PERIODICA POLYTECHNICA SER. EL. ENG. VOL. 49, NO. 3–4, PP. 163–174 (2005)

A NEW MATHEMATICAL FORMALISM FOR THE TTCN 3 CORE LANGUAGE

Szilárd JASKÓ

Department of Information Systems
University of Veszprém
H-8200 Veszprém, Egyetem út 10. Hungary
e-mail: szilard.jasko@weblab.hu

Received: Sept. 14, 2005

Abstract

Protocol testing is a very important field nowadays, because it provides a way to detect different kinds of errors in a communicating environment. Communicating Sequential Processes (CSP) could be efficiently applied for creating a process-based model for the test system. In this article we attempt to provide a CSP module for the most frequently used formal language in testing, the TTCN-3, which could make the testing process easier and cheaper. Another important question is the manufacturing time cost of the product. This article presents a test practise where time can be saved during this period.

Keywords: test system, CSP, test generating, test practise, URN for test.

1. Introduction

Creating test suits for new protocols is not an easy process. This time TTCN-3 is used for generating test cases. TTCN-3 is able to handle different presentation formats, but test process can not be done without human interaction. If the system under test would know its possible traces and would be capable to communicate a simple protocol, CSP could help in automatizing the testing process in a self-adaptive way. Now it is just a theory and it has not been proved yet. CSP is a process-based language, that is why the model-based approach of testing in CSP will also be process-based.

The aim of this article is to show how to create a CSP module for TTCN-3 and how to build up a complex system by using elementary CSP expressions. This can be done because CSP can be used for expressing a control signal based flow. The other part of this article shows a test practice. Time can be saved during the construction phase with the help of it.

The widely used TTCN's main properties and history are presented in Section 2. Then you can read about the new test practice in Section 3. Next CSP and its relation to TTCN are introduced. In Section 4 I give some examples for creating simple CSP expressions and I show how to connect them for building a more complex system. Finally, Section 5 is about our future work and the system's application possibilities.

2. The History of TTCN

In the middle of the 1980s ETSI and the telecommunication industry realized the need for a standardized formal testing language to ensure cooperation between different vendors' devices. It was necessary because of the growing popularity of the mainly heterogeneous telecommunication systems. To solve the problem of interoperability the first version of Tree and Tabular Combined Notation (TTCN) was standardized in 1995 and it was recommended to all telecommunication companies to generate test cases. This language was created especially for testing protocols and not for usual software development purposes. TTCN became widely used in the telecommunication sector.

The first version of TTCN was extended as TTCN-2 in 1997. This testing language was more useful for testing concurrent systems than the previous version. Moreover it was built up modularly, that's why it allows reusing tests between different projects and makes multi-user test suite development possible.

Breaking up with the tree and tabular form the third generation of TTCN [5, 7], Test and Test Combined Notation was standardized in 2001. TTCN-3 is better suited for testing 3rd generation protocols, where voice and (multimedia) data communications are dominant. It is also ideal to use in distributed applications. Test cases created in TTCN-3 core language can be presented in tabular, graphical or several other forms to the TTCN-3 user and support different types of data (e.g. ASN.1) (*Fig. 1*).

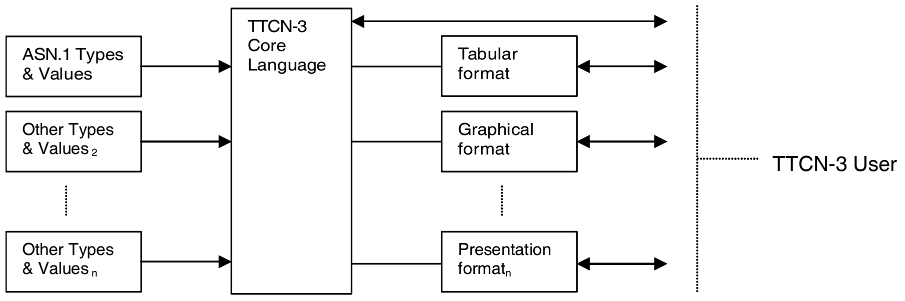


Fig. 1. Capabilities of TTCN-3

3. New Test Practice

Testing is a very important and expensive process in the area of telecommunication. The final price of the product is affected by the test practice used in the given flow. That is the cause to develop newer solutions. This accomplishment tries to save time and money compared with the practices used nowadays.

