

On Common Meta-Linguistic Aspects of Intrusion Detection and Testing

Krzysztof M. Brzezinski

Institute of Telecommunications
Warsaw University of Technology
Nowowiejska 15/19, 00-665 Warszawa, Poland
kb@tele.pw.edu.pl

Abstract: The current research into intrusion detection (ID) makes only marginal use of results obtained by the community concerned with formal verification/validation (V&V) and formal testing understood as its particular technique. To harmonize the ideas and methods used by separate research communities, we develop a discourse space (a taxonomy), in which the linguistic problems common to testing (in particular *passive* testing) and intrusion detection are captured. The main conceptual premises and consequences of adopting this taxonomy are discussed.

Keywords: intrusion detection, verification, testing, language, formal methods, TTCN

1 Introduction

We deal with distributed reactive systems, such as telecommunications networks. The generic problem of their design and operation is how to achieve (assure by design, check, impose at run-time, etc.) their *correct behaviour*. In this broad area, particular research communities develop their own methodology and terminology. The community rooted in traditional telecommunications networks with their standardized signalling protocols has strongly contributed to the development of formal methods and industry-strength tools, e.g., for rigorous testing [1]. The Internet community is guided by "rough consensus and working code", and is generally more inclined to adopt ad-hoc solutions based on programming practice. Each community tackles common problems in their own way, without seeking to adopt the results of the other, and even, due to the "confusion of tongues", without being fully aware that these *are* common problems. This leads to chaos and waste of resources (duplication of research efforts).

A step toward raising mutual awareness and harmonizing the approaches followed by both communities was made in [2], where *intrusion detection* (ID), an important issue in "new" telecommunications, was presented in the context of verification / validation. For a real system under examination (a *realization* - a hardware/software artefact), the appropriate verification technique is *testing*. Due to the pragmatics of intrusion detection, in [2] a particular testing concept and technique - *passive testing* was considered. It was shown how a standardized test language TTCN-3 (Testing and Test Control Notation [3]) can be effectively used in this doubly unconventional application: for passive testing (and not for *active* testing, for which this language was originally developed), and

applied to intrusion detection (which was not originally envisaged as one of its possible application areas). TTCN-3 was used as a high-level programming language for an *Intrusion Detection System* (IDS), and its commercial development environment provided an execution platform for this IDS. The idea seemed to serve well both communities, by extending the current scope of use of TTCN-3 (which complements other such attempts undertaken by the testing community), and by building awareness of the availability of a standardized linguistic solution that is "almost ready" to be used by the intrusion detection community¹.

The experiments showed the technical viability of the approach, but failed to provide the clear understanding of *what* was actually checked (tested), and *how* the applied testing technique and tool related to the ideas and elements present in other intrusion detection systems. A suitable taxonomy, or (more modestly) a *discourse space*, was thus needed. Superficially, the obvious candidate might be the popular, and often blindly cited, classification of ID techniques into *misuse detection*, *anomaly detection*, and a *specification-based* approach. However, in our opinion it is too arbitrary, non-orthogonal and terminologically confusing to be convincing. Instead, we generalize our previous work on testing and intrusion detection, by defining a discourse space that *joins* and *harmonizes* the concepts of both these research areas, in the context of a more general framework (or rather *through* this framework). It is important to stress that this unifying framework was meant to be "tight", i.e., placed at the immediately preceding level of generality, so that its concepts and terminology remain "close enough" to be acceptable and convincing. Obviously, a "too generic" common framework, like a set-theoretic setting, would be virtually useless in this unifying role - it is already present, and actually applied, with apparently no unifying effect. The construction of the proposed taxonomy and the accompanying discussion shows that *formal verification / validation* of the behaviour of a distributed system is such a viable unifying framework for both testing and intrusion detection.

The proposed discourse space is centered around the linguistic issues: it classifies the ways of expressing and reasoning about "interesting" (here - malicious) behaviours. It may thus be described as being placed at a *meta-linguistic* level.

¹Indeed, the presentation of [2] before the ID audience revealed a very apparent lack of such awareness, even if at least some members of the audience were found to have heard of TTCN, or have even used it for its primary, intended purpose.

The individual dimensions of this space will also be briefly related to the constructs and pragmatics of use of the TTCN-3 testing language; this aspect is directly complementary to [2].

This work is a thoroughly re-structured and extended sequel to [4]. The rest of the paper is organized as follows. In section 2 we provide a “philosophical” motivation for what might be otherwise interpreted as the abuse of terms and concepts. In section 3 we survey three disciplines: generic verification and validation (as a unifying conceptual level), testing, and intrusion detection. We also show how the intrusion detection setting can be described using the suitably interpreted (translated) testing terms. Section 4 identifies and describes the classes of concepts that make up the discourse space. In section 5 we discuss a particular feature of the discourse space - the passive character of an IDS. In section 6 our work is compared with other publications, and the related conceptual settings are briefly identified. Section 7 concludes the paper with a reflection on the rôle and use of a taxonomy.

2 Motivation

Freely adopting a set of basic concepts, assumptions, and a corresponding terminology, and thus a particular *language*, has always been the privilege of any scientific discipline, provided these are sound and used according to the principles of scientific investigation. It seems that the ways in which such language is actually used within a given community go beyond its nominal definition. On a philosophical note, one might think of Wittgenstein’s *language games* - patterns of use, understanding, and reacting to a certain language, which are acquired through various social processes and intuitively followed within a certain group [5]. As an illustration, consider the reaction to this author’s conjecture, presented before a prominent member of the measurement community, that testing and measurement are conceptually compatible and could be defined using a common language: “*This is not how we use the term*” [6]. Similarly, a typical reaction to the early expositions of our idea of harmonizing intrusion detection and testing was: “*Testing is about faults, while ID is about intrusions. Why should we want to treat uniformly such divergent concepts?*”. The general answer is: “*Because the main reason for regarding these concepts as divergent seems to be the differences in community-specific language games, not in the concepts themselves; because the similarities can be found and consistently described; and because it is useful to exploit these similarities.*” Indeed, the ideas presented herein have been found useful both conceptually (e.g., in teaching protocol engineering to undergraduate and postgraduate students) and technically (e.g., as a device for motivating the reuse of testing tools for intrusion detection [2]).

It should be clear that we do not intend to establish “superiority” of one discipline over the other. In particular, we do not insist that intrusion detection is a subset, or a special case, of testing. This position was unclear in [4], where, due to limited space, some shortcuts were made. In particular, to faithfully reflect our intentions, the statement: “*Intrusion detection is treated as a (passive) testing problem*” should read: “*Intrusion detection, with its assumptions and preconceptions, can be re-stated in terms of testing (esp. passive testing). This may require choosing combinations of testing concepts that might be strange and unnatural to the testing*

community, but which nevertheless remain valid and consistent.”

The aims of this work can be rephrased as characterizing a certain discipline in terms of its “inherent meaning”, rather than its conventional connotations. In this way the semantic and pragmatic extensions of the discipline, if consistent with this “inherent meaning”, can be easier to justify and accept. A discussion of such approach from an epistemological standpoint was presented in [7], where *measurement science* was characterized with respect to *generic evaluation* (in this work, we characterize both intrusion detection and testing with respect to verification / validation). [7] also cites a viewpoint of the opponents of the approach, who would have probably objected to the idea of harmonizing the testing and ID methodologies and vocabularies: “*It [i.e., measurement] is merely a word conventionally employed to denote certain ideas. To use it to denote other ideas does not broaden its meaning but destroys it: we cease to know what is to be understood by the term when we encounter it.*” However, if the “true meaning” of, e.g., intrusion detection were only conventional, then a suitable agreement (adopting a modified convention) would be sufficient for its extension.

