CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering
Department of Telecommunication Engineering

Automated Test Design in Multilayer Networks

Doctoral Thesis

Author:
Ing. Andrey Shchurov

Supervisor:
Ing. Radek Maříк, CSc.

Ph.D. Programme: *P2612 Electrical Engineering and Information Technology*

Branch of study: *2601V013 Telecommunication Engineering*

Prague, July 2017

Declaration of Authorship

I, Andrey Shchurov, declare that this thesis titled *Automated Test Design in Multilayer Networks* and the work presented in it are my own and has been generated by me as the result of my own original research. I confirm that:

1. This work was done wholly or mainly while in candidature for a research degree at the Czech Technical University in Prague.
2. Where any part of this thesis has previously been submitted for a degree or any other qualification at the University or any other institution, this has been clearly stated.
3. Where I have consulted the published work of others, this is always clearly attributed.
4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
5. I have acknowledged all main sources of help.
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

Abstract

Deployment of commercial computer networks sets high requirements for procedures, tools and approaches for comprehensive testing of these networks. However, in spite of the great efforts of many researchers, the process of test design/generation still tends to be unstructured and bound to the personal experience and/or intuition of individual engineers. To address this problem, the main research objective of this thesis is the automated design of abstract test specifications (test cases) for computer networks using the detailed design documentation (end-user requirements and technical specifications) as the data source. Based on the notions of: (1) model-based testing; and (2) system methodology, this thesis covers the following main goals:

- A formal model for test generation missions based on the concept of multilayer networks. Different layers (four layers in the case of basic releases and six layers in the case of extended releases) represent different (hardware, software, social, business, etc.) aspects of system architecture.
- A test case generation strategy which covers structural test cases. Test cases of this kind: (1) cover the system infrastructure including individual components and component-to-component interactions on all coexisting architectural layers; and (2) provide information for subsequent analysis to ensure that the used formal model is consistent with respect to test requirements.
- A test case generation strategy which covers nonfunctional test cases to ensure that: (1) system dependability mechanisms (fault tolerance or high availability) have been implemented correctly on all coexisting architectural layers; and (2) the system is able to provide the desired level of reliable services.

In turn, the quality of formal methods based on abstract models is limited by the quality of these models. Thus, to get the full advantages of model-based testing, it is necessary to completely eliminate the human factor from the process of model generation. To address this problem, a possible appropriate presentation format of architecture descriptions that allows automated development of the formal models is defined as a necessary part of the detailed design documentation of complex commercial computer networks.

Abstract

Využívání komerčních počítačových sítí klade vysoké nároky na procedury, nástroje a přístupy pro jejich důkladné testování. Navzdory velkému úsilí mnoha výzkumných pracovníků však proces navrhování a vytváření testů stále zůstává spíše nestrukturovaný a spočívající na osobních zkušenostech nebo intuici jednotlivých inženýrů. S ohledem na vyřešení tohoto problému je hlavním výzkumným cílem této práce automatizované navrhování abstraktních specifikací testů (testovacích případů) pro počítačové sítě s použitím detailní projektové dokumentace (požadavky koncového uživatele a technické specifikace) jako zdroje dat. Na základě myšlenek: (1) testování modelů; a (2) systémové metodologie zahrnuje tato práce následující hlavní cíle:

– Formální model pro účely generování testů, vycházející z konceptu vícevrstvých sítí. Různé vrstvy (čtyři vrstvy v případě základních verzí a šest vrstev u verzí rozšířených) představují různé aspekty (hardwarové, softwarové, sociální, obchodní atd.) systémové architektury.

– Strategie generování testovacích případů, pokrývající strukturální testovací případy. Testovací případy tohoto druhu: (1) zahrnují systémovou infrastrukturu včetně jednotlivých složek a interakcí mezi těmito složkami na všech koexistujících vrstvách architektury; a (2) poskytují informace pro následnou analýzu potvrzující, že je použitý formální model konzistentní s požadavky na testy.

– Strategie generování testovacích případů, pokrývající nefunkční testovací případy, aby bylo zaručeno, že: (1) mechanismy spolehlivosti systému (tolerance chyb nebo vysoká dostupnost) byly správně implementovány na všech koexistujících vrstvách architektury; a (2) systém je schopný zajistit požadovanou úroveň spolehlivých služeb.

Na druhou stranu, kvalita formálních metod založených na abstraktních modelech je omezena kvalitou těchto modelů. Pro plné využití všech výhod modelového testování je tedy nezbytné zcela vyloučit lidský faktor z procesu generování modelů. Za účelem vyřešení tohoto problému je jako nutná součást detailní projektové dokumentace komplexních komerčních počítačových sítí definován vhodný formát prezentace popisů architektury, který umožňuje automatizovaný vývoj formálních modelů.

Acknowledgements

This Ph.D. thesis has been originated within the framework of research and development activities at the Department of Telecommunication Engineering (Czech Technical University in Prague, Faculty of Electrical Engineering).

First of all, I would like to offer my deepest thanks to my supervisor, Ing. Radek Mařík, CSc. for his invaluable help, inspiring support and encouragement throughout this thesis. Interesting suggestions for the direction of research came from him together with the thorough verification of my ideas. Moreover, he taught me how to reason about important problems and present my ideas.

Deep respect I express to my work colleagues from SPC TRIGGER s.r.o., in particular Ing. Eduard Ryazhapov, SEO, for their support and testing of my ideas in practice.

My special thanks also go to Sharon Anne King for proofreading this dissertation and detecting a lot of grammar and spelling errors.

I also want to thank all my teachers in schools, colleges, and universities whose dedication and hard work helped lay the foundation for this work.

Last but not least, I would like to thank my family whose constant support and encouragement made achieving the goal of obtaining a Ph.D. possible.

Contents

List of Figures

List of Tables

Abbreviations

ADT	**A**rchitecture **D**riven **T**esting
COTS	**C**ommercial **O**ff-**T**he-**S**helf
DNF	**D**isjunctive **N**ormal **F**orm
FDT	**F**ormal **D**escription **T**echniques
IEC	**I**nternational **E**lectrotechnical **C**ommission
IEEE	**I**nstitute of **E**lectrical and **E**lectronics **E**ngineers
IETF	**I**nternet **E**ngineering **T**ask **F**orce
ISO	**I**nternational **O**rganization for **S**tandardization
ITU	**I**nternational **T**elecommunicationl **U**nion
IANA	**I**nternet **A**ssigned **N**umbers **A**uthority
MBT	**M**odel-**B**ased **T**esting
MSC	**M**essage **S**equence **C**hart
OSI RM	**O**pen **S**ystem **I**nterconnection **R**eference **M**odel
RFC	**R**equest **F**or **C**omments
SDL	**S**pecification and **D**escription **L**anguage
SDLC	**S**ystem **D**evelopment **L**ife **C**icle
SOA	**S**ervice-**O**riented **A**rchitecture
SPOF	**S**ingle **P**oint **O**f **F**ailure
SUT	**S**ystem **U**nder **T**est
TCP/IP	**T**ransmission **C**ontrol **P**rotocol/**I**nternet **P**rotocol
TCP/UDP	**T**ransmission **C**ontrol **P**rotocol/**U**ser **D**atagram **P**rotocol
TTCN	**T**esting and **T**est **C**ontrol **N**otation
UML	**U**nified **M**odeling **L**anguage
URN	**U**ser **R**equirements **N**otation
VLAN	**V**irtual **L**ocal **A**rea **N**etwork

Chapter 1

Introduction

1.1 Background

The world we've made as a result of the level of thinking we have done thus far creates problems that we cannot solve at the same level at which we created them.

—Albert Einstein

Computing systems have come a long way from a single processor to multiple distributed processors, from individual-separated systems to network-integrated systems, and from small-scale programs to sharing of large-scale resources. Moreover nowadays, virtualization and cloud technologies make another level of system complexity. In turn, computer networks that support these systems have evolved to incorporate more and more sophisticated capabilities [1] (see Figure 1.1). To paraphrase Einstein, nowadays we have the ability to create networks that are so complex that when problems arise they cannot be solved using the same sort of thinking that was used to create the networks [2]. In fact, computer networks created with this complexity often do not perform as well as expected and do not match end-user/customer requirements.

On the other hand, the consequences of failure and downtime have become more severe. Their failure may endanger human lives and the environment, do serious damage to major economic infrastructures, endanger personal privacy, undermine the viability of whole business sectors and facilitate crime [3]. As a consequence, the most difficult part of computer network deployment is the question of assurance (whether the network will

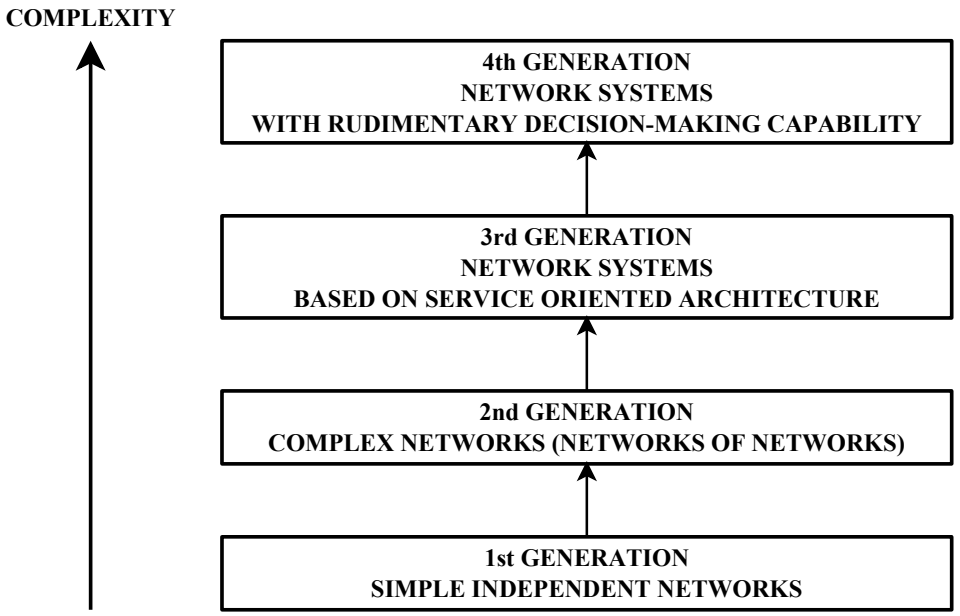


FIGURE 1.1: Generations of Networking [1].

work) and verification. If assurance is difficult, verification is even more difficult: it is a question of how to convince end-users/customers (and, in extremis, a jury) that a system is indeed fit for its requirements.

Generally, there is a practical means of failure detection (finding observable differences between the behaviors of implementation and what is expected on the basis of the technical specifications [4]) which can be highly effective if performed thoroughly. Despite the major limitation of testing[1], it is a necessary verification technique (it would be better to talk of a necessary and sufficient technique, but unfortunately in the case of complex systems a sufficient condition is theoretically unreachable [3]).

Hence, appropriate comprehensive testing plays a vital role in computer network development - it is necessary to determine a formal list of control objectives during the design phase of the System Development Life Cycle (SDLC) (see Figure 1.2) and, as the next step, to show that each component of this list undergoes a suitable amount of tests (at least one) during the implementation phase of the SDLC: i.e. it is necessary to have checklists [6].

[1] Testing is able only to show the presence of errors and never their absence [5].

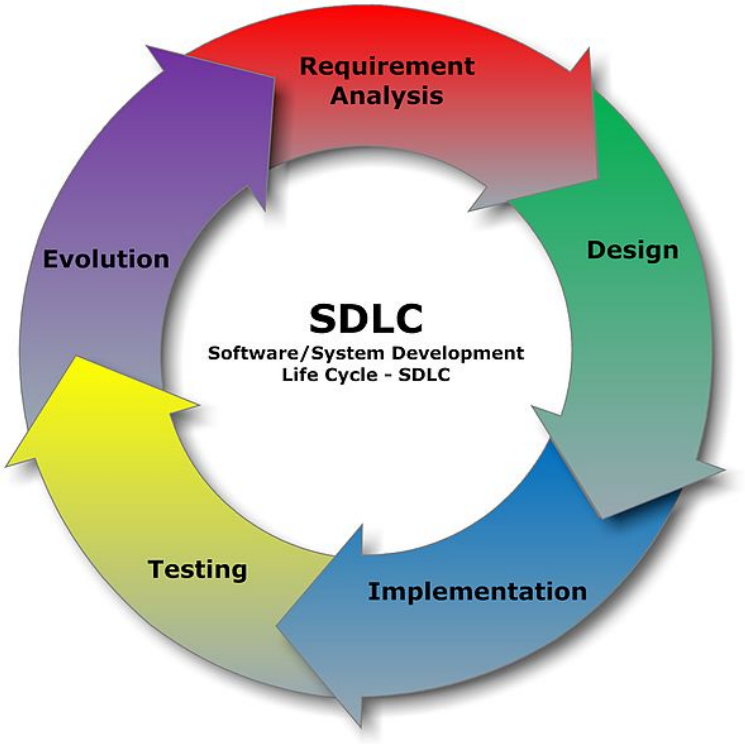


FIGURE 1.2: System Development Life Cycle [7].

1.2 Problem Statement

It isn't that they can't see the solution. It is that they can't see the problem.

—Gilbert Keith Chesterton

Applying a system methodology to network analysis [1] is a relatively new approach, particularly in the Internet Protocol (IP) world. The fundamental concept is that network architecture should take into account services/applications which this network provides and supports.

Historically, services/applications are the domain of system and software engineers. Respectively, computer networks are the domain of network engineers. As a consequence, system, software and network engineers have few common models or approaches and even their vocabularies (definitions) are different [3].

In fact, one of the most universal formal definitions of distributed systems, which was given by Tanenbaum and van Steen as *a collection of independent computers that appears to its users as a single coherent system* [8], can denote:

- a collection of components/products (hardware and software) - the viewpoint of the vendor community;
- a collection of the above plus external communication infrastructure - the viewpoint of the network engineer community;
- a collection of services/applications - the viewpoint of the software/system engineer community;
- all of the above plus end-users/customers - the viewpoint of the business community.

In practice, the confusion between these definitions is a fertile source of vulnerabilities in comprehensive testing. Broadly speaking, vendors focus on individual component testing problems only - but, in general, testing or qualification of elements of a system does not cover the system itself.

In turn, network engineers usually focus on network testing. In this case, ignoring services/applications influence is one of the most common causes of system problems [9]:

- If the network subsystem is not solid, services/applications cannot be responsive and reliable by definition.
- If the network subsystem is solid, but the services/applications do not provide required performance or functionality, end-users could perceive the network subsystem as unavailable or unreliable.

On the other hand, distributed systems differ from traditional software because components are dispersed across a network. Very often software/system engineers do not take this dispersion into account and this leads to the following false assumptions about computer networks [8]:

- networks are always reliable;
- latency is zero;
- bandwidth is infinite, etc...

In fact, the business (or integration) viewpoint brings all of the detailed elements of computer networks and the distributed systems they support together through a process of testing (or qualification) to achieve the valid systems for meeting the ultimate needs of the end-users/customers [10]. Hence, even if the main goal is the comprehensive testing of a computer network, analysis should cover the whole system.

It is important to note that this concept is completely supported by the most recent practical approaches such as Business-Driven Design [11] and Application Centric Design [12].

1.3 Work Objectives

Beware of false knowledge: it is more dangerous than ignorance.

—George Bernard Shaw

Despite the great efforts of many researchers, in the area of commercial system (specific areas such as the military, nuclear or aerospace industries are beyond the scope of this work) the process of test generation tends to be unstructured, barely motivated in the details, not reproducible, not documented, and bound to the ingenuity of individual engineers [4]. But in the case of complex or non-standard systems, personal experience and/or intuition are often inadequate. As a consequence, in the real world many systems have failed because:

- engineers had tested the wrong things;
- engineers had tested the right things but in the wrong way;
- some things had been just simply forgotten and had not been tested.

On the other hand, formal methods are mathematical techniques for developing software and hardware systems and can be used to conduct mathematical proofs of consistency of specification and correctness of implementation. Mathematical rigor enables users to analyze and verify abstract models at any part of the system life-cycle: requirements engineering, architecture design, implementation, maintenance and evolution [13]. These methods are particularly suitable for complex heterogeneous systems and are becoming

more and more important even if working engineers usually consider formal methods to be theoretical exercises that are widely taught in universities and not used anywhere in the real world.

The main research objective of this thesis is the automated design of test specifications for computer networks based on end-user requirements and technical specifications as a necessary part of detailed design documentation, i.e. during the design phase of SDLC (it is important to note that in practice computer networks are usually built not from scratch but from commercial off-the-shelf (COTS) hardware and software components). In this context, this thesis lies in the area of model-based testing (MBT).

The basic idea of MBT is that, instead of creating test cases manually, a selected algorithm generates them automatically from an abstract formal model (see Figure 1.3). In general, MBT involves the following major activities [4]:

- building the formal model from informal requirements or existing specification documents;
- defining test selection criteria and transforming them into operational test specifications or test cases;
- generating executable tests based on test cases;
- executing the tests (including conceiving and setting up adaptor components).

Based on the analysis of the overall tests development process, the resulting contributions of this thesis cover the first two major activities of MBT in three main areas:

- A formal model based on technical specifications to cover both hardware-based (system equipment) and software-based (system activities) aspects of a system under test (SUT). To evaluate the SUT as a whole, these aspects should be composed in such a way that their properties can be considered together. As a consequence, this composition has to: (1) preserve the properties of each aspect; and (2) represent interaction between aspects. On the other hand, the model must be sufficiently precise to serve as a basis for the generation of meaningful test cases [4],

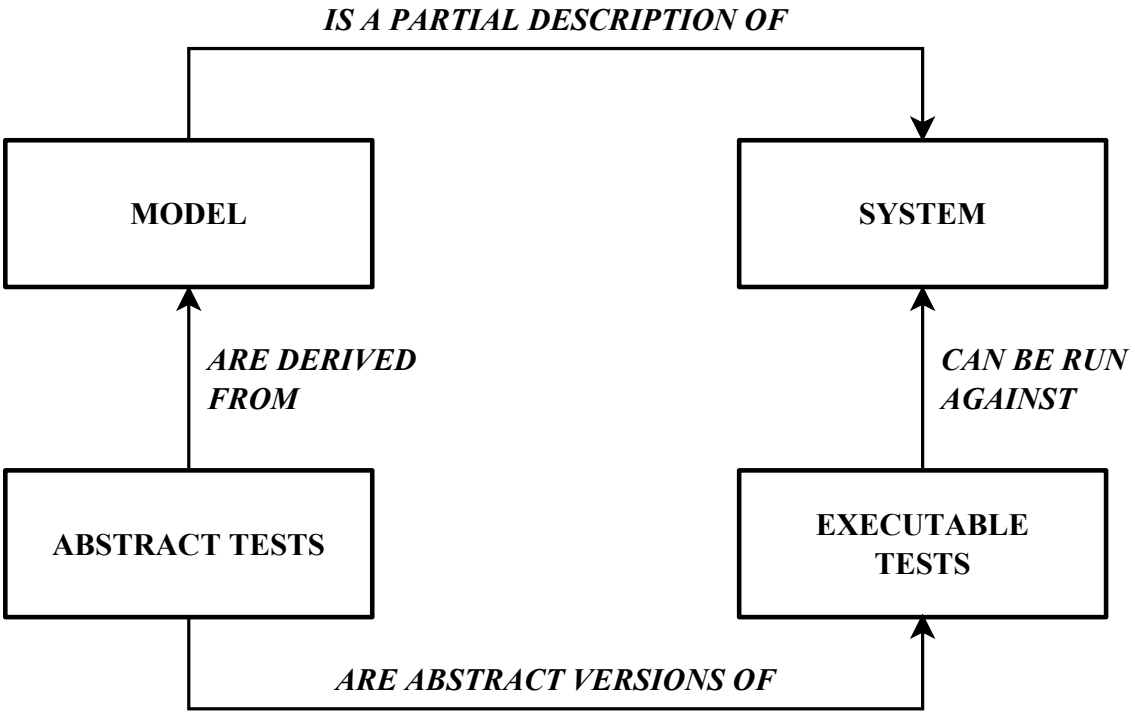


FIGURE 1.3: General model-based testing setting [14].

in other words the quality of model-based tests is limited by the quality of the model[2].

– A test case generation approach based on requirements coverage criteria. The test selection criteria are defined by: (1) end-user requirements; and (2) requirements derived from technical specifications, i.e. defined by technological solutions used to build the SUT. Generally, the test specifications should cover [15]: (1) structural tests which aim at the structure of the SUT; (2) functional tests[3]; and (3) nonfunctional (or extra-functional) tests which aim at assessing nonfunctional requirements such as reliability, load, and performance. In turn, test specifications scopes should cover [15]: (1) individual components; (2) component-to-component interactions; and, as a consequence, (3) the complete system.

– A formal model automated development approach. Generally, a SUT model must be correct in order to generate test case specifications accurately (as mentioned above, the quality of model-based tests is limited by the quality of the model).

[2]It is important to note that the model of the SUT can be used as the basis for test generation, but also can serve to validate requirements and check their consistency [4].

[3]In the case of COTS components, functional tests are usually prepared by vendors

Thus, to get full advantages of MBT, it is necessary to alleviate the burdens of learning model development and checking techniques for engineers and other non-technical stakeholders [16] or, ideally, completely eliminate the human factor

1.4 Terminology

A mistake is to commit a misunderstanding.

—Bob Dylan

In order to avoid misunderstandings and confusions, this section clarifies the usage of some key terms in this thesis.

In computer science, a system is: (1) a combination of interacting elements organized to achieve one or more stated purposes; or (2) an interdependent group of people, objects, and procedures constituted to achieve defined objectives or some operational role by performing specified functions [17]. The engineering definition is simpler: a system is a collection of components which cooperate in an organized way to achieve the desired result - the requirements [18]. It is important to note that this definition completely covers both computer networks and distributed systems. As a consequence, in this thesis the term *system under test (SUT)* (or just *system*) is used to denote a whole/complete system, i.e. a computer network and the distributed computing system that this network provides and supports, together.

There is no standard definition of model-based testing. In practice, the term *model-based testing (MBT)* is widely used today with subtle differences in its meaning. The most generic definition used in this thesis denotes MBT as the processes and techniques for the automatic derivation of abstract test cases from abstract models, the generation of concrete tests from abstract tests, and the manual or automated execution of the resulting concrete test cases [4].

In other words, the definition of MBT relates to the following definitions (see Figure 1.4 and Figure 1.5):

– Formal or abstract model. In computer science, a model is a representation of a real world process, device, or concept [17]. The engineering definition is quite

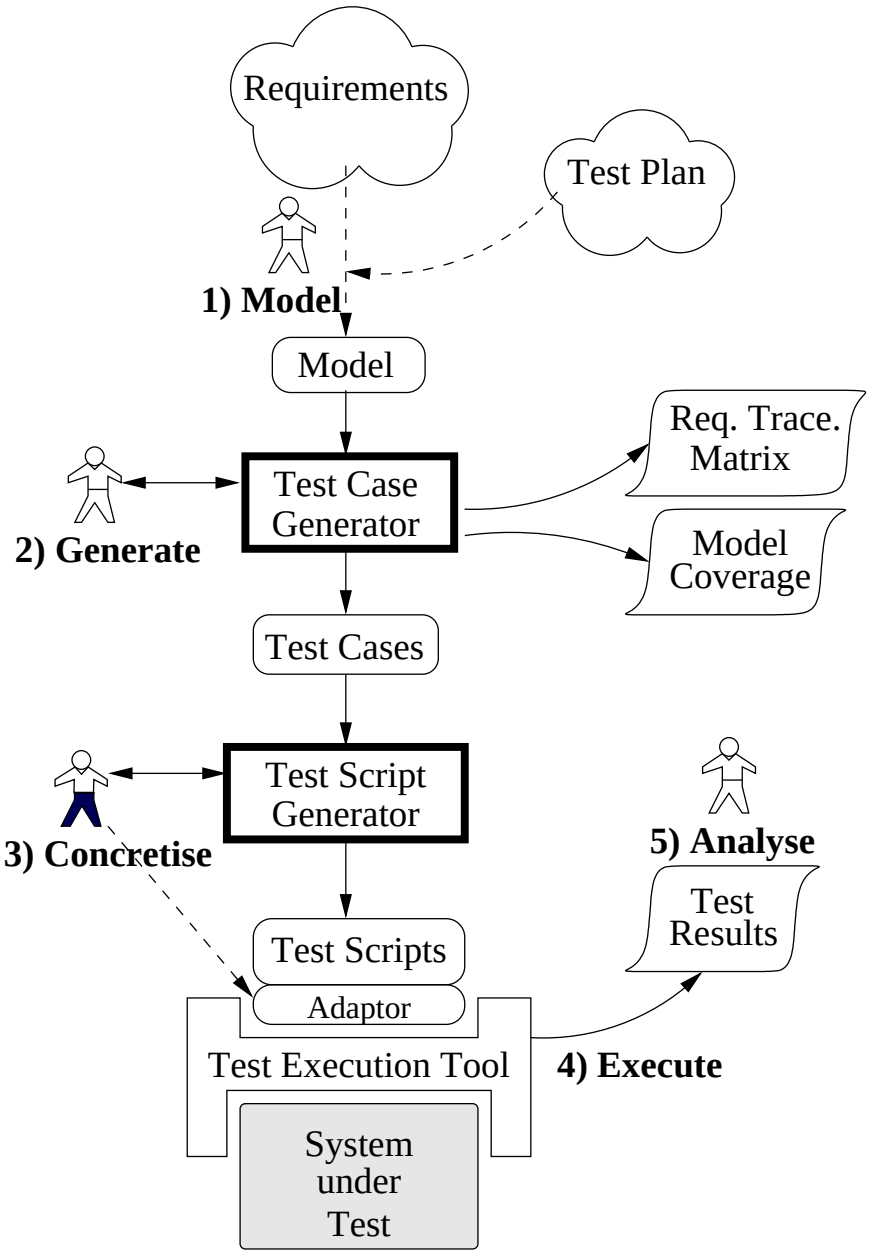


FIGURE 1.4: Model-based testing workflow [19].

similar: a system model is an abstract representation of certain aspects of the SUT [15]. In this thesis the term *formal model* (or just *model*) is used to denote the architecture viewpoint [20] as a simplified representation of the system with respect to the structure of the SUT.

– Test selection criteria. There is no definition based on standards. The engineering definition is quite simple: test selection criteria define the facilities that are used to control the generation of tests [15]. Generally, test selection criteria can relate:

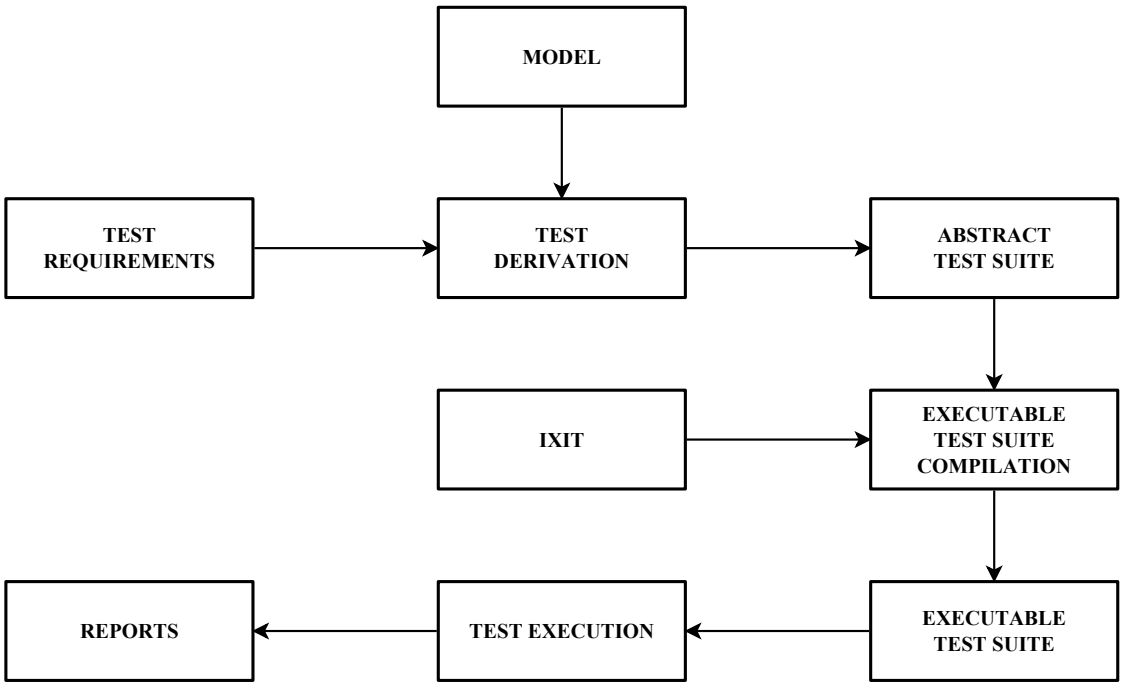


FIGURE 1.5: Model-based testing workflow [14]. Implementation extra information (IXIT) refers to information needed to convert an abstract test suite into an executable one (see also the *test bed* definition [17]).

(1) to a given functionality of the system; (2) to the structure of the model; (3) to data coverage heuristics; (4) to stochastic characterizations; or (5) to properties of the environment [4]. In this thesis the term *test requirements* is used to denote requirements coverage criteria[4] which relate to the structure of the system model.

– Test cases or abstract tests. In computer science, a test case[5] is a document specifying inputs, predicted results, and a set of execution conditions for a test item (an object of testing) [17]. The engineering definition denotes test cases as a collection of tests derived from a formal model on the same level of abstraction as the model[6] [14]. On the other hand, the most accurate definition of the nature of test cases (or abstract tests) in the area of model-based testing relates to the definition of test templates as formal statements of a constrained data space, i.e. test templates define sets of bindings of input variables to acceptable values [21]. As a consequence, in the context of this thesis the term *test case* is used to denote

[4]Other test selection criteria (data coverage criteria, random and stochastic criteria, fault-based criteria, etc... [15]) are beyond the scope of this thesis.

[5]It is important to note that the standard Std 24765:2010 does not distinguish between the definitions of *test case* and *test case specification* [17].

[6]These test cases are collectively known as an abstract test suite [14].

the result of application (binding) a test requirement (input variable) to a formal model (acceptable values).