3.1. Requirement Specification

The right requirement specification is the base of the planning. First, the goal of the work has to be determined and all changes have to be managed during the planning period. The requirement specification is particularly important in the big system's case. The user requirements have to determine the form of the goals and scripts. The goals include the non-functional requirement of the planning software. For example, it is a non-functional requirement if the system has to contain certification mechanism. The scripts include the functional requirements. For instance, what kind of communication flow is between the server and the client to realize the given certification. The big standardization organizations worked to create a standardized required management tool in the telecommunication area. The name of this standard is URN (User Requirement Notation) [6, 8]. It is an ITU-T standard and the other name of it is Z.150. It contains two parts: the non-functional, goal-oriented language and the functional use cases. One is called GRL (Goal-oriented Requirement Language) and the other is called UCM (Use Case Maps).

3.2. Testing in the Planning Phase

The testing phase is one of the most important parts in the life of all products. It can decide if the given product should be dropped into the market or some mistakes have to be corrected beforehand. This period is one of the most expensive parts of manufacture in the area of telecommunication. Therefore all methods should mean extra profit to the given industrial units, which help the companies to create cheaper products or faster manufacturing flow or both together.

One of these factors should be when the testing is used from the beginning of the planning phase. The testing can be prepared deliberately and expertly in this case. A further advantage is that the product development and the product specified test plan are parallel processes. Hence the parts of the system or the whole system are testable when it finishes, so time can be saved with the help of this practice.

3.2.1. An Example for Test Integration in GRL

First of all a goal is defined with the rules of GRL. For example a conformance test is the main goal. It means that the developed system is tested by a conformance test [1, 3, 4]. This kind of test checks the equivalence between the product and the given standard. It can be seen in *Fig. 2*.

One technical approach is defined in the next step. It has to be given how this goal would be solved. It can be given with the help of task. Everybody has to see the fact that this task provides a method only and not an exact solution. It can be seen in *Fig. 3* that the conformance test is done with the help of TTCN-3.

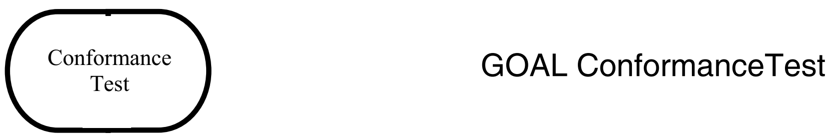


Fig. 2. A goal in GRL

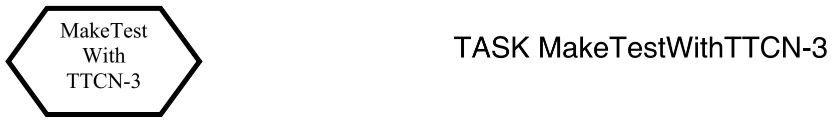


Fig. 3. A task in GRL

Then the resources needed for the test flow have to be defined. Only the TTCN-3 software is used in this example. So the resource is the following:

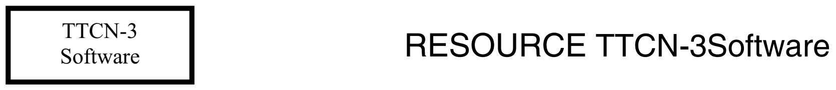


Fig. 4. Resource in GRL

Soft goals are given to testing with the help of GRL. The perfect and easy test is such a soft goal in this area. It is soft, because the perfect and easy are not an unambiguous concept in the world of testing.

