	etc.

Altsteps Summary	
Condensing	an altstep to receive unexpected responses

Test cases Summary	
Simple	a simple test case
Extended	an extended test case
Reducing	extended test case using a default

Details

Triangle

Fig. 5. Selected Documentation Elements for the Triangle Tests

Major extensions in version 4.1.1 and 4.2.1 have been the definition of **visibility** rules to control the reuse of definitions, i.e. public, private and friend, and the **import of import** statements. However, above all, **TTCN-3 packages** have been defined. The intention of TTCN-3 packages is to support additional concepts required by specific TTCN-3 targets such as application domains, without making them mandatory for all tool environments. Rather, these packages are optional and can be freely combined with the core language depending on specific requirements of a given application context. For example, a tool environment for automotive electronic control units may require the real-time package of TTCN-3, but not the behavior types package.

The Advanced Parameterization Package [15] defines static value parameterization of types, static type parameterization of types, templates, functions, altsteps and testcases, and default types for type parameters like default values/templates for value parameters. This allows e.g. to define message types with fields of "open" type or functions performing actions on data of "open" types as shown below:

```
type record Data <in type p_PayloadType>
{ Header hdr, p_PayloadType payload}
// a data record with header and payload field
// the hdr field is statically typed
// the payload field receives the type via the type parameter


type record of p_myType MyList <in type p_myType>;
// a list of p_MyType elements

function f_addElem <in type p_myType >
// adding an element to a list
( in MyList<p_myType> p_list, in p_myType p_elem)
 return MyList<p_myType>
{ p_list[lengthof(p_list)]:= p_elem; return p_list; }

f_addElem <integer> ({1,2,3,4}, 5);
// returning {1,2,3,4,5}
```

The Behaviour Types Package [16] allows to define types of functions, altsteps and testcases (FAT) as "prototypes" and to have module parameters, constants, variables and templates of FAT types and to store references to "real" FATs in them. References to FATs can be stored, passed as parameters and sent to other components. FATs can be called via their references.

```
type function MyFuncType ( in integer p1 ) return integer;
// definition of a function type
// with an integer parameter and an integer return
function f_myFunc ( in integer p_int ) return integer
{ return 2*p_int };
// definition of a concrete function compatible to MyFuncType
:
var MyFuncType v_func;
// definition of a variable of function type
v_func := f_myFunc;
// assignment of function f_myFunc to v_func
:
var integer x:= apply(v_func(10));
// execution of the function assigned to v_func, returning 20
```

The Configuration and Deployment Support Package [17] allows to have predefined static test components/configurations which exist across test cases. These "static" test configurations can be created and destroyed in the control part and used for several test cases sequentially. The test cases may add "dynamic" test components and connections to the test configuration but cannot destroy parts of the static test configuration.

```
configuration f_StaticConfig()
// the static test configuration f_StaticConfig
runs on MyMtcType
// having a main test component of MyMtcType
system MySystemType
// testing system under tests of MySystemType
{
     myStaticPTC:= MyPTCType.create static;
     // creation of the static parallel test component myStaticPTC
     map (myStaticPTC:PCO,system:PCO) static;
     // mapping the PTC port to the system port
}
:
testcase tc_test1 () execute on f_StaticConfig
// test case tc_test1 executing on f_StaticConfig
// having main test component, test system interface and parallel
// test component as defined above
{
     :
     myDynamicPTC:= MyPTCType.create;
     // creation of an additional parallel test component myDynamicPTC
     // which ceases to exist when the test case terminates
    :
}
testcase tc_test2 () execute on f_StaticConfig {…}
// test case tc_test1 executing on f_StaticConfig as well

control {
     var configuration myStaticConfig;
     // definition of the configuration variable myStaticConfig
     myStaticConfig := f_StaticConfig();     // configuration setup
     // setup of the static configuration
     execute(tc_test1());
     // execution of tc_test1 on the static configuration
     // the status of the static main and parallel test component
     // and of the statically mapped ports are kept for
     // test case tc_test2
     execute(tc_test2());
     // execution of tc_test2 on the static configuration
     // having the status when tc_test1 was completed
}
```

Last but not least, the Real-Time and Performance Testing Package [18] introduces system time progress (since the start of a test case), delays and timestamps. It allows specifying the required precision of time in the test system, to get the actual time, to suspend the execution of a test component until a given point in time, to specify ports with real-time requirements at which entering messages in the test system interface can be time stamped. The time stamps can be accessed and further used in the TTCN-3 specification.

```
module MyModule {
       :
     type port MyPortType message realtime
       // the message-based real time port type MyPortType
       // the received messages of ports of that type are time stamped
       {...}
       :
      port MyPortType myPort;
       // the message-based real time port myPort
       :
     var float v_specifiedSendTime:=1.0, v_sendTime, v_receiveTime;
       // time variables to store time stamps
     :
     wait (v_specifiedSendTime);
       // wait one second
     myPort.send(m_out);
       // send message m_out immediately
     v_sendTime:= now;
       // assign the current time to v_sendTime
      if (v_sendTime - v_specifiedSendTime > 0.01)
       // too late sending by the test system
      { … } // react accordingly
       :
     myPort.receive(t)-> timestamp v_receiveTime;
       // assign receive time to v_receiveTime
      if (v_receiveTime - v_sendTime > 10.0)
       // too late response from the system under test
      { … } // react accordingly
       :
} with {stepsize ″0.001″};
// defines the time precision of that module to be millisecond
```

This paper allows providing impressions of the recent extensions to TTCN-3 only. The interested reader is asked to refer to the packages their selves for further details.