– Test procedures or executable tests (test scripts). In computer science, test procedures are the detailed instructions for the setup, execution, and evaluation of results for a given test case [17]. The term is strictly relevant to the MBT definition but not used in this thesis (the process of executable test generation is beyond the scope of this thesis).

In computer science, dependability is trustworthiness of a computer system such that reliance can be justifiably placed on the service it delivers [17]. The original definition of dependability determines the system ability to deliver service that can justifiably be trusted [22] (this definition stresses the need for justification of trust). The engineering definition which is used in this thesis: *dependability* is the ability of a system to avoid service failures or the probability that a system will operate when needed [22].

The major category of dependability that relates to MBT is fault tolerance. In computer science, *fault tolerance* is the ability of a system or component to continue normal operation despite the presence of hardware or software faults [17]. We need to state here the difference between *fault tolerance* and *high availability*: a fault tolerant environment has no service interruption, while a highly available environment has minimal service interruption.

1.5 Thesis Goals

A goal is a dream with a deadline.

—Napoleon Hill

As mentioned above, the main research objective is the automated design of test specifications for commercial computer networks using the detailed design documentation (end-user requirements and technical specifications) as the data source. Based on the analysis of the model-based tests development processes, the resulting contributions of this thesis can be divided into four main goals that should be solved separately, but not in isolation:

– A formal model for test generation missions based on the concept of hierarchical multilayer networks. Different layers represent different (hardware or software) aspects of system architecture.

– A test case generation strategy[7] that covers structural/functional tests. Test cases of that kind: (1) cover the system infrastructure including individual components and component-to-component interaction on all coexisting architectural layers; and (2) check the internal consistency of the system technical specifications with respect to the end-user requirements.

– A test case generation strategy which covers nonfunctional tests to ensure that system dependability mechanisms (fault tolerance or high availability) have been implemented correctly on all coexisting architectural layers and the system is able to provide the desired level of reliable services.

– An appropriated presentation format of system architecture descriptions as a necessary part of the detailed design documentation (technical specifications) that allows automated development of the formal model for analysis and verifying of the system.

To accomplish these goals the thesis defines the methodology of system test case design based on the following general steps:

– At first, the system under test is modeled as a weighted graph structure. Vertices represent: (1) software components (such as application software and operated systems); and (2) hardware components (such as routers, switches, servers and PCs). Based on the concept of multilayer networks, edges represent: (1) interlayer component-to-component relations (such as web-browser-to-web-server or router-to-switch/server-to-switch interconnections); and (2) intralayer component-to-component relations (operated systems should fit application software and hardware platforms should fit operated systems). The graph labels represent the sets of facts (attributes) about their entities. The labels of the vertices determine the sets of communication protocols supported by the system components which are represented by the vertices (for example a WEB server can support http and https

[7] A test strategy (or test philosophy) establishes *what* should be tested and *why* [9].

protocols and a switch can support 10/100/1000BASE-T and 10GBASE-SR protocols). In turn, the labels of the graph edges determine the sets of communication protocols used for interlayer component-to-component interconnections which are represented by the edges (for example a web-browser-to-web-server interconnection uses the https protocol and a switch-to-switch interconnection uses the 10GBASE-SR protocol). Based on their nature, intralayer component-to-component relations do not have labels.

– Next, test requirements (test selection criteria) should be specified. The test requirements determine: (1) the system components which should exist in the SUT and their attributes (for example a system should contain a web-server and this web server should support the https protocol); and (2) the paths between system components which should exist in the SUT and their attributes (for example a router-to-switch path should exist and this path should use 1000BASE-T protocol). In general, the sets of attributes can be empty sets - in this case test requirements determine the fact of components or paths existence only.

– Finally, the test cases are the result of recursive applying of test requirements to the model. In general (based on the concept of multilayer networks), each test requirement can induce more than one test case. Firstly, a test requirement induces a test case for a given layer. Secondly, the test requirement propagates on the layer below using the intralayer component-to-component relations and induces a test case for this layer, and so on. As a consequence, each test requirement defined for the top architectural layer of the system model initiates at least one test case on all coexisting architectural layers (for example a test requirement for a web server induces test cases for: (1) the web-server itself; (2) its operated system; and (3) the hardware or virtualization platform which support the operated system) and cover computer networks and distributed computing systems, which these networks support, as whole systems.

1.6 Thesis Organization

To write simply is as difficult as to be good.

—William Somerset Maugham

This thesis is organized as follows. This chapter gives an overview and scope of the research topics of this thesis. It introduces the problems that the work is dealing with, its objectives, contributions and structure. Chapter 2 introduces the background and related work. Chapter 3 represents the formal system model based on the concept of multilayer networks. Next, Chapter 4 focuses on the test case generation strategy to cover structural/functional tests. Based on the previous chapters, Chapter 5 considers the test case generation strategy to cover fault-injection experiments based on analytical tools for system reliability assessment. Chapter 6 introduces the presentation format of architecture descriptions as a necessary part of the detailed design documentation (technical specifications), which allows automated development of the formal model, and the correlation between the formal model and this presentation format. Chapter 7 represents a case study. Finally, conclusion remarks and future research directions are given in Chapter 8.

Chapter 2

Related Work

Get your facts first, then you can distort them as you please.

—Mark Twain

The research presented in this thesis focuses on the automated design of test templates (specifications) for computer network comprehensive testing and thus spans the areas of:

- formal models of complex systems;
- model-based testing;
- dependability testing;
- presentation formats of system architecture description (design documentation).

This chapter presents the background and prior related research in each of these areas.

2.1 Formal Models

Over the years a lot of effort has been invested in creating formal models of complex systems. However, each model typically represents only one aspect of the entire system. To evaluate the system as a whole, these models must be composed in such a way that their properties can be considered together. As a consequence, this composition has to:

– preserve the properties of each individual model;

– represent interaction between individual models.

Nowadays, the modeling of complex systems can be roughly classified into two categories:

– decomposition of complex models (tree structures);

– multi-layer (composed) models.

2.1.1 Decomposition of Complex Models

Liu and Lee [23] and Eker et al. [24] represent a structured approach - hierarchically heterogeneous. Using hierarchy, they can divide a complex model into a tree of nested submodels (see Figure 2.1), which are at each level composed to form a network of interacting components (each of these networks are locally homogeneous[1], while different interaction mechanisms are specified at different levels in the hierarchy). One key concept of hierarchical heterogeneity is that the interaction specified for a specific local network of components covers the flow of data as well as the flow of control between them.

The three dimensional analysis (Yadav et al. [26]) decomposes a system structure into its physical elements and shows, in detail, how functional requirements can be fulfilled by individual product elements or groups of elements (see Figure 2.2). The functional requirements propagate from the requirements for the complete product down to the elements in a hierarchical manner. The mapping between physical elements and functional requirements shows which physical elements have impact on the same function or which single element has an impact on different functions. The time dimension (or damage behavior) helps in identifying which failure mechanisms have impact on physical elements and, as a consequence, on system functions.

Benz and Bohnert [27] define the Dependability Model as a model of use cases that are linked to system components they depend on. These models are constructed by identifying user cases or user interactions and then finding system functions, services and components which provide them. Once all system parts are found, the provision

[1] Homogeneous is uniform in composition or character (i.e. the type of components in a system and their interactions); one that is heterogeneous is distinctly nonuniform in one of these qualities [25].

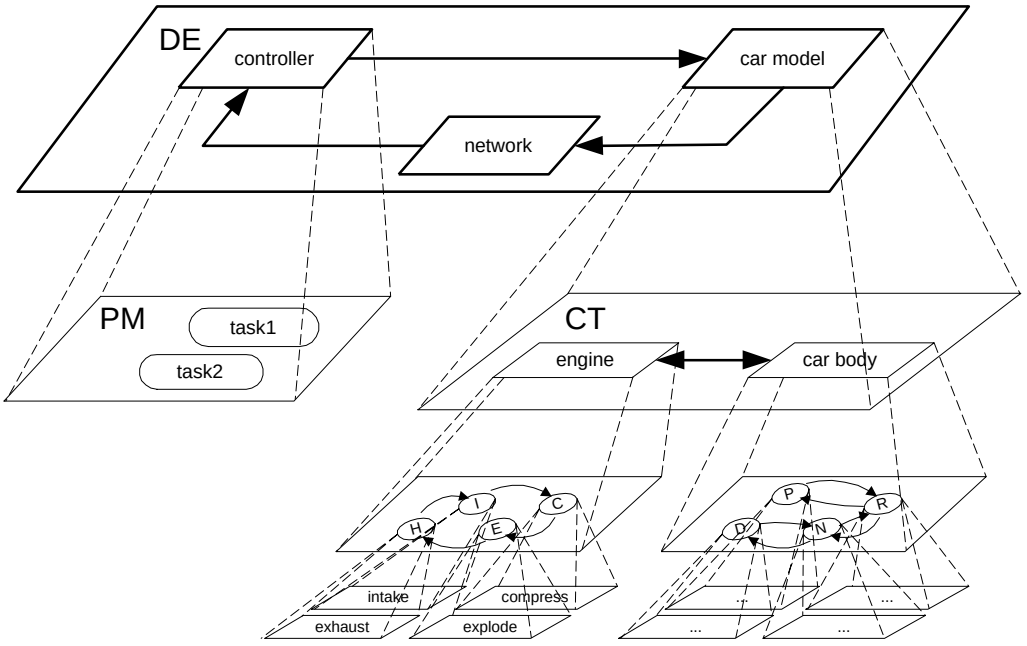


FIGURE 2.1: A hierarchical model for the engine control systems [23].

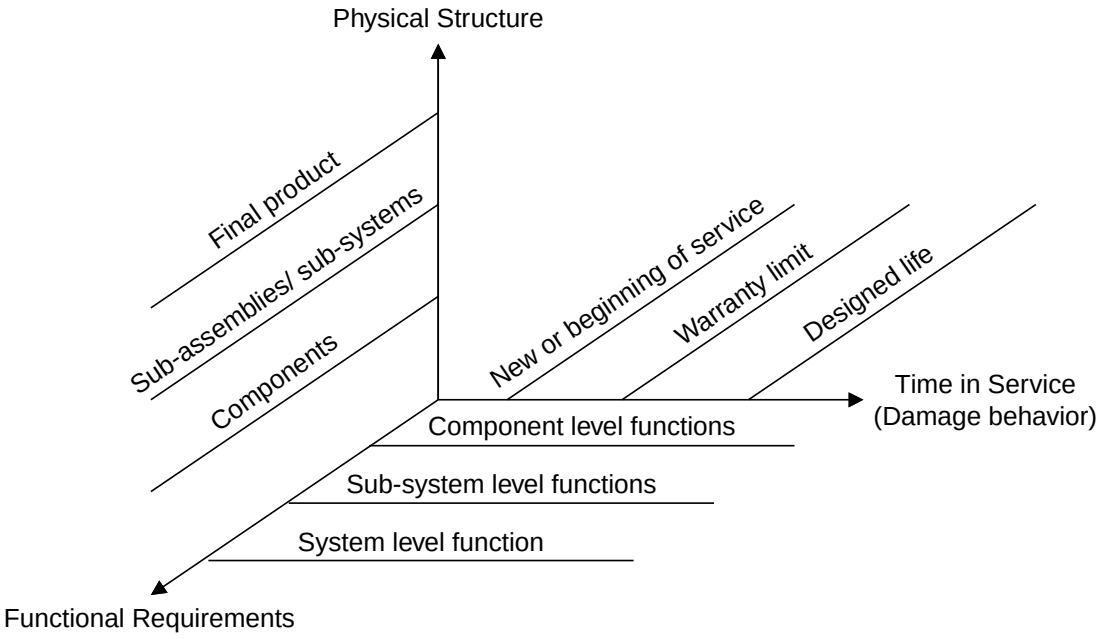


FIGURE 2.2: Three dimensions of system structures [26].

of use cases is modeled as links which show the dependability of user interactions on system components. Dependability models are shown either in a dependency table or in a dependency graph (see Figure 2.3) to show the different dependencies between user interactions, system functions, services and system resources.

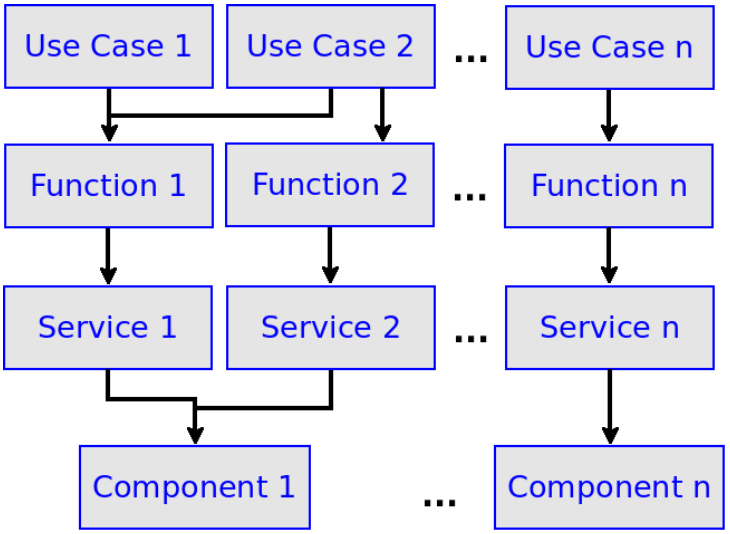


FIGURE 2.3: Dependency graph [27].

2.1.2 Multilayer Networks

One of the major goals of modern physics is providing proper and suitable representations of systems with many interdependent components, which, in turn, might interact through many different channels. As a result, interdisciplinary efforts of the last fifteen years with the aim of extracting the ultimate and optimal representation of complex systems and their underlying mechanisms have led to the birth of a movement in science, nowadays well-known as complex networks theory [28] [29] [30]. The main goals are [31]:

- the extraction of unifying principles that could encompass and describe (under some generic and universal rules) the structural accommodation;
- the modeling of the resulting emergent dynamics to explain what can be actually seen from the observation of such systems.

The traditional complex network approach is concentrated on cases when each system elementary unit (node or entity) is charted into a network node (graph vertex), and each unit-to-unit interaction (channel) is represented as a static link (weighted graph edge) that encapsulates all connections between units [32] [33] [34]. However, it is easy to realize that the assumption of encapsulation of different types of communication into a single link is almost always a gross oversimplification and, as a consequence, it can lead to incorrect descriptions of some phenomena that are taking place on real-world networks.

In turn, multilayer networks [34] [35] [31] explicitly incorporate multiple channels of connectivity and constitute the natural environment to describe systems interconnected through different types of connections: each channel (relationship, activity, category, etc...) is represented by a layer and the same node or entity may have different kinds of interactions (different set of neighbors in each layer). Assuming that all layers are informative, they can provide complementary information. Thus, the expectation is that a proper combination of the information contained in the different layers leads to a formal network representation (a formal model) which will be appropriate for applying the system methodology to network analysis.

Recent surveys in the domain of multilayer networks provided by Kivela et al. [35] and Boccaletti et al. [31] give a comprehensive overview of the existing technical literature and summarize the properties of various multilayer structures. However, it is important to note that the terminology referring to systems with multiple different relations has not yet reached a consensus - different papers from various areas use similar terminologies to refer to different models, or distinct names for the same model.

The significant case, which should be highlighted, is the multilayer model for studying complex systems introduced by Kurant and Thiran [36]. For simplicity, only a two-layer relationship is used (but the model can be extended to multi-layers). The lower-layer topology is called a physical graph and the upper-layer is called a logical graph (the physical and logical graphs can be directed or undirected, depending on the application). The number of nodes is equal for both layers. Every logical edge is mapped on the physical graph as a physical path. The set of paths corresponding to all logical edges is called mapping of the logical topology on the physical topology [36] (see Figure 2.4).

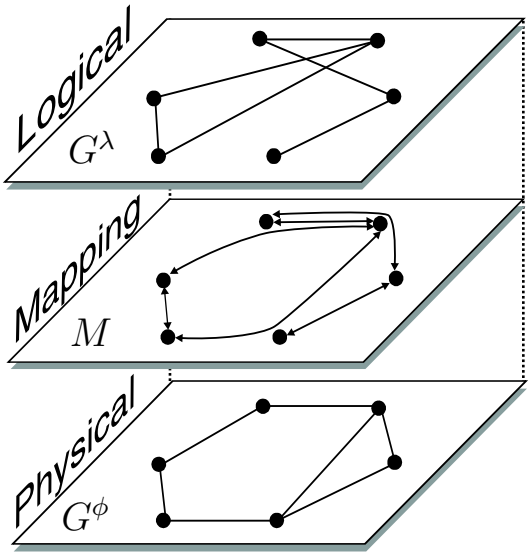


FIGURE 2.4: Multilayer model [36].

In the context of this thesis, the taxonomy of multilayer networks can be completely covered by four main dimensions [37] (see Figure 2.5):

- intralayer definition;
- intralayer topology;
- interlayer definition;
- interlayer topology.

Moreover, in terms of MBT (a formal definition of multilayer structures as the key component of MBT) the two dimensions which represent structural properties can be shown as a grid [37] (see Figure 2.6). The main intersection point denotes the basic formal definition [31]. In turn, the other three points can be described as special cases of the basic definition. It is important to note that this grid covers the majority of multilayer structures presented in the surveys [35] [31] (see Figure 2.6).

Despite the fact that the basic formal definition [31] (like the general form [35]) mainly targets transport, biologic (epidemic) and social networks, it can be used as a starting point and adapted and extended accordin to the goals of this thesis.

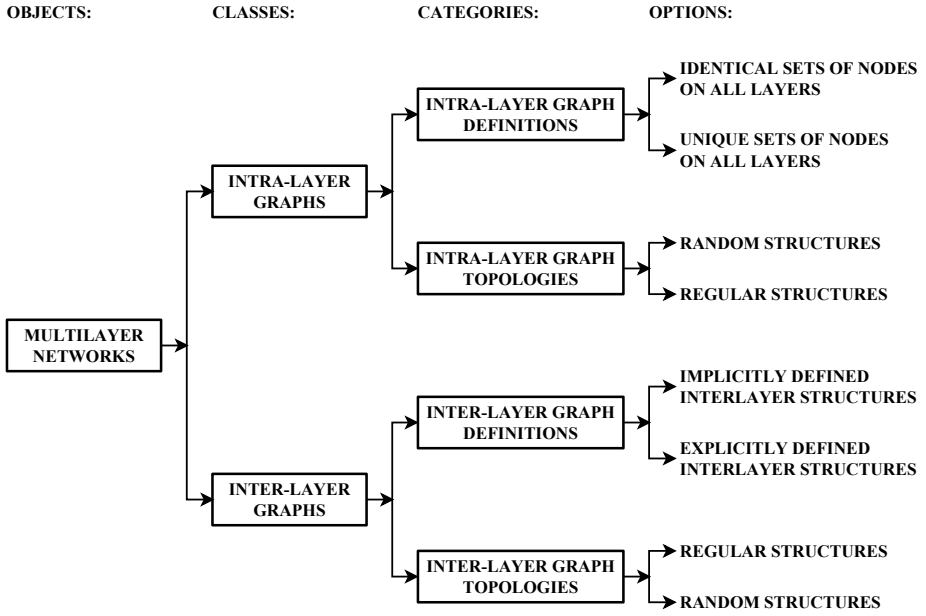


FIGURE 2.5: Taxonomy of multilayer networks [37].

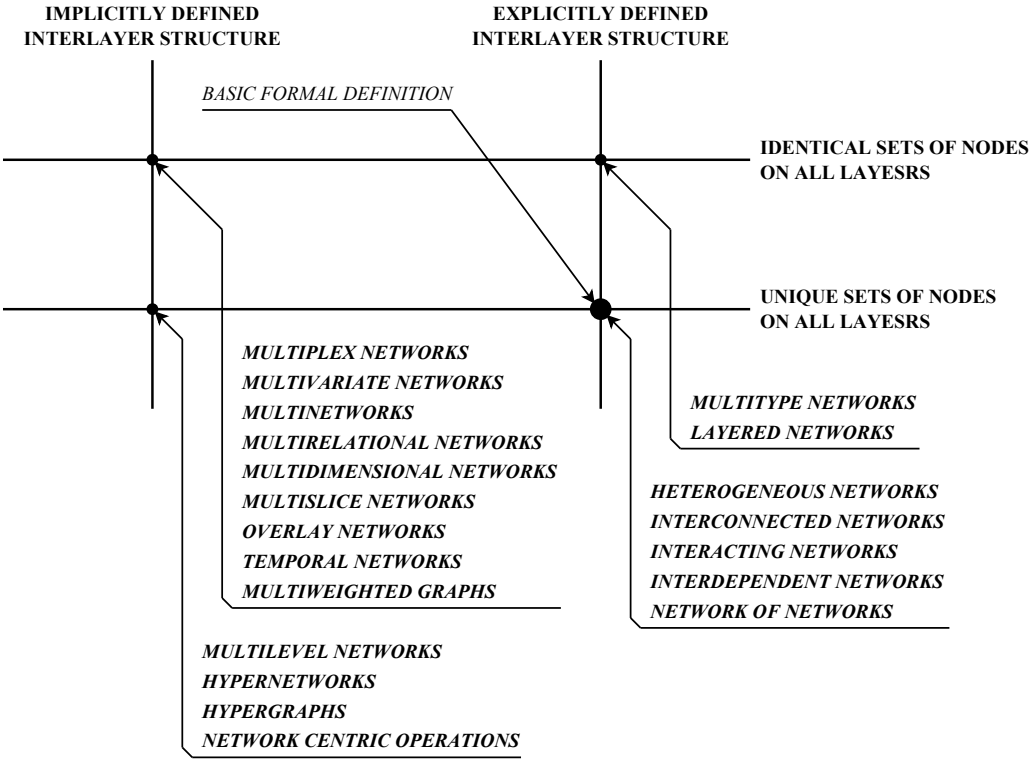


FIGURE 2.6: The grid of structural properties [37]. Data sources: Multiplex networks [34] [38]; Multivariate networks [39] [40]; Multinetworks [41]; Multirelational networks [42]; Multidimensional networks [43]; Multislice networks [44] [45]; Overlay networks [46]; Temporal networks [45] [47]; Multiweighted graphs [48]; Multilevel networks [49] [50]; Hypernetworks [51] [49]; Hypergraphs [52] [49]; Network centric operation [53]; Multiple networks [54] [55]; Layered networks [36] [56]; Heterogeneous networks [42] [57]; Interconnected networks [58] [59]; Interacting networks [60]; Interdependent networks [61] [62]; Network of networks [63].

2.2 Model-Based Testing

Recent surveys by Broy et al. [64], Dias-Neto et al. [65] and Hierons et al. [66] provide a comprehensive overview of the existing technical literature in the MBT field. MBT research in the domain of complex (hardware/software integrated) systems can be roughly classified into three categories [67] [66] [13]:

- MBT general approaches;
- MBT based on explicit models;
- MBT based on formal specifications.

MBT general approaches. El-Far and Whittaker [68] give a general introduction to principle, process, and techniques of model-based testing. Stocks and Carrington [21] define the term test templates and suggest that test templates can be defined as the base for test case generation and large test templates should be divided into smaller templates for generating more detailed test cases. In turn, Din et al. [20] represent the approach for architecture driven testing (ADT). A taxonomy of model-based testing approaches is provided by Utting et al. [4] (see Figure 2.7) and Zander et al. [15].

MBT based on explicit models. Offutt and Abdurazik [69] describe an approach to generating test cases from UML Statecharts for components testing. Hartmann et al. [70] extend the approach for integration testing and for test automation. Abbors et al. [71] represent a systematic methodology in the telecommunication domain. In turn, Peleska [72] introduces approaches to hardware/software integration and system testing.

MBT based on formal specifications. Bernot et al. [74] set up a theoretical basis for specification-based testing, explaining how formal specifications can serve as a base for test case generation. Dick and Faivre [75] propose to transform formal specifications into a disjunctive normal form (DNF) and then use it as the basis for test case generation. Donat [73] represents: (1) a technique for automatic transformation of formal specifications into test templates; and (2) a taxonomy for coverage schemes (see Figure 2.8). Hong et al. [76] show how coverage criteria based on control-flow or data-flow properties can be specified as sets of temporal logic formulas, including state and transition coverage as well as criteria based on definition-use pairs. A systematic method

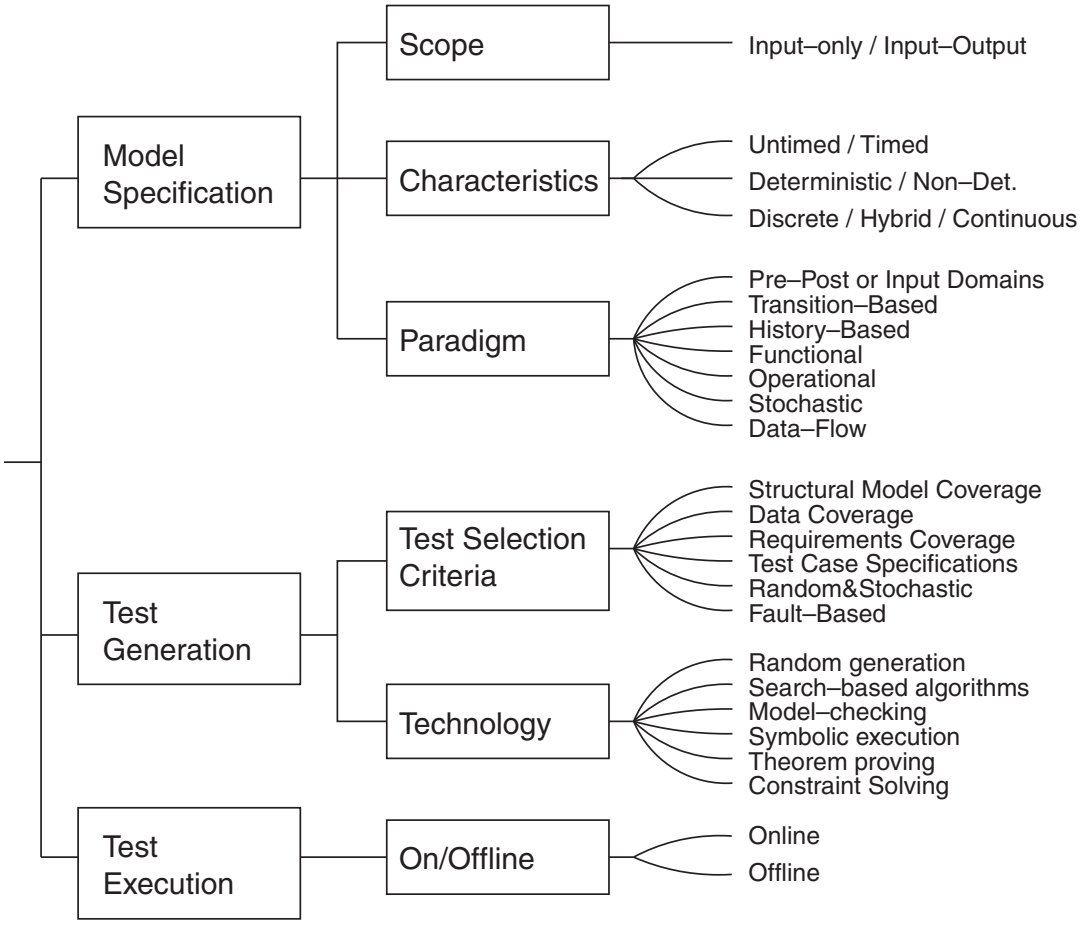


FIGURE 2.7: Taxonomy of model-based testing [4].

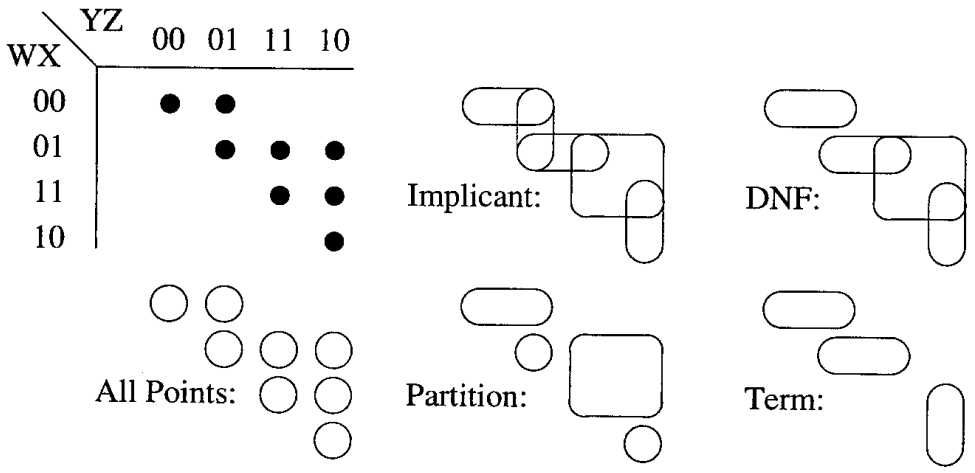


FIGURE 2.8: Taxonomy of coverage schemes [73].

presented by Liu and Shen [77] can be used for (1) identifying all possible scenarios; (2) formalizing informal requirements into formal operation specifications; and (3) testing based on these formal specifications (scenario-coverage strategy).

In the context of this thesis, the general principles of model-based testing provide the following basic notions:

- The MBT framework that defines major processes (see Figure 2.9): (1) building the formal model from detailed design documentation (technical specifications); (2) defining test selection criteria, i.e. transforming informal end-user and technical requirements into formal test requirements; (3) generation test cases (test specifications) based on formal test requirements; and (4) checking the internal consistency of the formal model with respect to the test requirements.
- Test scopes that cover: (1) individual components; and (2) component-to-component interactions.
- Test goals that cover: (1) structural tests; and (2) nonfunctional (or extra-functional) tests.

It is important to note that in the case of COTS components or groups of components, i.e. commercial products[2], functional tests are usually prepared by vendors. In the real world engineers are always under great financial and timing constraints and, as a consequence, have to rely on:

- vendors' and/or independent laboratories' information about products;
- vendors' conclusions of products' compliance with end-users'/customers' requirements;
- vendors' documentation (includes test descriptions).

On the other hand, model paradigms and test generation technologies that are based on formal specifications do not cover computer networks and distributed computing systems, which these networks support, as whole systems. Hence, they should be redefined according to the goals of this thesis.

[2]Commercial product is a product that can be sold, rather than one still being developed [78].