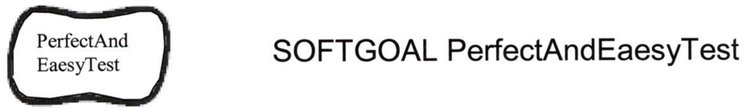


Fig. 5. A softgoal in GRL

If these GRL elements are linked together, the following figure comes as a result:

Very important information is got from this kind of description and this is the aim of testing. The environment of the test is shaped from the right GRL diagram. This process can be parallel with the planning and construction phases. That is one of the reasons, why time could be saved with the help of this practice.

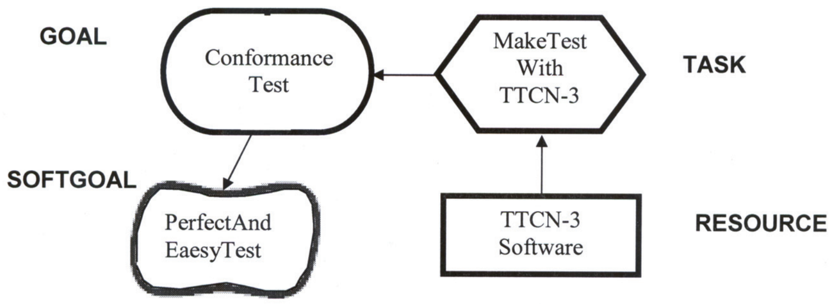


Fig. 6. An example for GRL description

3.2.2. An Example for Test Integration in UCM

The working flows of the given system are described graphically with the help of UCM. For example, a simplified phone connection establishment can be seen in *Fig. 7*.

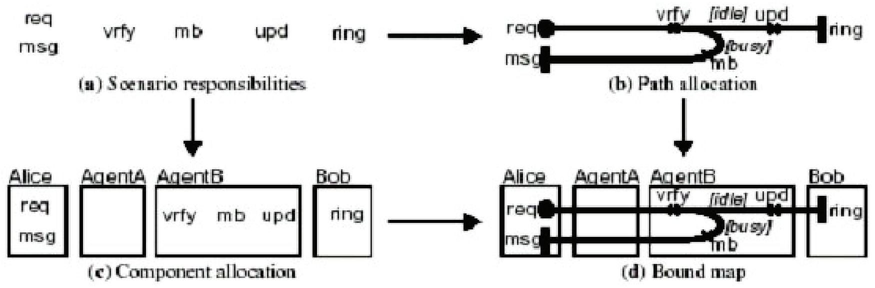


Fig. 7. An example for UCM description

There are two possibilities in this case. One is to ring the phone of Bob. The other is when Alice hears the busy tone. A very important information for testing is included in this kind of description way. The test points of the given system can be read out of it. The system under test is connected with the tester through those points. For example, interface of Alice is a point of the test, if Bob's phone would like to be tested.

4. CSP in Testing

A more effective test system could be created, if there would be an efficient tool for automatizing test methods. The behaviour and the working flow of the system have to be exact and provable for everybody. CSP [2] is such a tool. It is based on an exact mathematical formula and the correct behaviour of our system can be proved with the help of it. Furthermore 'intelligence' can be programmed in our CSP code. For example, a test system was described with the help of this mathematical tool in an earlier work [9, 10] that was able to generate a set of simple normal test cases automatically for an average protocol. Of course, the protocol was integrated in the test system before. This application of the CSP is totally new in the area of testing. That is the cause why the article has got the given title.

4.1. What is CSP?

CSP is a notation for describing concurrent systems where the component processes interact with each other by communication. The mathematical background of CSP is the process algebra. A system is built up by sequential processes which communicate with each other parallel.

Each process has its own alphabet which is the set of all communication events the process can use. CSP defines several operators. We can express sequential communicational events, recursion, different choices (deterministic, non-deterministic), and simultaneous behaviour of processes.

A P process is equal to *a* communicational event followed by a process behaving like Q that it is denoted by 'P = a -> Q'. If a choice between processes P and Q is made by the environment, it can be expressed by '$P\Box Q$'. 'P <x=3> Q' is a conditional choice. It means that P is executed if x is equal to 3, else process Q is chosen. The simultaneous execution of processes Q and P is denoted by 'Q || P'.

