Verification of Communication Protocols Based on Formal Methods Integration

Slavomír Šimoňák

Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice
Letná 9, 042 00 Košice, Slovakia
e-mail: slavomir.simonak@tuke.sk

Abstract: Communication protocols define the set of rules needed to exchange messages between communicating entities. Networked and distributed systems, built around communicating protocols, are widely used nowadays. Since such systems are often deployed in safety-critical applications, confidence in protocol correctness is highly required. We propose an approach based on formal method integration to support the modeling and analysis of communication protocols. Process algebra and Petri nets are used together to combine the best properties from both methods – the exceptional properties for system description offered by process algebra, and the powerful analytical properties of Petri nets. The ideas described within the paper are demonstrated by an example – the Trivial FTP (TFTP) protocol.

Keywords: protocol correctness; formal methods integration; Petri nets; process algebra

1 Introduction and Motivation

A protocol is a set of rules which must be followed in the course of some activity. Originally, the term was used in connection to human (more or less formal) activities. If the protocol is not followed, the activity will not be successful. Nowadays the term is increasingly used when the communication between computers, computer components or computer systems is considered [19]. Within the paper we will focus on this type of *communication protocols*. Communication protocols thus define the set of rules needed to exchange messages between two or more communicating entities [4, 5, 17]. Such protocols are elements of great importance when networked and distributed systems are considered [25]. Nowadays such systems are very common, and incorrect communication, or no communication at all, can cause complications, the severity of which can range from financial loss (internet banking, e-shopping) to issues like human health and lives (medical or transport systems).

2 Protocol Correctness

The mentioned risks are motivating factors for the development and use of protocols to ensure a correct information exchange between communicating entities. But what is the protocol *correctness,* and how can it be shown that the particular protocol is correct? We can look at the problem from different points of view. In the event that we want to check if the system complies with the requirements and performs the functions for which it is intended, we are talking about *validation*. A process used to determine if the system is consistent, adheres to standards, uses reliable techniques, performs the selected functions in the correct manner is referred to as *verification* [26, 10, 15]. Confidence in protocol correctness can be increased in different ways. *Testing* is one method and involves building a prototype and observing it, or observing the behavior of a real system. The main disadvantage of testing is that it can be used to show errors, but not to prove correctness. *Simulation* is a method based on the construction of an executable model of the system and its observation. Simulation requires the generation of test cases, which are difficult to design for complex systems. In the case of critical properties, simulation is not believed to provide sufficient confidence [2]. Methods such as simulation and testing are usually used with success to show the performance characteristics of the system considered. *Formal methods,* on the other hand, are mathematically-based techniques offering a framework in which systems can be specified, developed and analyzed in systematic manner. Generally, they are not suitable for assessing a system's performance, but there exist methods specifically designed for this purpose (e.g. Performance Evaluation Process Algebra – PEPA) [9]. Since performance analysis is not our goal here, we will focus our attention on employing formal methods in protocol correctness analysis further in this paper.

3 Related Works

Many attempts have been made to perform protocol analysis based on formal methods [7, 27, 2, 6, 13]. Process algebras, Petri nets and other methods have been employed for the specification and analysis of these systems. Process algebraic constructs, like basic operators for constructing finite processes (alternative and sequential composition), communication, encapsulation, abstraction and other operators, form a solid basis for specification and analysis of wide range of systems. In particular, they are suitable for the specification of communication protocols. The usual method for performing the system analysis in this case is the following: Firstly, the desired external behavior of the protocol is specified in the form of a process term (usually using basic operators and recursion). Next, the implementation of the protocol is specified in the form of a process term (usually including basic operators, parallel operators and recursion). Then, internal actions are forced to communicate using the encapsulation operation and the internal communication actions are made invisible using an abstraction operator, so effectively only the input/output relation of implementation is visible. Finally, using the axiom system, we try to show the terms are equal. By this equality we prove the desired external behavior and the input/output relation of the implementation are (rooted branching) bisimilar [7]. The above mentioned process enables us to show that the system meets properties included in the term validation.