3 Reasoning about behaviours

3.1 Verification and validation

In a formal design / development process, three classes of entities are normally distinguished: *R* - Requirements (generally - a “not necessarily executable”, abstract specification of *needs*), *S* - Specification (a description of a set of properties), and *I* - Implementation (a description of these properties that is “closer to being implemented”). Regardless of its detailed model, the development process can be abstractly perceived as a sequence of operations (transformations) on descriptions:

$$R \rightsquigarrow S^0 \rightarrow (I^1 \equiv S^1) \rightarrow (I^2 \equiv S^2) \rightarrow \dots \rightarrow I^F$$

Requirements are transformed, using domain knowledge, into an initial specification S^0 (a formal object), which is assumed to represent the intended properties of a system under development. An *n*-th development (implementation) step takes S^{n-1} , transforms it into implementation I^n , and presents this implementation as a specification S^n to the next implementation step. The process is intended to be finite: after the last step a *final* implementation I^F is obtained. This distinguished implementation, also referred to as a *realization* [8], is a hardware/software artefact that is certainly not formal (it does not belong to the universe of formal objects). Some initial intermediate implementations, and thus specifications, will normally be formal.

The correspondence between I^n and S^{n-1} , or an implementation and its (immediate) specification, can be denoted as: (*I* correctly implements *S*). The correctness criterion in the development process generally pertains to some notion of preservation of properties. When both objects are formal, then this correspondence becomes an *implementation relation*: $I \text{ imp } S$, also denoted: $I \preceq S$. If *I* is informal, then the \preceq relation can still be defined on *S* and a *model* of *I*: $m_I \preceq S$. That such a model always exists, and can be expressed in a notation “compatible” with *S*, is an assumption².

²In formal testing this is known as the “test hypothesis” due to Bernot [8]. For other hypotheses (assumptions) of formal testing see [9].

The implementation relation can be defined in a number of ways. Normally, it is a *preorder*: a relation that is reflexive (S is its own, correct implementation) and transitive (an implementation of an implementation is a correct implementation of an initial specification), but not necessarily symmetric (the development trajectory is directional). It is important to note that a correct implementation is usually not required to be *equivalent* to its specification w.r.t. its behaviour: among the development decisions there are *reductions* (e.g., resolving nondeterminism, choosing options) and *extensions* (adding behaviours that are not specified, but useful and not contradicting any properties already stated by a specification).

Checking whether $I \preceq S$ holds for a given I and S is one of the fundamental, generic development tasks. It comes in two flavours: *verification* and *validation*:

- In verification, S (denoted S_D) is a *design specification* - it expresses properties agreed upon in the development process. S_D may be “unproductive”, e.g., if the requirements capture phase was flawed.
- In validation, S expresses any properties that reflect the “real” expectations; it must necessarily be different from the design specification (otherwise validation simply becomes verification). In this context, calling the \preceq relation “imp” sounds awkward; it might be better to change the notation to \preceq and call it “comp” (is.compatible) or “corr” (corresponds_to), although technically it will remain the same relation.

The properties that we want to consider are *behavioural* properties. Behaviour is an aspect of a system that is common to all its development phases (i.e., the behaviour of a physical system and its “perfect”, abstract model is in principle indistinguishable - it is “the same” behaviour). Let us focus on a particular class of systems exhibiting behaviour. A distributed system M is composed of a set of active, non-distributed (at a given abstraction level) entities, communicating over passive channels. This corresponds to a generic protocol model attributed to Merlin (1982), where system entities are protocol “sides”, and also to a generic model of a network system in which intrusions can take place. An *instance* of behaviour results from an *execution* (a *run*) of a system. It yields a *trace* σ - a sequence of *events* $e \in E_M$ that are *observable* in a system: $\sigma \in E_M^*$. *System behaviour* is a set of behaviour instances that M can ever manifest. The *imp* (resp. *comp*) relation is defined on systems that exhibit behaviour.

To define and assess (compare) behaviours, two general approaches are available:

- *extensional characterization*: behaviour is defined by what an external observer can perceive; two systems are behaviour-equivalent if, after an arbitrarily long observation, an observer cannot tell them apart;
- *intensional characterization*: behaviour is fully defined by the internal structure (a “generator”) of a system; a relation on behaviours can be established by comparing these structures.

Verification and validation techniques can now be divided into:

1. *manipulations* on mathematical objects (basically - an intensional approach): if both S and I are formally expressed, e.g., as automata or Labelled Transition Systems,

then a given *imp* relation can be checked using the formal trace-based methods [8]; alternatively, if the structure of I is known, and S is expressed as a set of properties (e.g., temporal logic formulae), then the task can be translated into *model checking* - checking, if I is a *model* of required properties (if I *has*, or *satisfies*, these properties);

2. *purposeful observation* of behaviour (basically - an extensional approach): observations (tests, experiments) pertain to the behaviour “itself”, and the knowledge of the internal structure of a system is not indispensable.

The applicability of these techniques depends on the development phase. Both formal manipulation (a) and testing (b) are available when I and S are still formal objects (i.e., at least during a large initial part of the development process). With a *realization* (a final implementation), which is not a formal object, option (a) becomes elusive, and testing becomes the only available V&V method³.

Treating formal manipulation and testing as two complementary techniques of verification and validation seems to be quite original. Much more popular is linking the term “verification” with formal manipulation, and treating testing as a fundamentally different (and inferior) technique, opposed to verification. This approach is being extended to intrusion detection - e.g., in [10] intrusion detection is presented as an “*alternative approach ... besides testing and verification*”. This is exactly what we propose to avoid.

3.2 Testing

The essence of testing has been succinctly presented in [11]: “*What is testing? Testing is an operational way to check the correctness of a system implementation by means of experimenting with it. Tests are applied to the implementation under test in a controlled environment, and, based on observations made during the execution of the tests, a verdict about the correct functioning of the implementation is given*”. In order to decide about correctness, it is useful to develop an understanding of *incorrectness*. A *fault* is the irregularity (a structural deficiency, a “bug”) that is present in an implementation, such as, e.g., a changed (mutated) transition in an automaton. If, and when, during the execution of a system this irregularity is exercised, the resulting visible behaviour may be different from the behaviour that would have resulted in the absence of a fault. This visibly different behaviour is a *failure*. A fault may result from a human *error* (carelessness, omission, lack of experience) or from environmental conditions (e.g., flooding an electronic device). A fault⁴ may

³This statement should be taken with caution. In some “extreme” testing techniques, the extensional approach is used to reconstruct (by automatically learning Finite State Automata) the hypothetical structure of an observed realization, so that formal manipulation techniques can be applied [41], [25]. On the other hand, “regular” formal testing is in fact a hybrid technique: the structure of I is (and remains) unknown, but the structure of S is known, and serves as a source for deriving the tests that steer further observations.

⁴Other definitions of incorrectness-related terms, quite different from ours, also exist. According to some of them [12], an error is any changed behaviour (discrepancy between the expected and actual value, result, event; deviation from normal operation) that is caused by a fault and may go unnoticed, and a failure is the effect of error(s) escalating / prop-

be *permanent* (it “sits there” for good) or *intermittent* (it “happens”).

The generic aim of testing is to *detect failures*. When a failure (i.e., a failure-symptomatic behaviour) is detected, the existence of a fault is *inferred*. An implementation that exhibited a failure is considered *faulty*, and thus incorrect. It is usually expected that tests will aid in *diagnosing faults* - that they will help to *locate*, *identify*, and ultimately - *correct* faults. Despite these expectations, testing proper (e.g., telecommunications-oriented conformance testing [13]) is not involved at all with abductive (explanatory) activities inherent in diagnosis - “Activities like *pinpointing and correcting faults are out of the scope of testing*” [14]. On the other hand, detecting a failure is a natural prerequisite for the identification of its *root cause* [12]. In [13] it is explicitly stated that a failed test only indicates the occurrence of a failure (and, by inference, the existence of a fault *somewhere* within *I*).