A communicational event can be a signal which is received or sent through a channel. CSP notation for output is 'c!x' where c is the name of the channel and x is the signal's name. Input is denoted by '?', therefore 'c?x' means, signal x is received on channel c.

CSP has also predefined processes, some of them are RUN, SKIP, STOP.

One important property of CSP is that it gives us a tool for determining the trace of processes. The trace of a process means the set of the possible sequences of events, which the process can perform. For example if process P is defined by 'P = a -> P', then traces(P) is: traces(P)={<>, <a>, <a,a>, <a,a,a>, <a,a,a,a>, ...}

Finally there are tools for checking CSP implementations. Animators make it possible to write arbitrary process descriptions and to interact with them, while refinement checkers explore all of the states of a process.

4.2. The Connection between CSP and TTCN-3

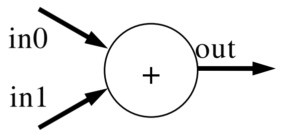
The efficiency of the developed system will be better if it can be connected to a good and widely used test tool. TTCN is the only one standardized test system in the area of telecommunication. Because of the modular structure of TTCN-3 there is the theoretical chance to connect a CSP module to the TTCN-3 core language. The core is programmed with the help of it. Hence the automatized test methods in CSP can directly control the core language. More time and money can be saved with the aid of this technology. Notice that there is an additional advantage: the final test system will be self-adaptive, so it will be able to work without human interaction. Engineers have to control only this process and nothing else.

4.3. Control Signal Based Process Flow and the Modular System

CSP is a versatile and flexible mathematical formula. It is proved by the fact that an expression can be described in many ways. A simple example can be seen in the followings. The addition operator is used for this work.

$$ADD(x,y)= (x+y) \rightarrow ADD(x,y)$$

Perhaps it is the simplest addition method. We can observe that this realization is similar to the function principle of programming language. The *ADD* process has two parameters and it adds them. The next implementation is the 'channel', it gets the information from the channel to be processed, then the processed data are put also onto the channel. We can see an example for this working mechanism in *Fig. 8.*

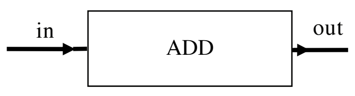


```
PlusInt (in0, in1, out) = (in0?x0 -> SKIP || inl?x1 -> SKIP);
        out!(x0 + x1) -> PlusInt (in0, in1, out)
```

Fig. 8. The ADD process

The parameters are the names of the channels used for communication. The system waits until it gets the data to be processed from the 'in0' and 'in1' input channels. If both items arrive, the *ADD* process will add them together and puts the sum on the 'out' output channel. The advantage of the previous two solutions would be combined in this work. Variables and values could be given by parameters. Data arriving on coming from the channel should start the working flow of the module. This data provides information for the system if it is necessary. This combined technology is called control signal-based process flow and we can see an example for it in *Fig. 9.*

We can see clearly that the process runs if and only if it gets a control signal from channel 'in'. If it happens, the module will sum 'value1' and 'value2' and the result of the action will be copied to 'variable'. Then it sends the control signal on channel 'out'. It is worth to note that the basic operations are working as modules. For example one basic module is the addition. A complex system can be built from the basic elements similarly to a Lego game.

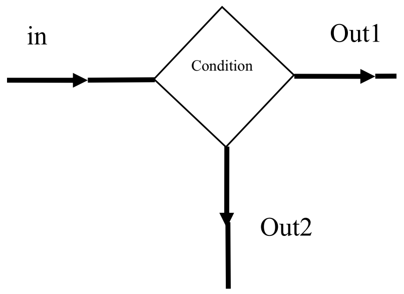


```
ADD(in, out, variable, value1, value2) = in?x ->
variable= value1+value2 -> out!x->
ADD(in, out, variable, value1, value2)
```