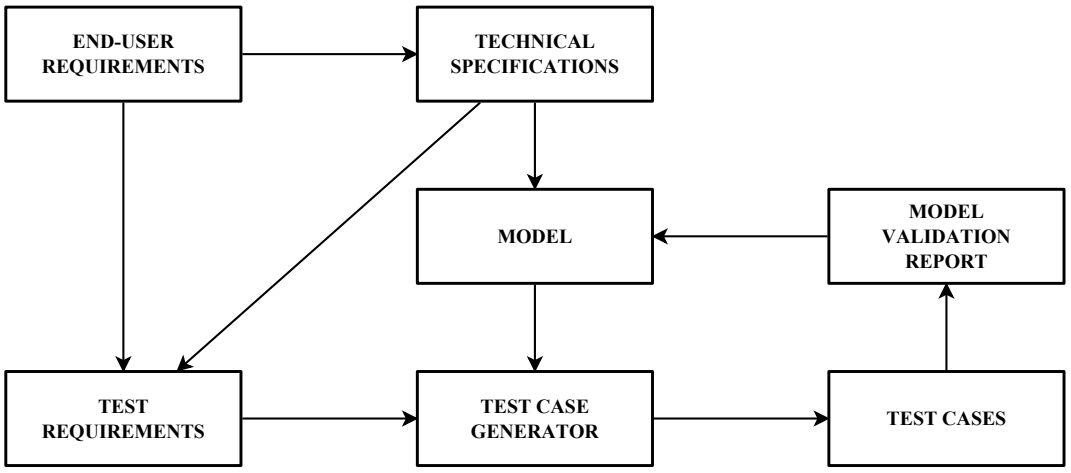


FIGURE 2.9: MBT framework in the context of this thesis.

2.3 Dependability Testing

The key factor of fault tolerance (or fault transparency [8]) is preventing failures due to system architectures and it addresses the fundamental characteristic of dependability requirements in two ways [79]:

- replication, i.e. providing multiple identical instances of the same component and choosing the correct result on the basis of a quorum (voting);
- redundancy, i.e. providing multiple identical instances of the same component and switching to one of the remaining instances in case of a failure (failover).

As a consequence, a system must be validated to ensure that its replication/redundancy mechanism has been correctly implemented and the system will provide the desired level of reliable service. Fault injection (the deliberate insertion of faults into a system to determine its response [80] [81]) offers an effective solution to this problem. Fault-injection experiments provide a means for understanding how these systems behave in the presence of faults (the monitoring of the effects the injected faults have on the system's final results).

Fault injection research in the domain of computing systems can be roughly classified into three categories [80] [82] [83]:

- physical fault injection [84] [85];

- software fault injection [86] [87];
- fault simulation (environment fault injection) [88] [83] [89] [90].

The mixed-mode simulation, where the system is hierarchically decomposed for simulation at different abstraction levels, is particularly useful in the case of complex distributed systems.

In turn, strategies for the fault-injection experiments are generally based on methods for assessing system reliability (identifying potential faults and determining the resulting error effects) [26] [91] [27] [92].

Nowadays, models for assessing reliability of distributed systems can be roughly classified into [93]:

- user-centric models;
- architecture-based models;
- state-based models.

2.3.1 User-centric models

Generally, user-centric models can be defined as the top-down or service-oriented approach (i.e. the viewpoint of the business/end-user community) to the reliability of distributed systems [94] [95] [96]. As reliability of any system has direct impact on the system usage, so these models focus on user/subscriber and provider behavior and basically work on the principle of evaluating transmission time to compute the execution time of each file or program under real conditions running in a distributed environment. As a consequence, the system reliability is based on the operational or usage profile of the given set of services.

The common analytical tool for user-centric models is time-based models (founded on the queueing theory) [97] [96]. User-centric approaches can be characterized as multi-stage problem solving processes where the system is conceived in terms of user behavior.

2.3.2 Architecture-based models

In contrast to the user-centric models, architecture-based models can be defined as the bottom-up or hardware-based approach (the viewpoint of the engineering community) to the reliability of distributed systems. In turn, they can be classified into: (1) component-oriented models; and (2) communication-oriented models.

Component-oriented models represent distributed systems as a composition of multiple processors but completely ignore the failures of communications and assume that the communication channels (links) among the processors are perfect [98] [99] [100]. Without considering communication failures, the exchanged information between components (software and hardware) must always be correct. In this case, the problem of distributed system reliability can be reduced to a parallel-series structure. In turn, the parallel-series reliability is easy to calculate [101] [100] [102] [103]. Such condition may be a good approximation for a system that exchanges only a little information among nodes, such as those where the processors do only their own jobs (no intensive data transmission).

The analytical tool for component-oriented models is reliability block diagrams (one of the conventional and most common tools of system reliability analysis [102] [103]).

In contrast to the component-oriented models, *communication-oriented models* consider the communication failures and assume that the components themselves (the nodes of networks) are always perfect [104] [100] [103]. They suppose that the system failures are caused by communication failures on channels (links) while the components (or nodes) cannot fail during the executing of programs. Such condition is a good approximation for cases where the communication time dominates the time of program execution or the components are highly reliable in comparison to the channels.

The analytical tool for communication-oriented models is network diagrams (commonly used in representing communication networks consisting of individual links [100]).

An additional effective analytical tool for architecture-base models is fault tree diagrams (the underlying graphical model in fault tree analysis) [105] [100] [102]. Whereas the reliability block diagrams and network diagrams are mission success oriented, the fault tree shows which combinations of the component failures can lead to system failures. And fault tree diagrams can describe the fault propagation in systems. However, repair

and maintenance (two important operations in system analysis) cannot be expressed using a fault tree formulation.

2.3.3 State-based models

The first generation of state-based models that considered both node failures and link failures have a common assumption - the operational probabilities of nodes or links are constant without considering bandwidth and content (constant-reliability models) [106] [107] [108]. However, this assumption of the constant-reliability of elements is not suitable in practice. Intuitively, downloading a larger file from a remote site will have a higher risk of failure than downloading a smaller file through the same link [109].

The most recent models relax this assumption for the elements (nodes and links). Instead, they assume that the failures of elements follow Poisson processes, so that the more time an element works (including execution and communication), the less reliable that element is [109] [110] [111]. In addition, the traditional models study the network topology by physical links and nodes that are static without considering dynamic changes of components and logic structures. To solve these problems, recent models use a virtual structure instead of physical structure [110] [111].

The analytical tool for state-based models is Markov models [100] [102]. To deal with all sorts of errors such as time-out failures, blocking failures, network failures, etc. (which can occur during operations of execution and communication), a hierarchical model must be used. This model suggests tackling various errors in different layers and uses Markov state principle to map layers into different physical states [93].

The general approach (common to all types of models) is to treat reliability as a complex problem and to decompose the distributed system into a hierarchy of related subsystems or components. Rebaiaia and Ait-Kadi [112] provide a survey of methods, algorithms and software tools. However, it is important to note that the reliability evaluation problem is NP-complete and, as a consequence, the generation of an exact solution is very problematic.

In turn, Kurant et al. [56] represent the dynamic analysis (or fault-injection simulation) which provides a means for understanding how two-layer complex systems which come

from the fields of communication (the Internet), transportation (the European railway system) and biology (the human brain) behave in the presence of faults. It includes:

- successive removals of vertices/edges from the model on the physical layer;
- impact assessments of those removals (disruption on the physical layer might destroy a substantial part of the upper logical layer which is mapped on it, rendering the whole system useless in practice).

Despite the fact that the dynamic analysis [56] mainly targets two-layer complex systems, it can be used as a starting point and adapted and extended according to the goals of this thesis

2.4 Presentation Formats

The universal requirement for design documentation is simple - the documentation should be based on standards like each and every formal document. Generally, the choice between international and regional standards depends on the state and/or corporate legislation but, fortunately, the majority of regional standards replicate their international predecessors.

The Formal Description Techniques (FDT) [113] are based on a technical language for unambiguous specification and description of the behavior of telecommunication systems. The main FDTs include: Specification and Description Language (SDL) [114], Message Sequence Chart (MSC) [115], User Requirements Notation (URN) [116], and Testing and Test Control Notation (TTCN) [117]. However, FDTs are intended to specify the behavioral aspects of software-intensive systems, not their architectures [114]. Furthermore, they do not cover the structure of design documentation.

The current revision of IEEE Std 1362-1998 (R2007) [118] standard represents a Concept of Operations (ConOps). ConOps is a user-oriented document that describes characteristics of to-be-delivered systems from the end-users' (or integrated systems) point of view. It also specifies recommended graphical tools (charts and diagrams).

The standard ISO/IEC Std 15288:2015 [119] establishes a common process framework for describing the life cycle of man-made systems. It defines a set of processes and associated terminology for the full life cycle, including *Architectural Design Process* (or the process of elaboration of design documentation). In turn, the standard ISO/IEC Std 15289:2011 [120] specifies the purpose and content of service management information items (documentation). It defines the life cycle data of ISO/IEC Std 15288:2015 by relating tasks and activities to the generic types of information items such as *descriptions* and *specifications* (the main information components of design documentation). Furthermore, conceptualization of system architectures assists the understanding of the system essence and key properties pertaining to its behavior, composition and evolution, which in turn affect concerns such as the feasibility, utility and maintainability of the system. As a consequence, the standard ISO/IEC Std 42010-2011 [121] specifies architecture viewpoints, architecture frameworks and architecture description languages for use in architecture descriptions.

The latest revision of IEEE Std 1233-1998 [122] and ISO/IEC Std 29148:2011 [123] standards specify the engineering processes of the identification, organization, presentation and modification of system requirements. These processes address conditions for incorporating operational concepts, design constraints and design configuration requirements into technical specifications.

In the context of this thesis, it is important to note that these international standards establish *what* should be contained in design documentation but not *how*: possible formats of information items or, at least, guidance on selecting appropriate presentations are not included in the scope of these standards. As a consequence, an appropriated presentation format of system architecture descriptions as a necessary part of the detailed design documentation (technical specifications) that allows automated development of the formal model should be defined according to the goals of this thesis.

Chapter 3

Formal Model

The journey of a thousand miles begins with one step.

—Lao Tzu

The multilayer approach for the modeling of complex systems covers two main areas:

- multilayer models (architecture formal representation);
- reference models (layers definition).

3.1 Multilayer Model

A type of multilayer network of particular relevance for computer networks is a *hierarchical multilayer network* [35], in which the bottom layer constitutes a *physical network* and the remaining layers are *virtual layers* that operate on top of the physical layer. Hence, the formal definitions of multilayer networks [31] [35] can be used as a starting point. However, these definitions support a wide spectrum of arbitrary relationships between different layers[1]. The necessary condition of top-down consistency can be provided by the concept of layered networks [36]. In turn, this concept is based on the facts:

[1]The arbitrary relationships between different layers are not a common case in the domain of computing systems. For example: in general, services/applications cannot communicate with hardware components directly [124].

– for each node on a given layer there is a corresponding node (or nodes) on the layer below;

– for each path between two nodes on a given layer there is a path (or paths) between the corresponding nodes on the layer below.

As a consequence, the formal basic definitions [31] (see Figure 2.6) should be adapted to the hierarchical top-down approach:

Definition 1. *Let the graph M denote the system under test (SUT) as a multilayer projection network:*

$$M = (V, E)$$

where M is a multi-layered 3D graph (see Figure 3.1), derived from the SUT specification; $V(M)$ is a finite, non-empty set of components of SUT; and $E(M)$ is a finite, non-empty set of component-to-component interconnections. In turn:

$$V = \bigcup_{\alpha=1}^{L} V^{\alpha}$$

and:

$$E = \left(\bigcup_{\alpha=1}^{L} E^{\alpha} \right) \bigcup \left(\bigcup_{\alpha=2}^{L} E^{\alpha,(\alpha-1)} \right)$$

where V^{α} is a finite, non-empty set of components of SUT on layer α; E^{α} is a finite, non-empty set of intralayer component-to-component interconnections on layer α; $E^{\alpha,(\alpha-1)}$ is a finite, non-empty set of interlayer relations (so called projections) between components of the layer α and the layer below $(\alpha-1)$; and L is the number of SUT layers $(1 \leq \alpha \leq L)$.

Therefore, two main elements of multilayer networks are (1) intra-layer graphs and (2) inter-layer graphs [35].

In general, the intralayer subgraph G^{α} of M can be represented as [31]:

$$G^{\alpha} = (V^{\alpha}, E^{\alpha})$$

where V^{α} is a finite, non-empty set of components on layer α; and $E^{\alpha} \subseteq V^{\alpha} \times V^{\alpha}$ is a finite, non-empty set of intralayer component-to-component interconnections on layer α. However, in practice, intralayer subgraphs G^{α} are usually not monolithic structures:

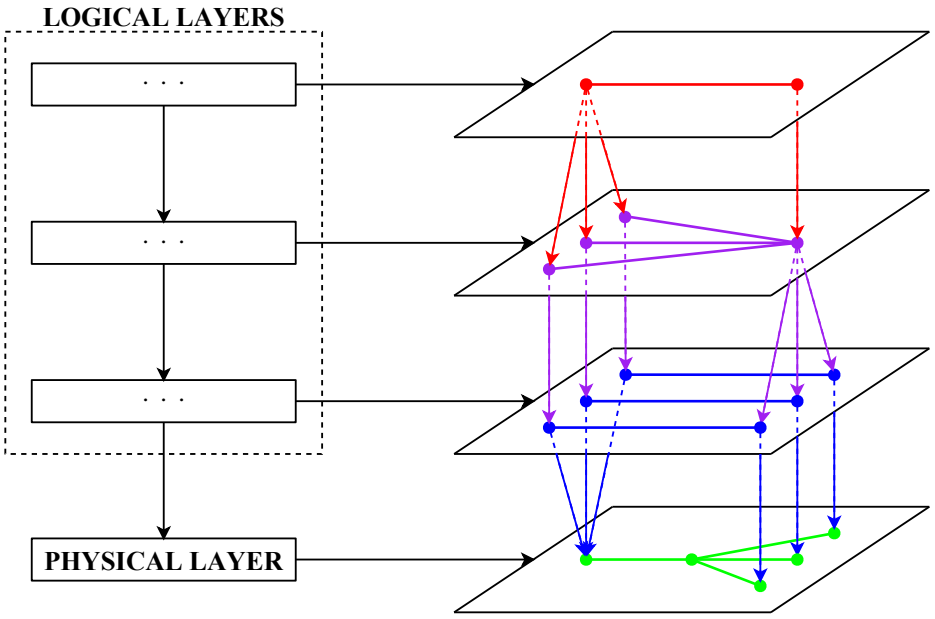


FIGURE 3.1: Hierarchical multilayer model [125] [126].

a set of protocols is predefined for each (physical or virtual) layer[2]. Moreover, these protocols can support different topologies. As a consequence, each intralayer subgraph G^{α} consists of a fixed set of components connected by different types of information links.

Types of multilayer network of particular relevance for this case are *multiplex* [34] [38] [31] or *multidimensional* [43] [31] networks, in which different layers represent different types of component-to-component interconnections. In the context of this thesis, specific types of interconnections and their properties are described by graph labels which, in turn, are represented by subsets of attributes. Hence:

Definition 2. *Let the subgraph G^{α} denote a layer of SUT as:*

$$G^{\alpha} = \left(V^{\alpha}, E^{\alpha}, S_V^{\alpha}, S_E^{\alpha}\right)$$

where G^{α} is a labeled intralayer subgraph of M; V^{α} is a finite, non-empty set of components on layer α; $E^{\alpha} \subseteq V^{\alpha} \times V^{\alpha}$ is a finite, non-empty set of intralayer component-to-component interconnections on layer α; S_V^{α} is a vertex label set for layer α; and S_E^{α}

[2]For example: a wireless access point (AP) must support at least two different protocols: (1) one for wired and (2) one for wireless communications.

is an edge label set for layer α. In this case:

$$S_V^\alpha = \bigcup_{v_i^\alpha \in V^\alpha} S_i^\alpha$$

where $S_i^\alpha \subset \boldsymbol{S^\alpha}$ is a finite non-empty set of specifications of SUT components (a set of supported communication protocols) that defines the label of the vertex v_i^α of G^α; and $\boldsymbol{S^\alpha}$ is the universal set of all possible communication protocols on layer α. Let $S_{i,j}^\alpha \subset \boldsymbol{S^\alpha}$ be a finite non-empty set of specifications of component-to-component interconnections (the set of used communication protocols) that defines the label of the edge $\left\langle v_i^\alpha, v_j^\alpha \right\rangle$ of G^α. Also, let G_β^α be a sub-subgraph which is defined by a given communication protocol $s_\beta^\alpha \in S_E^\alpha \subset \boldsymbol{S^\alpha}$; and $E_\beta^\alpha \subseteq E^\alpha$ be a finite, non-empty set of intralayer component-to-component interconnections on sub-layer β of layer α. In this case, G^α is represented as a multiplex network (see Figure 3.2):

$$G^\alpha = \bigcup_{\beta=1}^{|S_E^\alpha|} G_\beta^\alpha$$

and:

$$G_\beta^\alpha = \left(V^\alpha, E_\beta^\alpha\right)$$

In turn:

$$S_E^\alpha = \bigcup_{\left\langle v_i^\alpha, v_j^\alpha \right\rangle \in E^\alpha} S_{i,j}^\alpha$$

and:

$$S_{i,j}^\alpha = \bigcup_{\substack{\left\langle v_i^\alpha, v_j^\alpha \right\rangle \in E^\alpha \\ s_\beta^\alpha \in S_i^\alpha \\ s_\beta^\alpha \in S_j^\alpha}} \left\{ s_\beta^\alpha \right\}$$

If an edge $\left\langle v_i^\alpha, v_j^\alpha \right\rangle \in E^\alpha$ belongs to G_β^α then both components v_i^α and v_j^α support this protocol, i.e. $s_\beta^\alpha \in S_i^\alpha$ and $s_\beta^\alpha \in S_j^\alpha$ (each pair of components v_i^α and v_j^α can be connected by at most $|S_E^\alpha|$ possible edges).

The following assumptions can decrease the complexity of interlayer subgraphs in practice:

– In the context of this thesis, loops (or edges $\langle v_i^\alpha, v_i^\alpha \rangle$ of G^α) represent the internal structures of components (internal data flows) which are beyond the scope of this

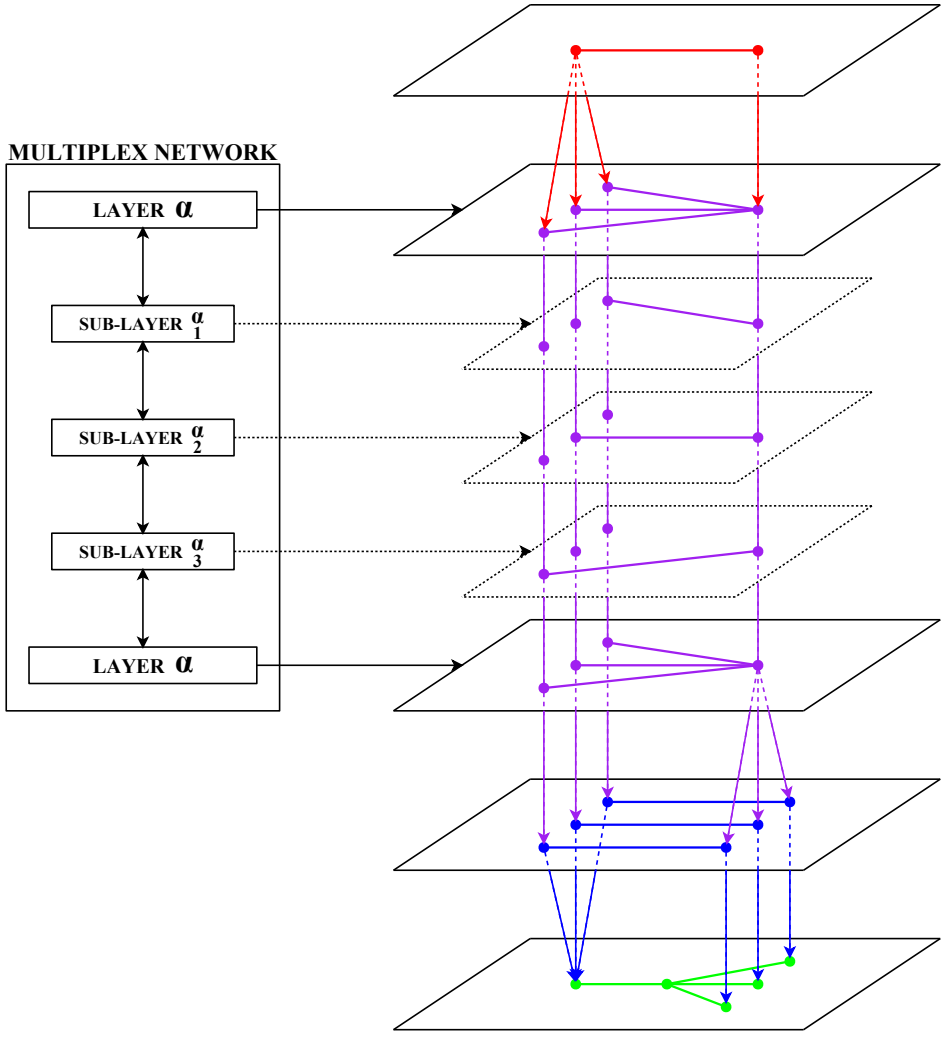


FIGURE 3.2: Intralayer subgraph representation as a multiplex network [126].

work, i.e. $\langle v_i^\alpha, v_i^\alpha \rangle \equiv v_i^\alpha$ (loops are disallowed).

– All multiple component-to-component interconnections between the same components v_i^α and v_j^α of G^α that are based on the same communication protocol (the representation of link combining/aggregation technologies [127]) should be represented as an edge $\left\langle v_i^\alpha, v_j^\alpha \right\rangle$ of G^α. In the context of this thesis, the label of the edge $S_{i,j}^\alpha$ represents both: (1) the used communication protocol; and (2) the used aggregation protocol.

– Also in the context of this thesis, in the case of the analysis of an intralayer subgraph G^α as a whole (i.e. without separation G^α into constituent sub-subgraphs), all multiple component-to-component interconnections between the same components v_i^α and v_j^α of G^α that are based on different communication protocols is

represented as one edge $\left\langle v_i^\alpha, v_j^\alpha \right\rangle$ of G^α. In turn, the label of the edge $S_{i,j}^\alpha$ represents all used communication protocols.

Hence, pro forma intralayer subgraphs can be defined as *simple graphs*[3].

In order to avoid misunderstandings, the following examples clarify the usage of label symbols:

- The vertex label S_1^3 represents the set of communication protocols that is supported by the component $v_1^3 \in V^3$ on the layer 3. In turn, the label S_V^3 represents the set of communication protocols that is supported by all components $v_i^3 \in V^3$ on the layer 3, i.e. $S_1^3 \subset S_V^3$. Moreover, the set $\boldsymbol{S^3}$ represents all possible communication protocols (standard and proprietary) that can be used on the layer 3, i.e. $S_1^3 \subset S_V^3 \subset \boldsymbol{S^3}$.
- The edge label $S_{1,5}^3$ represents the set of communication protocols that is used for communication between adjacent components v_1^3 and v_5^3 (the edge $\langle v_1^3, v_5^3 \rangle \in E^3$) on the layer 3. In turn, the label S_E^3 represents the set of communication protocols that is used for communication between all adjacent components $\left\langle v_i^3, v_j^3 \right\rangle \in E^3$ on the layer 3, i.e. $S_{1,5}^3 \subset S_E^3 \subset \boldsymbol{S^3}$.

Definition 3. *Let the subgraph $G^{\alpha,(\alpha-1)}$ denote a cross-layer of SUT as*

$$G^{\alpha,(\alpha-1)} = \left(V^\alpha, V^{(\alpha-1)}, E^{\alpha,(\alpha-1)}\right)$$

where $G^{\alpha,(\alpha-1)}$ is an interlayer bipartite subgraph of M; V^α is a finite, non-empty set of components on layer α, $V^{(\alpha-1)}$ is a finite, non-empty set of components on layer $(\alpha-1)$; and $E^{\alpha,(\alpha-1)} \subseteq V^\alpha \times V^{(\alpha-1)}$ is a finite, non-empty set of interlayer relations (all sets of projections) between components of the layer α $(2 \leq \alpha \leq L)$ and the layer below $(\alpha-1)$.

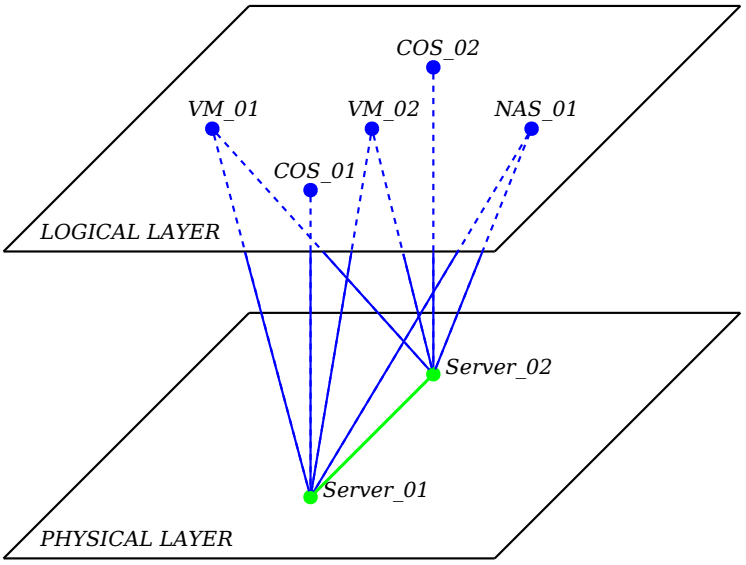
The degree (or valency) of vertices of $G^{\alpha,(\alpha-1)}$ represents the technological solutions which were used to build the system [125]:

- $d\left(v_i^\alpha\right) > 1; v_i^\alpha \in V^\alpha$ - clustering technology representation (see Figure 3.3);
- $d\left(v_j^{(\alpha-1)}\right) > 1; v_j^{(\alpha-1)} \in V^{(\alpha-1)}$ - virtualization and replication technologies representation (see Figure 3.4 and Figure 3.5);

[3] A simple graph is an undirected graph in which both multiple edges and loops are disallowed [128].

– $d\left(v_i^{\alpha}\right) = d\left(v_j^{(\alpha-1)}\right) = 1; \left\langle v_i^{\alpha}, v_j^{(\alpha-1)} \right\rangle \in E^{\alpha,(\alpha-1)}$ - a special case of dedicated components.

Hence, pro forma computer networks cannot be defined as *hierarchical multilayer networks*[4] but just *multilayer networks*.



Server_XX - Host (Cluster Member)
COS_XX - Console Operating System (Host Operating System)
VM_XX - Virtual Machine (Guest Operating System or Container)
NAS_XX - Network-Attached Storage (Network File System)

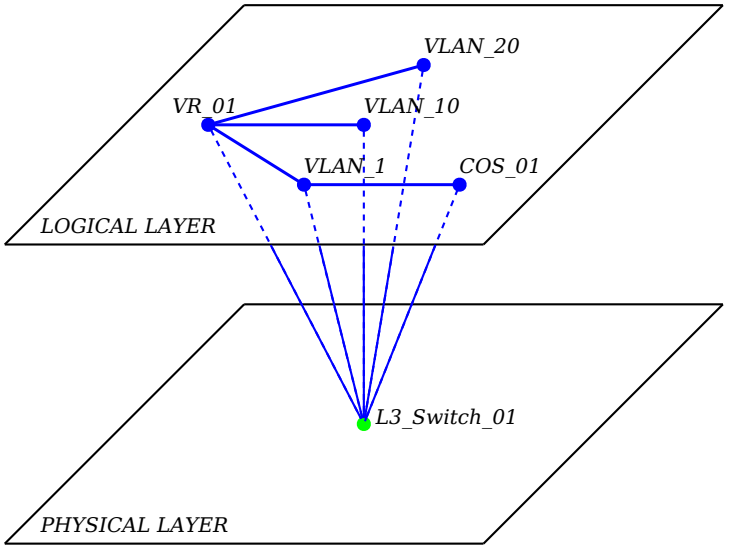
FIGURE 3.3: Hardware cluster example [125].

Based on Definition 2 and Definition 3, SUT can be represented as:

$$M = \left(\bigcup_{\alpha=1}^{L} G^{\alpha}\right) \bigcup \left(\bigcup_{\alpha=2}^{L} G^{\alpha,(\alpha-1)}\right) = \left(\bigcup_{\alpha=1}^{L} \left(\bigcup_{\beta=1}^{|S_E^{\alpha}|} G_{\beta}^{\alpha}\right)\right) \bigcup \left(\bigcup_{\alpha=2}^{L} G^{\alpha,(\alpha-1)}\right)$$

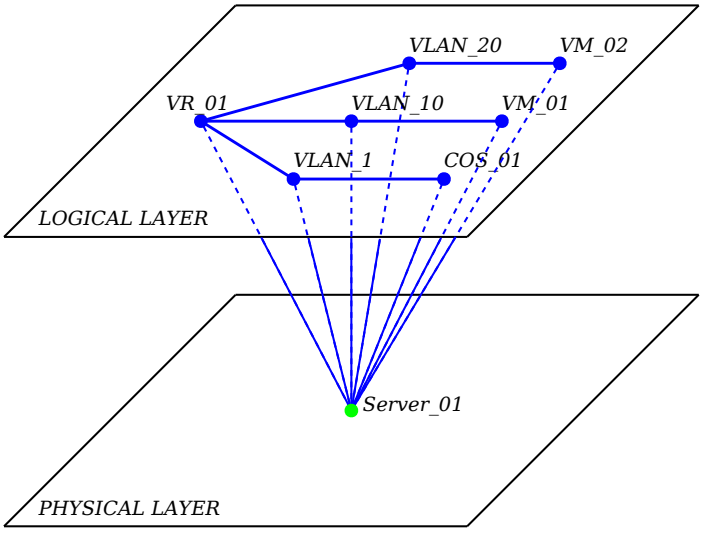
From the perspective of MBT, intralayer subgraphs G^{α} are the main source of initial data for the test case generation process; and interlayer subgraphs $G^{\alpha,(\alpha-1)}$ make this process consistent on all layers of the formal model (see Section 4.1).

[4]Hierarchy is a structure in which components are ranked into levels of subordination; each component has zero, one, or more subordinates; and no component has more than one superordinate component [17].