In the context of testing, implementation *I* is called *Implementation Under Test* (IUT), and a system of with IUT is a part (in which *I* is embedded) is referred to as *System Under Test* (SUT). Testing consists in evaluating (the behaviour of) an IUT in particular, purposely chosen circumstances (conditions). The usual paradigm of *active* testing considers this activity as consisting of three elements: (a) *stimulating* a SUT in order to place a IUT in one of the pre-defined conditions (states); (b) *observing* the behaviour of SUT in this situation; and (c) *comparing* the observation with a *model*, in order to issue a *verdict* - P (Pass), F (Fail), or I (Inconclusive). In testing parlance, the “pre-defined conditions” are related to *test purposes* [15]. Defining a finite and effective set of test purposes for a given model and a given kind of tests is one of the fundamental issues in testing theory and practice.

In *passive testing*, the test system does not *force* an IUT towards achieving test purposes - instead, it waits for an IUT to “place itself” in one of the conditions, in which it is possible to issue a verdict (or: where the behaviour would be symptomatic). It is thus inherently *non-invasive*, or *non-intrusive*. This framework effectively harmonizes the concepts of active and passive testing, although passive testing is still not universally recognized as a valid testing technique (note that the definition of testing recalled earlier clearly pertains to active, and *only* active, tests). Passive testing is fundamentally different from *generic monitoring* [16] in that it automatically issues verdicts according to clearly defined criteria that specify what is, and what is not, “correct”.

In practical testing, the observations should be effective. For active testing, this amounts to generating a finite set of test cases (a test suite) that would be, under certain assumptions, *complete*, i.e., both *sound* (all the correct implementations are accepted; if Fail is issued, then *I* is certainly incorrect) and *exhaustive* (all the incorrect implementations are rejected; if an implementation passes the tests, then it is certainly correct). It is easy to see that a sound test suite may issue *false positive verdicts* (and thus accept some incorrect implementations), and an exhaustive test suite may issue *false negative verdicts* (and thus flag some correct implementations as faulty). A complete test suite, which fully characterizes the sets of correct and faulty implementations, is usually too expensive both to design and run - in testing, obtaining a false

agating to the point that a mission (a service) of a system is compromised.

verdict is a calculated risk. Note that avoiding false verdicts is a hot research topic in intrusion detection, while in testing it has been researched for at least the last 25 years.

Conducting active tests is a finite activity - after all the tests have been run, the verdicts are assumed to hold, while the implementation may mutate immediately, or may start to display the behaviour that was “hidden” during the test campaign. Passive tests, which are by definition non-intrusive, may be performed continuously⁵, and are always ready to detect new failures. This seems to be the adequate setting for detecting failures that result from intrusions (attacks).

3.3 Intrusion detection

Intrusions (attacks) are undesired, hostile, malicious, destructive (in general - *insecure*, and thus *incorrect*) behaviours of an information system. We deal with the problem of *detecting* these behaviours using an Intrusion Detection System. For completeness, and as a reference point, we will briefly recount the basic elements of the current IDS methodology. The approaches to the construction of an IDS, tightly linked to the main intrusion detection *paradigms*, are usually divided into [17], [10], [18], [19]:

- *misuse detection* (a.k.a. *signature-based*, or *knowledge-based* approach), in which an IDS looks for patterns of behaviour that correspond to pre-specified attack signatures;
- *anomaly detection* (a.k.a. *behaviour-based* approach), in which an IDS looks for deviations from the “normal” behaviour (or traffic profile), usually described by statistical measures; what is to be considered “normal” is established autonomously in the learning phase, and the decision on flagging an intrusion is taken basing on similarity coefficients / metrics;
- *specification-based* approach, in which an IDS detects deviations from a pre-defined correct (legitimate) logical behaviour.

With regard to the topological relation to a system under observation and the source of processed data, intrusion detection systems are broadly classified as: *host-based* (HIDS), in which events that constitute the observed behaviour come directly from the active component(s) of a system, and *network-based* (NIDS), in which the behaviour of a system is inferred from the messages observed in the network. Other, less convincing expositions of the NIDS/HIDS dichotomy also exist. According to [20], a NIDS “is responsible for the entire network”, while a HIDS is only “responsible for a host on which it resides”.

The classification recounted above, although well established, raises many questions concerning its conceptual and terminological aspects:

- All these approaches are essentially *behaviour-based* (i.e., they all consist in capturing and assessing a visible behaviour of a system). Giving this name to just one of them is misleading.

⁵A concept that tries to bridge the differences between active and passive testing in this aspect, is *on-the-fly* active testing [11], in which the next test event (to be received or sent by the tester) is algorithmically determined during the execution of a test, depending on the previous event (which obviously could not have been determined off-line).

- An intrusion is, semantically, a misuse (with an additional pragmatic characterization concerning “bad intentions”). Again, all the approaches aim at detecting this.
- The definition of the specification-based approach implicitly contains several strong assumptions: that a specification concerns only the logical (functional) aspect of behaviour; that a specification is “positive”, i.e., it expresses only the correct behaviour; and that a specification is exhaustive, i.e., it expresses the “whole” correct behaviour. In telecommunications, specifications of protocols / services are rarely like that.
- Virtually all published expositions of the specification-based paradigm insist that the specification of correct (non-attack) behaviour be developed manually - “*derived... from other descriptions of protocols*” [19], and the relation between this attack-related specification and a formal protocol specification is quite obscure. Apparently, unlike in “traditional” telecommunications, the Internet community does not assume the pre-existence of a formal, complete protocol specification⁶. This may amplify the impression of separation between verification/testing (which uses the “normal” protocol specification) and ID (which, in this perspective, requires a special treatment).

To bring some order to numerous linguistic approaches and solutions for intrusion detection, [22] classifies the languages used in this field into: *event languages* for describing / constructing elementary “events” from the observed data (packets); *response languages* to specify actions to be taken in reaction to a detected attack, *reporting languages* to specify a convenient format for attack reports; *correlation languages* to reason about attacks at a “meta” level (i.e., when the elementary events are the attack detection reports); *exploit languages* for specifying the procedure for performing an attack (this is in fact a dual problem of intrusion detection); and *detection languages*, a.k.a. *attack languages* that provide mechanisms and abstractions for identifying the manifestation of an attack.

In the sequel we deal only with the event and detection languages, which are both required if a working IDS is to be realized. The following languages of this kind are characteristic of the Internet/ID community: EFSM automata [23], REE - Regular Expressions for Events [24], BSML - Behavioural Monitoring Specification Language [18], self-constructed Finite State Automata [25], Parallel Environment (PE) grammars [10], STATL - Attack Language for State-based Intrusion Detection [22], TDL/FDL - Trace / Filter Description Language [26], the Bro language [27], NERL - Network Event Recognition Language [28]. This variety is discouraging, and gives the impression of “re-inventing the wheel”. All these languages are being assessed and compared against each other using arbitrary arguments and subjective experiences, without a deeper reflection on *what* they really express. The taxonomy developed in the sequel is meant to serve as a basis for more organized comparisons.

⁶“Every program should come with a specification of its intended behaviour ... [but] we believe it is likely to remain unrealized for some time to come” [21].