Fig. 9. Control signal-based process flow

4.4. An Example

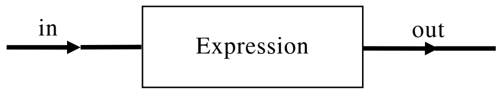
The way of joining together the modules of the control signal-based process flow and the functional connection between TTCN-3 and this formalism will be shown in a simple example. Two basic modules can be seen at the beginning: the 'path-choice' and then the 'expression'.



```
PathChoice(in, out1, out2, condition) =
in?x -> out1!x <condition> out2!x ->
PathChoice(in, out1, out2, condition)
```

Fig. 10. The path-choice module

The 'path-choice' basic module of CSP code and the graphical description can be seen in *Fig. 10*, too. If this module gets a control signal from channel 'in', the process will run. This flow is the following: if the given condition is true, it will send the control signal on channel 'out1', otherwise it will send the control signal on channel 'out2'.



```
Expression(in, out) = in?x -> (arbitrary
CSP process set) -> out!x ->
Expression(in, out)
```

Fig. 11. The expression module

Another module can be seen in *Fig. 11* and any CSP code can be integrated in the control signal-based process flow with the help of it. After the additional code execution the control comes back to the system. If these two elements are combined, an 'if' function will be received. The role of 'if' function can be seen in *Fig. 12*.

Fig. 12. If function in CSP

We can see clearly from this example that it is possible to build complex systems from basic modules. If this element gets the control, the following will happen: If the condition is true, 'expression 1' will run, otherwise 'expression 2' will run. Of course 'expression 1' and "expression 2" include the wanted functions. Next we can see the 'if' function in TTCN-3 core language. Those two expressions are equivalent.

If *(condition) statementblock1* ***else*** *statementblock2*

During the research the equality between all the functions of TTCN-3 core language and the control signal-based CSP code will be proved.

5. Possibility of further Development/Expansion/Extension

The fundamentals of the system and the schedule of the research have already been created. The results have been very promising till now. If a protocol knows its own CSP traces, a normal test case can be generated totally automatically from traces at the present state of the research, because the traces include the communication and working rules. One of the most important topics in the extension of this test is generating a method to other cases. For example, those should be erroneous behaviour, performance test etc. Although it is important to prove the following beforehand: TTCN-3 core language is programmed directly from CSP. The efficiency can be made better if the issue of the automatization process is given immediately to a nowadays used test system. The steps of the research are these: The equivalence between the TTCN-3 core language and the CSP tool should be proved. The idea of provement is the following: the TTCN-3 core language will be modelled by a state machine. If it is a finite state machine (FSM), the equivalence will be proved, because the CSP uses an infinite state machine and it has a wider range set than FSM. If it is infinite, the functions of the TTCN-3 core language will be linked by functions of the equivalent CSP. Then CSP code can be translated to the core. If it is finished, a compiler will be developed that can transform the mathematical description to the running code totally automatically. Then the test system will be extended to work in an average environment and it will be able to generate most kinds of test cases automatically. Finally the domains of the research will be joined together to create a working system. It will be able to generate test cases for the protocols that are integrated to the system. It means that the tested system has to know extra information that are the traces of the given protocol and the working mechanism of a simple communication protocol. The tester gets the additional information from the tested machine with the help of this protocol [9, 10]. The system will be able to translate the generated tests to TTCN-3 core language. This way such a test system is created that can be applied immediately. The research will be finished if the test tool will be able to recognize the totally unknown protocol family and the protocol itself and then it will able to generate test cases.

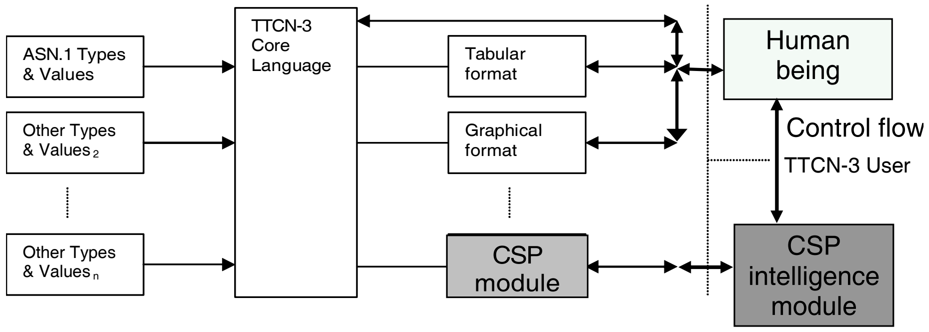


Fig. 13. The frame work of the tester

The frame work of this system can be seen in the *Fig. 13*. The innovation in this hierarchical structure is two new modules. They are the CSP intelligence module and the CSP module.

CSP intelligence module has to solve the following functions: First of all this is connected to the tested system and it asks for the traces of the protocols that the tested system use. In the next step, the module recognizes the protocols and generates test cases from the traces. Finally it builds a test description according to the control signal principle.

CSP module is a programming interface like the graphical format or tabular format. This module uses the control signal description method in CSP. Graphical description is a part of this module so users can see through the test description easily.

Therefore, the CSP module is the programming interface between the test system and the TTCN-3 core language. One part of the test system is the CSP intelligence module. This CSP tool generates the test cases from the given traces. Of course it has to include some kind of artificial intelligence, because the test generating process is very complex and this problem can not be solved without multiple logical units. That is where the name of this module comes from. Human beings can control this automatic flow during 'control flow' channels/interfaces. The control is a central question, because if a machine is able to test totally automatically, an occurrent faulty work will lead to blind corner consequences.

6. Conclusion

In this article a new approach for testing was introduced. CSP process algebra was applied for building a new module for TTCN-3. It could be connected to TTCN core language. By the help of this process-based test system protocols could be tested without human interaction, moreover self-adaptively. I have shown how to begin the building of the CSP module, how to create CSP expressions for every TTCN expression and how to build a complex system with their help. Based on this model I want to develop the whole CSP-based test system to get a more efficient and faster test process.

The other part of this article has given a test practice and the test phase could start earlier than the practice used nowadays. Two very important informations could be seen from the beginning with the help of this technique: they were the aim of the test strategy and the point of the test. These two things were enough to start the preparation of the testing phase. Therefore there is good chance to save time during the construction process.

Nomenclature

ASN.1	– Abstract Syntax Notation One
CSP	– Communicating Sequential Processes
ETSI	– European Telecommunications Standards Institute
GRL	– Goal & oriented Requirement Language
ISO	– International Organization for Standardization
ITU	– International Telecommunication Union
TTCN	– Tree and Tabular Combined Notation (Testing and Test Control Notation)
UCM	– Use Case Maps
URN	– User Requirement Notation

References

For a paper citation:

[1] BAUMGARTEN, B.– GIESSLER, A., OSI Conformance Testing Methodology and TTCN, *Elsevier Science Ltd*, 11, Jan. 1994.

For a book citation:

[2] ROSCOE, A.W., The Theory and Practice of Concurrency, *Prentice Hall International Series in Computer Science ISBN 0-13-674409-5*, 1997.

[3] ETS 300 406, Methods for Testing and Specification; Protocol and Profile Conformance Testing Specification, *Standardization Methodology, ETSI*, 1995.

[4] LINN, R. J. – UYAR, M. U., Conformance Testing Methodologies and Architecture for OSI Protocols, *IEEE Computer Society Press.*, 1994, pp. 93–152.
[5] The Tree and Tabular Combined Notation version 3 (TTCN-3), Core language, *ITU-T Recommendation Z.140*, July 2001.
[6] User Requirements Notation (URN) - Language Requirements and Framework, *ITU-T Recommendation Z.150*, 2003
For an URL citation:
[7] http://www.telelogic.com/products/tau/TTCN/index.cfm
[8] http://www.itu.int/itudoc/itu-t/workshop/framewrk/009/78074.html
For a conference citation:
[9] JASKÓ, SZ., New Views in the Testing of Protocols, *Miskolc, Microcad 2003*.
[10] JASKÓ, SZ. – DULAI, T. – MUHI, D. – TARNAY, K. Generating Test Case(s) in CSP (Communicating Sequential Processes), *Miskolc, Microcad 2004*.