L3_Switch_XX - Multilayer Ethernet Switch
COS_XX - Console Operating System (Firmware)
VLAN_XX - Virtual Local Area Network
VR_XX - Virtual Router

FIGURE 3.4: Network virtualization example [125].



Server_XX - Virtualization Host
COS_XX - Console Operating System (Hypervisor or Host Operating System)
VM_XX - Virtual Machine (Guest Operating System)
VLAN_XX - Virtual Local Area Network
VR_XX - Virtual Router

FIGURE 3.5: Host virtualization example [125]. The existence of virtual routers (VR) depends on implementation details: OpenStack [129] supports them but VMware vSphere [130].

3.2 Reference Models

The ISO/OSI Reference Model (OSI RM) [131] was developed years ago for application developers, equipment manufacturers and network protocol vendors as an open standard for constructing network devices and applications/services that can work together. The model partitions computing systems into seven abstraction layers:

1. Physical Layer;

2. Data Link Layer;

3. Network Layer;

4. Transport Layer;

5. Session Layer;

6. Presentation Layer;

7. Application Layer.

However, this conceptual model has never been implemented in practice. Instead, the increasing popularity of TCP/IP based networking has led hardware and software developers to use the TCP/IP Protocol Suite (or Five-layer Reference Model) [132] [133] [134], the five layers of which are based on OSI RM - Layers 5 through 7 are collapsed into the Application Layer (see Figure 3.6).

On the other hand, network architecture representation should be as clean and simple to understand as it can be. As a consequence (in contrast to the developer community) the business community (end-users) faces the following challenges [125]:

- Physical Layer and Data Link Layer cannot be separated in the case of commercial off-the-shelf (COTS) network equipment.
- Transport Layer and Application Layer cannot be separated in the case of COTS software.

Moreover, end-users do not need services and applications themselves - they need tools to solve their business problems. However, neither OSI RM nor TCP/IP Protocol Suite

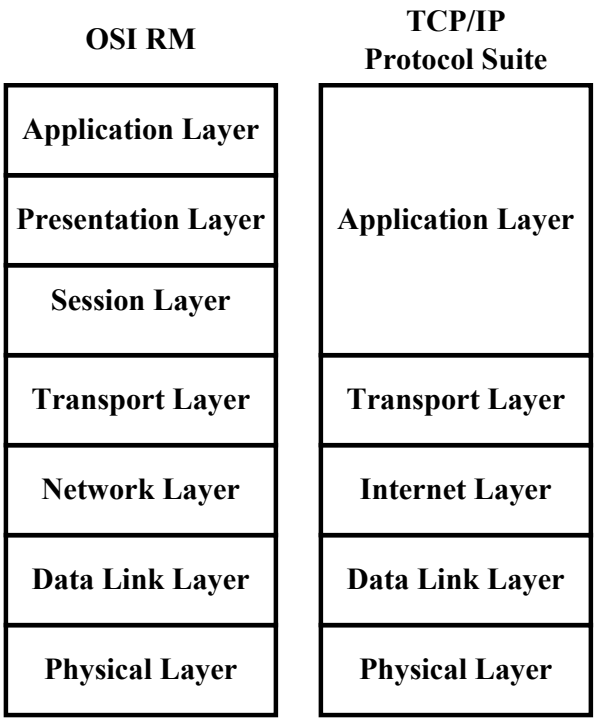


FIGURE 3.6: ISO/OSI Reference Model [131] and TCP/IP Protocol Suite (Five-layer Reference Model) [132] [133] [134].

provides a layer to represent the increased viewpoint of end-users (business goals). In fact, a common joke is that OSI RM should have three additional layers [6]:

8. User Layer;

9. Financial Layer;

10. Political Layer.

In practice, computer networks focus on solving problems at Layer 10 (but they are usually limited by Layer 9).

As a consequence, we should define an additional layer that can represent the increased system functionality or business goals. The basic multilayer reference model is shown in Figure 3.7. From the viewpoint of the hierarchical multilayer network, the physical layer constitutes a physical network and the logical, service and functional layers are virtual layers that operate on top of the physical layer.

Unfortunately, the basic model does not take into account the environment impact that might be critical in some cases[5]. The problem can be solved by two additional layers [136]:

- The engineering environment layer. This layer defines external engineering systems that are vital for normal operation of physical networks.
- The social environment layer (or Layer 8 of OSI RM [6]). This layer defines organization infrastructures or *human networks* [53].

It is important to note that all these additional layers - functional, social and engineering - lie beyond the ISO/OSI RM and the TCP/IP Protocol Suite but they provide a necessary complement to it with regard to applying the system methodology to network analysis.

Hence, the extended multilayer reference model can be stated as follows (see Figure 3.7):

- *The functional (or ready-for-use system) layer* defines functional components and their interconnections – the increased viewpoint of end-users/customers. This layer is based on functional models [1]:
  - service-provider architectural model [1];
  - intranet/extranet architectural model [1];
  - single-tiered/multi-tiered architectural model [1];
  - end-to-end architectural model [1].

  The number of sub-layers (see Definition **??**) reflects the number of system business goals. In turn, the lower bound of the sub-layer number strictly relies on the fact that each and every computing system must completely satisfy the two following viewpoints:
  - the viewpoint of the business community (end-users) - a system must be useful, i.e. it must solve (not create) business problems;

[5]For example in the case of security testing. It is obvious that security testing should cover all threats defined by the current revision of ISO/IEC 27005:2011 standard [135]. It is important to note that this list of typical threats covers both aspects (software-based and network-based) of computing systems but not only these aspects

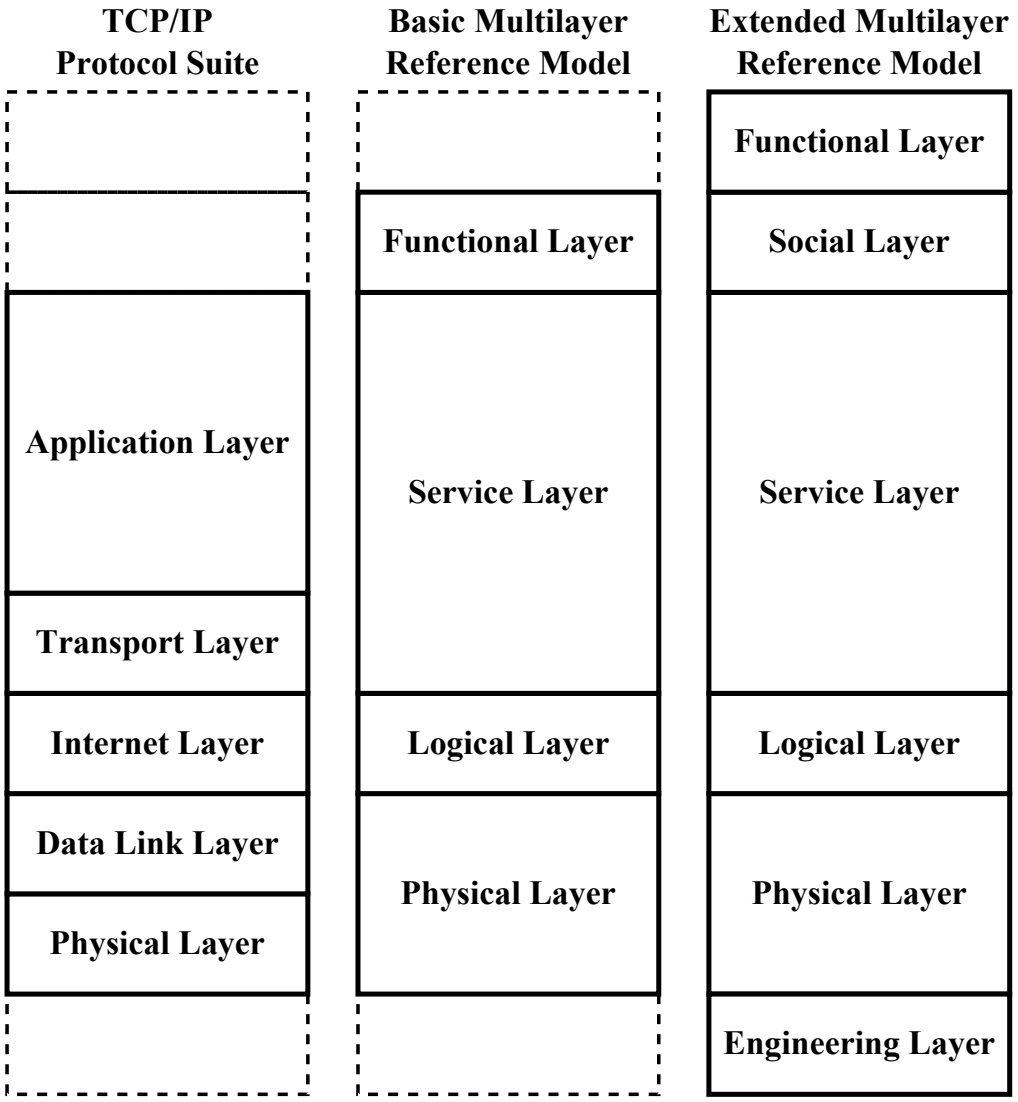


FIGURE 3.7: Multilayer reference models [126].

- the viewpoint of the IT personnel - a system must be fully controlled, maintainable and repairable.

On the other hand, the experimental psychological work based on a large number of experiments related to sensory perception concludes that humans can process about 5-9 levels of complexity [137]. This result can be used as the upper bound.

– *The social layer (optional)* defines organization infrastructures or *human (social) networks* [53]. It represents people or groups of people and their working relationships based on electronic communications (other types of communications are beyond the scope of this thesis). The layer is based on flow-based models [1]:

  - centralized architectural model [8];
  - decentralized architectural model [8].

  Social sub-layers represent different types of electronic communication such as video- and audio-conference, instant messaging and electronic mail, etc. As a consequence, the number of social sub-layers is defined by the institution/corporate requirements and/or state legislation.

- *The service layer* defines software-based components (services/applications) and their interconnections. It is based on flow-based models [1]:
  - client-server or centralized architectural model [8];
  - peer-to-peer or decentralized architectural model [8];
  - hybrid architectural model [8].

  In this case, sub-layers reveal services/applications and their associated TCP/UDP data flows (each sub-layer represents data flows for a particular number of TCP/UDP ports). In general, the upper bound is strictly defined by IANA as the range of TCP/UDP ports[6] [138].

- *The logical layer* defines logical (virtual) components and their interconnections. It is based on topological models [1]:
  - LAN/MAN/WAN architectural model [139];
  - core/distribution/access architectural model [140].

  Two main sub-layers represent logical topologies defined by IPv4 [141] and IPv6 [142] protocols. In turn, each logical topology might be divided (but it is not required) into subnets and their associated VLANs.

- *The physical layer* defines hardware (physical) components and their interconnections. Like its predecessor, this layer is based on topological models [1]. Two main sub-layers represent topologies defined by wired [143] and wireless [144] infrastructures. In turn, each physical topology might be decomposed (but it is not required) in accordance with used protocols and their associated technologies.

[6] In the domain of computer networks the upper bound can be defined by the range of standardized (well-known) and registered TCP/UDP ports only.

– *The engineering layer (optional)* defines external engineering systems (power supply systems, climate control systems, physical security systems, etc.), that are vital for normal operation of physical networks, and their relation with the physical networks. It is based on topological models [1], where engineering systems and physical networks are represented as individual components. In general, engineering sub-layers should represent different types of external engineering systems. However, in contrast to the logical and physical layers, the engineering layer usually does not require complicated architecture solutions, i.e. there are no reasons to divide it into sub-layers.

Chapter 4

Structural Test Case Generation Strategy

If you can't describe what you are doing as a process, you don't know what you're doing.

—William Edwards Deming

According to the goals of this thesis (see Section 1.5), the description of the structural test case generation strategy covers the following areas:

- The framework of the test case generation strategy including the detailed description of key elements.
- The formal definitions of the key elements and their relations.
- The formal definitions of test case generation strategy based on requirements coverage criteria which aim at the structure of the SUT.

4.1 Framework of Test Case Generation Strategy

The MBT framework which is used in this thesis is shown in Figure 2.9. In turn, Figure 4.1 represents the framework of structural test case generation strategy for a given layer of the formal model. The key elements of the both frameworks are:

- the formal model;

– test requirements;

– test cases.

According to the goals of this thesis (see Section 1.5), the formal model and test requirements are completely based on the SUT detailed design documentation (the unambiguous relations between these elements and the design documentation are discussed in Chapter 6). Unfortunately, design documentation might contain mistakes (it happens very often in practice). In turn, the proposed approach enables data processing based on wrong input representations (not the correct only). As a consequence, the criteria of consistency which allow selection of correct representations (errors/bugs detection) should be defined in conjunction with the framework key elements.

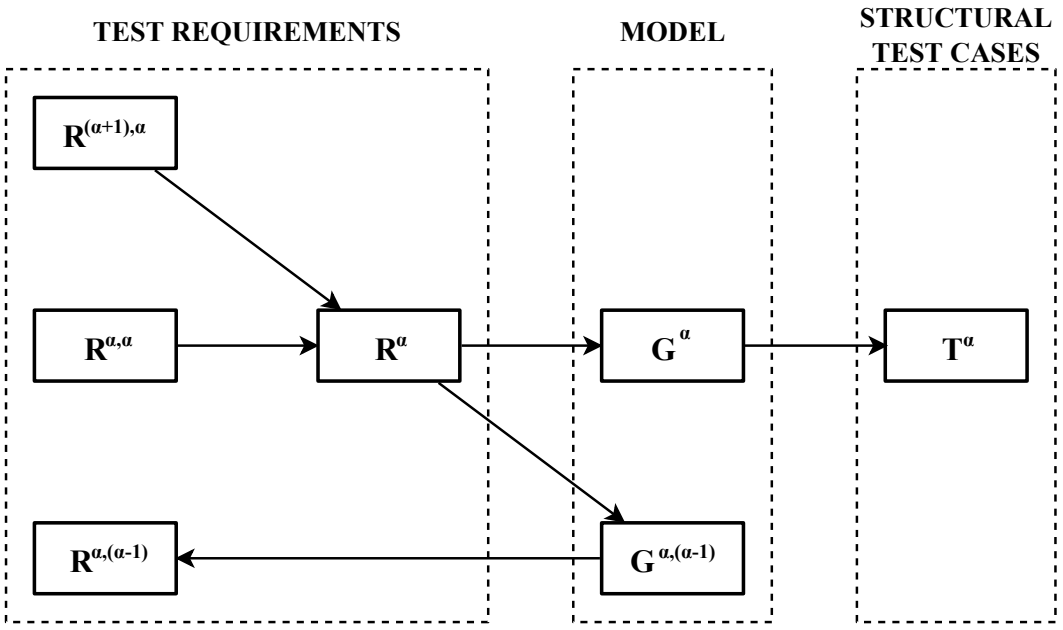


FIGURE 4.1: The framework of the structural test case generation strategy for a given layer α of the formal model where G^{α} is an intralayer subgraph; $G^{\alpha,(\alpha-1)}$ is an interlayer subgraph; $R^{(\alpha+1),\alpha}$ is a set of interlayer projections of test requirements from upper layers to layer α; $R^{\alpha,\alpha}$ is a set of intralayer test requirements (or the set of test requirement defined for layer α); R^{α} is a resulting set of test requirements for layer α); $R^{\alpha,(\alpha-1)}$ is a set of interlayer projections of test requirements from layer α to the layer below; and T^{α} is a set of test cases (abstract test specifications) which relate to the structure of the formal model on layer α.

4.1.1 Formal Model

As mentioned above (see Section 1.3), the quality of formal methods based on abstract models is limited by the quality of these models. Hence, the internal consistency of the formal model (as a part of the model validation report - see Figure 2.9) should be verified during the structural test case generation activities.

In the context of this thesis, the definition of the model internal consistency strictly relies on the following notions:

- The definition of consistency as the ability of parts of a system or component to be asserted together without contradiction [17].
- The definition of a communication protocol as a set of conventions that govern the interaction of processes, devices, and other components within a system [17].
- The concept of layered networks [36], i.e. the fact that a node on a given layer depends on a corresponding node (or nodes) on the layer below (with the exception of the bottom layer).
- The fact that the existence of isolated components is strictly against the definitions of computer networks [132] and distributed systems [8].

These ideas are formalized in Criterion 1:

Criterion 1. *The formal model based on the concept of multilayer networks is internally consistent on a given layer α iff:*

- *each vertex v_i^α of intralayer subgraphs G^α is incident with at least one edge of G^α, i.e. $d\left(v_i^\alpha \in G^\alpha\right) \geq 1$;*
- *each pair of adjacent vertices v_i^α and v_j^α of G^α which are incident with the edge $\left\langle v_i^\alpha, v_j^\alpha \right\rangle$ of G^α supports at least one common communication protocol, i.e. $S_{i,j}^\alpha \subseteq S_i^\alpha$; $S_{i,j}^\alpha \subseteq S_j^\alpha$ and $S_{i,j}^\alpha \neq \emptyset$;*
- *each vertex v_i^α of interlayer subgraphs $G^{\alpha,(\alpha-1)}$ $(2 \leq \alpha \leq L)$ is incident with at least one edge of $G^{\alpha,(\alpha-1)}$, i.e. $d\left(v_i^\alpha \in G^{\alpha,(\alpha-1)}\right) \geq 1$.*

In general, there are two possible results of applying Criterion 1:

- The formal model is internally consistent on a given layer. In this case, the model consistency with respect to the test requirements (the model external consistency) on the given layer should be checked during the structural test case generation activities (see Section 4.1.3).
- The formal model is internally inconsistent on a given layer. This case explicitly represents the existence of errors/bugs (at least one) in design documentation (technical specifications). As a consequence:
  - the design documentation should be corrected;
  - the formal model should be re-built and then re-checked using Criterion 1.

Limitations:

- The formal model based on the concept of multilayer networks is intended to specify heterogeneous structures and their properties. The behavioral aspects of computer networks have to be described using different techniques (these aspects are beyond the scope of this thesis).
- The SUT detailed design documentation should cover all coexisting architectural layers (according to the basic or extended multilayer reference model - see Section 3.2). Otherwise, the building (generation) of the formal model and, as a consequence, the application of the system methodology to network analysis is impossible.

4.1.2 Test Requirements

In general, formal models can define an infinite number of potential test cases due to their internal structures [64]. It turn, test selection criteria define the conditions that are used to control the generation of test cases, i.e. they determine what kind of test cases should be extracted from a possibly infinite universal set of all possible test cases.

Based on the fact that the usefulness of network/distributed systems does not depend on any particular part of these systems, but emerges from the way in which their components interact [23] [145], the structural model coverage criteria (which use the structure of formal models to select the test cases [15]) can be used as a starting point. In this case,

the strongest coverage criterion is the path coverage criterion[1]. Nevertheless, the full path coverage is in general impossible to achieve and impractical for real life testing [64].

In practice, different categories should be combined to complement one another so as to achieve the best test coverage [15]. The system methodology (see Section 1.2) allows eliminating the disadvantages of the path coverage criterion by using the requirements-based criteria. In this case, the test suite should cover only the paths which are defined by: (1) end-user requirements; and (2) requirements derived from technical specifications, i.e. defined by technological solutions used to build the SUT[2].

In turn, the standard ISO/IEC/IEEE Std 29148:2011 [123] defines:

- the term *requirement* as a statement which translates or expresses a need and its associated constraints and conditions;
- the term *condition* as a measurable qualitative or quantitative attribute that is stipulated for a requirement.

As a consequence, in the context of this thesis formal test requirements should determine: (1) objects as associated elements of the SUT structure; and (2) associated conditions of these objects (or requirement attributes). As mentioned above (see Section 3.1), the interlayer relations: (1) determine how the topological properties on different layers affect each other and, as a consequence, (2) represent technologies used to build the system (virtualization, clustering, replication, etc.). As a consequence, the result of applying the concept of layered networks [36] to the definition of the test requirements (see Section 1.4) introduces two sources of test requirements for a given layer of the formal model (see Figure 4.1):

$$R^{\alpha} = \left(R^{\alpha,\alpha} \cup R^{(\alpha+1),\alpha}\right)$$

where R^{α} is a set of test requirements for the given layer α; $R^{\alpha,\alpha}$ is a set of intralayer test requirements (or the set of test requirement defined for the layer α); and $R^{(\alpha+1),\alpha}$ is

[1]The path coverage criterion is satisfied by a test suite iff for selected paths in a formal model, the test suite contains at least one test case which covers this path [64].

[2]Requirements of that kind are defined by technological solutions that are used to build the SUT (i.e. they can be derived from technical specifications). For example: The technological solution represents the virtualization platform as VMware vSphere 6.0. As a consequence, the derived technical requirements state that ESXi instance on the service layer must communicate with: (1) Dynamic Name System (DNS) service; and (2) Network Time Protocol (NTP) service [146].

a set of interlayer projections of test requirements from upper layers to the layer α. In the context of this thesis, $R^{(\alpha+1),\alpha}$, $R^{\alpha,\alpha}$ (and $R^{\alpha,(\alpha-1)}$ - see Figure 4.1) have the same formal operational specifications (the same presentation format).

Limitation:

- The techniques of automated transforming of informal end-user requirements into formal test requirements are beyond the scope of this thesis. The problem requires a separate analysis - even in the case of relatively simple systems, it may not be a routine exercise in practice.
- Pro forma, the set of intralayer test requirements might be an empty set for all coexisting architectural layers of the formal model with the exception of the top layer (in practice, this layer usually represents the system business goals - see Section 3.2). The absence of formal test requirements for this layer makes applying the system methodology to network analysis impossible - there is no starting point for the test case generation strategy.

4.1.3 Test Cases

As mentioned above (see Section 1.4), test cases (or abstract test specification) are the results of applying (binding) test requirements to the formal model. As a consequence:

- Similar to the test requirements, the test cases should determine:
  - objects as associated elements of the formal model;
  - associated specifications of these objects.
- The presentation format of the test requirement objects (or elements of the SUT structure) should be fully compatible with the presentation format of the test case objects (or elements of the formal model).
- The presentation format of the requirement attributes should be fully compatible with the presentation format of the specifications of the formal model.

In the context of the thesis, these ideas are formalized in Criterion 2:

Criterion 2. *A test requirement induces a test case on a given layer α iff:*

– *the object defined by the test requirement for the layer α binds an element (at least one) of the formal model on the layer α;*

– *the specifications of the bound element match the requirement attributes on the layer α.*

In turn, system decomposition into objects which interact is a common baseline for all technologies for the design and implementation of distributed systems [121]. These two aspects (components and links) of knowledge can be defined as associated elements of the intralayer subgraphs: (1) SUT components as vertices; and (2) SUT links (or communication channels[3]) as paths.

It is important to note that communication channels should be represented by the paths in the multilayer (3D) graph, i.e. two SUT components can communicate iff there is a path between these components. Cycles and, as a consequence, trails and walks[4] cannot exist in computer networks which usually have the necessary protection mechanisms (such as Spanning-Tree Protocols and Routing Protocols [2] [132] [133]).

Also in the context of this thesis, the definition of the model consistency with respect to the test requirements (the model external consistency) strictly relies on the following notions:

– The definition of consistency as the ability of parts of a system or component to be asserted together without contradiction [17].

– The definition of a system as a collection of components (machine, software, human, etc.) which cooperate in an organized way to achieve a desired result - the end-user requirements [18].

These ideas are formalized in Criterion 3:

Criterion 3. *The formal model based on the concept of multilayer networks is externally consistent on a given layer α with respect to the test requirements iff each test requirement defined for the layer α initiates at least one test case on the layer α.*

[3] A communication channel (or channel) is a configuration of stubs, binders, protocol objects and interceptors providing a binding between a set of interfaces to basic engineering objects, through which interaction can occur [17].

[4] A (v_0, v_i)–*walk* in a graph G is an alternating sequence $[v_0, e_1, v_1, e_2, \ldots, v_{(i-1)}, e_i, v_i]$ of vertices and edges from G with $e_l = \langle v_{(l-1)}, v_l \rangle$. In a *closed walk* $v_0 = v_i$. A *trail* is a walk in which all edges are distinct. A *path* is a trail in which also all vertices are distinct. A *cycle* is a closed trail in which all vertices except v_0 and v_i are distinct. [128]

In general, there are two possible results of applying Criterion 3:

- The formal model is externally consistent with respect to the test requirements on a given layer. In this case, all test requirements defined for the layer are covered by test cases (abstract test specifications) which relate to the structure of the formal model on the layer. The set of test cases should be included in the SUT design documentation.
- The formal model is externally inconsistent on a given layer. This case explicitly represents the existence of errors/bugs (at least one) in design documentation (test requirements and/or technical specifications). As a consequence:
  - the design documentation should be corrected;
  - the formal model should be re-built and then re-checked using Criterion 1 (if necessary) and Criterion 3.

Criterion 4. *The formal model based on the concept of multilayer networks is consistent with respect to the test requirements iff:*

- *there is at least one test requirement defined for the top architectural layer of the formal model;*
- *the formal model is internally consistent on all coexisting architectural layers;*
- *the formal model is externally consistent with respect to the test requirements on all coexisting architectural layers.*

In the context of the thesis, Criterion 4 defines the model validation report - see Figure 2.9.

Limitations:

- The quality of test cases is limited by the quality of formal models and test requirements.

– Based on Criteria 1 and 3 it is possible to detect the potential sources of primary (incorrect design) and secondary (incorrect requirements) faults[5]. The potential sources of command faults (the behavioral aspects of computer networks) are beyond the scope of this thesis due to the properties of the formal model.

4.2 Formal Definitions

The following formal definitions should be determined according to the framework of structural test case generation strategy (see Figure 4.1):

– model-based definitions;

– definitions of test requirements;

– definitions of test cases.

These definitions will be used later as integral components of test generation strategies.

4.2.1 Model-Based Definitions

As mentioned above (see Section 4.1.3), two SUT components can communicate if there is a path in the formal model between these components. In turn, dependable computing systems incorporate protection mechanisms to tolerate faults that could cause systems failures (see Section 2.3). As a consequence, in general, there are some paths (at least one) between each pair of components which can communicate.

Definition 4. *Let $P^{\alpha}_{i,j}$ denote the set of communication channels (data flows) between a pair of SUT dedicated components v^{α}_{i} and v^{α}_{j} of G^{α} which can communicate as follows:*

$$P^{\alpha}_{i,j} = \bigcup_{k=1}^{K^{\alpha}_{i,j}} \left\{ p^{\alpha}_{i,j,k} \right\}$$

[5]In the domain of computers: primary faults occur when errors result in the computer output not meeting its specification (incorrect design); secondary faults occur when the computer gets input that differ from what was anticipated or designed (incorrect requirements); and command faults occur when the computer responds to erroneous inputs that are expected but occur at the wrong time or in the wrong order. [3]

where $p^{\alpha}_{i,j,k}$ *is a* k^{th} $(v^{\alpha}_0, v^{\alpha}_i)$*-path*[6] *in* G^{α}*; and* $K^{\alpha}_{i,j}$ *is the finite number of duplicated (parallel/redundant) paths* $p^{\alpha}_{i,j,k}$.

In other words, each pair of SUT components v^{α}_i and v^{α}_j can be connected by at most $K^{\alpha}_{i,j}$ possible $\left(v^{\alpha}_i, v^{\alpha}_j\right)$-paths (the value of the variable $K^{\alpha}_{i,j}$ is dependent on the layer topology)[7]. In turn:

$$P^{\alpha} = \bigcup P^{\alpha}_{i,j}$$

where P^{α} is the completed set of communication channels on a given layer α.

4.2.2 Definitions of Test Requirements

As mentioned above (see Section 4.1.2), the framework of structural test case generation strategy for a given layer of the formal model introduces two sources of test requirements (see Figure 4.1):

$$R^{\alpha} = \left(R^{\alpha,\alpha} \cup R^{(\alpha+1),\alpha}\right)$$

where R^{α} is a set of test requirements for the given layer α; $R^{\alpha,\alpha}$ is a set of intralayer test requirements (or the set of test requirements defined for the layer α); and $R^{(\alpha+1),\alpha}$ is a set of interlayer test requirements (or the set of interlayer projections of test requirements from upper layers to the layer α). It is important to note that R^{α} represents the union of test requirements. The possible test requirement aggregation is beyond the scope of this thesis.

Moreover, test requirements should cover (see Section 4.1.3): (1) SUT components; and (2) SUT communication channels. Hence, the sets of test requirements $R^{\alpha,\alpha}$ and $R^{(\alpha+1),\alpha}$ on layer α can be defined as:

$$R^{\alpha,\alpha} = \left(R^{\alpha,\alpha}_{comp} \cup R^{\alpha,\alpha}_{link}\right)$$

and:

$$R^{(\alpha+1),\alpha} = \left(R^{(\alpha+1),\alpha}_{comp} \cup R^{(\alpha+1),\alpha}_{link}\right)$$

[6] A $(v^{\alpha}_0, v^{\alpha}_i)$-path in a graph G^{α} is an alternating sequence $[v_0, e_1, v_1, e_2, \ldots, v_{(i-1)}, e_i, v_i]$ of vertices and edges from G^{α} with $e_l = \langle v_{(l-1)}, v_l \rangle$ in which all vertices and edges are distinct [128].

[7] In the real engineering world under financial constraints commercial systems are usually based on redundant architectures [147], i.e. in most cases $K^{\alpha}_{i,j} = 2$.

where $R_{comp}^{\alpha,\alpha}$ is a set of intralayer test requirements of SUT components; $R_{link}^{\alpha,\alpha}$ is a set of intralayer test requirements of SUT communication channels; $R_{comp}^{(\alpha+1),\alpha}$ is a set of interlayer test requirements of SUT components; and $R_{link}^{(\alpha+1),\alpha}$ is a set of intralayer test requirements of SUT communication channels. In turn:

Definition 5. *Let* $R_{comp}^{\alpha,\alpha} = \left\{ r_{n,comp}^{\alpha,\alpha} \right\}$ *denote the set of intralayer test requirements for SUT components as a set of triplets (3-tuples):*

$$R_{comp}^{\alpha,\alpha} = \left\{ \left(v_i^{\alpha}, A_i^{\alpha}, A_i^{\alpha,(\alpha-1)} \right) \right\}$$

where v_i^{α} *is a component of SUT on layer* α*;* $A_i^{\alpha} \subset \boldsymbol{S^{\alpha}}$ *is a set of required attributes for* v_i^{α}*; and* $A_i^{\alpha,(\alpha-1)} \subset \boldsymbol{S^{(\alpha-1)}}$ *is a set of required attributes for any interlayer projection of* v_i^{α} *on layer* $(\alpha - 1)$*.*

Definition 6. *Let* $R_{link}^{\alpha,\alpha} = \left\{ r_{n,link}^{\alpha,\alpha} \right\}$ *denote the set of intralayer test requirements for SUT communication channels as a set of quadruples (4-tuples):*

$$R_{link}^{\alpha,\alpha} = \left\{ \left(v_i^{\alpha}, v_j^{\alpha}, A_{i,j}^{\alpha}, A_{i,j}^{\alpha,(\alpha-1)} \right) \right\}$$

where v_i^{α} *and* v_j^{α} *is a pair of SUT dedicated components on layer* α *which must communicate;* $A_{i,j}^{\alpha} \subset \boldsymbol{S^{\alpha}}$ *is a set of required attributes for* $\left(v_i^{\alpha}, v_j^{\alpha} \right)$*-path; and* $A_{i,j}^{\alpha,(\alpha-1)} \subset \boldsymbol{S^{(\alpha-1)}}$ *is a set of required attributes for any interlayer projection of* $\left(v_i^{\alpha}, v_j^{\alpha} \right)$*-path on layer* $(\alpha - 1)$*.*

Definition 7. *Let* $R_{comp}^{(\alpha+1),\alpha} = \left\{ r_{n,comp}^{(\alpha+1),\alpha} \right\}$ *denote the set of interlayer projections of test requirements* $R_{comp}^{(\alpha+1),(\alpha+1)} = \left\{ \left(v_k^{(\alpha+1)}, A_k^{(\alpha+1)}, A_k^{(\alpha+1),\alpha} \right) \right\}$ *for SUT components from layer* $(\alpha + 1)$ *to layer* α *as a set of triplets:*

$$R_{comp}^{(\alpha+1),\alpha} = \left\{ \left(v_i^{\alpha}, A_i^{\alpha}, A_i^{\alpha,(\alpha-1)} \right) \right\}$$

where $v_k^{(\alpha+1)}$ *is a component of SUT on layer* $(\alpha+1)$*;* v_i^{α} *is a corresponding component of* $v_k^{(\alpha+1)}$ *on layer* α*;* $A_i^{\alpha} \subset \boldsymbol{S^{\alpha}}$ *is a set of required attributes for* v_i^{α}*; and* $A_i^{\alpha,(\alpha-1)} \subset \boldsymbol{S^{(\alpha-1)}}$ *is a set of required attributes for any interlayer projection of* v_i^{α} *on layer* $(\alpha - 1)$*.*

Definition 8. *Let* $R_{link}^{(\alpha+1),\alpha} = \left\{ r_{n,link}^{(\alpha+1),\alpha} \right\}$ *denote the set of interlayer projections of test*

requirements $R_{link}^{(\alpha+1),(\alpha+1)} = \left\{\left(v_k^{(\alpha+1)}, v_l^{(\alpha+1)}, A_{k,l}^{(\alpha+1)}, A_{k,l}^{(\alpha+1),\alpha}\right)\right\}$ *for SUT communication channels from layer* $(\alpha+1)$ *to layer* α *as a set of quadruples:*

$$R_{link}^{(\alpha+1),\alpha} = \left\{\left(v_i^{\alpha}, v_j^{\alpha}, A_{i,j}^{\alpha}, A_{i,j}^{\alpha,(\alpha-1)}\right)\right\}$$

where $v_k^{(\alpha+1)}$ *and* $v_l^{(\alpha+1)}$ *is a pair of SUT dedicated components on layer* $(\alpha+1)$ *which must communicate;* v_i^{α} *and* v_j^{α} *is a pair of corresponding components of* $v_k^{(\alpha+1)}$ *and* $v_l^{(\alpha+1)}$ *on layer* α*;* $A_{i,j}^{\alpha} \subset \boldsymbol{S^{\alpha}}$ *is a set of required attributes for* $\left(v_i^{\alpha}, v_j^{\alpha}\right)$*-path; and* $A_{i,j}^{\alpha,(\alpha-1)} \subset \boldsymbol{S^{(\alpha-1)}}$ *is a set of required attributes for any interlayer projection of* $\left(v_i^{\alpha}, v_j^{\alpha}\right)$*-path on layer* $(\alpha-1)$*.*

The presentation format of the requirement attributes (see Definitions 5 - 8) should be fully compatible with the presentation format of the specifications of the formal model (see Definition 2), i.e. $A_i^{\alpha}, A_{i,j}^{\alpha} \subset \boldsymbol{S^{\alpha}}$ and $A_i^{\alpha,(\alpha-1)}, A_{i,j}^{\alpha,(\alpha-1)} \subset \boldsymbol{S^{(\alpha-1)}}$.

In general, a set of requirement attributes can be an empty set. In this case, the test requirement expresses the need for object (component or communication channel) existence only.

As mentioned above (see Section 4.1.2), the interlayer relations (projections) determine how the topological properties on different layers affect each other. In the context of the thesis, this fact is represented by the following functions:

Definition 9. *The function* $\mu_{comp}^{(\alpha+1),\alpha} : R_{comp}^{(\alpha+1),(\alpha+1)} \times G^{(\alpha+1),\alpha} \to R_{comp}^{(\alpha+1),\alpha}$ *is defined as follows:*

$$\mu_{comp}^{(\alpha+1),\alpha}\left(\left(v_k^{(\alpha+1)}, A_k^{(\alpha+1)}, A_k^{(\alpha+1),\alpha}\right)\right) = \bigcup_{\substack{\left\langle v_k^{(\alpha+1)}, v_i^{\alpha}\right\rangle \in G^{(\alpha+1),\alpha} \\ A_i^{\alpha} = A_k^{(\alpha+1),\alpha},\, A_i^{\alpha,(\alpha-1)} = \emptyset}} \left\{\left(v_i^{\alpha}, A_i^{\alpha}, A_i^{\alpha,(\alpha-1)}\right)\right\};$$

In other words, for each test requirement $\left(v_k^{(\alpha+1)}, A_k^{(\alpha+1)}, A_k^{(\alpha+1),\alpha}\right) \in R_{comp}^{(\alpha+1),(\alpha+1)}$ on layer $(\alpha+1)$ the function $\mu_{comp}^{(\alpha+1),\alpha}$ determines the finite set of all possible corresponding triplets $\left(v_i^{\alpha}, A_i^{\alpha}, A_i^{\alpha,(\alpha-1)}\right) \in R_{comp}^{(\alpha+1),\alpha}$ on layer α where $A_i^{\alpha} = A_k^{(\alpha+1),\alpha}$ and $A_i^{\alpha,(\alpha-1)} = \emptyset$. If this set is an empty set then the formal model M is inconsistent according to Criterion 1.

Definition 10. *The function* $\mu_{link}^{(\alpha+1),\alpha} : R_{link}^{(\alpha+1),(\alpha+1)} \times G^{(\alpha+1),\alpha} \rightarrow R_{link}^{(\alpha+1),\alpha}$ *is defined as follows:*

$$\mu_{link}^{(\alpha+1),\alpha}\left(\left(v_k^{(\alpha+1)}, v_l^{(\alpha+1)}, A_{k,l}^{(\alpha+1)}, A_{k,l}^{(\alpha+1),\alpha}\right)\right) = \bigcup_{\substack{\left\langle v_k^{(\alpha+1)}, v_i^{\alpha}\right\rangle \in G^{(\alpha+1),\alpha} \\ \left\langle v_l^{(\alpha+1)}, v_j^{\alpha}\right\rangle \in G^{(\alpha+1),\alpha} \\ A_{i,j}^{\alpha} = A_{k,l}^{(\alpha+1),\alpha}, A_{i,j}^{\alpha,(\alpha-1)} = \emptyset}} \left\{\left(v_i^{\alpha}, v_j^{\alpha}, A_{i,j}^{\alpha}, A_{i,j}^{\alpha,(\alpha-1)}\right)\right\};$$

In other words, for each test requirement $\left(v_k^{(\alpha+1)}, v_l^{(\alpha+1)}, A_{k,l}^{(\alpha+1)}, A_{k,l}^{(\alpha+1),\alpha}\right) \in R_{link}^{(\alpha+1),(\alpha+1)}$ on layer $(\alpha+1)$ the function $\mu_{link}^{(\alpha+1),\alpha}$ determines the finite set of all possible corresponding quadruples $\left(v_i^{\alpha}, v_j^{\alpha}, A_{i,j}^{\alpha}, A_{i,j}^{\alpha,(\alpha-1)}\right) \in R_{link}^{(\alpha+1),\alpha}$ on layer α where $A_{i,j}^{\alpha} = A_{k,l}^{(\alpha+1),\alpha}$ and $A_{i,j}^{\alpha,(\alpha-1)} = \emptyset$. If this set is an empty set then the formal model M is inconsistent according to Criterion 1.

The sets of test requirements R_{comp}^{α} and R_{link}^{α} can be defined as:

$$R_{comp}^{\alpha} = \left(R_{comp}^{\alpha,\alpha} \cup R_{comp}^{(\alpha+1),\alpha}\right)$$

$$R_{link}^{\alpha} = \left(R_{link}^{\alpha,\alpha} \cup R_{link}^{(\alpha+1),\alpha}\right)$$

where R_{comp}^{α} is the set of test requirements of SUT components for the given layer α; and R_{link}^{α} is the set of test requirements of SUT communication channels for the given layer α.

4.2.3 Definitions of Test Cases

Similarly to test requirements, test cases should cover (see Section 4.1.3): (1) SUT components; and (2) SUT communication channels, i.e. the completed set of test cases T^{α} for a given layer α (see Figure 4.1) can be defined as:

$$T^{\alpha} = \left(T_{comp}^{\alpha} \cup T_{link}^{\alpha}\right)$$

Hence:

Definition 11. *Let* $T^{\alpha}_{comp} = \left\{t^{\alpha}_{n,comp}\right\}$ *denote the set of test cases of SUT components on layer* α *as a set of pairs:*

$$T^{\alpha}_{comp} = \left\{\left[\left(v^{\alpha}_{i}, S^{\alpha}_{i}\right), A^{\alpha}_{i}\right]\right\}$$

where v^{α}_{i} *is a component of SUT on layer* α*;* $S^{\alpha}_{i} \subset \boldsymbol{S^{\alpha}}$ *is the set of specifications of* v^{α}_{i}*; and* $A^{\alpha}_{i} \subset \boldsymbol{S^{\alpha}}$ *is the set of required attributes for* v^{α}_{i}*.*

In other words, each test case of that kind represents a SUT component whose characteristics or configuration should be verified according to corresponding required attributes.

The next definition is based on the fact that if there is a set of required attributes for a path then each set of specifications (labels) of edges which constitute this path should match the set of required attributes[8].

Definition 12. *Let* $T^{\alpha}_{link} = \left\{t^{\alpha}_{n,link}\right\}$ *denote the set of test cases of SUT communication channels on layer* α *as a union of pairs:*

$$T^{\alpha}_{link} = \left\{\left[\left(\bigcup_{\left\langle v^{\alpha}_{(l-1)}, v^{\alpha}_{l}\right\rangle \in p^{\alpha}_{i,j,k}} \left(\left\langle v^{\alpha}_{(l-1)}, v^{\alpha}_{l}\right\rangle, S^{\alpha}_{(l-1),l}\right)\right), A^{\alpha}_{i,j}\right]\right\}$$

where $p^{\alpha}_{i,j,k}$ *is a* k^{th} $\left(v^{\alpha}_{i}, v^{\alpha}_{j}\right)$*-path between the pair of SUT dedicated components* v^{α}_{i} *and* v^{α}_{j} *on layer* α*;* $v^{\alpha}_{(l-1)}$ *and* v^{α}_{l} *is a pair of adjacent components on layer* α *which constitute the path* $p^{\alpha}_{i,j,k}$*;* $S^{\alpha}_{i,j,k} \subset \boldsymbol{S^{\alpha}}$ *is the set of specifications of the edge* $\left\langle v^{\alpha}_{(l-1)}, v^{\alpha}_{l}\right\rangle \in p^{\alpha}_{i,j,k}$*; and* $A^{\alpha}_{i,j} \subset \boldsymbol{S^{\alpha}}$ *is the set of required attributes for* $\left(v^{\alpha}_{i}, v^{\alpha}_{j}\right)$*-paths.*

In other words, each test case of that kind represents a SUT communication channel whose characteristics or configuration (as entity) should be verified according to corresponding required attributes. In practice, if a communication channel is functioning as entity (i.e. (1) the channel is in operational state; and (2) all its characteristics match the required attributes) then the assumption can be made that all elements which constitute the channel are also functioning and, as a consequence, they should not be verified individually due to financial and time constraints. Otherwise, if the communication channel is not functioning then the test case of that kind provides the necessary information

[8]This fact is based on the max-flow min-cut (Ford-Fulkerson) theorem [128].

about elements which constitute the channel to narrow the field of potential problems according to the *Follow-the-Path Troubleshooting Method*[9] [148].

In some cases, diagnostic tools do not allow the testing of paths on the physical architectural layer [148]. As a consequence, each test case of SUT communication channels on this layer should be divided into subset of test cases of component-to-component interconnections which constitute the channels, i.e.:

$$t^{\alpha}_{n,link} = \left[\left(\bigcup_{\left\langle v^{\alpha}_{(l-1)}, v^{\alpha}_{l} \right\rangle \in p^{\alpha}_{i,j,k}} \left(\left\langle v^{\alpha}_{(l-1)}, v^{\alpha}_{l} \right\rangle, S^{\alpha}_{(l-1),l} \right) \right), A^{\alpha}_{i,j} \right]$$
$$= \bigcup_{\left\langle v^{\alpha}_{(l-1)}, v^{\alpha}_{l} \right\rangle \in p^{\alpha}_{i,j,k}} \left[\left(\left\langle v^{\alpha}_{(l-1)}, v^{\alpha}_{l} \right\rangle, S^{\alpha}_{(l-1),l} \right), A^{\alpha}_{i,j} \right]$$

where layer α represents the physical architectural layer according to multilayer reference models (see Figure 3.7).

The next definitions are completely based on Criterion 2 (see Section 4.1.3):

Definition 13. *The function* $\varphi^{\alpha}_{comp} : R^{\alpha}_{comp} \times G^{\alpha} \to T^{\alpha}_{comp}$ *is defined as follows:*

$$\varphi^{\alpha}_{comp}\left(\left(v^{\alpha}_{i}, A^{\alpha}_{i}, A^{\alpha,(\alpha-1)}_{i} \right) \right) = \begin{cases} \left[\left(v^{\alpha}_{i}, S^{\alpha}_{i} \right), A^{\alpha}_{i} \right] & \text{if } A^{\alpha}_{i} \subseteq S^{\alpha}_{i} \\ \emptyset & \text{otherwise} \end{cases}$$

In other words, for each test requirement $\left(v^{\alpha}_{i}, A^{\alpha}_{i}, A^{\alpha,(\alpha-1)}_{i} \right) \in R^{\alpha}_{comp}$ on a given layer α the function φ^{α}_{comp} determines the pair $\left(v^{\alpha}_{i}, S^{\alpha}_{i} \right) \in G^{\alpha}$ whose characteristics match the required attributes, i.e. $A^{\alpha}_{i} \subseteq S^{\alpha}_{i}$. If this pair does not exist then the formal model M is inconsistent according to Criterion 3.

Definition 14. *The function* $\varphi^{\alpha}_{link} : R^{\alpha}_{link} \times G^{\alpha} \to T^{\alpha}_{link}$ *is defined as follows:*

$$\varphi^{\alpha}_{link}\left(\left(v^{\alpha}_{i}, v^{\alpha}_{j}, A^{\alpha}_{i,j}, A^{\alpha,(\alpha-1)}_{i,j} \right) \right) = \bigcup_{p^{\alpha}_{i,j,k} \in P^{\alpha}_{i,j}} \left\{ \widehat{t^{\alpha}_{n,link}} \right\}$$

[9] The follow-the-path approach first discovers the actual traffic path all the way from source to destination. Next, the scope of troubleshooting is reduced to just the links and devices that are actually in the forwarding path. [148]

where:

$$\widehat{t^{\alpha}_{n,link}} = \begin{cases} t^{\alpha}_{n,link} & if\ \forall \left\langle v^{\alpha}_{(l-1)}, v^{\alpha}_{l} \right\rangle \in p^{\alpha}_{i,j,k} : A^{\alpha}_{i,j} \subseteq S^{\alpha}_{(l-1),l} \\ \emptyset & otherwise \end{cases}$$

In other words:

- For each test requirement $\left(v^{\alpha}_{i}, v^{\alpha}_{j}, A^{\alpha}_{i,j}, A^{\alpha,(\alpha-1)}_{i,j}\right) \in R^{\alpha}_{link}$ on a given layer α the function φ^{α}_{link} determines the finite set $P^{\alpha}_{i,j}$ of all possible $\left(v^{\alpha}_{i}, v^{\alpha}_{j}\right)$-paths $p^{\alpha}_{i,j,k} \in P^{\alpha}_{i,j}$ between the pair of SUT dedicated components v^{α}_{i} and v^{α}_{j} which should communicate. If this set is an empty set then the formal model M is inconsistent according to Criterion 3.

- In turn, for each path $p^{\alpha}_{i,j,k}$ of $P^{\alpha}_{i,j}$ the function φ^{α}_{link} determines the finite set of all possible pairs $\left(\left\langle v^{\alpha}_{(l-1)}, v^{\alpha}_{l} \right\rangle, S^{\alpha}_{(l-1),l}\right) \in G^{\alpha}$ whose elements $\left\langle v^{\alpha}_{(l-1)}, v^{\alpha}_{l} \right\rangle$ constitute the path $p^{\alpha}_{i,j,k}$ and whose characteristics match the required attributes, i.e. $A^{\alpha}_{i,j} \subseteq S^{\alpha}_{(l-1),l}$. If this set does not cover the path $p^{\alpha}_{i,j,k}$ completely then the formal model M is inconsistent according to Criterion 3.

4.3 Structural Test Case Generation Strategy

The detailed framework of the structural test case generation strategy is shown in Figure 4.2. Two main steps of the strategy on a given layer α can be defined as follows:

- The set of interlayer test requirements $R^{(\alpha+1),\alpha}$ (see Definitions 7 and 8) is the result of recursive applying of: (1) the intralayer test requirements $R^{(\alpha+1),(\alpha+1)}$ (see Definitions 5 and 6); and (2) the interlayer test requirements $R^{(\alpha+2),(\alpha+1)}$ (see Definitions 7 and 8) to the interlayer subgraph $G^{(\alpha+1),\alpha}$ (see Definition 3):

$$R^{(\alpha+1),\alpha} = \mu^{(\alpha+1),\alpha}\left(R^{(\alpha+1)}\right) = \mu^{(\alpha+1),\alpha}\left(R^{(\alpha+1),(\alpha+1)} \cup R^{(\alpha+2),(\alpha+1)}\right)$$

  The process is described by Definitions 9 and 10 and Criterion 1.

- The set of system infrastructure test cases T^{α} on the layer α (see Definitions 11 and 12) is the result of applying of: (1) the intralayer test requirements $R^{\alpha,\alpha}$

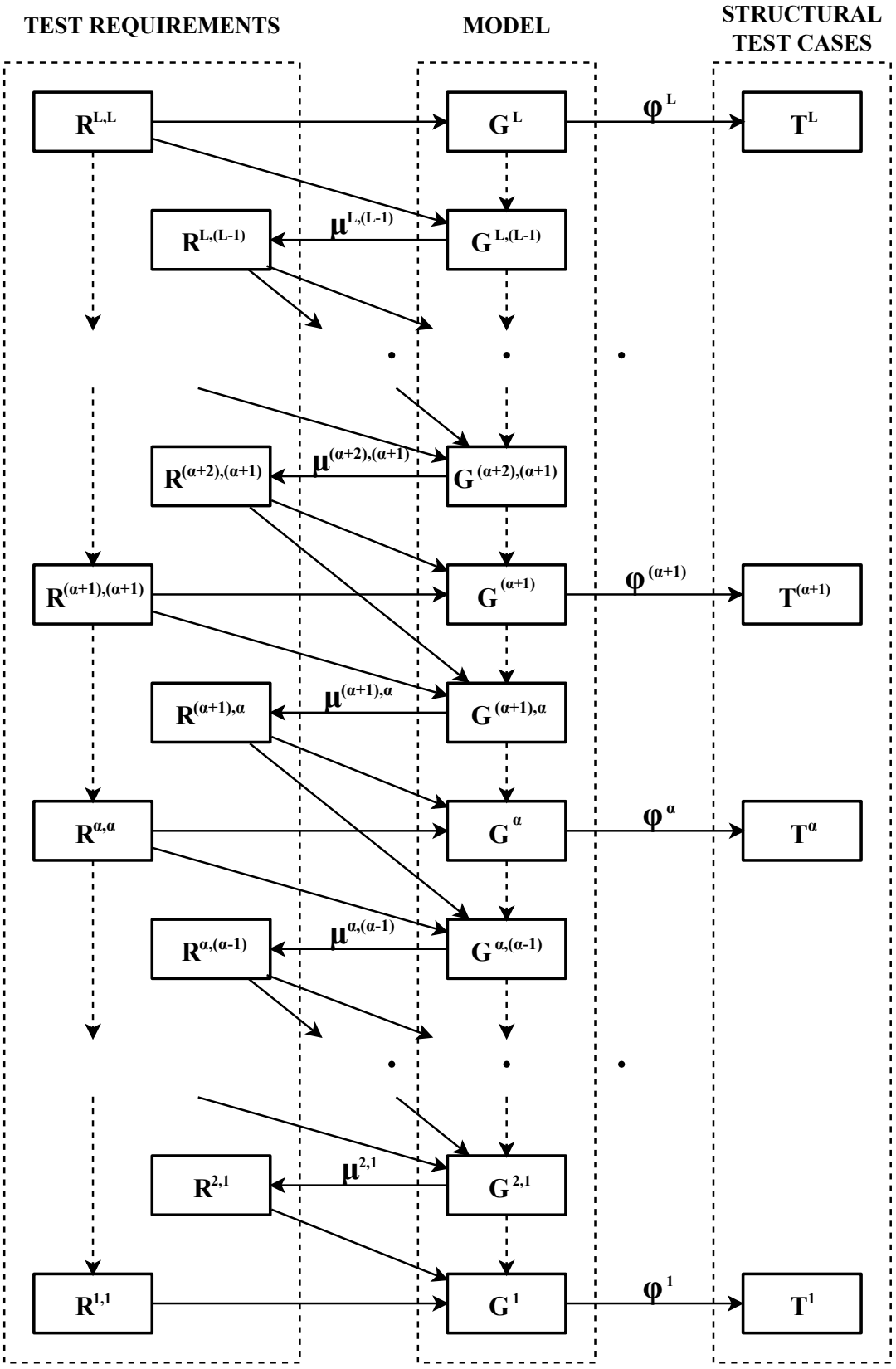


FIGURE 4.2: Graphical representation of the structural test case generation strategy.

(see Definitions 5 and 6); and (2) the interlayer test requirements $R^{(\alpha+1),\alpha}$ (see Definitions 7 and 8) to the intralayer subgraph G^{α} (see Definition 2):

$$T^{\alpha} = \varphi^{\alpha}\left(R^{\alpha}\right) = \varphi^{\alpha}\left(R^{\alpha,\alpha} \cup R^{(\alpha+1),\alpha}\right)$$

The process is described by Definitions 13 and 14 and Criterion 3.

The important properties of the structural test case generation strategy are represented by the following propositions:

Proposition 1. *If the formal model based on the concept of multilayer networks is consistent with respect to the test requirements then the set of test requirements R^{α} defined for a given layer α cannot be an empty set on all coexisting architectural layers.*

Proof. Let a formal model M be consistent according to Criterion 4. Hence, there is at least one test requirement defined for the top architectural layer L of the model (see Criterion 4), i.e.:

$$R^{L} = R^{L,L} \neq \emptyset$$

Each test requirement covers: either (1) a SUT component as a vertex of M (see Definition 5); or (2) a SUT communication channel as a pair (initial and terminal) of vertices of M (see Definition 6). In turn, each vertex of M on a given layer has at least one corresponding neighbor on the layer below (see Criterion 1). Hence, for each test requirement for a given layer there is at least one corresponding test requirement on the layer below, i.e. if $R^{L} \neq \emptyset$ then $R^{L,(L-1)} = \mu^{L,(L-1)}\left(R^{L}\right) \neq \emptyset$. As a consequence:

$$R^{(L-1)} = \left(R^{(L-1),(L-1)} \cup R^{L,(L-1)}\right) \neq \emptyset$$

The result of sequential repetitions of these steps on all coexisting architectural layers can be formally represented as follows:

$$R^{\alpha} \neq \emptyset; 1 \leq \alpha \leq L$$

□

Proposition 2. *If the formal model based on the concept of multilayer networks is consistent with respect to the test requirements then the set of test cases T^α generated on a given layer α cannot be an empty set on all coexisting architectural layers.*

Proof. Let a formal model M be consistent according to Criterion 4. Hence, there is at least one test requirement defined for a given layer α on all coexisting architectural layers (see Proposition 1), i.e.:

$$R^\alpha \neq \emptyset; 1 \leq \alpha \leq L$$

In this case according to Criterion 3:

$$T^\alpha = \varphi^\alpha \left(R^\alpha\right) \neq \emptyset; 1 \leq \alpha \leq L$$

□

Proposition 3. *If the formal model based on the concept of multilayer networks is consistent with respect to the test requirements then each test requirement defined for the top architectural layer of the formal model initiates at least one test case on all coexisting architectural layers.*

Proof. Let a formal model M be consistent according to Criterion 4. In this case:

- (according to Proposition 1) for each test requirement defined for the top architectural layer of the model there is at least one corresponding test requirement on all coexisting architectural layers below;
- (according to Criterion 3) each test requirement for a given layer initiates at least one test case on the given layer.

□

In the context of this thesis, the part of the SUT covered by test cases on layer α can be represented by the induced subgraph $\widehat{G^\alpha}$ of G^α as follows:

$$\widehat{G^\alpha} = \left(\widehat{V^\alpha}, \widehat{E^\alpha}\right)$$

where $\widehat{V^{\alpha}} \subseteq V^{\alpha}$ is a set of SUT components covered by the set of test cases of SUT components T^{α}_{comp} on layer α; and $\widehat{E^{\alpha}} \subseteq E^{\alpha}$ is a set of SUT component-to-component interconnections covered by the set of test cases of SUT communication channels T^{α}_{link} on layer α.

In general, each intralayer subgraph should be completely covered by test cases, i.e. $\widehat{G^{\alpha}} \equiv G^{\alpha}$. In this case, the fact $\widehat{G^{\alpha}} \subset G^{\alpha}$ might explicitly represent the existence of errors/bugs (at least one) in design documentation (test requirements and/or technical specifications), i.e. the fact that the set of test requirements is inconsistent with respect to the formal model. Unfortunately, in the case of upgrading existing infrastructures, these infrastructures: (1) should be covered by formal models; but, on the other hand, (2) should be verified with respect to the new components only (not completely) due to financial and time constraints. As a consequence, it makes this possible criterion useless in practice.

Chapter 5

Nonfunctional Test Case Generation Strategy

You know you have a distributed system when the crash of a computer you've never heard of stops you from getting any work done.

—Leslie Lamport

In the context of this thesis, the nonfunctional tests should ensure that system dependability mechanisms have been implemented correctly and, as a consequence, the SUT is able to provide the desired level of reliable (dependable) services.

As mentioned above (see Section 2.3), fault-injection experiments provide a means for understanding how the SUT behaves in the presence of faults (the monitoring of the effects of the injected faults). As a consequence in practice (or in the world of imperfect sensing and switching components[1]), the nonfunctional (dependability) tests are represented by fault-injection experiments as follow:

- Component fault injection to ensure that:
  - the sensing mechanism is able to (1) detect a component failure, and (2) trigger the switching mechanism, i.e. sensing mechanism verification;

[1]Perfect sensing and switching devices are just a mathematical abstraction. In practice, system reliability is totally dependent on the quality of the sensing and switching components [102].

  - the switching mechanism is able to reconfigure SUT topology (reroute communication channels) in the case of component failure, i.e. switching mechanism verification.

- Component repair (or inverse fault injection) to ensure that:
  - the sensing mechanism is able to (1) detect a component *resurrection* (successful replacement or recovery), and (2) trigger the switching mechanism (if necessary);
  - the switching mechanism is able to restore SUT initial topology (if necessary).

Hence, the nonfunctional (dependability) test case generation strategy defines the set of SUT components which should be targeted by fault-injection experiments (fault-injection targets) as a set of fault-injection test cases. In turn, in the context of the formal model, each fault-injection experiment represents the removal of a fault-injection target (a SUT component) and its incident edges from the formal model.

According to the goals of this thesis (see Section 1.5), the description of the nonfunctional (dependability) test case generation strategy covers the following areas:

- The framework of the test case generation strategy.
- The formal definitions of the key elements and their relations.
- The formal definitions of test case generation strategy.

5.1 Framework of Test Case Generation Strategy

Figure 5.1 represents the framework of test case generation strategy for a given layer of the formal model. In general, this framework is based on the concept of the dynamic analysis represented by Kurant et al. [56]. Nevertheless, it is important to note that the successive dynamic analysis of all components of the formal model can be impractical for real life testing (even in the case of relatively simple systems) due to computational complexity. In this case, the structural test case generation strategy (see Chapter 4) provides information for subsequent analysis with respect to the test requirements.

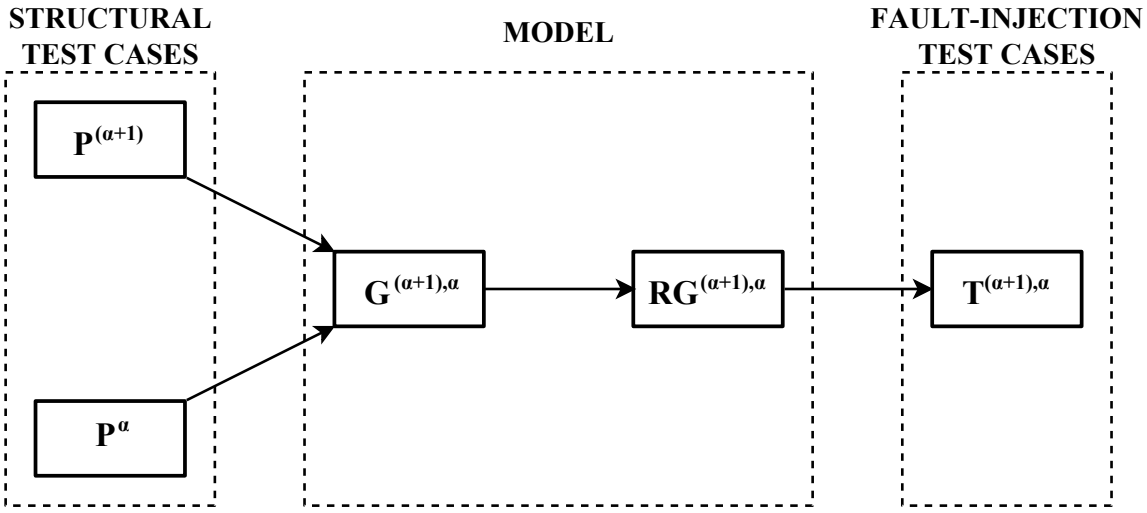


FIGURE 5.1: The framework of the nonfunctional (dependability) test case generation strategy for a given layer α of the formal model where $G^{(\alpha+1),\alpha}$ is an interlayer subgraph; $P^{(\alpha+1)}$ is a set of communication channels (data flows) on layer $(\alpha+1)$; P^{α} is a set of communication channels (data flows) on layer α; $RG^{(\alpha+1),\alpha}$ is a recovery group on layer α; and $T^{(\alpha+1),\alpha}$ is a set of fault-injection test cases on layer α.

As mentioned above (see Section 1.4), the original definition of dependability determines the system's ability to deliver service that can justifiably be trusted [22]. In the context of this thesis, the system's ability to deliver a service is represented by a path (or paths) between the corresponding SUT components (see Section 4.1.3).

In turn, the concept of layered networks [36] strictly relies on the fact that a path between two nodes on a given layer depends on a path (or paths) between the corresponding nodes on the layer below. In general, there are three possible results of applying the concept:

– A path on a given layer has exactly one corresponding path on the layer below. In this case, SUT components which constitute the corresponding path represent single points of failure[2]. In the context of the formal model, the removal of any SUT component which represents a single point of failure and its incident edges completely destroys the corresponding path.

– A path on a given layer has more than one (at least two) corresponding paths on the layer below. Furthermore, these corresponding paths have no common components (i.e. the corresponding paths are totally independent). In this case, SUT

[2] A single point of failure (SPOF) is any single component in a system that can fail and cause a service to become unavailable [91].

components which constitute the corresponding paths represent a recovery group[3]. In the context of the formal model, the removal of any SUT component which represents a member of a recovery group and its incident edges: (1) destroys only one corresponding paths; and, as a consequence, (2) leaves the other corresponding paths unaffected. In other words, recovery groups represent SUT components which constitute corresponding parallel/redundant paths, i.e. provide topological redundancy[4] for the path on a given layer.

– A path on a given layer has more than one (at least two) corresponding paths on the layer below. In turn, these corresponding paths have common components. In this case:

  – common components represent single points of failure (the removal of the component of that kind and its incident edges destroys all corresponding paths);

  – other remaining components represent a recovery group (the removal of the component of that kind and its incident edges destroys only some corresponding paths).

These results are formalized in Criterion 5:

Criterion 5. *A vertex v_l^α represents a single point of failure for the set of communication channels (data flows) $P_{i,j}^\alpha$ between a pair of SUT components v_i^α and v_j^α iff v_l^α and its incident edges $\left\langle v_{(l-1)}^\alpha, v_l^\alpha \right\rangle$ and $\left\langle v_l^\alpha, v_{(l+1)}^\alpha \right\rangle$ are integral parts of each path $p_{i,j,k}^\alpha \in P_{i,j}^\alpha$.*

The important property of single points of failure is represented by the following proposition:

Proposition 4. *If a formal model based on the concept of multilayer networks is consistent with respect to the test requirements then the removal of a vertex which represents a single point of failure and its incident edges renders this model inconsistent with respect to the test requirements.*

Proof. Let: (1) a formal model M be consistent according to Criterion 4; (2) G^α be an intralayer subgraph of M; and (3) $r_{n,link}^{\alpha,\alpha} = \left(v_i^\alpha, v_j^\alpha, A_{i,j}^\alpha, A_{i,j}^{\alpha,(\alpha-1)}\right)$ be an intralayer test

[3] A recovery group is the granularity at which a service can be recovered after service failures [91].

[4] Redundancy is the presence of auxiliary components in a system to perform the same or similar functions as other elements for the purpose of preventing or recovering from failures [17].

requirement defined for layer α. In this case, according to Criterion 2, $r^{\alpha,\alpha}_{n,link}$ binds a set of $\left(v^{\alpha}_{i}, v^{\alpha}_{j}\right)$-paths (at least one) in G^{α} between the pair v^{α}_{i} and v^{α}_{j}. In turn, according to Criterion 5, v^{α}_{l} and its incident edges $\left\langle v^{\alpha}_{(l-1)}, v^{\alpha}_{l}\right\rangle$ and $\left\langle v^{\alpha}_{l}, v^{\alpha}_{(l+1)}\right\rangle$ are integral parts of each $\left(v^{\alpha}_{i}, v^{\alpha}_{j}\right)$-path. Hence, $\left(v^{\alpha}_{i}, v^{\alpha}_{j}\right)$-paths do not exist in graph $G^{\alpha} - v^{\alpha}_{l}$ and, as a consequence, $r^{\alpha,\alpha}_{n,link}$ does not bind an object in $G^{\alpha} - v^{\alpha}_{l}$. Hence, the formal model $M - v^{\alpha}_{l}$ is inconsistent according to Criterion 4. □

In the context of this thesis, the definition of the fault-injection targets (SUT components which should be targeted by fault-injection experiments) strictly relies on the following notions:

- The fact that single points of failure do not define any dependability mechanism that should be verified.
- The fact that a component of SUT which is a member of a recovery group for one service can be a single point of failure for another service (or services). In this case, the component should be defined as a single point of failure.

These ideas are formalized in Criterion 6:

Criterion 6. *A vertex v^{α}_{i} of the formal model M which is consistent with respect to the test requirements represents a fault-injection target on a given layer α iff the formal model $M - v^{\alpha}_{i}$ is also consistent with respect to the test requirements.*

In general, there are three possible results of applying Criterion 6:

- (1) SUT is announced as a dependable system; and (2) there are a finite number of fault-injection targets defined by the dynamic analysis. This case represents a dependable (at least partially) design. The set of fault-injection targets should be included in the SUT design docu-mentation as fault-injection test cases.
- (1) SUT is announced as a dependable system; (2) there is no one fault-injection target defined by the dynamic analysis. This case explicitly represents the existence of errors/bugs (at least one) in design documentation (test requirements and/or technical specifications). As a consequence:
  - the design documentation should be corrected;

– the formal model should be re-built and then re-checked using Criterion 4 (if necessary) and Criterion 6.

Limitations:

– The formal model based on the concept of multilayer networks is intended to specify heterogeneous structures and their properties. The user-centric aspects of dependability (see Section 2.3 have to be described using different techniques (these aspects are beyond the scope of this thesis).

– The announcement about system dependability should be made explicitly as a part of end-user requirements.

5.2 Formal Definitions

As mentioned above (see Section 5.1), in the context of the formal model, recovery groups on a given layer represent SUT components which provide topological redundancy for the upper layer. In this case:

Definition 15. *Let* $RG^{(\alpha+1),\alpha}$ *denote the recovery group on layer* α *as set of SUT components:*

$$RG^{(\alpha+1),\alpha} = \left\{ v_i^{\alpha} \right\}$$

where v_i^{α} *is a component of SUT on layer* α *which provides topological redundancy for layer* $(\alpha + 1)$.

A special case of recovery group components is end-user components (hardware and software) as a part of end systems[5]. As mentioned above (see Section 5.1), the system's ability to deliver a service is represented by paths between: (1) the SUT component which represents the service and its projections on the layers below; and (2) the end-user components (as service subscribers) and their projections on the layers below[6].

[5]The formal definition of end systems (or hosts) includes: (1) servers; (2) desktop computers; and (3) mobile computers [133]. In the context of this thesis, the definition of end-user components covers desktop and mobile computers only.

[6]In the context of this thesis, a service is available until there is at least one path between the service (and its projections on the layers below) and an end-user component (and its projections on the layers below) on all coexisting architectural layers.

However, dependability mechanisms (replication or redundancy) are normally not used for end-user components. This statement is based on two main reasons:

- economic reason - dependability mechanisms tend to increase the system cost;
- technical reason - dependability mechanisms might increase complexity to the point where the dependability mechanisms themselves contribute to accidents [3].

In practice, end-user components and their projections should be removed from the recovery group and, as a consequence, eliminated from the analysis[7].

As mentioned above (see Section 5.1), the successive dynamic analysis of all components of the formal model can be impractical due to the fact that single points of failure do not define any dependability mechanism that should be verified. In turn, the structural test case generation strategy (see Chapter 4) provides information to determine recovery groups. As a consequence, the next function is completely based on: (1) the formal definition of the set of SUT communication channels (see Definition 4); and (2) the definition of the symmetric difference of sets as the set of elements which are in either of the sets and not in their intersection [149]:

Definition 16. *The function* $\omega^{(\alpha+1),\alpha} : P^{(\alpha+1)} \times P^{\alpha} \times G^{(\alpha+1),\alpha} \rightarrow RG^{(\alpha+1),\alpha}$ *is defined as follows:*

$$\omega^{(\alpha+1),\alpha}\left(p_{l,m,n}^{(\alpha+1)}, P^{\alpha}\right) = \left[\bigcup_{\substack{\left\langle v_l^{(\alpha+1)}, v_i^{\alpha} \in G^{(\alpha+1),\alpha}\right\rangle \\ \left\langle v_m^{(\alpha+1)}, v_j^{\alpha} \in G^{(\alpha+1),\alpha}\right\rangle \\ p_{l,m,n}^{(\alpha+1)} \in P^{(\alpha+1)} \\ p_{i,j,k}^{\alpha} \in P^{\alpha}}} V_{i,j,k}^{\alpha}\right] \setminus \left[\bigcap_{\substack{\left\langle v_l^{(\alpha+1)}, v_i^{\alpha} \in G^{(\alpha+1),\alpha}\right\rangle \\ \left\langle v_m^{(\alpha+1)}, v_j^{\alpha} \in G^{(\alpha+1),\alpha}\right\rangle \\ p_{l,m,n}^{(\alpha+1)} \in P^{(\alpha+1)} \\ p_{i,j,k}^{\alpha} \in P^{\alpha}}} V_{i,j,k}^{\alpha}\right]$$

where $p_{l,m,n}^{(\alpha+1)} \in P^{(\alpha+1)}$ *is a* n^{th} $\left(v_l^{(\alpha+1)}, v_m^{(\alpha+1)}\right)$*-path on layer* $(\alpha+1)$*;* $p_{i,j,k}^{\alpha} \in P^{\alpha}$ *is a corresponding path of* $p_{l,m,n}^{(\alpha+1)}$ *on layer* α*; and* $V_{i,j,k}^{\alpha}$ *is the set of SUT components which constitute the path* $p_{i,j,k}^{\alpha}$ *on layer* α *.*

[7] In the context of this thesis, end-user components are completely covered by structural test cases (see Chapter 4).

In other words, for each path $p_{l,m,n}^{(\alpha+1)} \in P^{(\alpha+1)}$ on layer $(\alpha+1)$ the function $\omega^{(\alpha+1),\alpha}$ determines the finite set of components v_i^α on layer α which provides topological redundancy for the path $p_{l,m,n}^{(\alpha+1)}$ on layer $(\alpha+1)$.

In the context of this thesis, fault-injection test cases represent SUT components which should be targeted by fault-injection experiments. Hence:

Definition 17. *Let $T^{(\alpha+1),\alpha}$ denote the set of fault-injection test cases on layer α as a set of SUT components:*

$$T^{(\alpha+1),\alpha} = \left\{ v_i^\alpha \right\}$$

where v_i^α is a component of SUT on layer α which should be targeted by fault-injection experiments.

As mentioned above (see Section 5.1), a component of SUT which is a member of a recovery group for one service can be a single point of failure for another service (or services). To address this problem, the next definitions are completely based on Criterion 6:

Definition 18. *The function $\theta^{(\alpha+1),\alpha} : RG^{(\alpha+1),\alpha} \rightarrow T^{(\alpha+1),\alpha}$ is defined as follows: A SUT component v_i^α of M is targeted by fault-injection experiments, i.e. $v_i^\alpha \in T^{(\alpha+1),\alpha}$ iff:*

- *graph M is consistent with respect to the test requirements (according to Criterion 4);*
- *v_i^α is a member of the recovery group on a given layer α, i.e. $v_i^\alpha \in RG^{(\alpha+1),\alpha}$;*
- *subgraph $M - v_i^\alpha$ is also consistent with respect to the test requirements (according to Criterion 4).*

In other words, the function $\theta^{(\alpha+1),\alpha}$ eliminates single points of failure from the recovery group on a given layer α.

5.3 Test Case Generation Strategy

The detailed framework of the nonfunctional (dependability) test case generation strategy is shown in Figure 5.2. Two main steps of the strategy on a given layer α can be defined as follows:

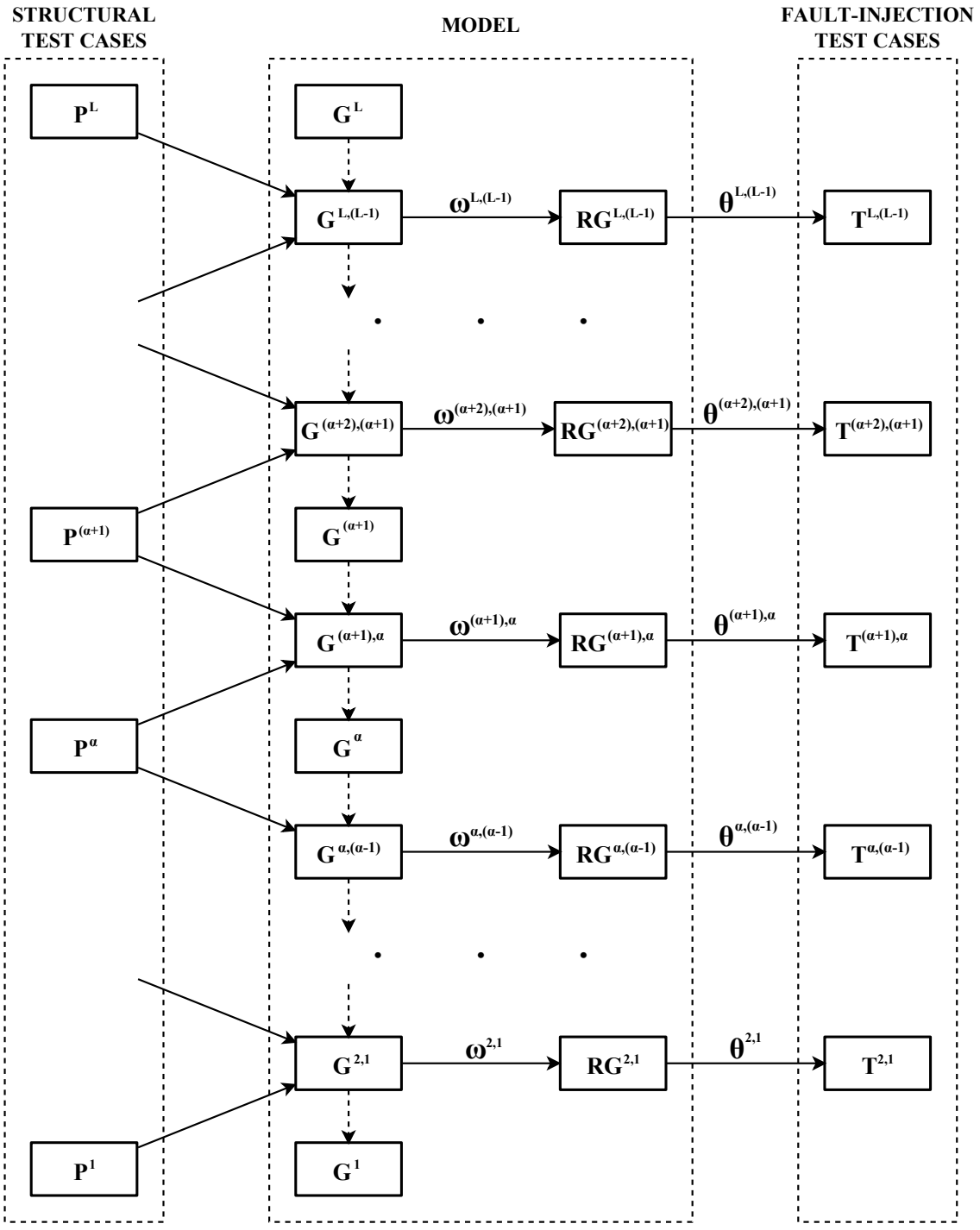


FIGURE 5.2: Graphical representation of the nonfunctional (dependability) test case generation strategy.

– The recovery group $RG^{(\alpha+1),\alpha}$ on the layer α (see Definition 15) is the result of applying of: (1) the set of SUT communication channels $P^{(\alpha+1)} \subseteq G^{(\alpha+1)}$ on the layer $(\alpha+1)$ (see Definitions 4 and 14); and (2) the set of SUT communication channels $P^{\alpha} \subseteq G^{\alpha}$ on the layer α (see Definitions 4 and 14) to the interlayer subgraph $G^{(\alpha+1),\alpha}$ (see Definition 3):

$$RG^{(\alpha+1),\alpha} = \omega^{(\alpha+1),\alpha}\left(T_{link}^{(\alpha+1)}, T_{link}^{\alpha}\right)$$

The process is described by Definition 16.

– The set of fault-injection test cases $T^{(\alpha+1),\alpha}$ on the layer α (see Definition 17) is the result of the dynamic analysis of the recovery group $RG^{(\alpha+1),\alpha}$ (see Definition 15) on the layer α:

$$T^{(\alpha+1),\alpha} = \theta^{(\alpha+1),\alpha}\left(RG^{(\alpha+1),\alpha}\right)$$

The process is described by Definition 18 and Criterion 6.

In general, recovery groups (after elimination end-user components and their projections) should not contain single points of failure (i.e. $T^{(\alpha+1),\alpha} \equiv RG^{(\alpha+1),\alpha}$). In this case, the fact $T^{(\alpha+1),\alpha} \subset RG^{(\alpha+1),\alpha}$ might explicitly represent the existence of errors/bugs (at least one) in design documentation (technical specifications). Unfortunately, the partial implementation of dependability mechanisms (for important services only) can be often met in practice due to financial constraints. As consequence, it makes this possible criterion useless in practice.

Chapter 6

Presentation Format

What you do today can improve all your tomorrows.

—Ralph Marston

In practice, model analysis requires specialized training, both: (1) in the models development (a model must be completely relevant to a system - a trusted model); and (2) in the interpretation of the analysis results. The human work involved in data transformation represents a major bottleneck due to its tendency to be relatively unsophisticated and repetitive, but persistently tricky and time-consuming at the same time [150]. Challenges in the analysis process that repeatedly occur in analysis efforts are

- discover necessary data;
- wrangle data into a desired format;
- profile data to verify its quality and suitability;
- report procedures to consumers of the analysis.

Thus, to get the full advantages of model analysis and verifying in the domain of complex systems, it is necessary to alleviate the burdens of learning model development and checking techniques for engineers and other non-technical stakeholders [16] or, ideally, completely eliminate the human factor.

There have been some attempts to make model development accessible to those who are not trained in formal methods. These include Formal Description Techniques [113]

based on a technical language for unambiguous specification and description of the behavior of telecommunication systems (see Section 2.4). However, FDTs are intended to specify the behavioral aspects of software-intensive systems only; the general parameters, which determine heterogeneous architectures and properties, have to be described using different techniques.

To address this problem, this section presents a possible appropriate presentation format of architecture descriptions as a part of detailed design documentation that provides unambiguous interrelation between the design documentation and the multilayer formal model. As a consequence, this presentation format could allow automated development of formal models for analysis and verifying of computer networks.

6.1 Presentation Format

As mentioned above (see Section 2.4), the current revisions of international standards establish what should be contained in design documentation but not how exactly. Nevertheless, Appendix I of ITU-T Recommendation L.72 [151] represents an example of a currently used presentation format of optical access network infrastructure descriptions[1]. It is important to note that this format covers the physical architecture completely and the logical architecture partially. However, this format is optimized for representation network infrastructures and, as a consequence, cannot be used: (1) to define a whole/-completed system (i.e. a computer network and services/applications which this network provides and supports); and (2) to represent technological solutions (hardware and software clusters, virtualization platforms, etc.) which are used to build the system.

To fill the gap, this section represents a set of design patterns[2] for unambiguous architecture description as a possible part of the detailed design documentation [153] [154].

Based on the concept of layered networks [36], the architecture of complex computer networks can be represented by three main design patterns (tables) on each layer:

Layer component specification. The layer component specification design pattern is used for the components detail representation. This design pattern should cover: (1) system

[1]This appendix does not form an integral part of the Recommendation.
[2]The term *design pattern* [152] aims to explicitly represent design knowledge that can be understood implicitly by skilled engineers and other non-technical stakeholders.

TABLE 6.1: Design Pattern of Layer Component Specifications.

Table columns		
No.	**Name**	**Description**
1	T1.α.1 Record Number	Record identification number (component index)
2	T1.α.2 Layer Identifier	Engineering (optional), physical, logical, service, social (optional) or functional layer (similar to T2.α.2 and T3.α.2)
3	T1.α.3 Component Assignment	Component functional description (if necessary)
4	T1.α.4 Component Identifier	Component name
5	T1.α.5 Vendor Identifier	Vendor contact name (for COTS components)
6	T1.α.6 Component Attributes	Component technical specifications (according to T2.α.8), i.e. supported protocols, IP addresses and masks, TCP/UDP ports, etc.
7	T1.α.7 Notes	Additional information (if necessary)

TABLE 6.2: Design Pattern of Intralayer Topology Specifications.

Table columns		
No.	**Name**	**Description**
1	T2.α.1 Record Number	Record identification number (link index)
2	T2.α.2 Layer Identifier	Engineering (optional), physical, logical, service, social (optional) or functional layer (similar to T1.α.2 and T3.α.2)
3	T2.α.3 Link Assignment	Component-to-component interconnection functional description (if necessary)
4	T2.α.4 Source Component Identifier	Component name according to T1.α.4
5	T2.α.5 Source Port Identifier	Component communication interface
6	T2.α.6 Target Component Identifier	Component name according to T1.α.4
7	T2.α.7 Target Port Identifier	Component communication interface
8	T2.α.8 Link Attributes	Technical specifications of component-to-component interconnection (according to T1.α.6), i.e. used protocols, IP addresses and masks, TCP/UDP ports, etc.
9	T2.α.9 Notes	Additional information (if necessary)

business goals for the functional layer; (2) persons or groups of persons for the social layer (optionally); (3) software-based components (services/applications) for the service layer; (4) virtual components (VM, VLAN, etc.) for the logical layer; (5) hardware-based components (equipment) for the physical layer; and (6) external engineering systems for the engineering layer (optionally). The unified table column structure specifies the necessary component properties and, therefore, includes (see Table 6.1):

– Record Number.

– Layer Identifier.

– Component Assignment.

TABLE 6.3: Design Pattern of Interlayer Topology Specifications.

Table columns		
No.	**Name**	**Description**
1	T3.α.1 Record Number	Record identification number (projection index)
2	T3.α.2 Layer Identifier	Engineering (optional), physical, logical, service, social (optional) or functional layer (similar to T1.α.2 and T2.α.2)
3	T3.α.3 Projection Assignment	Components interlayer relation functional description (if necessary)
4	T3.α.4 Source Component Identifier	Component name according to T1.α.4
5	T3.α.5 Target Component Identifier	Component name according to T1.α.4
6	T3.α.6 Distribution Index	Cross-layer technologies: (1) $N_n : 1_{n-1}$ - virtualization and replication; (2) $1_n : N_{n-1}$ - clustering; and (3) $1_n : 1_{n-1}$ - a special case of dedicated components
7	T3.α.7 Projection Attributes	Technical specifications of components interlayer relation (resources distribution across the network) technical specifications such as capacity metrics and modes (active/active, active/standby, etc.
8	T3.α.8 Notes	Additional information (if necessary)

– Component Identifier.

– Vendor Identifier.

– Component Attributes.

– Notes.

Intralayer topology specification. The intralayer topology specification design pattern is used for the layer topology detail representation. This pattern should cover architecture descriptions of: (1) functional models [1] for the functional (or ready-for-use system) layer; (2) flow-based models [1] for the social (optionally) and service layers; and (3) topological models [1] for the logical, physical and engineering (optionally) layers. The unified table column structure determines the intralayer links and, therefore, includes (see Table 6.2):

– Record Number.

– Layer Identifier.

– Link Assignment.

– Link Identifier:

    - Source Identifier:
        - Component Identifier.
        - Port Identifier.
    - Target Identifier:
        - Component Identifier.
        - Port Identifier.
- Link Attributes.
- Notes.

Interlayer topology specification. The interlayer topology specification design pattern is used for the resources distribution (cross-layer topology) detail representation. This pattern strictly relies on the concept of layered networks [36] that a node in a given layer depends on a corresponding node (or nodes) in the layer below. The unified table column structure defines the necessary properties of interlayer projections and, therefore, includes (see Table 6.3):

- Record Number.
- Layer Identifier.
- Projection Assignment.
- Projection Identifier:
    - Source Identifier:
        - Component Identifier on a Given Layer.
    - Target Identifier:
        - Component Identifier on the Layer Below.
- Distribution Index
- Projection Attributes.
- Notes.

TABLE 6.4: Formal Model and Design Pattern of Layer Component Specifications.

No.	Design pattern record	Formal model symbol $\boldsymbol{G^{\alpha} = (V^{\alpha}, E^{\alpha}, S_V^{\alpha}, S_E^{\alpha})}$
1	T1.α.1 Record Number	i
2	T1.α.2 Layer Identifier	α
3	T1.α.4 Component Identifier	$v_i^{\alpha} \in V^{\alpha}$
4	T1.α.6 Component Attributes	$S_i^{\alpha} \subset S_V^{\alpha} \subset \boldsymbol{S^{\alpha}}$

In turn, each table header structure should include: (1) Table Identifier; (2) Project Identifier; and (3) Facility Identifier.

In practice, these tables can be used (1) as independent documents or (2) as a database structure similar to ITU-T Rec L.72 [151].

6.2 Formal Model and Design Pattern Correlations

A model is any incomplete representation of reality - an abstraction [10]. In practice it means that design documentation usually contains much more data than we need to create models. In our case, from the perspective of the formal abstract model:

– The layer component specification is a node list (see Table 6.1): each row represents a node (vertex) in the graph and columns contain attributes (node labels). Data structures correlation between the formal model and this design pattern is shown in Table 6.4.

– The intralayer topology specification is an adjacency list or a relational table (see Table 6.2): each row represents an edge in the graph and columns contain incident (source and target) nodes among other attributes (edge labels). Data structures correlation between the formal model and this design pattern is shown in Table 6.5.

– The interlayer topology description is an adjacency list or a relational table (see Table 6.3): each row represents an edge in the graph and columns contain incident (source and target) nodes among other attributes. Data structures correlation between the formal model and this design pattern is shown in Table 6.6.

As mentioned above (see Section 1.3), the quality of formal methods based on abstract models is limited by the quality of these models. In the context of this thesis, complex

Table 6.5: Formal Model and Design Pattern of Intralayer Topology Specifications.

No.	**Design pattern record**	**Formal model symbol** $\boldsymbol{G^{\alpha} = (V^{\alpha}, E^{\alpha}, S_V^{\alpha}, S_E^{\alpha})}$	
1	T2.α.1 Record Number	n	
2	T2.α.2 Layer Identifier	α	
3	T2.α.4 Source Component Identifier	$v_i^{\alpha} \in V^{\alpha}$	$e_n^{\alpha} = \langle v_i^{\alpha}, v_j^{\alpha} \rangle \in E^{\alpha}$
4	T2.α.6 Target Component Identifier	$v_j^{\alpha} \in V^{\alpha}$	
5	T2.α.8 Link Attributes	$S_{i,j}^{\alpha} = (S_i^{\alpha} \cap S_i^{\alpha}) \subset S_E^{\alpha} \subset \boldsymbol{S^{\alpha}}$	

Table 6.6: Formal Model and Design Pattern of Interlayer Topology Specifications.

No.	**Design pattern record**	**Formal model symbol** $\boldsymbol{G^{\alpha,(\alpha-1)} = \left(V^{\alpha}, V^{(\alpha-1)}, E^{\alpha,(\alpha-1)}\right)}$	
1	T3.α.1 Record Number	n	
2	T3.α.2 Layer Identifier	α	
3	T3.α.4 Source Component Identifier	$v_i^{\alpha} \in V^{\alpha}$	$e_n^{\alpha,(\alpha-1)} = \left\langle v_i^{\alpha}, v_j^{(\alpha-1)} \right\rangle \in$
4	T3.α.5 Target Component Identifier	$v_j^{(\alpha-1)} \in V^{(\alpha-1)}$	$\in E^{\alpha,(\alpha-1)}$

Table 6.7: Test Requirements for SUT Components and Design Pattern of Layer Component Specifications.

No.	**Design pattern record**	**Formal model symbol** $\boldsymbol{R_{comp}^{\alpha,\alpha} = \left\{\left(v_i^{\alpha}, A_i^{\alpha}, A_i^{\alpha,(\alpha-1)}\right)_n\right\}}$
1	T1.α.1 Record Number	n
2	T1.α.2 Layer Identifier	α
3	T1.α.4 Component Identifier	$v_i^{\alpha} \in V^{\alpha}$
4	T1.α.6 Component Attributes	$A_i^{\alpha} \subset \boldsymbol{S^{\alpha}}; A_i^{\alpha,(\alpha-1)} \subset \boldsymbol{S^{(\alpha-1)}}$

Table 6.8: Test Requirements for SUT Communication Channels and Design Pattern of Intralayer Topology Specifications.

No.	**Design pattern record**	**Formal model symbol** $\boldsymbol{R_{link}^{\alpha,\alpha} = \left\{\left(v_i^{\alpha}, v_j^{\alpha}, A_{i,j}^{\alpha}, A_{i,j}^{\alpha,(\alpha-1)}\right)_n\right\}}$
1	T2.α.1 Record Number	n
2	T2.α.2 Layer Identifier	α
3	T2.α.4 Component Identifier	$v_i^{\alpha} \in V^{\alpha}$
4	T2.α.6 Component Identifier	$v_j^{\alpha} \in V^{\alpha}$
5	T2.α.8 Component Attributes	$A_{i,j}^{\alpha} \subset \boldsymbol{S^{\alpha}}; A_{i,j}^{\alpha,(\alpha-1)} \subset \boldsymbol{S^{(\alpha-1)}}$

network architecture can be unambiguously represented using a set of tables (design patterns) that should be included as a necessary part of the detailed design documentation of a computer network. In turn, this set of tables provides unambiguous definition of the formal model (3D graph) for analysis and verifying of the network structure. As a consequence, the human factor can be completely eliminated from the data transformation processes during the formal model generation activities - the process can be done in automated mode using the detailed design documentation as input data. In this case,

the formal model is completely relevant to the design documentation (a trusted model from the viewpoint of network/system designers).

It is important to note that MBT techniques can be used for automated validation the formal model internal consistency with respect to the end-user requirements [4]. In the case of successful validation, the formal model is completely relevant to the end-user requirements (a trusted model from the viewpoint of end-users/customers). To accomplish such a goal and to make the model being completely applicable to the test generation strategies (to ensure full compatibility), the formal operational specifications of: (1) end-user requirements; and (2) derived technical requirements (see Figure 2.9) must be based on the same design patterns that the technical specifications (see Table 6.1 and Table 6.2). Data structures correlation between the test requirements and the design patterns is shown in Table 6.7 and Table 6.8.

Limitations:

- Similar to other formal methods, the proposed approach has no future outlook without the support of standardization communities (at least as a corporation standard).
- As mentioned above (see Section 4.1.2), the techniques of automated transforming informal end-user requirements into formal operation specifications are beyond the scope of this paper. The problem requires a separate analysis - even in the case of relatively simple systems, it may not be a routine exercise in practice.

Chapter 7

A Case Study

The most effective way to do it is to do it.

—Amelia Earhart

In 1845 (developing a new world view), Karl Marx introduced the definition of practice as the criterion of truth[1] [155]. According to the criterion, this chapter represents a case study which is based on a project performed within the framework of development activities at the company SPC TRIGGER s.r.o. [156]. In the context of this thesis, the case study represents the part of the project which covers the viewpoint of the business community (see Section 3.2) - the virtualization platform. In turn, the viewpoint of the IT personnel which covers:

- replacement of x86 desktop PCs by thin clients;
- using of the existing centralized management tool for the new infrastructure;

is beyond the scope of the case study[2].

7.1 Project Description

The project background was based on the following notions:

[1] The question of whether objective truth can be attributed to human thinking is not a question of theory but is a practical question. Man must prove the truth, i.e. the reality and power, the this-worldliness of his thinking in practice. [155]

[2] These parts of the project contain sensitive information and cannot be presented in the public domain.

- the customer is an industrial institution;
- the vision for the future of IT is the transformation of the existing infrastructure based on virtualization technologies to increase its agility and availability;
- the project was initiated by the local IT unit as a pilot project for detailed acquaintance with VMware vSphere virtualization platform;
- the initial consolidation project targets 16 x86 servers and 48 x86 desktop personal computers (personal computers are replaced by thin clients).

The main goals of the project were:

- reduction of hardware cost through the consolidation of physical servers;
- improvement of operational efficiency by increasing uptime and resiliency of services/applications and reducing services/applications recovery time;
- providing high availability of services/applications.

The following figures represent the used design requirements (partially):

- end-user requirements - see Figure 7.1;
- end-user constraints - see Figure 7.2;
- design assumptions - see Figure 7.3.

In turn, Figure 7.4 represents the example of the requirements derived from technical specifications, i.e. defined by technological solutions used to build the SUT (see Section 4.1.2).

7.2 Architecture Design

As mentioned above, the case study covers the viewpoint of the customer business community which includes[3]:

[3]Due to lack of space, this case study covers only one application server and two desktop personal computers. In practice, the project covered sixteen application servers and forty-eight thin clients.

ID	End-User Requirements
R101	*Business agility and flexibility should be increased*
R102	*Cost of doing business should be decreased*
R103	*Interaction delay (INTD) must be less than human response time (HRT)*
R104	*Availability of services is defined as 99.9 percent during core business hours*
R105	*Consolidate the existing 16 physical application servers down to 3 servers*
R106	*Centralized management tool must be used for the new infrastructure*
R107	*Separate management VLANs must be used for management traffic*
R108	*Server hardware maintenance should not affect application uptime*
R109	*Provide N+1 redundancy to support hardware failure during normal operation*

FIGURE 7.1: A Case Study - Example of End-user requirements.

ID	End-User Constraints
C101	*VMware vSphere Essentials Plus Kit has been preselected as the virtualization platform*
C102	*Dell servers have been preselected as the compute platform*
C103	*Two 10G ports should be used per server*
C104	*VMware Virtual SAN tool has been preselected as the storage solution*
C105	*D-Link managed switches have been preselected as the network platform*

FIGURE 7.2: A Case Study - Example of End-user constraints.

ID	Assumptions
A101	*Sufficient power, cooling, and floor/rack space is available in the existing datacenter to support the new infrastructure during normal and maintenance operations*
A102	*Sufficient 10G ports are available in the existing core switches to support the new infrastructure*
A103	*System services (DNS, NTP and DHCP) are available in the existing infrastructure to support the new services and applications*

FIGURE 7.3: A Case Study - Example of Design assumptions.

ID	Source	Derived Technical Requirements
T101	C101	*All components of the virtualization platform must communicate with DNS service*
	A103	
T102	C101	*Selected components of the virtualization platform - hyperviser (ESXi) and managenet (vCenter) servises - must communicate with NTP service*
	A103	
T103	C101	*Hyperviser services of the virtualization platform (ESXi) must communicate with the managemnt service of the virtualization platform (vCenter)*
	R106	
	R108	
	R109	
T104	C101	*Hyperviser services of the virtualization platform (ESXi) must communicate with vSphere Desktop Client with Update Manager*
	R106	
T105	C101	*vSphere Desktop Client and vSphere Web Client must communicate with the managemnt service of the virtualization platform (vCenter)*
	R106	
T106	C101	*Separate management VLAN must be used for the virtualization platform (vSphere) management traffic*
	R107	
T107	C101	*Separate management VLAN must be used for the live migration (vMotion) management traffic*
	R107	
	R108	
	R109	
T108	C101	*Separate management VLAN must be used for the storage area network (vSAN) management traffic*
	C104	
	R107	
	R108	
	R109	

FIGURE 7.4: A Case Study - Example of Derived technical requirements.

- VMware vSphere 6.0 virtualization platform [130];
- Application server based on Apache HTTP Server 2.4.17 [157] and Ubuntu Server 14.04 LTS [158];
- Two end-user components based on:
  - Internet Explorer 11.0.24 and Microsoft Windows 7 Professional SP1 [159];
  - Mozilla Firefox 51.0.2 for Ubuntu [160] and Ubuntu Desktop 14.04 LTS [158];
- Supporting network infrastructure.

The architecture design used for this test case according to the basic multilayer reference model (see Figure 3.7) is represented by the following figures:

- functional architectural layer - see Figure 7.5;
- service architectural layer - see Figure 7.6;
- logical architectural layer - see Figure 7.7;
- physical architectural layer - see Figure 7.8.

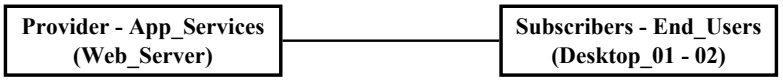


FIGURE 7.5: A Case Study - Functional architectural layer.

Next, the following figures illustrate the examples of detailed design documentation - technical specifications - based on the predefined design patterns (see Section 6.1):

- layer component specifications - see Figure 7.9;
- intralayer topology specifications - see Figure 7.10;
- interlayer topology specifications - see Figure 7.11.

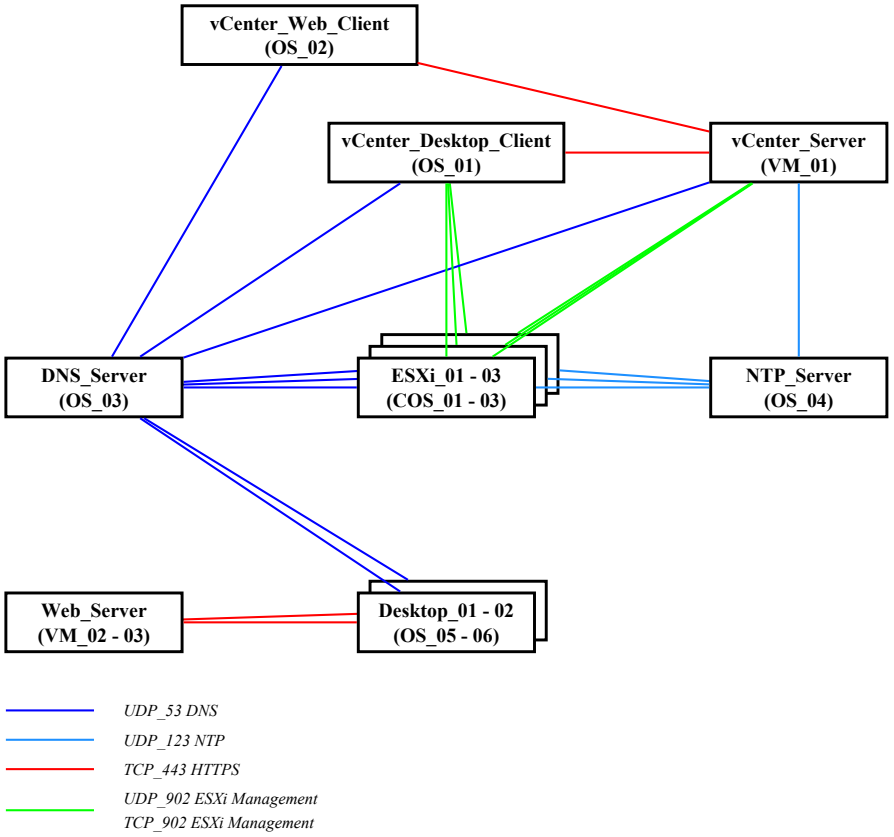


FIGURE 7.6: A Case Study - Service architectural layer.

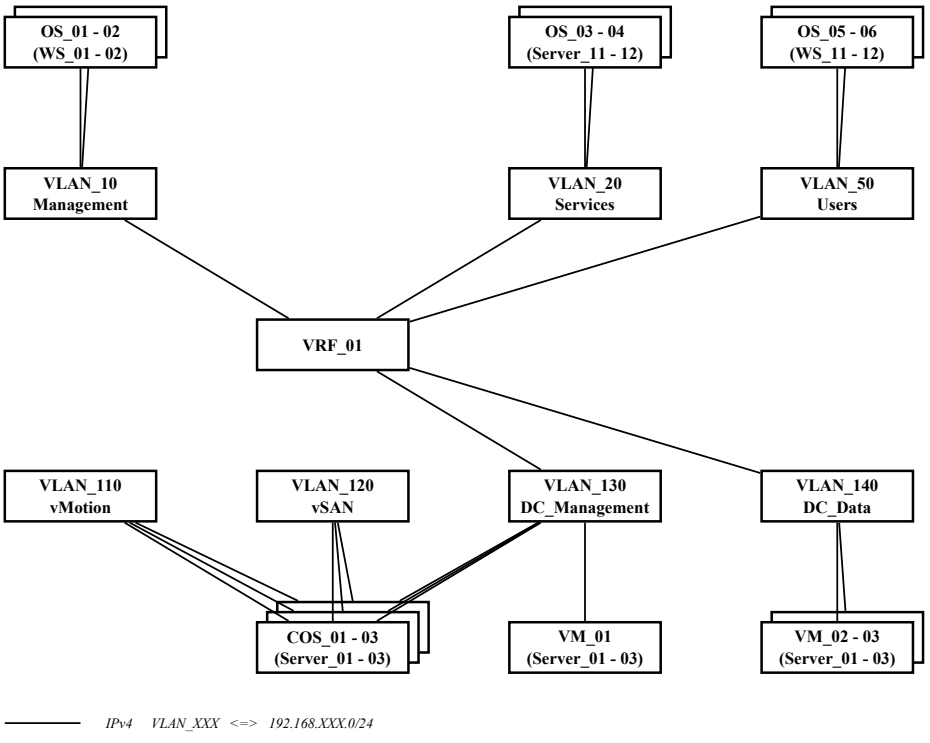


FIGURE 7.7: A Case Study - Logical architectural layer.

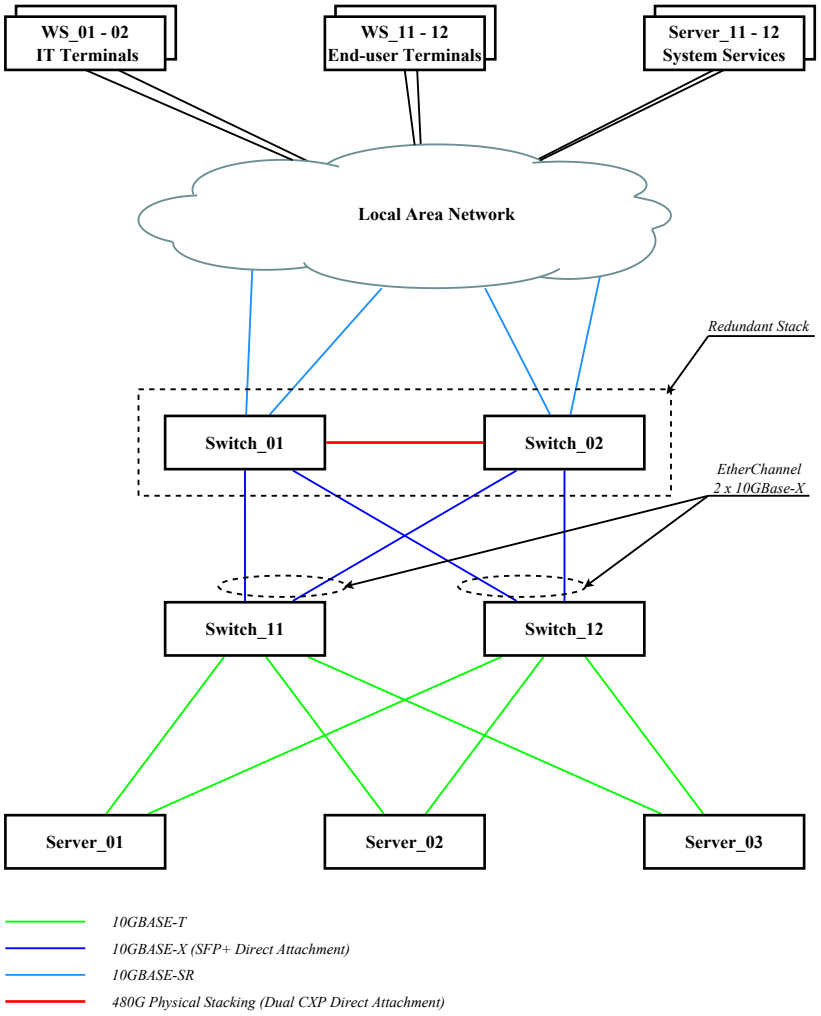


FIGURE 7.8: A Case Study - Physical architectural layer.

Record Number	Layer Identifier	Component Assignment	Component Identifier		Vendor Identifier	Component Attributes (3-tuples)			Notes
			Type	Index					
i	α	-	v_i^{α}		-	S_i^{α}			-
T1.Lα.01	**T1.Lα.02**	**T1.Lα.03**	**T1.Lα.04**		**T1.Lα.05**	**T1.Lα.06**			**T1.Lα.07**
1	4	Provider of Application Services	Provider	01	-	HTML/XML	INTD=100ms	Tasks=64	-
...	...	...	...	...	...	...	...	...	...
1	3	Apache HTTP Server	WEB	01	Apache Software Foundation	TCP 443	-	-	-
2	3	VMware vCenter Server	vCenter	01	VMware	UDP 53	-	-	-
						UDP 123	-	-	
						TCP 443	-	-	
						UDP/TCP 902	-	-	
3	3	VMware ESXi hypervisor 6.0	ESXi	01	VMware	UDP 53	-	-	-
						UDP 123	-	-	
						UDP/TCP 902	-	-	
...	...	...	...	...	...	...	...	...	...
1	2	VMware ESXi hypervisor 6.0	COS	01	VMware	IPv4	192.168.110.11	255.255.255.0	-
						IPv4	192.168.120.11	255.255.255.0	
						IPv4	192.168.130.11	255.255.255.0	
2	2	SUSE Linux Enterprise Server 12	VM	03	Novell	IPv4	192.168.140.12	255.255.255.0	-
3	2	VLAN vMotion	VLAN	110	-	IPv4	192.168.110.0	255.255.255.0	-
4	2	Virtual Router	VRF	01	-	IPv4	192.168.0.0	255.255.0.0	-
...	...	...	...	...	...	...	...	...	...
1	1	Dell PowerEdge R730xd Rack Server	Server	01	Dell	10GBASE-T	Full Duplex	-	-
2	1	DXS-3600-32S 10 Gigabit Managed Switch	Switch	01	D-Link	10GBASE-X	Full Duplex	-	-
						10GBASE-SR	Full Duplex	-	
						120G CXP	-	-	

FIGURE 7.9: A Case Study - Example of Layer component specifications.

Record Number	Layer Identifier	Link Identifier								Link Attributes (3-tuples)			Notes
		Source Identifier				Target Identifier							
		Component Identifier		Port Identifier		Component Identifier		Port Identifier					
		Type	Index	Type	Index	Type	Index	Type	Index				
n	α	v_i^{α}		-		v_j^{α}		-		S_{ij}^{α}			-
T2.La.01	T2.La.02	T2.La.04		T2.La.05		T2.La.06		T2.La.07		T2.La.08			T2.La.09
1	4	Provider	01	-	-	Subscriber	01	-	-	HTML/XML	INTD=100ms	Tasks=64	-
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1	3	ESXi	01	UDP	53	DNS	01	UDP	53	UDP 53	-	-	-
2	3	ESXi	01	UDP	123	NTP	01	UDP	123	UDP 123	-	-	-
3	3	ESXi	01	TCP/UDP	902	vCenter	01	TCP/UDP	902	UDP/TCP 902	-	-	-
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1	2	OS	01	IPv4	192.168.10.101/24	VLAN	10	IPv4	192.168.10.0/24	IPv4	192.168.10.0	255.255.255.0	-
2	2	COS	01	IPv4	192.168.110.11/24	VLAN	110	IPv4	192.168.110.0/24	IPv4	192.168.110.0	255.255.255.0	-
3	2	VM	01	IPv4	192.168.130.1/24	VLAN	130	IPv4	192.168.130.0/24	IPv4	192.168.130.0	255.255.255.0	-
4	2	VLAN	140	IPv4	192.168.140.0/24	VRF	01	IPv4	192.168.0.0/16	IPv4	192.168.0.0	255.255.0.0	-
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1	1	Switch	01	120G CXP	01, 02	Switch	02	120G CXP	01, 02	120G CXP	-	-	Stacking Ring
2	1	Switch	01	10GBASE-X	16	Switch	11	10GBASE-X	09	10GBASE-X	Full Duplex	-	EtherChannel 1
3	1	Switch	11	10GBASE-T	01	Server	01	10GBASE-T	01	10GBASE-T	Full Duplex	-	-
4	1	WS	01	1000BASE-T	01	LAN	00	1000BASE-T	00	1000BASE-T	-	-	-

FIGURE 7.10: A Case Study - Example of Intralayer topology specifications.

Record Number	Layer Identifier	Projection Assignment	Projection Identifier				Distribution Index	Projection Attributes	Notes
			Source Component Identifier		Target Component Identifier				
			Type	Index	Type	Index			
n	α	-	v_i^{α}		$v_j^{(\alpha-1)}$		-	-	-
T3.La.01	T3.La.02	T3.La.03	T3.La.04		T3.La.05		T3.La.06	T3.La.07	T3.La.08
1	4	-	Provider	01	WEB	01	1:1	-	-
2	4	-	Subscriber	01	Desktop	01	1:2	-	-
					Desktop	02	1:2	-	-
...	...	...	...	...	...	...	...	...	...
1	3	-	WEB	01	VM	02	1:2	Active/Standby	AppServer Cluster 1
					VM	03	1:2	Standby/Active	AppServer Cluster 1
2	3	-	ESXi	01	COS	01	1:1	-	-
3	3	-	vCenter	01	VM	01	1:1	-	-
...	...	...	...	...	...	...	...	...	...
1	2	-	OS	01	WS	01	1:1	-	-
2	2	-	COS	01	Server	01	1:1	-	-
3	2	-	VM	01	Server	01	1:3	-	-
					Server	02	1:3	-	
					Server	03	1:3	-	
4	2	-	VLAN	110	Switch	01	1:4	-	-
					Switch	02	1:4	-	
					Switch	11	1:4	-	
					Switch	12	1:4	-	

FIGURE 7.11: A Case Study - Example of Interlayer topology specifications.

7.3 Test Cases

The multilayer model derived from the detailed design documentation is shown in Figure 7.12.

The examples of detailed design documentation - technical specifications - based on the predefined design patterns (see Section 6.1) are illustrated by the following figures:

- test requirements for SUT components - see Figure 7.13;
- test requirements for SUT communication channels - see Figure 7.14;

Finally, Figure 7.15 - 7.17 represent the examples of the application of test generation strategies (check lists):

- test cases of SUT components - see Figure 7.15;

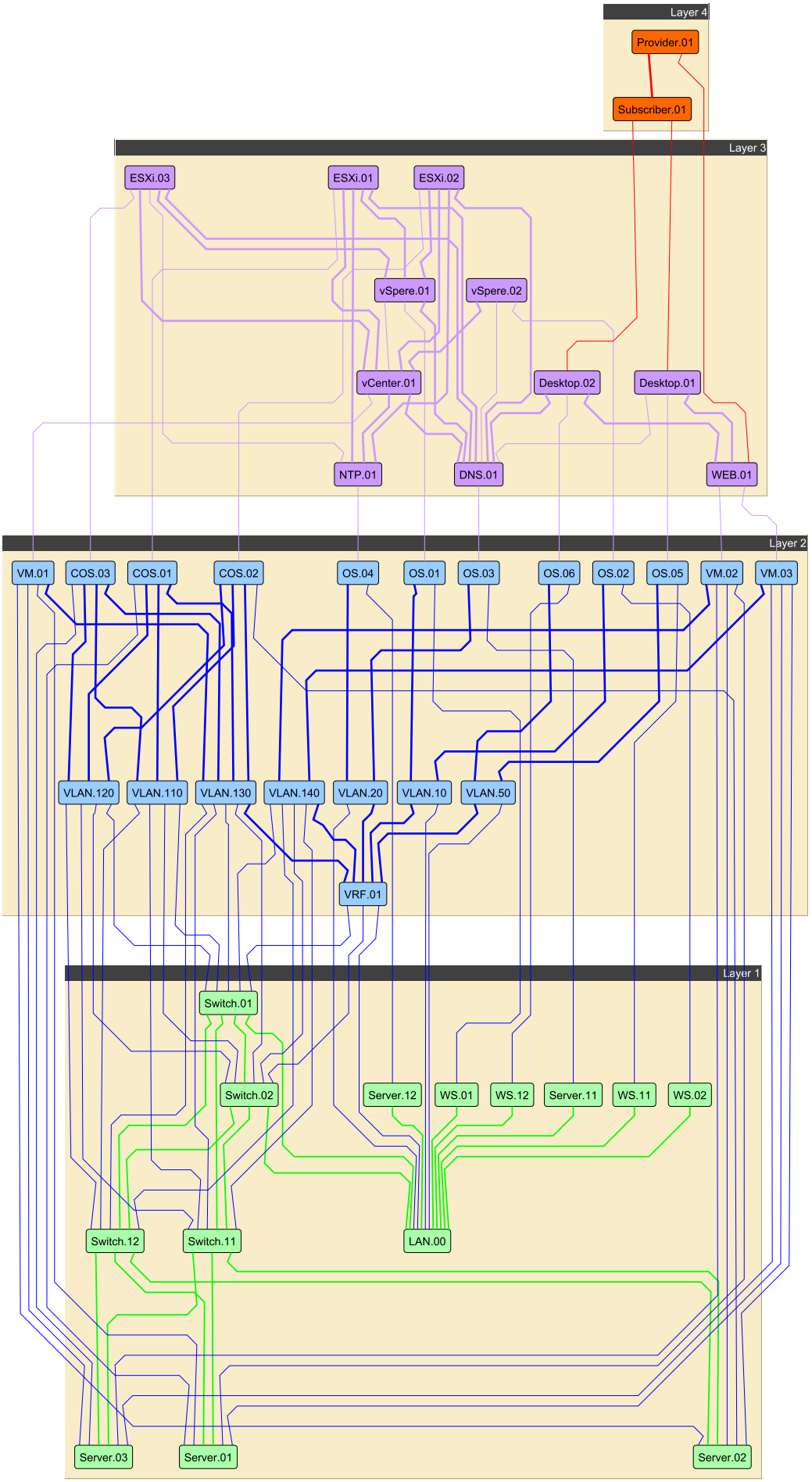


FIGURE 7.12: A Case Study - Multilayer model.

Record Number	Layer Identifier	Requirement Assignment	Component Identifier		Requirement Attributes (3-tuples)						Notes
			Type	Index							
i	α	-	v_i^α		A_i^α			$A_i^{\alpha,(\alpha-1)}$			-
T1.Lα.01	**T1.Lα.02**	**T1.Lα.03**	**T1.Lα.04**		**T1.Lα.06**						**T1.Lα.07**
1	4	R105	Provider	01	HTML/XML	-	-	-	-	-	End User Requirement
2	4	R105	Subscriber	01	-	-	-	-	-	-	End User Requirement
…	…	…	…	…	…	…	…	…	…	…	…
1	3	R105	WEB	01	TCP 443	-	-	-	-	-	End User Requirement
2	3	C101, R106, R108, R109	vCenter	01	-	-	-	-	-	-	End User Requirement
3	3	C101	ESXi	01	-	-	-	-	-	-	End User Requirement
…	…	…	…	…	…	…	…	…	…	…	…
1	2	C101	OS	01	-	-	-	-	-	-	End User Requirement
2	2	C101	COS	01	-	-	-	-	-	-	End User Requirement
3	2	C101, R108, R109	VM	01	-	-	-	-	-	-	End User Requirement
4	2	C101, R107, R108, R109	VLAN	110	-	-	-	-	-	-	End User Requirement
…	…	…	…	…	…	…	…	…	…	…	…
1	1	C102, C103	Server	01	10GBASE-T	-	-	-	-	-	End User Requirement
2	1	C102	WS	01	-	-	-	-	-	-	End User Requirement
3	1	C105	Switch	01	-	-	-	-	-	-	End User Requirement

FIGURE 7.13: A Case Study - Example of Test requirements for SUT components.

Record Number	Layer Identifier	Requirement Assignment	Link Identifier				Requirement Attributes (3-tuples)						Notes
			Source Identifier		Target Identifier								
			Type	Index	Type	Index							
n	α	-	v_i^α		v_j^α		A_{ij}^α			$A_{ij}^{\alpha,(\alpha-1)}$			-
T2.Lα.01	**T2.Lα.02**	**T2.Lα.03**	**T2.Lα.04**		**T2.Lα.06**		**T2.Lα.08**						**T2.Lα.09**
1	4	R103, R104, R105	Subscriber	01	Provider	01	HTML/XML	INTD ≤ 200ms	Tasks ≥ 16	TCP 443	-	-	End User Requirement
…	…	…	…	…	…	…	…	…	…	…	…	…	…
1	3	T101	DNS	01	ESXi	01	UDP 53	-	-	IPv4	-	-	Technical Requirement
2	3	T102	NTP	01	ESXi	01	UDP 123	-	-	IPv4	-	-	Technical Requirement
3	3	T103	vCenter	01	ESXi	01	UDP/TCP 902	-	-	IPv4	-	-	Technical Requirement
4	3	T105	vCenter	01	vSpere	01	TCP 443	-	-	IPv4	-	-	Technical Requirement
…	…	…	…	…	…	…	…	…	…	…	…	…	…
1	2	T107	VLAN	110	COS	01	IPv4	192.168.110.0	255.255.255.0	10GBASE-T	-	-	Technical Requirement
2	2	T108	VLAN	120	COS	01	IPv4	192.168.120.0	255.255.255.0	10GBASE-T	-	-	Technical Requirement

FIGURE 7.14: A Case Study - Example of Test requirements for SUT communication channels.

– test cases of SUT communication channels - see Figure 7.16;

– fault-injection test cases - see Figure 7.17.

Test cases of SUT components on layer α $T_{comp}^\alpha = \{[(v_i^\alpha, S_i^\alpha), A_i^\alpha]_n\}$								
Record Number	**Layer Identifier**	**Component Identifier**	**Component Attributes (3-tuples)**			**Requirement Attributes (3-tuples)**		
n	α	v_i^α	S_i^α			A_i^α		
1	1	Server, 01	10GBASE-T	Full Duplex	-	10GBASE-T	-	-
2	1	Switch, 01	10GBASE-X	Full Duplex	-	-	-	-
			10GBASE-SR	Full Duplex	-			
			120G CXP	-	-			
3	1	WS, 01	10BASE-T	-	-	-	-	-
			100BASE-T	-	-			
			1000BASE-T	-	-			
…	…	…	…	…	…	…	…	…
1	2	OS, 01	IPv4	192.168.10.101	255.255.255.0	-	-	-
2	2	COS, 01	IPv4	192.168.110.11	255.255.255.0	-	-	-
			IPv4	192.168.120.11	255.255.255.0			
			IPv4	192.168.130.11	255.255.255.0			
3	2	VLAN, 110	IPv4	192.168.110.0	255.255.255.0	-	-	-
…	…	…	…	…	…	…	…	…
1	3	WEB, 01	TCP 443	-	-	TCP 443	-	-
2	3	vCenter, 01	UDP 53	-	-	-	-	-
			UDP 123	-	-			
			TCP 443	-	-			
			UDP/TCP 902	-	-			
3	3	ESXi, 01	UDP 53	-	-	-	-	-
			UDP 123	-	-			
			UDP/TCP 902	-	-			

FIGURE 7.15: A Case Study - Example of Test cases of SUT components.

Test cases of SUT communication channels on layer α

$$T^{\alpha}_{llnk} = \left\{ \left[\left(\bigcup_{\langle v^{\alpha}_{(l-1)}, v^{\alpha}_{l} \rangle \in p^{\alpha}_{l,j,k}} \left(\langle v^{\alpha}_{(l-1)}, v^{\alpha}_{l} \rangle, S^{\alpha}_{(l-1),l} \right) \right), A^{\alpha}_{l,j} \right]_{n} \right\}$$

Record Number	Layer Identifier	Path Identifier	Attributes of component-to-component interconnections (3-tuples)			Requirement Attributes (3-tuples)		
n	α	$p^{\alpha}_{l,j,k} = \{\langle v^{\alpha}_{(l-1)}, v^{\alpha}_{l} \rangle\}$	$S^{\alpha}_{(l-1),l}$			$A^{\alpha}_{l,j}$		
1	1	(Switch, 01; Switch, 11)	-	-	-	-	-	-
		<Switch, 01; Switch, 11>	10GBASE-X	Full Duplex	-			
2	1	(Switch, 11; Server, 01)	-	-	-	-	-	-
		<Switch, 11; Server, 01>	10GBASE-T	Full Duplex	-			
3	1	(WS, 01; LAN, 00)	-	-	-	-	-	-
		<WS, 01; LAN, 00>	1000BASE-T	-	-			
…	…	…	…	…	…	…	…	…
1	2	(VLAN, 110; COS, 01)	-	-	-	IPv4	192.168.110.0	255.255.255.0
		<VLAN, 110; COS, 01>	IPv4	192.168.110.0	255.255.255.0			
2	2	(VM, 01; COS, 01)	-	-	-	IPv4	-	-
		<VM, 01; VLAN, 130>	IPv4	192.168.130.0	255.255.255.0			
		<VLAN, 130; COS, 01>	IPv4	192.168.130.0	255.255.255.0			
3	2	(OS, 01; COS, 01)	-	-	-	IPv4	-	-
		<OS, 01; VLAN, 10>	IPv4	192.168.10.0	255.255.255.0			
		<VLAN, 10; VRF, 01>	IPv4	192.168.10.0	255.255.255.0			
		<VRF, 01; VLAN, 130>	IPv4	192.168.130.0	255.255.255.0			
		<VLAN, 130; COS, 01>	IPv4	192.168.130.0	255.255.255.0			
…	…	…	…	…	…	…	…	…
1	3	(DNS, 01; ESXi, 01)	-	-	-	-	-	-
		<DNS, 01; ESXi, 01>	UDP 53	-	-			
2	3	(vCenter, 01; ESXi, 01)	-	-	-	-	-	-
		<vCenter, 01; ESXi, 01>	UDP/TCP 902	-	-			
2	3	(vSphere, 01; vCenter, 01)	-	-	-	-	-	-
		<vSphere, 01; vCenter, 01>	TCP 443	-	-			
3	3	(Desktop, 01; WEB, 01)	-	-	-	TCP 443	-	-
		<Desktop, 01; WEB, 01>	TCP 443	-	-			

FIGURE 7.16: A Case Study - Example of Test cases of SUT communication channels.

Fault-injection test cases on layer α

$$T^{(\alpha+1),\alpha} = \{v^{\alpha}_{l}\}$$

Record Number	Layer Identifier	Recovery Group	Fault-Injection Test Cases
n	α	$RG^{(\alpha+1),\alpha}$	$T^{(\alpha+1),\alpha}$
1	1	Server, 01; Server, 02; Server, 03; Switch, 11; Switch, 12; Switch, 01; Switch, 02; WS, 11; WS, 12	Server, 01; Server, 02; Server, 03; Switch, 11; Switch, 12; Switch, 01; Switch, 02
2	2	VM, 02; VM, 03; OS, 05; OS, 06	VM, 02; VM, 03
3	3	Desktop, 01; Desktop, 02	-

FIGURE 7.17: A Case Study - Example of Fault-injection test cases.

In general, the result of applying test generation strategies shows that a surprisingly large number of test cases are required to fully cover the structure of this *extremely simple* model - see Table 7.1.

As mentioned above, the case study covers only one application server and two desktop personal computers. In practice, the project covered sixteen application servers and forty-eight thin clients[4]. The results of the application of test generation strategies in practice is shown in Table 7.2.

By and large, this result can easily reflect the existence of the large amount of potential faults in commercial systems. Increasing system complexity and fierce market competition on time-to-market and cost make comprehensive testing of complex network

[4]One of the main project goals was the replacement of desktop personal computers by thin clients.

TABLE 7.1: The result of applying test generation strategies.

No.	**Model Cardinality**				**Test Case Numbers**		
	Architectural layer	α	$\lvert V^{\alpha}\rvert$	$\lvert E^{\alpha}\rvert$	$\lvert T^{\alpha}_{comp}\rvert$	$\lvert T^{\alpha}_{link}\rvert$	$\lvert T^{(\alpha+1),\alpha}\rvert$
1	Functional layer	4	2	1	–	–	–
2	Social layer	–	–	–	–	–	–
3	Service layer	3	11	22	7	20	–
4	Logical layer	2	20	23	13	28	2
5	Physical layer	1	14	19	9	15	7
6	Engineering layer	–	–	–	–	–	–
Total:					**29**	**63**	**9**
					101		

TABLE 7.2: The result of applying test generation strategies in practice.

No.	**Test Case Numbers**				
	Architectural layer	α	$\lvert T^{\alpha}_{comp}\rvert$	$\lvert T^{\alpha}_{link}\rvert$	$\lvert T^{(\alpha+1),\alpha}\rvert$
1	Functional layer	4	–	–	–
2	Social layer	–	–	–	–
3	Service layer	3	70	258	–
4	Logical layer	2	139	312	16
5	Physical layer	1	57	63	7
6	Engineering layer	–	–	–	–
Total:			**266**	**633**	**23**
			922		

systems very difficult (or even impossible) without appropriate formal methods.

Chapter 8

Conclusion and Future Work

Aim for the moon. If you miss, you may hit a star.

—W. Clement Stone

The last chapter of this thesis is divided into two sections. The first section contains a general summary (i.e. what was done in accordance with the thesis goals). Then, the second part emphasizes two aspects of future work - the next and future possible steps

8.1 What Was Done

Who seeks shall find.

—Sophocles

Deployment of commercial computer networks sets high requirements for procedures, tools and approaches for comprehensive testing of these networks. At the same time, despite the great efforts of many researchers, the process of test design/generation still tends to be unstructured and bound to the personal experience and/or intuition of individual engineers. However, in the case of complex or non-standard networks, personal experience and/or intuition are often inadequate. As a consequence, in the real world many computer networks have failed because:

– engineers had tested the wrong things;

– engineers had tested the right things but in the wrong way;

– some things had been just simply forgotten and had not been tested.

To address this problem, the main research objective of this thesis is the automated design of test specifications (test cases) for computer networks using the detailed design documentation (end-user requirements and technical specifications) as the data source. Based on the model-based approach, the following main goals have been solved separately, but not in isolation:

– A formal model for test generation missions was defined based on the concept of multilayer networks. The model is the layered 3D-graph which can be derived directly from the SUT technical specification. Different layers (four layers in the case of basic releases and six layers in the case of extended releases) represent different (hardware, software, social, business, etc.) aspects of SUT architecture. In turn, interlayer relations: (1) represent the technological solutions which were used to build the SUT; and (2) make the layered model consistent. This model completely covers all layers of OSI Reference Model (moreover, it covers some additional layers beyond the OSI RM) and, as a consequence, both software-based and network-based aspects of computer networks with regard to applying the system methodology to network analysis.

– Using the models of this kind and the graph theoretical metrics, both static and dynamic network anal-yses were performed. In the context of this thesis, the static analysis determines the structural test case generation strategy. This strategy is based on the top-down approach and uses test requirements as source data. The top-down approach utilizes the concept of layered networks the concept of layered networks, which strictly relies on the facts that: (1) for each node on a given layer there is a corresponding node (or nodes) on the layer below; and (2) for each logical path between two nodes on a given layer there is a path (or paths) between the corresponding nodes on the layer below. As a consequence, test cases of this kind: (1) cover the system infrastructure including individual components and component-to-component in-teraction on all coexisting architectural layers; and (2) provide information for subsequent analysis to ensure that the formal model is consistent with respect to the test requirements.

– In turn, the dynamic analysis (or fault injection simulation) provides a means for understanding how the SUT behaves in the presence of faults. In the context of this thesis, the dynamic analysis determines the nonfunctional (dependability) test case generation strategy. The strategy is also based on the top-down approach and uses structural test cases as source data. The analysis includes two main steps: (1) successive removals of vertices and their incident edges from the formal model (fault injection experiments); and (2) impact assessments of those removals on the model consistency – disruption on an arbitrary layer might destroy a substantial part of the upper layer (or layers) that are mapped on it, rendering the whole network useless in practice. In other words, test cases such as these define SUT components on all coexisting architectural layers as objects of fault injection experiments. In general, they provide information for subsequent analysis to ensure that: (1) system dependability mechanisms have been implemented correctly on all coexisting architectural layers and, as a consequence, (2) the system is able to provide the desired level of reliable services.

– It is important to note that the quality of formal methods based on abstract models is limited by the quality of these models. Thus, to get the full advantages of model-based testing, it is necessary to alleviate the burdens of learning model development and checking techniques for engineers and other non-technical stakeholders or, ideally, completely eliminate the human factor. To address this problem, a possible appropriate presentation format of architecture descriptions was defined as a necessary part of the detailed design documentation of computer networks: complex network architecture can be unambiguously represented based on the set of tables (design patterns). In turn, this set of tables provides an unambiguous definition of the formal model (3D graph) for analysis and verification of the network structure. As a consequence, the human factor was completely eliminated from the data transformation processes during the formal model generation activities – the process was done in automated mode using the detailed design documentation as data source. The resulting formal model is completely relevant to the design documentation (a trusted model from the viewpoint of network/system designers).

Conclusions:

– The unambiguous formal presentation of a completed computing system (i.e. a computer network and the distributed computing system that this network provides and supports) based on the concept of multilayer networks (the layered formal model) is possible.

– Applying the system methodology to network analysis based on the layered formal model en-sures that computer network architecture is completely consistent with respect to: (1) distributed computing system architecture which this network provides and supports; (2) system functional (business) goals.

– Formal model automated generation using the detailed design documentation as data source is possible.

– Test case automated generation based on the layered formal model is possible on all coex-isting architectural layers. The resulting test cases can be used for: (1) verification of a complet-ed computing system (or a computer network only); (2) validation that the system (or network) technical specifications are consistent with respect to the end-user requirements.

– The techniques of the transformation of informal end-user requirements into formal operation specifications (test requirements) require a separate detailed analysis[1] - even in the case of relatively simple systems, it is not a routine exercise in practice.

8.2 Future Work

The more we do, the more we can do.

—William Hazlitt

In general, the goals of this thesis do not cover the problem of cybersecurity in spite of the facts that: (1) the problem exists; and (2) it is growing every year [162]. Nowadays, computer networks have critical security requirements. As mentioned above, their failure may endanger human lives and the environment, do serious damage to major economic infrastructures, endanger personal privacy, undermine the viability of whole business sectors and facilitate crime [3]. As a consequence, future work will target the problem of security testing.

[1] A possible solution might lay in the domain of Artificial Intelligence - Ontological Engineering [161]

The preliminary analysis shows that test cases of SUT communication channels (see Definition 12) contain information (see Figure 7.16) which can be used for the following purposes:

- Automated generation of network security rules (access lists and firewall configurations).
- Checking the network security rules consistency with respect to system functional (business) goals[2].

This result can be used as a basis for the next possible steps. On the other hand, it is important to note that these network security rules do not cover all possible threats. In turn, the current revision of ISO/IEC 27005:2011 [135] standard contains a list of typical threats, which covers both aspects (software-based and network-based) of computer networks, but not only these aspects.

The preliminary analysis [136] shows that the set of typical threats can be partitioned with regard to the extended multilayer reference model (see Section 3.2) and the protected objects - SUT components and SUT communication channels (see Figure 8.1). The result of the layer-by-layer mapping from the set of typical threats to the set of protected objects might be a basis for the necessary security checklists - a set of security (penetration) test cases.

[2]Generally, it is impossible to separate: (1) components/communication channels failures; and (2) components/communication channels security misconfiguration. In both cases, these components/communication channels are in failure state.

THREATS	EXTENDED REFERENCE MODEL LAYERS									
	Components					Communication channels				
	1	2	3	4	5	1	2	3	4	5
T 0.01 Fire	x									
T 0.02 Unfavorable climatic conditions	x									
T 0.03 Water	x									
T 0.04 Pollution, dust, corrosion	x									
T 0.05 Natural disasters	x									
T 0.06 Environmental disasters	x					x				
T 0.07 Major events in the environment	x					x				
T 0.08 Failure or disruption of the power supply	x									
T 0.09 Failure or disruption of communication networks		x					x			
T 0.10 Failure or disruption of mains supply	x									
T 0.11 Failure or disruption of service providers								x		
T 0.12 Interfering radiation	x					x				
T 0.13 Intercepting compromising emissions	x					x				
T 0.14 Interception of information / espionage				x					x	
T 0.15 Eavesdropping		x					x			
T 0.16 Theft of devices, storage media and documents	x									
T 0.17 Loss of devices, storage media and documents					x					
T 0.18 Bad planning or lack of adaptation					x					x
T 0.19 Disclosure of sensitive information				x					x	
T 0.20 Information or Product from an unreliable source					x					
T 0.21 Manipulation of hardware and software					x					
T 0.22 Manipulation of information				x						
T 0.23 Unauthorized access to IT systems			x	x						
T 0.24 Destruction of devices or storage media					x					
T 0.25 Failure of devices or systems		x								
T 0.26 Malfunction of devices or systems		x								
T 0.27 Lack of resources	x					x				
T 0.28 Software vulnerabilities or errors			x	x						
T 0.29 Violation of laws or regulations					x					x
T 0.30 Unauthorized use or administration of devices and systems		x	x	x			x	x	x	
T 0.31 Incorrect use or administration of devices and systems			x	x				x	x	
T 0.32 Abuse of authorizations			x	x				x	x	
T 0.33 Absence of personal					x					
T 0.34 Attack	x									
T 0.35 Coercion, extortion or corruption					x					
T 0.36 Identity theft					x					
T 0.37 Repudiation of actions					x					
T 0.38 Abuse of personal data					x					
T 0.39 Malicious software				x						
T 0.40 Denial of service				x						
T 0.41 Sabotage					x					
T 0.42 Social Engineering					x					
T 0.43 Replay of messages				x						
T 0.44 Unauthorized entry to premises	x									
T 0.45 Data loss				x						
T 0.46 Loss of integrity of sensitive information				x						

FIGURE 8.1: Partitioned list of typical threats [136].

Appendix A

Author's Publications

Parts of this thesis have been resulted in the following publications:

- Journals with Impact Factors / Web of Science (WoS) Journals:
  - Shchurov A. and Mařík R.: A Multilayer Approach to Commercial Computer Networks Testing. *IEEE Access*. 2017, vol. 5, p. 11083-11099. ISSN 2169-3536.
- Other Journals:
  - Shchurov A. and Mařík R.: A Trusted Model of Complex Computer Networks. *Journal of ICT Standardization*. 2016, vol. 3, no. 3, p. 201-230. ISSN 2245-800X
  - Shchurov A. and Mařík R.: A Simple Taxonomy of Multilayer Networks. *International Journal of Computer Trends and Technology*. 2016, vol. 28, no. 1, p. 76-80. ISSN 2231-2803
  - Shchurov A.: A Multilayer Model of Computer Networks. *International Journal of Computer Trends and Technology*. 2015, vol. 26, no. 1, p. 12-16. ISSN 2231-2803
  - Shchurov A. and Mařík R.: Dependability Tests Selection Based on the Concept of Layered Networks. *International Journal of Scientific and Engineering Research*. 2015, vol. 6, no. 1, p. 1165-1174. ISSN 2229-5518

- Shchurov A., Mařík R. and Khlevnoy V.: A Formal Approach to Network/Distributed Systems Complex Testing. *International Journal of Computer Trends and Technology.* 2015, vol. 22, no. 2, p. 76-80. ISSN 2231-2803
- Shchurov A.: Industrial Computing Systems: A Case Study of Fault Tolerance Analysis. *International Journal of Computer Trends and Technology.* 2015, vol. 21, no. 1, p. 50-55. ISSN 2231-2803
- Khlevnoy V. and Shchurov A.: A Formal Approach to Distributed System Security Test Generation. *International Journal of Computer Trends and Technology.* 2014, vol. 16, no. 3, p. 121-127. ISSN 2231-2803
- Shchurov A.: A Formal Model of Distributed Systems for Test Generation Missions. *International Journal of Computer Trends and Technology.* 2014, vol. 15, no. 3, p. 128-133. ISSN 2231-2803
- Shchurov A. and Mařík R.: A Formal Approach to Distributed System Tests Design. *International Journal of Computer and Information Technology.* 2014, vol. 3, no. 4, p. 696-705. ISSN 2279-0764

– Conference Proceedings:

- Shchurov A. and Mařík R.: A Presentation Format of Architecture Description Based on The Concept of Multilayer Networks. In *Proceedings of ITU Kaleidoscope Academic Conference: Trust in the Information Society (K-2015).* Geneva: ITU, 2015, p. 227-232. ISBN 978-92-61-15821-7

Bibliography

[1] James D. McCabe. *Network Analysis, Architecture, and Design.* Morgan Kaufmann, 3rd edition, 2007.

[2] Priscilla Oppenheimer. *Top-Down Network Design.* Cisco Press, 3rd edition, 2010.

[3] Nancy G. Leveson. *Safeware: system safety and computers.* ACM, 1995.

[4] Mark Utting, Alexander Pretschner, and Bruno Legeard. A taxonomy of model-based testing approaches. *Softw. Test. Verif. Reliab.*, 22(5):297–312, August 2012.

[5] Edsger Wybe Dijkstra, C. A. R. Hoare, and Ole-Johan Dahl. *Structured Proguamming.* A.P.I.C. studies in data processing, no. 8. Academic Press, 1972.

[6] Thomas A. Limoncelli, Christina J. Hogan, and Strata R. Chalup. *The Practice of System and Network Administration.* Addison Wesley, 2nd edition, 2007.

[7] Wikipedia. Introduction to Software Engineering/Process/Life Cycle, . URL https://en.wikibooks.org/wiki/Introduction_to_Software_Engineering/Process/Life_Cycle.

[8] Andrew S. Tanenbaum and Maarten van Steen. *Distributed Systems: Principles and Paradigms.* Prentice Hall Press, 3rd edition, 2013.

[9] Robert W. Buchanan Jr. *The art of testing network systems.* Wiley Publishing, 1996.

[10] Dennis M. Buede. *The Engineering Design of Systems: Models and Methods.* Wiley Publishing, 2nd edition, 2009.

[11] Russ White and Denise Donohue. *The Art of Network Architecture: Business-Driven Design.* Cisco Press, 1st edition, 2014.

[12] Shaun L. Hummel. *Cisco Design Fundamentals: Multilayered Design Approach For Network Engineers*. Cisco Press, 1st edition, 2015.

[13] Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui, and John Fitzgerald. Formal methods: Practice and experience. *ACM Comput. Surv.*, 41(4):19:1–19:36, October 2009.

[14] Wikipedia. Model-based testing, . URL https://en.wikipedia.org/wiki/Model-based_testing.

[15] Justyna Zander, Ina Schieferdecker, and Pieter J. Mosterman. *A Taxonomy of Model-Based Testing for Embedded Systems from Multiple Industry Domains*, pages 1–22. CRC Press, 2011.

[16] Daniel Aceituna, Hyunsook Do, and Sudarshan Srinivasan. A systematic approach to transforming system requirements into model checking specifications. In *Companion Proceedings of the 36th International Conference on Software Engineering (ICSE Companion 2014)*, pages 165–174, June 2014.

[17] ISO/IEC/IEEE Std 24765:2010(E) - Systems and software engineering - Vocabulary, 2010.

[18] Elizabeth Hull, Ken Jackson, and Jeremy Dick. *Requirements Engineering*. Springer, 3rd edition, 2011.

[19] Mark Utting and Bruno Legeard. *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann, 2007.

[20] George Din, Klaus-Dietrich Engel, and Axel Rennoch. An approach for test derivation from system architecture models applied to embedded systems. In *In Proceedings of the 2nd Workshop on Model-based Testing in Practice (MoTiP 2009), In Conjunction with the 5th European Conference on Model-Driven Architecture (ECMDA 2009)*, pages 23–32, June 2009.

[21] P. Stocks and D. Carrington. A framework for specification-based testing. *Software Engineering, IEEE Transactions on*, 22(11):777–793, November 1996.

[22] A Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *Dependable and Secure Computing, IEEE Transactions on*, 1(1):11–33, January 2004.

[23] Jie Liu and Edward A. Lee. A component-based approach to modeling and simulating mixed-signal and hybrid systems. *ACM Trans. Model. Comput. Simul.*, 12 (4):343–368, October 2002.

[24] J. Eker, J.W. Janneck, E.A Lee, Jie Liu, Xiaojun Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Yuhong Xiong. Taming heterogeneity - the ptolemy approach. *Proceedings of the IEEE*, 91(1):127–144, January 2003.

[25] Wikipedia. Homogeneity and heterogeneity, . URL https://en.wikipedia.org/wiki/Homogeneity_and_heterogeneity.

[26] Om Prakash Yadav, Nanua Singh, and Parveen S. Goel. Reliability demonstration test planning: A three dimensional consideration. *Reliability Engineering & System Safety*, 91(8):882–893, August 2006.

[27] Konstantin Benz and Thomas Bohnert. Dependability modeling framework: A test procedure for high availability in cloud operating systems. In *Vehicular Technology Conference (VTC Fall), 2013 IEEE 78th*, pages 1–8, September 2013.

[28] Steven H. Strogatz. Exploring complex networks. *Nature*, 410:268–276, March 2001.

[29] Reka Albert and Albert-Laszlo Barabasi. Statistical mechanics of complex networks. *Rev. Mod. Phys.*, 74(1):47–97, January 2002.

[30] Mark Newman. The structure and function of complex networks. *SIAM Review*, 45(2):167–256, May 2003.

[31] S. Boccaletti, G. Bianconi, R. Criado, C.I. del Genio, J. Gomez-Gardenes, M. Romance, I. Sendina-Nadal, Z. Wang, and M. Zanin. The structure and dynamics of multilayer networks. *Physics Reports*, 544(1):1–122, November 2014.

[32] Geoffrey G. Xie, Jibin Zhan, David A. Maltz, Hui Zhang, Albert Greenberg, Gisli Hjalmtysson, and Jennifer Rexford. On static reachability analysis of ip networks. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 3, pages 2170–2183, March 2005.

[33] Petr Matousek, Jaroslav Rab, Ondrej Rysavy, and Miroslav Sveda. A formal model for network-wide security analysis. In *Engineering of Computer Based Systems,*

2008. ECBS 2008. 15th Annual IEEE International Conference and Workshop on the, pages 171–181, March 2008.

[34] Manlio De Domenico, Albert Sole-Ribalta, Emanuele Cozzo, Mikko Kivela, Yamir Moreno, Mason A. Porter, Sergio Gomez, and Alex Arenas. Mathematical formulation of multilayer networks. *Phys. Rev. X*, 3(4):041022, December 2013.

[35] Mikko Kivela, Alex Arenas, Marc Barthelemy, James P. Gleeson, Yamir Moreno, and Mason A. Porter. Multilayer networks. *Journal of Complex Networks*, 2(3): 203–271, July 2014.

[36] Maciej Kurant and Patrick Thiran. Layered complex networks. *Phys. Rev. Lett.*, 96(13):138701, April 2006.

[37] Andrey A. Shchurov and Radek Marik. A simple taxonomy of multilayer networks. *International Journal of Computer Trends and Technology*, 31(1):20–24, January 2016.

[38] Luis Sola, Miguel Romance, Regino Criado, Julio Flores, Alejandro Garcia del Amo, and Stefano Boccaletti. Eigenvector centrality of nodes in multiplex networks. *Chaos*, 23(3):033131, September 2013.

[39] V. Stroele, J. Oliveira, G. Zimbrao, and J.M. Souza. Mining and analyzing multirelational social networks. In *Computational Science and Engineering, 2009. CSE '09. International Conference on*, volume 4, pages 711–716, August 2009.

[40] Philippa E. Pattison and Stan Wasserman. Logit models and logistic regressions for social networks: Ii. multivariate relations. *British Journal of Mathematical and Statistical Psychology*, 52:169–193, November 1999.

[41] Matteo Barigozzi, Giorgio Fagiolo, and Diego Garlaschelli. Multinetwork of international trade: A commodity-specific analysis. *Phys. Rev. E*, 81:046104, April 2010.

[42] Deng Cai, Zheng Shao, Xiaofei He, Xifeng Yan, and Jiawei Han. Community mining from multi-relational networks. In *In Proceedings of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 445–452, October 2005.

[43] M. Berlingerio, M. Coscia, F. Giannotti, A. Monreale, and D. Pedreschi. Foundations of multidimensional network analysis. In *Advances in Social Networks Analysis and Mining (ASONAM), 2011 International Conference on*, pages 485–489, July 2011.

[44] Peter J. Mucha and Mason A. Porter. Communities in multislice voting networks. *Chaos*, 20(4):041108, December 2010.

[45] Peter J. Mucha, Thomas Richardson, Kevin Macon, Mason A. Porter, and Jukka-Pekka Onne. Community structure in time-dependent, multiscale and multiplex networks. *Science*, 328:876–878, May 2010.

[46] Sebastian Funk and Vincent A. A. Jansen. Interacting epidemics on overlay networks. *Phys. Rev. E*, 81(3):036118, March 2010.

[47] Petter Holme and Jari Saramaki. Temporal networks. *Physics Reports*, 519(3): 97–125, March 2012.

[48] Matthew Rocklin and Ali Pinar. Latent clustering on graphs with multiple edge types. In *Proceedings of the 8th International Conference on Algorithms and Models for the Web Graph*, pages 38–49, May 2011.

[49] Regino Criado, Miguel Romance, and M. Vela-Perez. Hyperstructures, a new approach to complex systems. *I. J. Bifurcation and Chaos*, 20(3):877–883, March 2010.

[50] Peng Wang, Garry Robins, Philippa Pattison, and Emmanuel Lazega. Exponential random graph models for multilevel networks. *Social Networks*, 35(1):96–115, January 2013.

[51] Francesco Sorrentino. Synchronization of hypernetworks of coupled dynamical systems. *New Journal of Physics*, 14(3):033035, March 2012.

[52] Claude Berge. *Hypergraphs: Combinatorics of Finite Sets*. North Holland, 1989.

[53] Ann Wong-Jiru. *Graph Theoretical Analysis of Network-centric Operations Using Multi-layer Models*. BiblioScholar, 2012.

[54] Mark Newman. Mixing patterns in networks. *Phys. Rev. E*, 67:026126, March 2003.

[55] Alexei Vazquez. Spreading dynamics on heterogeneous populations: Multitype network approach. *Phys. Rev. E*, 74(6):066114, December 2006.

[56] Maciej Kurant, Patrick Thiran, and Patric Hagmann. Error and attack tolerance of layered complex networks. *Phys. Rev. E*, 76(2):026103, August 2007.

[57] Ding Zhou, Sergey A. Orshanskiy, Hongyuan Zha, and C. Lee Giles. Co-ranking authors and documents in a heterogeneous network. In *Proceedings of the 2007 Seventh IEEE International Conference on Data Mining*, pages 739–744, October 2007.

[58] Mark Dickison, S. Havlin, and H. E. Stanley. Epidemics on interconnected networks. *Phys. Rev. E*, 85(6):066109, June 2012.

[59] Anna Saumell-Mendiola, M. Angeles Serrano, and Marian Boguna. Epidemic spreading on interconnected networks. *Phys. Rev. E*, 86(2):026106, August 2012.

[60] Jonathan F. Donges, Hanna C. H. Schultz, Norbert Marwan, Yong Zou, and Jurgen Kurths. Investigating the topology of interacting networks - theory and application to coupled climate subnetworks. *Eur. Phys. J. B*, 84(4):635–651, April 2011.

[61] Sergey V. Buldyrev, Roni Parshani, Gerald Paul, H. Eugene Stanley, and Shlomo Havlin. Catastrophic cascade of failures in interdependent networks. *Nature*, 464: 1025–1028, April 2010.

[62] Roni Parshani, Sergey V. Buldyrev, and Shlomo Havlin. Interdependent networks: Reducing the coupling strength leads to a change from a first to second order percolation transition. *Phys. Rev. Lett.*, 105(4):048701, July 2010.

[63] Jianxi Gao, Sergey V. Buldyrev, Shlomo Havlin, and H. Eugene Stanley. Robustness of a network of networks. *Phys. Rev. Lett.*, 107(19):195701, November 2011.

[64] Manfred Broy, Bengt Jonsson, Joost-Pieter Katoen, Martin Leucker, and Alexander Pretschner. *Model-Based Testing of Reactive Systems: Advanced Lectures (Lecture Notes in Computer Science)*. Springer, 2005.

[65] Arilo C. Dias Neto, Rajesh Subramanyan, Marlon Vieira, and Guilherme H. Travassos. A survey on model-based testing approaches: A systematic review.

In *Proceedings of the 1st ACM International Workshop on Empirical Assessment of Software Engineering Languages and Technologies: held in Conjunction with the 22Nd IEEE/ACM International Conference on Automated Software Engineering (ASE) 2007*, pages 31–36, November 2007.

[66] Robert M. Hierons, Kirill Bogdanov, Jonathan P. Bowen, Rance Cleaveland, John Derrick, Jeremy Dick, Marian Gheorghe, Mark Harman, Kalpesh Kapoor, Paul Krause, Gerald Luttgen, Anthony J. H. Simons, Sergiy Vilkomir, Martin R. Woodward, and Hussein Zedan. Using formal specifications to support testing. *ACM Comput. Surv.*, 41(2):9:1–9:76, February 2009.

[67] S.R. Dalal, A. Jain, N. Karunanithi, J. M. Leaton, C.M. Lott, G.C. Patton, and B.M. