EXTERNAL CONFORMANCE REQUIREMENTS: CONCEPTS, METHODS AND TOOLS

Rafał Artych
Krzysztof M. Brzeziński
Institute of Telecommunications
Warsaw University of Technology
{ rartych,kb } @tele.pw.edu.pl

Abstract In this paper we propose to re-use and extend the elements of standard conformance testing methodology, in order to cover particular needs of a telecommunications Network Operator. In this way we intend to contribute to the “industrial relevance” of formal techniques. We present the idea and formal notation of external requirements imposed by a Network Operator on protocol implementations. We also report on the support tool ORB.

Keywords: Validation, protocols, requirements, conformance testing, PICS, ORB

1. INTRODUCTION

In order to achieve (preserve, prove) correctness in particular phases of system’s life-cycle, various theories, methods and processes (applications of methods) have been researched and applied. Some of them, like conformance testing, have become state-of-the-art. However, the applicability of individual methods and their acceptable level of formality depend on the context. What is obvious in a university environment, may be rejected as unacceptable by a commercial network operator.

This problem has been recognized and analyzed, mostly in relation to the testing phase of the system life-cycle [1, 2]. Accordingly, after the initial emphasis on fundamental theories (such as test generation algorithms, testing architectures, defining and formalizing the TTCN language), the interest of researchers has visibly turned to making the available methodology more

industrially relevant: acceptable and actually used. This is also the general framework of our work, which concentrates on the role of techniques related to protocol conformance testing in the activities of a Network Operator.

We are not aiming at making any particular subset of conformance testing techniques more industrially relevant *per se*. Instead, we propose to apply a “leverage strategy”, which can be summarized as follows:

- identify an informal element within the system life-cycle: a weak link in the “formality chain” that makes other, well developed formal techniques relatively unattractive to the industrial user;
- formalize this element of the life-cycle, and develop the appropriate tools that would support and automate its execution;
- respect, as far as possible, the well established procedures of the user and their need for flexibility (which also pertains to the formal, or at least organized treatment of informal elements).

Our conjecture is that, when a weak link is formalized, the user will see sense in actively using other, already developed formal techniques for other parts of the process, because their use will then become natural.

We concentrate on a particular example of this general strategy. We consider the system life-cycle as seen from the perspective of a commercial Operator of a public telecommunications network, pursuing its own commercial goals on a deregulated, competitive market. This life-cycle differs from the usually considered design and implementation-oriented process [3], because the Operator does not design or implement the system and does not place the equipment on the market for sale. Formal techniques are available for stating the abstract properties of a telecommunications protocol and for establishing conformance of a given implementation to a standardized definition of this protocol. However, to demand “conformance with ETSI 300xxx” is grossly insufficient for the planning and construction of a particular telecommunications network. For contract procedures it is necessary to specify which protocols are to be implemented, which protocol standards are to be followed and which options are to be chosen in a particular piece of equipment, according to its planned role and in line with the Operator’s strategy of growth and service deployment.

The common practice of the Operator has been either to refrain from stating their own requirements (“conformance with protocol standards and national requirements”), or to produce lengthy, verbose textual documents under the common name of Technical Requirements, which were in an obscure relation to international standards and to each other. Both solutions were obviously inadequate. Therefore, we will treat the equipment procurement phase as a “weak point” mentioned before. Any mistakes or misunderstandings in this

phase can have serious consequences for interoperability of equipment and the overall quality of service.

In this paper we make the following contributions: we introduce the notion of *Operator Requirements* (ORs), identify their relation to requirements inherent in a specification of a protocol, formalize both the notation of ORs and the process of their design and use, describe the support tool ORB, and report on the initial experience with introducing the formalized approach into the activities of a real Network Operator.

We draw from the generic techniques related to conformance testing, as described in ISO 9646 [4] ETS300406, ETR212 and [5]. Our technique of expressing the ORs is similar to a protocol profile. Note that we are not dealing with conformance testing itself; in particular, we are not interested in whether conformance tests are actually conducted or how the test suite has been derived.

The need to formalize and automate other parts of industrial processes related to testing has been identified and addressed previously, e.g. in the context of ATS management [6]. To our knowledge, the particular set of concepts presented in this paper has not been considered before, apart from our own preliminary attempt [7].

The rest of this paper is organized as follows. In section 2 we identify two different classes of requirements: integral and external, and discuss the particular needs of a Network Operator. In section 3 we introduce the proposed structure and notation of Operator Requirements (ORs). Section 4 illustrates how the structure of ORs can be applied to generic problems. In section 5 we briefly discuss the design cycle of ORs. Section 6 is devoted to the presentation of the ORB tool, developed to support and automate this design cycle. Section 7 concludes the paper.

2. TYPES OF REQUIREMENTS

Two notions (models) of conformance prevail: more theoretical [8, 9], based on implementation relations, and more practical (pragmatic) [4], based on a collection of conformance requirements that will have to be satisfied by an implementation. We adopt the latter model, which is the basis of “industrial” conformance testing, as described in ISO 9646. Within a system life-cycle an inherent discontinuity point can be identified, in which the system under development changes its nature (fig. 1): from abstract object (*S* – specification) to real equipment (*I* – implementation). Due to this discontinuity point, the organized empirical observation (testing) is considered as the only relevant method of revealing whether the implemented system is “correct”. However, “the notion of validity of an implementation w.r.t. a specification is not expressed in the specification itself” [9]. Similarly, one could say that requirements (which define the link between *S* and *I*) are fundamentally different from the specifi-

cation (which is abstract and should not refer to the implementation).

Figure 1 Requirements in system design cycle

Let us introduce two kinds of requirements: *integral* (R_i in fig. 1) and *external* ($R_a, R_b, \dots$). Integral requirements are inherently related to the specification of a protocol. These requirements are considered in the methodology of conformance testing. Two types of integral requirements are distinguished: *static* and *dynamic* (fig. 2). A Static Conformance Requirement is defined as “one of the requirements that specify the limitations on the combinations of implemented capabilities...”[8]. Static requirements are reflected in an ICS proforma (Implementation Conformance Statement), which is a part of a standard. Technically, static requirements are modal qualifiers that assign a Status to every capability/feature of a protocol (M—Mandatory to be implemented, O—Optional, X—prohibited from being implemented, etc.). To state the implemented capabilities, the Supplier has to fill in the Support column in the proforma (this is the prerequisite for test case selection and subsequent testing). Dynamic conformance requirements specify the permitted observable behaviour within the implemented capabilities, and cannot be easily extracted (separated) from the specification.

Figure 2 Relations between requirements and specification

Standard specifications, with their associated integral requirements (e.g. ETSI standards), do not contain any self-reference, or obligation that they should be applied in a given context. The implementation options allowed by

static requirements are fixed for a given protocol, irrespective of whether a particular choice makes sense or is desired in a given piece of equipment.

To address the *need* to conform, external requirements are also necessary. External requirements are disjoint from the integral ones and cannot be deduced from a specification. They may be imposed by different organizations and for various reasons (e.g. international or national normative requirements, private user requirements). Let us consider two examples of external requirements.

Normative (regulatory) international requirements w.r.t. the terminal equipment, and terminal equipment only (TBR documents), contain clauses such as: “We do not require that integral requirement X be fulfilled (in fact we don’t care), because it is not essential from our point of view”. Such statement does not mean that the integral requirement in question becomes in any way invalid, but does indicate that it is inappropriate for regulatory use and will not be further investigated [5].

The most concise contents of private requirements of a Network Operator could state: “It is mandatory for system X, intended for our network, to implement the protocol standard Y (i.e. to fulfil all its integral requirements), AND to fulfil all normative requirements”. However, such briefness will almost certainly not match the real needs of the Operator.

In general, external requirements may be less restrictive or more restrictive than integral requirements, or may pertain to other properties of the system, not covered by integral requirements (e.g. when protocol X, and not Y, is required to be implemented). It is also generally true that the behaviour of a protocol within its individual capabilities (i.e. the dynamic requirements) will not be subject to any intentional changes. Therefore external requirements are essentially static requirements.

In the sequel we will concentrate on the external requirements imposed by a Network Operator (Operator Requirements – ORs). The discussed concepts may also be applicable to other kinds of external requirements. The crucial observation is that a Network Operator has the right to establish its private requirements that pertain to devices installed in its network, according to its own commercial and technical policy.

The general aims of Operator Requirements are: to impose consistency and interoperability within a particular network and at its borders, to document the technical capabilities of a network as a whole and individual systems within this network, and as a template for technical clauses in contracts (in order to make them more effective). The Operator is bound by legal provisions and will not install in its network any devices that are contrary to normative regulations. This also usually means that ORs cannot contradict the integral requirements. Therefore, the main question is: how to ensure that resulting ORs are both internally consistent and compatible with other valid requirements (integral and regulatory). Obviously, it seems expedient to take advantage of as many

elements of the standardized methodology as possible, such as: specifying (or rather reflecting) a set of conformance requirements in an ICS-like document, formulating a profile by introducing Requirement Lists (RL) that change the status of protocol features in ICS documents, identifying and concentrating on essential external requirements [5], and using the methodology of conformance testing: existing test suites, existing test instruments and existing facilities at test houses (the ORs will simply result in a particular parameterization of a test suite).

The main idea of ORs is to change the status of chosen protocol capabilities (features), while leaving the status of other features as defined in a base specification. The change is done in reference to the standard ICS documents (PICS) that are part of the standard. For example, it will usually be necessary to state that it is mandatory for the implementation to support both the incoming call and the outgoing call capability, even if this does not follow from any standard (fig. 3). The technique that we intend to use is not new and basically

Figure 3 Example of external requirements imposed upon integral ones

amounts to specifying a *profile*. However, the notion of a profile will have to be extended. For example, the Operator will need to state that given services shall be supported as well. Also note that any implementation status of any feature is meaningful only if a protocol itself is required by the Operator to be implemented. In this way the ORs can be very concise: they do not repeat, rephrase or interpret the contents of a standard protocol specification (unlike formerly used, narrative "Technical Requirements"), but directly identify the differences.

3. STRUCTURE AND NOTATION

ORs are constructed from a set of tables of three types: DP (Declaration of Parameters), RL (Requirements List) and CS (Conformance Statement). The structure of CS and RL tables had originally been defined elsewhere [4], but for the purpose of ORs was suitably modified. DP tables are used to specify the status of items defined by the Operator, and to document the support of these items (or commitment to these items) by the Operator. CS tables serve to specify

the status of items defined by the Operator, and to capture the support answers given by an external party (such as an equipment supplier). RL tables are used to change the original status of an item defined in a protocol specification to another status, indicated by the Operator.

The entries in the tables are linked to form a hierarchical structure. The structure itself is defined for each application (instance) of ORs. The rules of the correct linking of objects within the tables are referred to as *Requirements Calculus*. The structure of tables is illustrated below. Note that the structure of DP and CS tables is identical (ICS type), but the use of these tables differs.

ICS(DP/CS):

⟨header⟩					
Item	⟨Item description⟩	Reference	Conditions	Operator's status	Status

RL:

⟨header⟩						
No.	Item	⟨Item description⟩	Status	Conditions	Operator's status	Ref.

Before going into more details, we present a tiny sample of real ORs:

Table CS28. HOLD - general					
References to [141-1], unless stated otherwise					
Item	General service aspects	Reference	Conditions for status	Op's status	Status
1	not sending notifications to the remote user	[141-1]9.2.1	DP1/2 else	x o	

Table RL22. Requirements on items used in HOLD PICS [141-2]						
No.	Item	Feature with modified status	Status	Conditions for status	Op's status	Ref.
5	A1/R3.2	support requirements for interworking with private ISDNs	o	not CS28/1 and DP1/4 else	m o	

The values of Status fields belong to a set

$STAT = \{i, m, o, oe.i, om.i, x, na, nr, c\}$, where:

i irrelevant (out of scope)

m mandatory

o optional

oe.i optional: exactly one item needs to be implemented

om.i optional: one or more items need to be implemented

x excluded (implementation prohibited)

na not applicable

nr not a requirement

c (only in RL tables) any previous status (e.g. conditional)

Support fields in DP and CS tables may assume values from a set $SUPP = \{\text{Yes}, \text{No}, -, \emptyset\}$. Value ‘-’ denotes “no support answer required”. Value “ $\emptyset$ ” (empty field) is assumed before any support answer is given. This value,

if used in condition expressions, is treated in a special way (note that the Requirements Calculus involves multi-valued logic).

The actual status is calculated by evaluating predicates and conditions. A predicate is associated with a table. If a predicate evaluates to FALSE, then all the status values in the table assume value 'i' (in case of RL tables 'i' is interpreted as "ignore any change in the original status"). A single item (entry) in a table may have multiple status values, associated with multiple condition expressions in a multi-line arrangement (see the mini-example of real ORs). Subsequent expressions are evaluated from top to bottom (TTCN-like), and the status associated with a TRUE expression becomes the current status. If all expressions evaluate to FALSE, the default 'na' status is applied. Both predicates and conditions are logical expressions built of support answers (Yes=TRUE, No=FALSE). The support answers are identified by reference ($ref \in REF$) to a support field in one of the existing tables within the ORs (reference by $\langle \text{table_identifier} \rangle / \langle \text{item_number} \rangle$, e.g. DP1/2) or to a support field in an external PICS table (e.g. [092-2]A1/R1 refers to a specific support field in ETS300092-2). To find a support value, the function $supp : REF \rightarrow SUPP$ is applied.

Function $stat_check : STAT \times STAT \rightarrow \{TRUE, FALSE\}$ controls the correctness (validity) of status changes specified in RL tables. The value $stat_check(s1, s2) = TRUE$ indicates that a change from the original status $s1$ to the new status $s2$ is allowed w.r.t. integrity and consistency of ORs. The current status of an item determines the correct values of a support answer, according to function $supp_check : STAT \times SUPP \rightarrow \{TRUE, FALSE\}$. The answer is acceptable only if $supp_check$ evaluates to TRUE.

We use the following definition of both functions (interesting results can be obtained by tuning this definition):

	<i>stat_check</i>						<i>supp_check</i>		
	Operators's status:						support answer:		
base status	na	nr	m	o	x	i	Y	N	—
na	T	F	F	F	F	F	F	F	T
nr	T	T	F	T	T	T	T	T	T
m	F	F	T	F	F	F	T	F	F
o	F	F	T	T	T	F	T	T	F
x	F	F	F	F	T	F	F	T	F
c	T	F	T	T	T	F	-	-	-
i	T	T	T	T	T	T	T	T	T

4. USE PATTERNS

DP and CS tables. ORs may be used for two purposes:

(a) as a general declaration (specification) of technical and service-related properties of a network;

(b) as a specification of technical requirements imposed on a particular piece of equipment.

For each type of application the DP tables, which store the parameters of ORs, are used in a different way. DP tables, which are always filled by the Operator, can be fully completed (to indicate the current state of the network and its global properties) or partially completed, with the Support column left blank — this column will be completed (still by the Operator) in order to characterize the required type of equipment.

In application (a), it is assumed that the blank Support fields in DP tables have, or may have, value ‘Y’. This means that the reader of ORs may expect a particular feature/capability to appear somewhere in the Operator’s network, but not necessarily in every piece of equipment. Application (b) corresponds to contract procedures: the Operator has to fully identify the type of equipment required. In this application status ‘o’ in a DP table means that the Operator has freedom to point to a particular capability as required or not required, while the support value ‘Y’ means that the Operator does consider the capability as required.

Figure 4 Indirect parameterization

In this use pattern, each DP table is accompanied by a similar CS table, as shown in fig. 4. The Operator could state its requirements directly in the Status field of a CS table. However, the use pattern described above allows the Operator to benefit from the same semantic support mechanisms that also serve the Supplier, who fills the Support fields of CS tables.

Phased introduction of a protocol. Another use pattern addresses the tricky problem of the phased introduction of capabilities into the network. Consider the following fragment of ORs:

DP:	condition	Op.status	Support
phase_1	—	i	Y
phase_2	—	i	N
CS:	condition	Op.status	Support
feature_a	phase_2	M	
	phase_1	O	
RL:	status	condition	Op.status
feature_b	M	phase_1	NR
feature_c	O	phase_2	M

This example shows how informal, non-protocol items (such as the identification of a phase of network development) can be formally accommodated in ORs. Feature_a, defined by the Operator, becomes mandatory to be implemented in phase 2. In the earlier phase it remains optional (note the order in which condition expressions are evaluated). Feature_b, which is mandatory according to integral requirements, in the initial phase of network development becomes “not required”. Feature_c will become mandatory later.

5. ORS AS A PROCESS

ORs, as an entity, must undergo a full design cycle under the control of the Operator. This design cycle is orthogonal to the design cycle of the system (compare fig. 1). The OR design cycle has its own peculiarities, but there is no reason why it should not follow the well known general principles. The simplified, waterfall-like model of this life-cycle is outlined in fig. 5.

Figure 5 The life-cycle of ORs

The outcome of the OR design cycle can be directly used to parameterize the test suite in the process of conformance testing w.r.t. the ORs, which is a non-standard application of the standard methodology.

6. SUPPORT TOOL — THE ORB SYSTEM

Our experience shows that in most practical cases ORs are complex. Control over their correctness is a nontrivial task, which calls for a computer support tool. Therefore, the concepts presented above have been embodied in the ORB (Operator Requirements Backbone) support system. Technically, ORB is a database application implemented in Borland Delphi with BDE (Borland Database Engine), running under MS Windows 95/98. This simple execution platform is well suited for the job, and is readily available to the Operator.

Static structure of ORs. According to the declaration made in the Introduction, ORs handled by ORB should closely follow the metaphor of a familiar technical document. A single OR document handled by ORB, called a project, is a sequence of chapters. Each chapter consists of its title, a sequence of textual notes, and an arbitrary number of DP, CS and RL tables. Chapters form a tree (i.e. a chapter may contain an arbitrary number of subchapters, which are chapters themselves):

```

ORB-project := { Chapter }+
Chapter := Title { Chapter-element }*
Chapter_element := Text_note | Table | Chapter
Title := text
Text_note := text
Table := DP_table | CS_table | RL_table
DP_table := ICS_table
CS_table := ICS_table

```

(‘+’ means: one or more, ‘*’ means: zero or more)

Altogether, a project is equivalent to a technical annex of a contract, an internal document of the Operator, etc. To exploit the familiar metaphor, documents produced by ORB are formatted exactly as expected by the technical personnel of the Operator. The document is also produced in a portable format (HTML), in which it can be made available for corporate-wide browsing, as well as in a format compatible with the MS Word text editor.

ORB Functionality. The waterfall model presented in fig. 5 pertained to a single project and a single thread of activity. ORB supports the realistic version of the life-cycle (fig. 6), which takes into account the multitude of projects and their versions under development. Project management functions are an important part of system functionality. Individual phases of the OR life-cycle are handled by separate modules — Actors, with the following functionality:

- **CREATOR (C):** create a generic OR document for a particular signalling protocol (create and modify the structure of ORs; edit the contents of tables; perform automatic semantic checks)
- **PARAMETERIZER (P):** prepare a requirements document for a given contract (parameterize the ORs by filling in the remaining Support fields of DP tables; check correctness)
- **DECLARATOR (D):** declare the implementation of capabilities (fill in the blank Support fields in CS tables; assign the values from standard PICS documents to “external variables” of ORB; calculate condition values from the actual Support answers)

- **VERIFIER (V):** verify the support answers (perform full Static Conformance Review; generate the Conformance Report)
- **BROWSER (B):** browse the project in any phase of its life-cycle.

Figure 6 The OR life-cycle supported by ORB

ORB in use. The ORB toolset consists in fact of two separate systems: the full ORB system (which contains all the Actors) and the ORBik system with a single actor (DECLARATOR only), intended primarily for the Equipment Supplier.

a) A designer, using the CREATOR, creates generic Operator Requirements for a given signalling protocol (or rather for a profile of signalling protocols to be used in a given signalling relation). The final version of ORs in this phase reflects a current state of the network, but is not parameterized (some Support fields in DP tables are empty). The project is then finalized (changes its phase) and can no longer be modified by its designer.

b) When a need arises to procure a piece of network equipment of a given type, a “contract officer” of the Operator prepares a parameterized version of Requirements (fills in the remaining free Support fields in the DP tables), using the PARAMETERIZER. The same project can be used to prepare multiple, differently parameterized versions (e.g. for different Suppliers). The finalized parameterized version is saved on diskette and is handed over to a Supplier, together with the ORBik system.

c) The Supplier, using the DECLARATOR module of ORBik, declares the implemented capabilities in the Support fields of the CS tables. Apart from the ORs themselves, the Supplier also fills in the standard PICS proformas, identified by the ORs. The DECLARATOR points to particular support answers in these PICS documents that have to be copied into the electronic document handled by ORBik. The finalized version of ORs is saved on diskette and handed over to the Operator, together with standard PICS documents.

d) In parallel, or as part of earlier design activities, the designer may use the DECLARATOR built into the ORB system to perform “what-if” simulations.

e) Using the VERIFIER, the “contract officer” checks the correctness of support answers. Static Conformance Review [4] is performed automatically, and its results are included in a Conformance Report.

f) The personnel of the Operator can browse any of the OR documents, using the BROWSER. The available functions are hypertext-like, which is invaluable in a large document of a technical nature.

g) To allow for real-life events, the Administrator (a user with special rights) can recover the original version of the document at any development phase.

Semantic support. ORB implements the Requirements Calculus, which allows for extensive automatic semantic checks on the structure of ORs and values given by the User. The creation of ORs is the crucial stage. For example, internal consistency of ORs requires that there shall be no circular dependencies involving condition expressions, for items in RL tables the *stat_check* function shall evaluate to TRUE for any modified value of the Status

field, and for each declared parameter the *supp_check* function shall evaluate to TRUE. The example of inconsistency in ORs is shown in fig. 7.

Figure 7 Example of circular reference

In real ORs, “multi-hop” dependencies are commonplace, and manual discovery of possible loops is a hopeless task. For this single reason, the ORB tool appears to be the enabling component of the whole approach. In other stages, the *supp_check* function is extensively used. The Static Conformance Review is successful if for each item:

$supp_check(status(item), support(item)) = TRUE$.

7. CONCLUSION

In this paper we proposed a particular “leverage strategy” for making formal methods more relevant to the industry. The application of this strategy to conformance testing in the context of a Network Operator leads to the following effects:

- the elements of conformance testing methodology used to formally state the ORs, are introduced for well understood commercial reasons;
- conformance testing w.r.t. the ORs (e.g. for internal acceptance purposes) becomes a task for a test laboratory run by the Operator;
- consequently, the Operator is compelled to acquire the methodology of conformance testing;
- similarly, the formal aspect of protocol standards (SDL diagrams, PICS proformas), which is the prerequisite of conformance testing methodology, becomes less obscure.

Introducing the formalized Operator Requirements into procedures and structures of the Operator was a methodical, step-wise, and sometimes painful process. The proposed structure of ORs was first applied “manually”, to DSS1

and INAP protocols. After a period of confusion, the improvement over the previously used informal documents was acknowledged, but the inconvenience of the paper form of ORs became evident. The introduction of ORB answered this criticism. The first real application of ORB was to handle (transcribe) the ORs for DSS1 protocols. The ORs themselves had been developed “manually”, and were available on paper. The transcription process went effortlessly. ORB almost immediately found a few logical errors (inconsistencies) in what was believed to be the final version of ORs and was about to be approved. This demonstrates non-triviality of the problem and usefulness of the tool. The Operator is currently making arrangements for field trials of the ORB system in its R&D department. In section 6 we reported on features that appear to be minor technical details (such as the printout format). However, it has become clear to us that this kind of minor details, rather than the complexity of semantic mechanisms built into the tool, is essential for winning acceptance with traditionally-minded personnel. Neither the methodology of formulating and expressing the ORs, nor the support toolset are complete. We are currently working on the Requirements Calculus, and the ORB tool is being linked with the Means of Testing (MOT), which will allow for the semi-automatic test selection.

Acknowledgments

The ORB system was designed and implemented in LTiv (Laboratory of Testing and Verification, Warsaw University of Technology) on contract from the TP S.A., the leading Polish network operator. The authors wish to thank the TP S.A. for the kind permission to publish information about the system, and anonymous referees for helpful remarks.

References

- [1] Lai, R., Leung W.(1995) Academic and industrial protocol testing - the gap and the means of convergence. *Computer Networks and ISDN Systems*, 27 (1995), pp.537-547
- [2] Lai, R. (1996) How could research on testing of communicating systems become more industrially relevant? In *Proc. IWTCs'96*, pp.3-13, Darmstadt, Germany
- [3] Making Better Standards; A guide from Technical Committee MTS. ETSI, 1996
- [4] ITU-T/ISO 9646; Conformance Testing Methodology and Framework
- [5] ETSI TCR-TR 017 MTS; Technical Basis for Regulation (TBR) specification methodology

- [6] Danet, P.-Y., Desecures, E. (1995) Management and maintenance of TTCN Abstract Test Suites. In Proc. IWPTS'95, pp.367-375, Evry, France
- [7] Brzeziński, K.M. (1998) Towards Formality of Technical Requirements for Protocols. In Proc. EWDC'98, pp.71-74, Gdańsk, Poland
- [8] ITU-T Z.500/ISO Framework on Formal Methods in Conformance Testing
- [9] Leduc, G. (1992) A framework based on implementation relations for implementing LOTOS specifications. Computer Networks and ISDN Systems, 25 (1992), pp.23-41