5 TTCN-3 Tomorrow: An Outlook

TTCN-3 has progressed a lot in the past. It is successfully used across various domains. A plethora of tools is available that support TTCN-3 in different scale. Further test experts are looking into the adoption of TTCN-3 in additional domains such as scientific computing or cloud computing. Still, more needs to be done. In particular people need to be trained in order to enable an efficient use and adoption of this test technology. The established TTCN-3 Certificate provides a good basis – however more reading and training material and training courses should be provided. Also, TTCN-3 is rarely lectured at universities – although free tools help in this. In particular, it is not enough to spread the knowledge about TTCN-3 and its success stories, but also a thorough methodology including guidelines and best practices for the application of TTCN-3 should be provided.

A strong basis will be the further maintenance and evolution of TTCN-3. Although already a quite exhaustive and powerful set of testing concepts is supported by TTCN-3, still more requirements appear – for example to have native support for object-oriented data types. However, like in the past every new concept is very critically reviewed and discussed before being added to the core language or to one – if not to a new – extension package of TTCN-3 in order to keep TTCN-3 handy for the users and tool vendors.

Thinking further, our focus is on establishing TTCN-3 as a middleware for automated tests, like we know it with SQL for databases or with IDL for object-oriented systems. Even if software-based systems differ in its area of application and in its tests, a common number of key concepts for the development of testing solutions can be identified. On this common number of key concepts, implemented in TTCN-3, a corresponding test middleware, dedicated methods and tools for the different application areas can be efficiently and effectively developed. In my view, testing should more and more be seen as an engineering discipline, using common automated methods with educated personnel. The times of manual and proprietary testing solutions should be replaced sooner or later.

For this purpose, the methodology in the application of TTCN-3 should be better mediated and the tools should be oriented even stronger on the needs of the testing personnel. Up-to-date methods of software development should also be offered for the development of testing systems. This includes, among others, refactoring, metrics, verifications, debugging, and even simulations and tests. A major development is the model-based testing methodology [5], which in combination with the TTCN-3 middleware is exploiting its potential at a maximum.

References

1. Rao, G.G., Weigert, T.: Network Element Testing using TTCN-3: Benefits and Comparison. In: SDL Forum 2005, Grimstadt, Norway (June 2005)
2. Ji, W.: TTCN-3 test technique evaluation report, Technical Report, Joint Sino-German Institute, Beijing, China (Mai 2006)
3. Myers, G.J.: The Art of Software Testing. Wiley, Chichester (2004)
4. EUREKA TestNGMed Project: Test automation for next generation medical systems (2010), `http://www.testngmed.org` (Last access August 4, 2010)
5. ITEA D-MINT Project: Deployment of Model-Based Technologies to Industrial Testing (2010), `http://www.d-mint.org` (Last access August 4, 2010)
6. Din, G.: A Workload Realization Methodology for Performance Testing of Telecommunication Services, PhD Thesis, TU Berlin, Faculty Electrical Engineering and Computer Science (September 2008)
7. ETSI ES 201 873-1: Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language (July 2010)
8. ETSI ES 201 873-4: Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Operational Semantics (July 2010)
9. ETSI ES 201 873-5: Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 5: TTCN-3 Runtime Interface (TRI) (July 2010)
10. ETSI ES 201 873-6: Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 6: TTCN-3 Control Interface (TCI) (July 2010)
11. ETSI ES 201 873-7: Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 7: Using ASN.1 with TTCN-3 (July 2010)
12. ETSI ES 201 873-8: Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 8: The IDL to TTCN-3 Mapping (July 2010)

13. ETSI ES 201 873-9: Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 9: Use XML with TTCN-3 (July 2010)
14. ETSI ES 201 873-10: Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 10: TTCN-3 Documentation Comment Specification (July 2010)
15. ETSI ES 202 781: Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 Language Extensions: Configuration and Deployment Support (July 2010)
16. ETSI ES 202 784: Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 Language Extensions: Advanced Parameterization (January 2010)
17. ETSI ES 202 785: Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 Language Extensions: Behaviour Types (January 2010)
18. ETSI ES 202 782: Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 Language Extensions: TTCN-3 Performance and Real Time Testing (July 2010)