# Process-based model for a test system

Szilárd Jaskó[1], Tibor Dulai[1] , Dániel Muhi[1], Katalin Tarnay[2,1]
Internet-based Telecommunication Laboratory, Department of Information Systems
[1] University of Veszprém - Hungary
Egyetem u. 10, H-8200 Veszprém
Hungary
[2] Nokia-Hungary
Köztelek u. 6, H-1092 Budapest
Hungary
*jaskosz@freemail.hu, tibor.dulai@irt.vein.hu, daniel.muhi@irt.vein.hu, tarnay.katalin@axelero.hu*

***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.**

## I. 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's why the model based approach of testing in CSP will also be process based.

The aim of this article we show how to create a CSP module for TTCN-3 and how to build up a complex system by using elementary CSP expressions. It can be done because CSP can be used for expressing a control signal based flow.

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

## II. THE HISTORY OF TTCN

In the middle of the 1980s ETSI and the telecommunication industry realised 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 was recommended for all telecommunication companies to generate test cases. This language was created especially for testing protocols and not for usual software development purposes. TTCN became used widely 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], 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) (Figure 1).

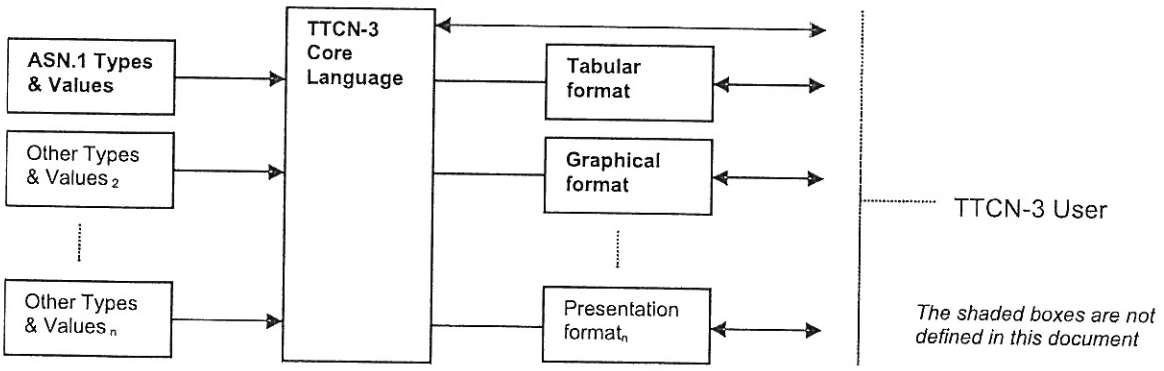



Fig. 1. Capabilities of TTCN-3

## III. 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 formalism 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.

### *A. 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 behaves 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 c 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.

### *B. The connection between CSP and TTCN-3*

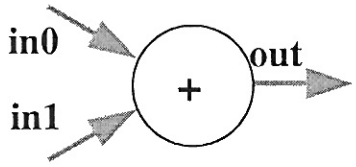
The efficiency of the developed system will be better if that can be connected to a good and widely used test tool. TTCN is the only one standardized test system in the area of telecommunication. Because of the modular structure of TTCN-3 there is the theoretical chance to connect a CSP module to the 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.

### *C. Control signal based process flow and the modular system*

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

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

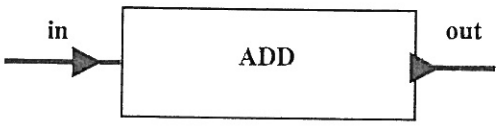
Perhaps it is the simplest addition method. We can observe that this realization is similar to the function principle of programming language. The *ADD* process has two parameters and it adds them. The next implementation is the "channel", it gets the information from the channel to be processed, then the processed data is put also onto the channel. We can see an example for this working mechanism in Figure 2.




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

Fig. 2. 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 Figure 3.



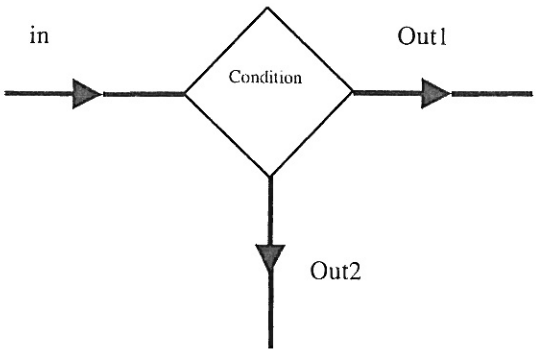

ADD(in, out, variable, value1, value2) = in?x ->
variable= value1+value2 -> out!x->
ADD(in, out, variable, value1, value2)

Fig. 3. Control signal based process flow

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.

*D. An example*

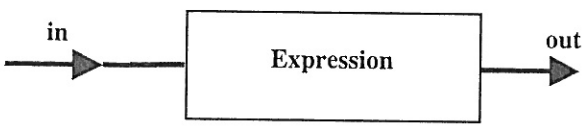
The way of joining together the modules of the control signal based process flow and the functional connection between TTCN-3 and this formalism will be showed in a simple example. Two basic modules can be seen at the first time: the "path-choice" and after the "expression".




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

Fig. 4. The path-choice module

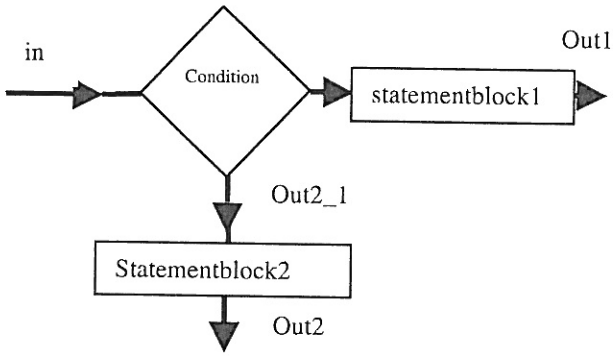
The "path-choice" basic module of CSP code and the graphical description can be seen in Figure 4, too. If this module gets a control signal from channel "in", the process will run. This flow is the following: if the given condition is true, it will send the control signal on channel "out1", otherwise it will send the control signal on channel "out2".




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

Fig. 5. The expression module

Another module can be seen in Figure 5 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 Figure 6.




If(in, out1, out2, condition) =
PathChoice(in, out1_1, out2_1, condition)| |
Expression(out1_1, out1) □
PathChoice(in, out1_1, out2_1, condition)| |
Expression(out2_1, out2)

Fig. 6. 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.

## IV. FUTURE WORK

The fundamentals of the system and the schedule of the research have already been created. The results have been very hopeful 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. One of the most important topics in the extension one this test is generating a method to other cases. 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. 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 protocol has to know its own traces and the working mechanism of a simple communication 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.

## V. 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 test without human interaction, moreover self-adaptively. We have showed 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 we want to develop the whole CSP-based test system to get a more efficient and faster test process.

## VI. REFERENCES


For a paper citation:

[1] Bernd Baumgarten and Alfred Giessler, "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] Richard J. Linn with M. Umit Uyar, "Conformance testing methodologies and architecture for OSI protocols", I E E E 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

For an URL citation:

[6] http://www.telelogic.com/products/tau/SDL/index.cfm

[7] http://www.uml.org/

[8] http://www.rational.com/uml/resources/documentation/index.jsp

For a conference citation:

[9] Jaskó Szilárd, "New views in the testing of protocols", *Miskolc, Microcad 2003*

[10] Jaskó Szilárd, Dulai Tibor, Muhi Dániel, Dr. Tarnay Katalin, "Generating Test Case(s) in CSP (Communicating Sequential Processes)", *Miskolc, Microcad 2004*