3.4 The intuition of harmonization

We are now ready to build the intuition behind the uniform treatment of testing and intrusion detection. The general idea of this work is to treat uniformly the cases, in which the correctness of a black-box realization is to be assessed (verified / validated) on-line, empirically, in a non-intrusive way, basing on a formalized notion of correctness (i.e., not *ad-hoc*). Two cases of interest are: passive testing in general, and intrusion detection. The main rôle of general passive testing is to flag faulty implementations by detecting failures of their behaviour. For intrusion detection, these terms need be re-interpreted⁷, e.g., in the following way:

- A failure is an attack-related behaviour that is *visibly different* from the correct (attack-free) behaviour; if the effects of an intrusion do not ever transpire to the level, at which they can be perceived by observing the behaviour of an IUT, then passive testing (and any empirical verification method in general) will be futile.
- A “fault” that makes the failures happen is the intruder. He himself (or the effects of his earlier presence) is the structural element - a “mutation” - that is introduced into the implementation.
- Note that the IUT may already be faulty for other reasons. At this (operational) phase, it may still contain some lingering implementation faults, resulting from the defective development process (to be found by on-line verification) and conceptual faults, resulting from the misinterpretation of the “real” needs and environmental conditions (to be found by on-line validation, but likely to remain forever, or until the redesign of an implementation). An “intruder fault” is yet another type (or class) of faults.
- Each class of faults may (and usually does) require the use of a different *incarnation* of the testing methodology. Its particular assumptions (e.g., fault models, a testing method, a test suite) can make it “blind” to other fault classes. In this way, faults of different classes may be looked for independently⁸. For intrusion-related testing, a realization may thus be considered initially faultless (i.e., with no intruder fault).
- A fault may be intermittent, causing failures to occur apparently at random, although in some, often obscure way depending on the behaviour trajectory of a system (an “intruder fault” usually *is* intermittent). This is an acknowledged research problem in testing. For active testing (e.g., conformance testing [13]), one of the implicit test hypotheses (assumptions) is that the faults are “static”: during the test campaign a fault set is expected not to change, in order not to introduce new failure modes. As a consequence, at the start of a test campaign an IUT is either faulty or faultless. This particular variant of testing methodology does not seem to easily handle intermittent faults. However, there are other variants, like passive testing, that do.

⁷But not re-defined. This is not really necessary!

⁸E.g., logical faults and performance faults. However, it is pointless to conduct performance tests if *I* is logically faulty. Therefore the independence (orthogonality) of testing w.r.t. particular fault classes is not guaranteed. This problem is *different* from the multiple-fault case, and it has not been researched thoroughly yet.

We have shown that the intrusion detection concept can be described (interpreted) using the testing terms, without having to re-define them. This suggests that there is nothing inherently “unnatural” in our harmonization attempt.

4 The joint discourse space

Building upon the intuition gained in the previous section, we now proceed to developing a more organized set of concepts. These concepts pertain to the linguistic devices used for both testing and intrusion detection - the features of a language for describing “interesting” behaviours, and the ways of actually using these features. As a starting point, we adopt the following assumptions and conventions:

1. Intrusions are specific behaviour manifestations of a realization (a physical implementation) I .
2. The (non-intrusion-related) behaviour of I is specified in a, normally standardized, *design specification* S_D , from which I is supposed to have been derived, or implemented, in the development process.
3. Intrusion manifestations can be (directly or indirectly) captured and specified in an attack-related *reference specification* S_A , which is normally developed specifically for the purpose of intrusion detection; note that, at this point, we do not assume that S_A is actually “written” by a human.
4. Intrusion detection is a testing-based verification/validation problem of establishing whether $I \lesssim S_A$ ⁹.
5. An additional assumption is that, in the absence of intrusion, I is considered correct w.r.t. its design specification: $I \preceq S_D$.

4.1 Primary dimensions

4.1.0.1 A. The semantics of S_A A reference specification may express (specify):

1. *abnormal* (disallowed, irregular, hostile) behaviour;
2. *normal* (allowed, intended, correct) behaviour.

A reference specification that states the correct behaviour may be understood to indirectly specify incorrect behaviour (*any* behaviour that is not directly specified). However, the concept and mechanism of detection can be in each case quite different, due to the expected asymmetry of those two kinds of behaviours.

[29] provides a detailed, although highly theoretic and general analysis of both schemes that are termed *negative detection* and *positive detection*, respectively.

It is also conceivable that S_A is hybrid in that it specifies both abnormal and normal behaviour.

4.1.0.2 B. The logical aim The direction of reasoning (a verification / testing paradigm) can be [30]:

1. to *falsify* (refute) a hypothesis: $I \lesssim S_A$;
2. to *verify* (confirm) this hypothesis.

⁹For brevity, we resort to a certain informality; to be more precise, we should have employed a formal *model* of I : $m_I \lesssim S_A$.

In testing, a working hypothesis is that behaviour of I *corresponds to* that expressed by S_A , or, alternatively, that I is correct w.r.t. S_A by the \lesssim relation. Note that, if a specification represents a disallowed behaviour, then I is “correct” if it is found to indeed misbehave. In the tradition of the model-checking community, this statement could be dually formulated as: “ S_A is a correct model of I ”. This alternative formulation fits the setting of natural sciences (where a model is constructed so as to explain natural phenomena, and can serve as a theory until empirically falsified - it is *the model* that is being falsified or verified). However, it is counter-intuitive in the area of engineering, where artefacts are developed *from* models and judged *against* the models, which requires that at some point in the development process a model be assumed correct by definition.

The hypothesis can be falsified or confirmed (verified). The conceptual validity of these operations has been discussed by philosophers. One of the viewpoints, advocated, e.g., by Karl Popper, is (very briefly) that a hypothesis (or a tentative theory) can serve as a theory until it is empirically falsified: falsification is logically decisive, while no amount of positive outcomes of experiments (or tests) can conclusively prove the theory. This position has seeped into the testing theory: it is often said of testing that its aim is to reveal incorrectness (and thus falsify a theory), and not to prove correctness of an implementation. Similarities can also be found in the theory of temporal logic of behavioural properties, where division is made into safety properties (those that are “finitely refutable”), and liveness properties that hold iff, despite any possible finite failure, there is always a (possibly infinite) continuation of behaviour such that the formula defining a property is satisfied. Intuitively, the safety properties are “testable”, while the liveness properties are not. Indeed, it has been shown in [31] that passively testable properties (i.e., those for which passive testing is, in principle, complete, i.e., sound and adequate - with no false verdicts) are a subclass of safety properties.

The popular belief that testing cannot give conclusive positive results is not really constructive in practice. Under certain hypotheses (assumptions as to how the implementation under test behaves), testing *can* be complete - if the final verdict is Pass, then it is formally proved that I is correct - this result is valid as long as the hypotheses are (believed to be) true. On the other hand, consider a specification, according to which a system is correct if it *eventually* delivers a certain result (service, behaviour pattern, etc). Obviously, if this result happens to be obtained during a test, then a Pass verdict is a conclusive proof of correctness. In other words, a hypothesis was that a system *can* behave as specified; when this behaviour is demonstrated, we are done with the verification.

Superficially it may appear that, to detect a failure, the only two viable combinations of concepts are: normal behaviour specified - refutation, and: abnormal behaviour specified - confirmation. However, when we look at the language mechanisms of, e.g., the TTCN language, then we may discover an asymmetry that makes these schemes not equivalent. In TTCN, test verdicts are not completely conventional, but rather carry a particular semantics. During the execution of a particular test case, a temporary verdict can only change from Pass through Inconclusive to Fail, not the other way

round. Similarly, the joint verdict of a test suite depends on the verdicts of individual test cases, and it changes to Fail if *at least one* test case yields a Fail. This conceptual choice is consistent with the idea of finitely *refutable behaviour* - if a failure is found, it cannot be “unfound” regardless of any further behaviour.

Note that a failure detection scheme that is based on monitoring a heartbeat stream of *confirmations of correct behaviour* seems to be a legitimate option (although, surprisingly, it is not covered at all by the standard taxonomies of ID). This discovery shows why there are two separate concepts: A and B in our taxonomy.

4.1.0.3 C. Class (aspect) of behaviour expressed by S_A A reference specification may express aspects of behaviour that are:

1. *qualitative* (functional, logical), i.e., based on the order and presence/absence of events in a trace;
2. *quantitative* (performance-related), i.e., involved with counting, timing, integrating, statistical measures, etc.

These two aspects call for different methods and approaches. For example, the TTCN language was initially designed for qualitative conformance tests. There are efforts at extending its scope to performance testing [32], but with the current semantics of the language this requires adopting certain discipline (pragmatics) for the use of language constructs [33].

4.1.0.4 D. Locality of behaviour S_A may refer to:

1. the *local* behaviour of a particular entity - a “protocol side” (e.g., “entity A receives message x and then sends message y”);
2. the *global* (joint) behaviour of a subset of implementation entities (e.g., “entity A sends message x, and then entity B sends message y”).

Recall that the assumed model of a system consists of a number of active entities, each of which is non-distributed (local, and sequential) at a given abstraction level. The joint (compound) behaviour of multiple system entities strongly depends on the properties of communication channels. Although in principle it can also be transcribed as a sequential process (implied by a global reachability graph), for testing purposes this is normally impractical due to the phenomenon of state space explosion.

4.1.0.5 E. Observability This dimension pertains to the relation between I (IUT for testing) and an IDS (a test system - TS for testing). The elements of behaviour that are expressed by S_A will have to be matched against the real events visible through a *context* - a SUT that generally impairs observations. These real events may be:

1. *communicated* to a test system (unrestricted, perfect observability); this is possible if a TS is co-located with I , and/or I is suitably instrumented;
2. *inferred* by a test system (restricted observability); a TS observes the *effects* of events executed by I (e.g., the messages present in a physical communication channel constituting a part of the context).

This aspect closely corresponds to the division into HIDS (a) and NIDS (b). In a HIDS, the causal dependencies between the events of the observed behaviour are naturally preserved, while the inferences made by a NIDS may be inaccurate or incorrect. Such observation infidelity is a specific research problem for intrusion detection, as commented in [34]: “*Indeed, the inability to counter such trace infidelities ... has been identified as a significant vulnerability of intrusion detection systems*”. An incorrect inference of a behaviour trace from inaccurate observations can result in false negatives and false positives being issued, in addition to, and independently from, such verdicts issued due to the inadequacy of criteria used for detecting an intrusion-related behaviour. The same problem has been researched in the context of testing [35].

In traditional, telecommunications-oriented conformance testing [13], it has always been accepted that a tester is a separate device that attaches to a system under test at its existing interfaces, not the interfaces explicitly prepared for the testing purposes. This is one aspect of the *black-box assumption*, according to which testing of externally visible behaviour must not rely on any additional knowledge of the structure of I or any additional functionality provided solely for the purpose of testing (another aspect is that the knowledge of the structure of I , even if available, must not be used for deriving, or generating, the tests).

Note that S_A may account for message reordering, e.g., by specifying partially ordered traces - this clearly requires some linguistic support. Let us consider the case of TTCN. Its semantics assumes FIFO queues as a model of message ports. The order of messages handed over to a test program through a particular port (point of observation) is preserved until a message is received using a TTCN operation (**receive**, **trigger**). There are two general approaches to observational infidelity in TTCN: either to program its handling directly into a test case, possibly with the aid of automatic transformations applied to the fragments of code [36] (this idea was also proposed for the NER language [37], [28]), or to delegate the task of re-constructing the “correct” trace to the peripherals of a test system (in the architecture of a TTCN test system - a SUT adapter).

TTCN provides some mechanisms to deal with the irrelevant or “wrong” order of messages awaiting reception:

- distinguishable sets of messages (e.g., coming from different sources, or “sides”) need not be interleaved on a single port - they may be presented on different ports, which avoids building up an artificial, and possibly misleading, order;
- the **interleave** statement allows for a very succinct expression of a set of nested alternatives (its semantics is actually very intricate);
- the **check** operation is a **receive** without actually removing a message from the top of a queue (e.g., to check if this message should be received “right now”).

The separation of trace analysis from trace preparation (conditioning, so that a test program is fed a reconstructed, “perfect” trace) has been investigated in [16],[38], but its possible application in the TTCN environment needs further research.

4.1.0.6 F. Relation between S_D and S_A These two specifications may be related to each other as follows:

1. *the same relative level*: S_D and S_A are of the same "kind" - they both define behaviour on the same syntactic and semantic level; as a special case, we may have $S_A = S_D$;
2. *meta-level*: S_A is placed at a different - normally "higher" - level than S_D ; it includes additional information, expectations and constraints pertaining not to the behaviour itself, but rather to *meta*-behaviour of I (the "behaviour of behaviours"), or the pragmatics of system operation.

Define an (*execution*) *policy* as a predicate \hat{W} on sets of traces [39]. A set of traces $\Sigma \subseteq E^*$ follows a policy \hat{W} iff $\hat{W}(\Sigma)$. Given a system M , if every set of traces Σ_M resulting from the execution(s) of this system follows a policy \hat{W} , then we say that system M *satisfies* this policy, write $M \models \hat{W}$. A policy is a *property* if policy satisfaction can be decided by evaluating a single predicate W separately for each individual trace. For a policy to be a property, such predicate (defined on traces, not sets of traces) must exist:

$$\exists W : E^* \rightarrow \{T, F\}. \forall \Sigma (\hat{W}(\Sigma) \Leftrightarrow \forall \sigma \in \Sigma. W(\sigma)).$$

In protocol engineering, as applied to practical telecommunications, a design specification S_D typically refers to a *single* instance of infinitely recurring behaviour. This instance corresponds to a single realization of a system's *mission*: the handling of a single call, session, transaction, flow, etc. A design specification defines, directly or indirectly, the valid traces for any single instance. The members of this set are different not only due to the usual parameterization (e.g., different addresses, payload, and time of occurrence). It is also normal for S_D to be nondeterministic; e.g., a connection may be set up or rejected due to the momentary lack of internal resources. It is tempting to assign modal qualifiers to those different traces: accepting a call is "good", while rejecting it is "bad". However, for a non-deterministic protocol specification, all the allowed traces are equally correct

Let us now return to the definition of this dimension of the discourse space. In case (a), S_A is used to capture the essence of an attack-related behaviour as a *property* of each single sub-execution. The case of $S_A = S_D$ is quite common in research on passive testing [40], [16], where any trace not accepted by a design specification indicates a failure. In general, S_A may define as attack-symptomatic (and thus inadmissible) some traces that are correct w.r.t. S_D .

In case (b), S_A may define a *policy*. This policy may state, e.g., that "too many" half-open connections are invalid, even if each one is formally correct, and any single such connection is not attack-symptomatic.

Yet another, alternative explanation would be to treat case (a) as a verification problem, and case (b) as a validation problem. An intrusion-related behaviour may be correct (successful verification against a reference of the first kind), but invalid (failed validation against the expectations expressed by a reference of the second kind). The division is not absolute and "stationary", as it is quite possible that experiences from intrusion detection activities could lead to reconsidering the contents of a design specification: over time, behaviours that were found harmful may be explicitly marked as incorrect.

4.1.0.7 G. Dynamics A reference specification S_A may be:

1. *static* - accepted as entered, and unchanged throughout the operation of an IDS / TS;
2. *evolving* - automatically modified (built, refined, adapted) by an IDS / TS during its operation.

A test suite used for typical active tests [13], or a reference specification used for mainstream passive testing [40] are examples of a static reference. A typical example of an evolving reference specification can be found in the self-learning anomaly detection approach. An interesting case of an evolving reference is when this reference is actually built (learned) from scratch, and then either stays unchanged or further evolves. In [21], a specification of a program is learned (actually - derived automatically from the known source code) and then used as a fixed logical reference, as in the specification-based approach. In this case one may argue that the phase in which a reference is built does not actually take place during the operation of an ID system. Another approach is more "clean": a reference specification (normally in the form of an automaton) may be learned from the observations of the behaviour of a system [41], [25]. In this case the already learned reference might be used for detecting failures, while a refined version of this reference is being prepared basing on further observations¹⁰.

4.1.1 Secondary dimensions

The secondary, but important dimensions of the discourse space are revealed when a generic IDS / TS is decomposed into three conceptual layers:

- a *problem-oriented domain*, in which a reference specification S_A is defined with a specific aim, here - for intrusion detection (but also, e.g., for autonomous supervision of network operation, for the arbitration of interoperability defects in multi-operator networks, etc.);
- a generic *abstract behaviour testing domain*, in which a test program TP for the evaluation of behaviour is written in some "high-level testing language" TL, such as TTCN-3 [2];
- a run-time *detection engine domain* DE.

The third tier determines the paradigm of test program execution. A test program TP may be executed according to the "standard understanding" of the execution mechanisms of programs. Another alternative is to have a "detection engine" play the role of a real-time pre-processor or filter. In this hybrid approach, a test program expressed in a test language may be transformed into a different form / paradigm (usually for the sake of run-time performance), or some functionality of a test program may be delegated to a detection engine, and thus must be expressed using the mechanisms of this engine. In practice it is a distinguishable conceptual and practical problem. For example, [42] reports on a hybrid approach in which parts of a test program are delegated to field-programmable hardware. We feel this issue should be mentioned (for completeness), as it might influence the linguistic mechanisms. However, it has not yet developed into a convincing dimension of the discourse space.

¹⁰This is actually a fresh idea, not covered in [41] and [25], which raises many questions (e.g., what if the refined version of a reference becomes "infected" by an ongoing intrusion), but may be worth further consideration.

4.1.1.1 H. Coding of a reference specification The relations between a reference specification S_A and a test program may be:

1. *direct* - S_A is expressed directly by a test program written in a test language (TL);
2. *multi-tier* (indirect) - S_A is expressed in some suitable domain-specific language DL and then transformed into a TL program.

In the direct approach (a), the specification of reference behaviour is embedded in a test program written "manually", e.g., in TTCN-3. In the multi-tier approach (b), it is prepared in another formalism, e.g., as a finite state automaton (FSM, EFSM), which is characteristic of the mainstream passive testing research. This form is then translated automatically into the TTCN-3 program, using pre-defined transformation patterns. Note that the general need for a multi-tier approach has been acknowledged by the creators of TTCN-3: a very suitable alternative notation for intrusion detection is the MSC language, and its slightly redesigned form is used as an alternative, semantically equivalent, Graphical Presentation Format (GFT) of TTCN.

This dimension of a discourse space is concerned with the generality of a detection language. The intended general domain of most "attack languages" is intrusion detection, and problem-oriented domains are understood to refer to particular intrusion classes and environments. Such languages have emerged precisely because the general-purpose programming languages were found not to be expressive enough and convenient. However, TTCN-3 is not a typical general-purpose language, although its general domain (testing) is undeniably wider than intrusion detection. For TTCN, one of the possible problem-oriented domains is intrusion detection, for which it might, or might not, be suitable. In [2] it is argued that TTCN is suitable: the majority of problems that are considered to be directly expressible in a typical detection language appear to be also directly expressible (with a similar effectiveness and elegance) in TTCN. This opinion is founded on the experience of this author. However, the suitability of a language can also be assessed in a more organized and unbiased way. For example, [28] lists the desired properties of a "network event recognition" language as the ability to: easily express protocol state machines, be compiled to executable monitors, easily handle packet formats, naturally express modularity and layering, interface with C libraries in order to reuse the already developed network monitoring functionality, easily lend itself to analysis and transformation. Similarly, [22] considers the following "desirable properties" of a detection language: simplicity, expressiveness, rigor, extensibility, executability/translatability, portability, heterogeneity. Basing on these sets of properties, both authors argue for the superiority of their own languages: NERL and STATL, respectively. The TTCN test language was not considered at all, but (as shown in [2]) it also possesses all the required properties of both sets, not as an afterthought, but as its key design features. Considering the similarity of TTCN-3 language constructs (and even, to some degree, its look-and-feel) to some other attack detection languages (such as STATL and NERL) it is surprising how little interest in its use is displayed by the ID community. This may be partly attributed to the impression that TTCN is a testing language and thus "cannot" be

appropriate for ID - the impression that we challenge in [2] (on the basis of practical experiments) and in this work (on a conceptual level).

5 Discussion

The structure of the proposed discourse space is summarized in Table 1. As an example, the taxonomy is applied to: M - misuse detection, A - anomaly detection, S - the specification-based approach, E - the experiment with TTCN reported in [2], and G - a generic (mainstream) passive testing technique [40], [16], [31].

Table 1. IDS / testing taxonomy with examples

Concept	Value	M	A	S	E	G
A: semantics of ref.	a) abnormal					
	b) normal					
B: logical aim	a) to falsify					
	b) to verify					
C: class of behaviour	a) qualitative					
	b) quantitative					
D: locality	a) local					
	b) global (joint)					
E: observability	a) communicated					
	b) inferred					
F: rel. to design spec.	a) same level					
	b) meta-level					
G: dynamics of ref.	a) static					
	b) dynamic					
H: coding of ref.	a) direct					
	b) multi-tier					
?: passive / active						

One of the consequences of taking the passive testing perspective is that currently there is no "active / passive" dimension in our taxonomy - just a placeholder. According to the "behaviour after detection" of Debar's ID taxonomy [17], a *passive* IDS merely generates an alarm (in testing - a verdict), and an *active* IDS also reacts by taking a corrective or proactive actions. Apparently, "active" refers to what happens after detection, and not to any characteristics of the detection process itself, so this division seems to be a semantic abuse of intrusion *detection* - a suitable reaction is an intrinsic feature of intrusion *prevention* systems - IPS. We approach this issue by separating the V&V aspect of intrusion detection from the context of using the detection results, as described below.

Any verification/validation activity is a part of a loop of activities that generally aim at achieving or preserving the intended properties [16]. This observation also pertains to what we consider to be the incarnations of V&V methodology - testing and intrusion detection. The dynamics of this loop vary, from an open loop (purely for documentation purposes), through a long-term development (re-design) strategy, scheduled corrective actions after failed conformance / interoperability tests or analysis of intrusion detection results, up to real-time control. The nature of this feedback loop does influence some properties of a V&V method and tool (testing, monitoring, measurement system), in particular - the extra-functional properties, such as throughput, decision delay, ability to operate on-line (using only events actually observed as they are produced).

Reaction to a "failure" is the element of the context - a loop of activities, not of ID proper. However, in principle, a "truly

active” intrusion detection method is also conceivable. A test system might actively stimulate a SUT so that an intrusion-related behaviour, *if present*, becomes ”more symptomatic” and amenable to comparison with a suitable reference¹¹. We thus reserve a place for this dimension in our discourse space.

6 Related work

Building (or finding) a common conceptual base for both testing and intrusion detection, which is the central idea of this work, remains virtually unexplored. One of very few hints in this direction can be found [28], where the authors argue that the properties to be checked should be expressed in a formal specification language (which happens to be one of the prerequisites for formal verification), and list the potential applications of ”black box network monitoring” (apparently treated as a verification technique): performance testing, *intrusion detection*, and surveillance.

Another, only slightly more popular idea is to directly ”translate” testing into intrusion detection. Two variants of this approach can be distinguished: translating the concepts, and re-using the testing tools (including the TTCN test language and programming environments) in the context of intrusion detection; a sub-variant, which goes half-way, proposes to re-use the *active* testing tools in the context of *passive* testing.

In the area of translating the concepts, [28] vaguely remarks that ”an alternative is to look at network event recognition [which is presented as a technology that potentially could be used for ID] as a testing problem...[and handle it by]... *test trace analysis* [which is roughly equivalent to passive testing]”. In [23], ”a specification-based approach ...[is claimed to be able] to detect the anomalies in the implementation of the protocol as well as attacks...”. In this approach, ”attacks are assimilated to violations of the specification”. This paper, pertaining directly to intrusion detection, was actually written by researchers well known for their work in testing, and in particular - passive testing. However, apart from these hints, the authors have not expressly indicated the relevance of their previous work to ID.

The general need for the Internet community to acquire the TTCN technology is voiced in [44]. Some elements of passive testing using TTCN are discussed in [45], where, within the active test architecture, the selected interfaces (called measurement points) are dedicated to observation only. In [46] a TTCN-based system is proposed for on-line validation of the service deployment process, and the idea of ”*abusing the [TTCN-3] notation for online testing purposes*” is discussed. The authors also suggest the possible use of such system for intrusion detection. This possibility, which was only identified as an item for further work, was later discussed and tried out in [2].

There are other conceptual settings that, in our opinion, are related to both passive testing and intrusion detection, yet this affinity apparently has not been exploited¹². If intrusion detection is to be used as an element of a ”fast” loop

¹¹This is still different from the *active probing* idea, which consists in generating a purposely chosen stimulus (a *test transaction*) to diagnose a fault [43].

¹²As an illustration, consider a taxonomy of safety monitoring techniques [47]: within 119 references, not a single one refers to IDS (yet another ”hermetic” community).

(e.g., as a functional part of an intrusion *prevention* system, and not in the context of analyzing finite historical data), then it is tempting to relate its methodology to other conceptual settings for on-line behaviour assessment, such as: reactive monitoring [48], general safety monitoring of industrial control systems [47], EM - Execution Monitoring [39], [49]¹³, DES (Discrete Event Systems): their diagnosis [50] and supervisory control [51]. Here, for completeness, we can only identify these methodologies.

7 Conclusions

There have been many attempts at building a useful taxonomy of intrusion detection concepts and techniques [17], [52], [53]. Work on a taxonomy usually indicates a certain conceptual unrest, and a need for bringing order to a complicated matter. Any good taxonomy has a descriptive and explanatory power - it allows to better understand the subject matter. Our discourse space (which is in fact a pre-taxonomy) is the result of our discontent with the level of understanding induced by the existing taxonomies. We took a fresh look at the set of basic concepts from which a taxonomy of intrusion detection might be built. We made a ”guess”: *intrusion detection is a particular incarnation of a generic verification and validation methodology*. So is testing - this (not accepted by some) standpoint is the result of our previous work on formal verification / validation and testing of communication protocols. If so, then testing and intrusion detection are ”siblings” - they are clearly grown in different environments and may look different, but they should have common genes. We have concentrated on looking for these genes, and trying to see how the problems of one discipline could have been approached by the other. Our ”guess” was a good one - it led us to a consistent picture that was presented (admittedly - mostly in an informal way) in this work. The most useful feature of this picture is the clear path to the re-use of concepts and tools. Some practical aspects have already been achieved - the developed methodological base allowed us to more convincingly present the feasibility of using a testing language and tool (TTCN-3) in the domain of intrusion detection [2].

Our discourse space is a further projection of this general picture onto the linguistic aspects of intrusion detection and testing - showing *what* should be expressed, *how* it should be expressed, and *what to do* with these statements, seems to be more concrete and close to the level at which intrusion detection systems used to be described and compared to each other.

Any good taxonomy should also have a *generative* (predictive) potential [52]. Many combinations of concepts from our discourse space apparently define new classes of techniques and tools for intrusion detection. As indicated in section 4.1 C, we have ”discovered” a new ID technique that seems not derivable from existing taxonomies.

The presented discourse space may be developed into a full, detailed taxonomy - we intend to work in this direction. When this project matures, we will re-classify more thoroughly the existing approaches to intrusion detection.

¹³Stopping or modifying an execution considered as ”insecure”, as discussed in [49], corresponds to the idea of an Intrusion Prevention System (IPS). Surprisingly, this quite obvious parallel seems to remain unexplored.

Acknowledgment

This work was supported by the Polish Government research grant N517 008 31/1429.

References

- [1] Recommendation Z.500. *Framework on formal methods in conformance testing*. ITU-T, 1997.
- [2] K.M. Brzezinski. "Intrusion Detection as Passive Testing: Linguistic Support with TTCN-3". In *Proceedings of the 4th Int. Conf. on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, pp.79-88, 2007.
- [3] ETSI ES 201 873. *Methods of Testing and Specification; The Testing and Test Control Notation version 3; Parts 1-6*.
- [4] K.M. Brzezinski. "A Joint Meta-Linguistic Taxonomy of Intrusion Detection and Testing / Verification". In *Proceedings of SIS'2007*, 2007.
- [5] L. Wittgensten. *Philosophical Investigations*, Macmillan, New York, 1965.
- [6] R. Morawski, private communication, 2007.
- [7] L. Mari. "Epistemology of Measurement". *Measurement*, 34, pp.17-30, 2003.
- [8] J. Tretmans. "A Formal Approach to Conformance Testing". *Ph.D. Thesis*, Univ. Twente, 1992.
- [9] R.M. Hierons. "Comparing Test Sets and Criteria in the Presence of Test Hypotheses and Fault Domains", *ACM Transactions on Software Engineering and Methodology*, 11 (4), pp.427-448.
- [10] C. Ko, M. Ruschitzka, K. Levitt. "Execution Monitoring of Security-critical Programs in Distributed Systems: A Specification-based Approach". In *Proceedings of the IEEE Symposium on Security and Privacy*, 1997.
- [11] J. Tretmans, A. Belinfante. "Automatic Testing with Formal Methods". In *Proceedings of the 7th European Int. Conf. on Software Testing, Analysis and Review (EuroSTAR'99)*, 1999.
- [12] M. Steinder, A.S. Sethi. "A Survey of Fault Localization Techniques in Computer Networks", *Science of Computer Programming*, 53, pp.165-194, 2004.
- [13] ISO/OSI 9646. *Conformance Testing Methodology and Framework*.
- [14] P.D.L. Machado. "Testing from Structured Algebraic Specifications: The Oracle Problem". *Ph.D. Thesis*, Univ. of Edinburgh, 2000.
- [15] Y. Ledru, L. du Bousquet, P. Bontron, O. Maury, C. Oriat, M.-L. Potet. "Test purposes: adapting the notion of specification to testing". In *Proceedings of the 16th IEEE Int. Conf. on Automated Software Engineering (ASE2001)*, 2001.
- [16] K.M. Brzezinski. "Towards Practical Passive Testing". In *Proceedings of the IASTED Int. Conf. on Parallel and Distributed Computing and Networks (PDCN'05)*, 2005.
- [17] H. Debar, M. Dacier, A. Wespi. "Towards a Taxonomy of Intrusion-Detection Systems", *Computer Networks*, 31 (8), pp.805-822, 1999.
- [18] P. Uppuluri, R. Sekar R. "Experiences with Specification-based Intrusion Detection". In *Proceedings of RAID'2001*, 2001.
- [19] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwan, H. Yang, S. Zhou. "Specification-based Anomaly Detection: a New Approach for Detecting Network Intrusions". In *Proceedings of ACM CCS'02*, 2002.
- [20] K. Labib, V.R. Vemuri. "Detecting And Visualizing Denial-of-Service and Network Probe Attacks Using Principal Component Analysis". In *Proceedings of SAR'04*, 2004.
- [21] D. Wagner, D. Dean. "Intrusion Detection via Static Analysis". In *Proceedings of the IEEE Symposium on Security and Privacy*, pp.156-169, 2001.
- [22] S.T. Eckmann, G. Vigna, R.A. Kemmerer. "STATL: An Attack Language for State-based Intrusion Detection", *Journal of Computer Security*, 10 (1/2), pp.71-104, 2002.
- [23] J.-M. Orset, B. Alcalde, A. Cavalli. "An EFSM-Based Intrusion Detection System for Ad Hoc Networks". In *Proceedings of the 3rd Int. Symposium on Automated Technology for Verification and Analysis (ATVA'05)*, 2005.
- [24] R. Sekar, P. Uppuluri. "Synthesizing Fast Intrusion Prevention/Detection Systems from High-Level Specifications". In *Proceedings of USENIX'99*, 1999.
- [25] R. Sekar, M. Bendre, D. Dhurjati. "A Fast Automaton-Based Method for Detecting Anomalous Program Behaviors". In *Proceedings of the IEEE Symposium on Security and Privacy*, 2001.
- [26] R. Hofmann, R. Klar, B. Mohr, A. Quick, M. Siegle. "Distributed Performance Monitoring: Methods, Tools, and Applications", *IEEE Transactions on Parallel and Distributed Systems*, 5 (6), pp.585-598, 1994.
- [27] V. Paxson. "Bro: a System for Detecting Network Intruders in Real-Time", *Computer Networks*, 31 (23-24), 1999.
- [28] K. Bhargavan, C. Gunter. "Requirements for a Practical Network Event Recognition Language", *Electronic Notes in Theoretical Computer Science*, 70 (4), 2002.
- [29] F. Esponda, S. Forrest, P. Helman. "A Formal Framework for Positive and Negative Detection Schemes", *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, 34 (1), pp.357-373, 2004.
- [30] L. Heerink, E. Brinksma. "Validation in Context". In *Proceedings of the IFIP 15th Int. Symposium on Protocol Specification, Testing and Verification (PSTV'95)*, pp.209-224, 1995.
- [31] A.N. Netravali, K.K. Sabnani, R. Viswanathan. "Correct Passive Testing Algorithms and Complete Fault Coverage". In *Proceedings of the IFIP 23rd Int. Conf. on Formal Techniques for Networked and Distributed Systems (FORTE'2003)*, pp.303-318, 2003.
- [32] Z. Dai. "TimedTTCN-3, a Real-time Extension for TTCN-3". In *Proceedings of the IFIP 14th Int. Conf. on Testing of Communicating Systems (TestCom'02)*, 2002.
- [33] G. Ziegler, G. Rethy. "Performance Testing with TTCN-3". In *Proceedings of the TTCN-3 User Conference*, 2006.

- [34] K. Bhargavan, C.A. Gunter. "Network Event Recognition", *Formal Methods in System Design*, 27 (3), pp.213-251, 2005.
- [35] G. v.Bochmann, O. Bellal. "Test Result analysis with respect to formal specifications". In *Proceedings of the 2nd Int. Workshop on Protocol Test Systems*, pp.272-294, 1989.
- [36] R. Castanet, P. Laurecot. "Testing Real-Time Systems". In *Proceedings of the 15th World Conference on Nondestructive Testing*, 2000.
- [37] K. Bhargavan, S. Chandra, P.J. McCann, C.A. Gunter. "What Packets May Come: Automata for Network Monitoring". In *Proceedings of the Symposium on Principles of Programming Languages*, 2001.
- [38] K.M. Brzezinski. "Self-Tuned Passive Testers for Grey-Box Distributed Systems with Indefinite Communication Delays". In *Proceedings of the IASTED Int. Conf. on Parallel and Distributed Computing and Networks (PDCN'07)*, 2007.
- [39] F.B. Schneider. "Enforceable Security Policies", *ACM Transactions on Information and System Security*, 3 (1), pp.30-50, 2000.
- [40] D. Lee, A.N. Netravali, K.K. Sabnani, B. Sugla, A. John A. "Passive testing and applications to network management". In *Proceedings of the Int. Conf. on Network Protocols (ICNP'97)*, 1997.
- [41] H. Hallal, S. Boroday, A. Ulrich, A. Petrenko. "An Automata-based Approach to Property Testing in Event Traces". In *Proceedings of the 15th Int. Conf. On Testing of Communicating Systems (TestCom'03)*, 2003.
- [42] L. Deri. "Passively Monitoring Networks at Gigabit Speeds Using Commodity Hardware and Open Source Software". In *Proceedings of the Passive and Active Measurement Conference (PAM'03)*, 2003.
- [43] I. Rish, M. Brodie, S. Ma, N. Odintsova, A. Beygelzimer, G. Grabarnik, K. Hernandez. "Adaptive Diagnosis in Distributed Systems", *IEEE Trans. On Neural Networks*, 16 (5), pp.1088-1109, 2005.
- [44] A. Sabiguero, A. Baire, A. Floch, C. Viho C. "Using TTCN-3 in the Internet Community: an Experiment with the RIPng Protocol". In *Proceedings of the 2nd TTCN-3 User Conference*, 2005.
- [45] I. Schieferdecker, B. Stepien, A. Rennoch. "PerfTTCN, a TTCN Language Extension for Performance Testing". In *Proceedings of the 10th Int. Workshop on Testing of Communicating Systems (IWTCS)*, 1997.
- [46] P.H. Deussen, G. Din, I. Schieferdecker. "A TTCN-3 Based Online Test and Validation Platform for Internet Services". In *Proceedings of the 6th Int. Symposium on Autonomous Decentralized Systems (ISADS'03)*, 2003.
- [47] Y. Papadopoulos, J. McDermid. "Automated Safety Monitoring: A Review and Classification of Methods", *International Journal of Condition Monitoring and Diagnostic Engineering Management*, 4 (4), 2001.
- [48] M. Dilman, D. Raz. "Efficient Reactive Monitoring". In *Proceedings of IEEE INFOCOM*, 2001.
- [49] L. Bauer, J. Ligatti, D. Walker. "More Enforceable Security Policies". In *Proceedings of the Foundations on Computer Security Workshop (FCS'02)*, 2002.
- [50] S. Lafortune, D. Teneketzi, M. Sampath, R. Sengupta, K. Sinnamohideen. "Failure Diagnosis of Dynamic Systems: An Approach Based on Discrete Event Systems". In *Proceedings of the American Control Conference*, pp.2058-2071, 2001.
- [51] P.J.G. Ramadge, W.M. Wonham. "The Control of Discrete Event Systems", *Proceeding of the IEEE*, 77 (1), pp.81-98, 1989.
- [52] S. Axelsson. "Intrusion Detection Systems: A Survey and Taxonomy". *Technical Report TR:99-15*, Dept. of Computer Engineering, Chalmers Univ. of Technology, 2000.
- [53] J.S. Sherif, T.G. Dearmond. "Intrusion Detection: Systems and Models". In *Proceedings of the 11th IEEE Int. Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'02)*, 2002.

Author Biography

Krzysztof M. Brzezinski is an Assistant Professor at the Institute of Telecommunications, Warsaw University of Technology, Poland. He obtained his M.Sc. (for work on switching system management) and Ph.D. (for work on protocol design) in telecommunications from the same University in 1984 and 1995, respectively. His current research interests include formal verification and testing, requirements-based network development methodology, and interoperability problems in telecommunications networks of the ISDN, IN and NGN class. Dr. Brzezinski is the author of two patents, two books, and a number of other publications.