4 The Method

Instead of the verification process just described (or as an addition to it), we propose to use a method based on a combination of two formal methods: process algebra and Petri nets [22]. Our aim here is to use the best properties from both worlds – the exceptional properties for system description offered by process algebra, and the powerful analytical properties of Petri nets. We believe it is easier to create a description of a communication protocol using the constructs of process algebra. The main reason for our belief here is that de/composition and communication can be expressed more naturally using special algebraic constructs than using Petri nets. The larger and more complex the modeled system, the greater is the impact of this advantage. In the case of communication protocols, the system usually consists of communicating entities, medium and maybe other parts, which can be specified separately and put together by means of the parallel composition operator. On the other hand, we believe the analysis is usually better/easier done using Petri nets. Petri nets are a well-known formal method, mainly due to their valuable analytical properties and intuitive graphical representation [27, 3]. Two types of properties can be investigated using the Petri net models: properties which depend on the initial marking (behavioral properties) and those which are independent of it (structural properties). Problems connected with analysis of behavioral properties include reachability, boundedness, liveness, reversibility and home state, coverability and other problems. Structural properties, depending on the topological structures of Petri nets, on the other hand, hold for any initial marking or are concerned with the existence of certain firing sequences from some initial marking. Properties of this kind include structural boundedness, conservativeness, repetitiveness and consistency [14, 11]. Invariants of the system can be derived from the structure of the net, so the construction and analysis of the state space (which can be of great size) is not necessary here. Invariant-based analysis alone is a powerful tool for studying the structural properties of Petri nets.

Many different dialects of Petri nets are available today and differ by the properties such as *modeling power* and *decision power*. Modeling and decision power are in some respects antagonistic properties; by increasing modeling power, decision power usually decreases. In our case, ordinary Petri nets are considered, which represents a good balance between modeling and decision power.

The key element of the method proposed here is the automatic, semantic-preserving transformation of process-algebraic specification into the Petri net-based one. After the transformation is performed, the powerful analytical properties of Petri nets can be used. By the analysis we can disclose defects in the internal consistency and correctness of the specification, which can potentially be hidden within the specification.

5 The TFTP Protocol

The transformation method developed by the author, described in deeper detail, can be found in [20, 21]. Transformation is automated by tools like ACP2Petri and PATool [23], both developed at the author's home institution. ACP2Petri is the tool performing the transformation itself. It accepts a process algebraic, ACP-based [1] specification in the PAML language as the input and produces the corresponding Petri net in the standard PNML format, which is supported by various analytical tools, such as TINA, Netlab and PNtool2 [24, 16, 12].

The TFTP (Trivial File Transfer Protocol) [8, 18] was chosen as an example to demonstrate the ideas presented above. TFTP is a simple protocol to move files between machines. It is designed to be small and easy to implement, so it lacks most of the features of a regular FTP. The protocol only supports reading and writing files from/to a remote server. It cannot list directories and currently has no support for a user authentication. TFTP is a protocol with strict data transfer restrictions. When an error occurs, the current transfer is stopped and connection is terminated, so it is necessary to establish the connection and start the transfer again. Communication between the server and the client will be described at the level of packets exchange. According to the type of data within a packet, packets can be subdivided into types summarized in Table 1.

Table 1

TFTP packet types

Packet type	**Description**
RRQ	Read request
WRQ	Write request
D1	Data 1 (first packet of a file)
DL	Data L (last packet of a file)

DN	Data N (N-th packet of a file)
ACK0	Answer after the request for writing to the server was received
ACK1	Answer after receiving the first packet of the file
ACKN	Answer after receiving the N-th packet of the file
ERR	Error

All the packets of the communication serve one of the following purposes:

- To transfer the (parts of the) file, i.e. data packets (D1, DL, DN).
- To control the transfer, i.e. control packets (ACK0, ACK1, ACKN, ERR, RRQ, WRQ).

Basic TFTP functionality includes reading a file from the server and writing a file to the server, respectively. Let us describe those activities in deeper detail.

5.1 Reading a File from the Server

The operation is initiated by sending the RRQ packet by the client. The server can respond to this request in three ways:

- By sending ERR, if the file requested does not exist, or it is unable to read the file.
- By sending D1 – a positive answer to the request and a first packet of the file.
- By sending DL – a positive answer too, but also a signal, that this is the only packet of the file.

When a client receives the ERR packet, the reading of the file is terminated and it is able to send its new request. The client replies to D1 and DL (received as the first after RRQ packet) by sending the ACK1 packet. If server receives the ACK1 packet after sending DL, it terminates the connection and is ready for the next request. In the case that the server receives ACK1 after sending D1, it responds with the next part of the file in form of a DN or DL packet, where the latter of the two is used, when the last part of the file is to be sent. The client replies to the received DN by the ACKN packet, where N is the packet number, and to packet DL (when it is not only packet of the file), replies also with DN.

5.2 Writing a File to the Server

The writing operation is very similar to the reading one, with one significant exception: after receiving the writing request in form of the WRQ packet, the server replies by ACK0, which is a packet type reserved especially for this purpose only. The rest of communication runs analogically to the reading operation explained above.

The protocol operation from the client's point of view is depicted in Figure 1.

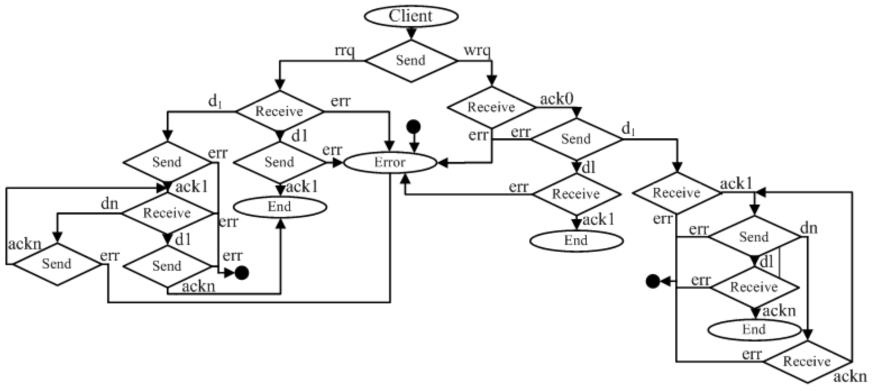

Figure 1
Protocol operation, client point of view

5.3 Formal Specification

Formal specification of the TFTP protocol is based on the analysis of its operation given above. The rules for naming the actions used within the specification are as follows. The first symbol of the action name is one of the three following: c (client), s (server) and m (medium); the second symbol is '_' (underscore); and the third one gives a type of communication (s-send, r-receive). The next symbols represent an abbreviation of the message (packet) type. Some packet types are used in both directions of communication, so the direction in these cases is expressed by the suffix w (write to server) or r (read from server). Packets of ERR type are distinguished similarly according to the direction by adding a suffix s (from the server) or c (from the client), respectively. The two operations supported by the TFTP protocol (read and write) are mutually independent, so it is possible to perform decomposition and specify (and analyze) each of them separately. The reading operation (TFTPR) specification only is given within the rest of this paper. In the case of interest in a whole TFTP specification, please refer to [8]. The following TFTPR specification is given in textual form [23] of process algebra ACP [1].

```
*Communication
gamma (c_srrq,m_srrq) = srrq         gamma (s_sdnr,m_sdnr) = sdnr
gamma (m_rrrq,s_rrrq) = rrrq         gamma (m_rdnr,c_rdnr) = rdnr
gamma (c_sack1r,m_sack1r) = sack1r   gamma (s_sdlr,m_sdlr) = sdlr
gamma (m_rack1r,s_rack1r) = rack1r   gamma (m_rdlr,c_rdlr) = rdlr
gamma (c_sacknr,m_sacknr) = sacknr   gamma (s_serrs,m_serrs) = serrs
gamma (m_racknr,s_racknr) = racknr   gamma (m_rerrs,c_rerrs) = rerrs
gamma (s_sd1r,m_sd1r) = sd1r         gamma (c_serrc,m_serrc) = serrc
gamma (m_rd1r,c_rd1r) = rd1r         gamma (m_rerrc,s_rerrc) = rerrc
```

We started with the definition of communication between actions, where the ACP-style binary communication function gamma gives the action that is the result of communication. Actions not essential from our point of view are hidden (encapsulated) further in order to concentrate on the protocol operation.

```
*Encapsulation
encset[H](c_srrq,m_srrq,m_rrrq,s_rrrq,c_sack1r,m_sack1r,m_rack1r,s_ra
ck1r,c_sacknr,m_sacknr,m_racknr,s_racknr,s_sd1r,m_sd1r,m_rd1r,c_rd1r,
s_sdnr,m_sdnr,m_rdnr,c_rdnr,s_sdlr,m_sdlr,m_rdlr,c_rdlr,s_serrs,m_ser
rs,m_rerrs,c_rerrs,c_serrc,m_serrc,m_rerrc,s_rerrc)
```

The specifications of client, server and the messages transferring medium are given. The recursive specifications used here reflect their repeated activity.

```
*Client
CRN=c_rdnr.(c_sacknr.CRN+c_serrc)+c_rdlr.(c_sacknr+c_serrc)+c_rerrs.C
CR1=c_rd1r.(c_sack1r.CRN+c_serrc)+c_rdlr.(c_sack1r+c_serrc)+c_rerrs.C
C = (c_srrq.CR1).C

*Server
SRN=s_sdnr.(s_racknr.SRN+s_rerrc)+s_sdlr.(s_racknr+s_rerrc)+s_serrs.S
SR1=s_sd1r.(s_rack1r.SRN+s_rerrc)+s_sdlr.(s_rack1r+s_rerrc)+s_serrs.S
S = (s_rrrq.SR1).S

RRQ = m_srrq.m_rrrq.RRQ
ACK1 = (m_sack1r.m_rack1r).ACK1
ACKN = (m_sacknr.m_racknr).ACKN
D1 = (m_sd1r.m_rd1r).D1
DN = (m_sdnr.m_rdnr).DN
DL = (m_sdlr.m_rdlr).DL
ERRS = (m_serrs.m_rerrs).ERRS
ERRC = (m_serrc.m_rerrc).ERRC

*Composition
TFTP = encaps[H](C||S||RRQ||ACK1||ACKN||D1||DL||DN||ERRS||ERRC)
```

At the end of the specification, all the components are put together by means of the parallel composition operation. The encapsulation set (H) is applied to the whole composition. The specification given above is translated into a machine readable, XML-based PAML format using the PATool [23]. The resulting algebraic specification is transformed subsequently into the corresponding Petri net (Figure 2) and stored in PNML format. The resulting Petri net generated by the ACP2Petri tool has a simple (matrix) layout and was edited manually to the form depicted in Figure 2 using the TINA Toolbox [24].

5.4 TFTPR Analysis

Petri net analysis can be used for the investigation of several properties of modeled systems, as was mentioned in Section 4 of the paper. The methods to analyze Petri net models may be subdivided into the following three groups: the coverability (reachability) tree method, the matrix-equation approach and the reduction/decomposition techniques [4, 11, 14].

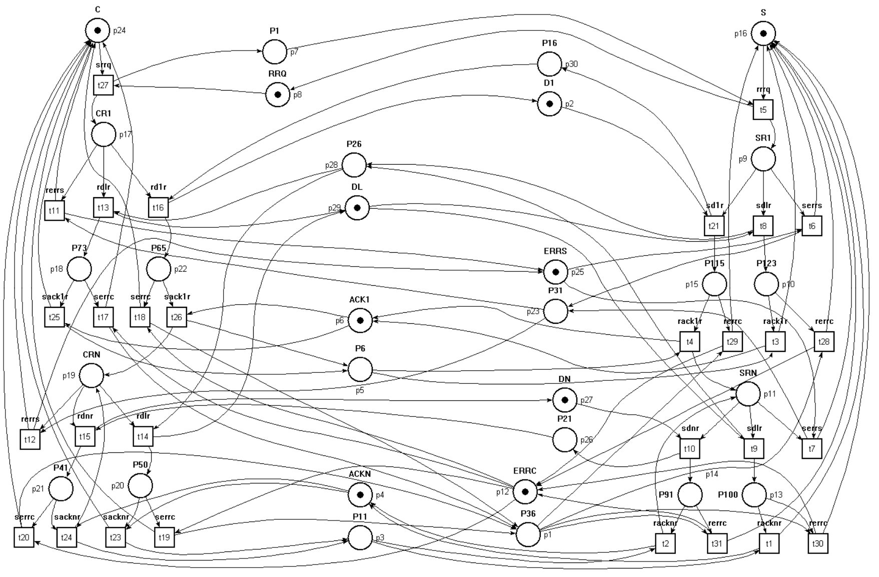

Figure 2
Petri net of TFTPR protocol

The execution of operations or occurrence of events within the modeled system is simulated by the firing of Petri net transitions. State changes of the system thus are reflected by the changes in distribution of tokens (marking) in places of Petri net. By analysis of these changes, one can study the dynamic behavior of the modeled system. For instance, when the sequence of messages (*srrq* – send read request, *rrrq* – receive read request, *sd1r* – send first packet of data read from the server, *rd1r* – receive first packet of data) is exchanged between a client and a server, starting from the initial state of our system (Figure 2), the client can respond by sending one of two messages – error (*serrc*) or acknowledgement (*sack1r*) respectively, depending on the successfulness of receiving the first packed of requested data file. Within the corresponding Petri net, the situation is modeled by two enabled transitions (*serrc* and *sack1r*), as depicted in Figure 3.

Invariants alone have numerous applications and form the basis for many necessary/sufficient conditions of Petri net model properties. There are a variety of tools available today which help with invariants calculation. In our case, the Netlab [16] tool was used and invariants of places and transitions calculated are summarized in Table 2 and Table 3, respectively.

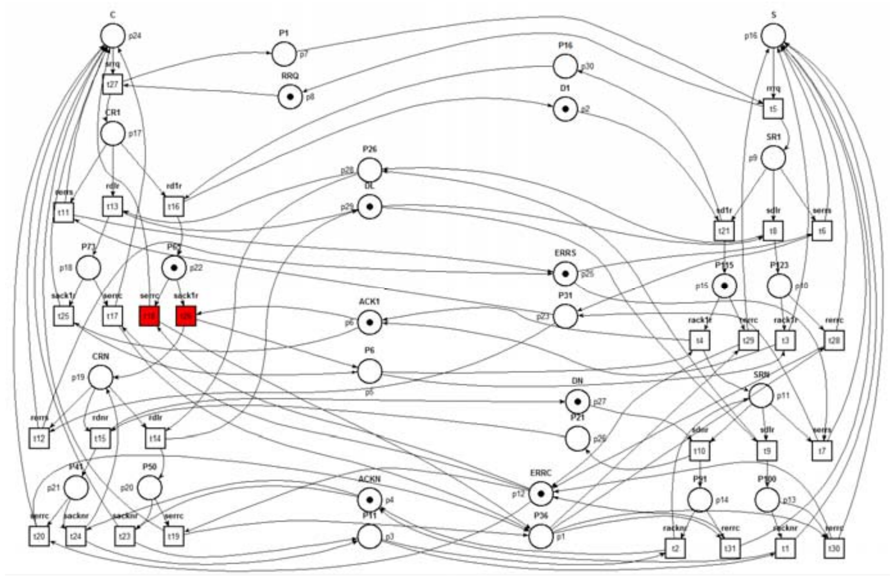

Figure 3

Studying the dynamic behavior using the Tina [24] stepper simulator

Table 2

Invariants of places

p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}	p_{13}	p_{14}	p_{15}	p_{16}	p_{17}	p_{18}	p_{19}	p_{20}	p_{21}	p_{22}	p_{23}	p_{24}	p_{25}	p_{26}	p_{27}	p_{28}	p_{29}	p_{30}
P36	*D1*	*P11*	*ACKN*	*P6*	*ACK1*	*P1*	*RRQ*	*SR1*	*P123*	*SRN*	*ERRC*	*P100*	*P91*	*P115*	*S*	*CR1*	*P73*	*CRN*	*P50*	*P41*	*P65*	*P31*	*C*	*ERRS*	*P21*	*DN*	*P26*	*DL*	*P16*
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1	1	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Moreover, Netlab provides the summary of net analysis results in textual form based on invariants and a reachability graph. Some of them are listed below in an abbreviated form.

- `Dead transitions (RG): none.`
- `Total deadlock (RG): none.`
- `Reversibility (RG, condensed): The net is reversible.`
- `Necessary conditions for invariants: There exists a non-negative T-invariant. Therefore, the necessary condition for reversibility is satisfied, and the net may be reversible.`
- `Partial deadlocks exist in the following sinks (RG, condensed): none.`

- Liveness (RG, condensed): The net is live.
- Necessary conditions for invariants: There exists a positive T-invariant. Therefore, the necessary condition for liveness is satisfied, and the net may be live.
- Boundedness (RG): The net is bounded.
- Sufficient conditions for invariants: There exists a positive P-invariant. Therefore, the sufficient condition for boundedness is satisfied, and the net is bounded.

PNtool2, as an addition to the invariant analysis, provides also the reachability analysis based on results of the research performed at the author's home institution [11, 12].

Table 3

Invariants of transitions

t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}	t_{16}	t_{17}	t_{18}	t_{19}	t_{20}	t_{21}	t_{22}	t_{23}	t_{24}	t_{25}	t_{26}	t_{27}	t_{28}	t_{29}	t_{30}
racknr	*sdnr*	*rerrs*	*rerrs*	*rdlr*	*rdlr*	*rdnr*	*rd1r*	*serrc*	*serrc*	*serrc*	*racknr*	*serrc*	*sd1r*	*sacknr*	*sacknr*	*sack1r*	*sack1r*	*srrq*	*rerrc*	*rerrc*	*rack1r*	*rerrc*	*rerrc*	*rack1r*	*rrrq*	*serrs*	*serrs*	*sdlr*	*sdlr*
0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	1	0	0	0	1	0	0	0	0	1	0	1	0	0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1	0	0	0
0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0	1	0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	0	0	0	1	0	0	1	0

Conclusions

Within this work a method for the specification and verification of communication protocols is discussed. It is based on combining process algebra and Petri nets in order to simplify both the production of protocol specification (using process algebra) and protocol analysis (using Petri nets). The Trivial FTP protocol is taken as an example of practical employment of the method proposed. In this case, the resulting Petri net has the properties (boundedness, liveness, deadlock freeness) which a correct communication protocol should have.

Our future activities will include several items to be explored. Firstly, a lot of time was spent editing the generated Petri net model into the shape depicted in Figure 2. We have made some attempts in the field of generating the layout of Petri net models, but there is still a lot of space to improve. Another idea is to incorporate the notion of time into the process of transformation. This would lead to an update of the transformation method (and the ACP2Petri tool implementing it) with the support of corresponding timed process algebra and Petri net formalisms. And finally, we plan to specify some other protocols to recognize the potential strengths and limits of the method proposed above and compare it more deeply with other approaches.

Acknowledgement

This work is the result of the project implementation: Development of the Center of Information and Communication Technologies for Knowledge Systems (ITMS project code: 26220120030) supported by the Research & Development Operational Program funded by the ERDF.

References

[1] Baeten, J. C. M., Weijland, W. P.: Process Algebra, Cambridge University Press, 1990

[2] Barjaktarovic, M., Shiu-Kai, C., Jabbour, K.: Formal Specification and Verification of Communication Protocols Using Automated Tools, Proceedings of ICECCS'95, pp. 246-253, 1995

[3] Češka, M., Marek, V., Novosad, P., Vojnar, T.: Petri nets, BUT, 2009

[4] Diaz, M.: Petri Nets: Fundamental Models, Verification and Applications, John Wiley and Sons, 2009

[5] East, I.: Computer Architecture and Organization, Pitman Publishing, 1990

[6] Edwards, J.: Process Algebras for Protocol Validation and Analysis, Proceedings of PREP 2001, pp. 1-20, Keele, England, 2001

[7] Fokkink, W.: Introduction to Process Algebra, Springer-Verlag, 2007

[8] Fürdős, F.: Verification of Communication Protocols, diploma thesis, Technical university of Košice, 2009

[9] Hillston, J.: Process Algebras for Quantitative Analysis, Proceedings of the 20th Annual IEEE Symposium on Logic in Computer Science (LICS' 05), pp. 239-248, Chicago, 2005

[10] Holzmann, G. J.: Design and Validation of Computer Protocols, Prentice-Hall, 1991

[11] Hudák, Š.: Reachability Analysis of Systems Based on Petri Nets, Elfa, Košice, 1999

[12] Hudák, Š., Zaitsev, D. A., Korečko, Š., Šimoňák, S.: mfdte/pntool – a Tool for the Rigorous Design, Analysis and Development of Concurrent and Time-critical Systems, Acta Electrotechnica et Informatica, Vol. 7, No. 4, 2007

[13] Lanet, J. L.: Using the B Method to Model Protocols, AFADL98 (LISI/ENSMA), pp. 79-90

[14] Murata, T.: Petri-Nets: Properties, Analysis and Applications, Proceedings of the IEEE, 77(4), 1989

[15] Oberkampf, W. L., Trucano, T. G., Hirsch, C.: Verification, Validation, and Predictive Capability in Computational Engineering and Physics,

Foundations for Verification and Validation in the 21st Century Workshop, Hopkins University, Maryland, 2002

[16] Petri net tool Netlab, available at: http://www.irt.rwth-aachen.de/en/fuer-studierende/downloads/petri-net-tool-netlab

[17] Sharp, R.: Principles of Protocol Design, Springer-Verlag, 2008

[18] Sollins, K.: The TFTP Protocol, 1992, available at: http://tools.ietf.org/html/rfc1350

[19] Szádeczky, T.: Problems of Digital Sustainability, Acta Polytechnica Hungarica, Vol. 7, No. 3, 2010

[20] Šimoňák, S.: Formal Methods Integration Based on Petri nets and Process algebra Transformations, PhD thesis, Technical University of Košice, 2003

[21] Šimoňák, S., Hudák, Š., Korečko, Š.: ACP2Petri: a Tool for FDT Integration Support, Proceedings of EMES'05, pp. 122-127, 2005

[22] Šimoňák, S., Hudák, Š., Korečko, Š.: Protocol Specification and Verification Using Process Algebra and Petri Nets, Proceedings of CSSim 2009, pp. 110-114

[23] Šimoňák, S., Peťko, I.: PATool – A Tool for Design and Analysis of Discrete Systems Using Process Algebras with FDT Integration Support, Acta Electrotechnica et Informatica, Vol. 10, No. 1, 2010, pp. 59-67

[24] TINA (Time Petri Net Analyzer) home, available at: http://homepages.laas.fr/bernard/tina/description.php

[25] Tomášek, M.: Language for a Distributed System of Mobile Agents, Acta Polytechnica Hungarica, Vol. 8, No. 2, 2011

[26] Verification and Validation, available at: http://en.wikipedia.org/wiki/Verification_and_validation

[27] Zaitsev, D. A., Zaitsev, I. D.: Verification of Ethernet Protocols via Parametric Composition of Petri Net, 12th IFAC Symposium on Information Control Problems in Manufacturing, pp. 122-127, 2006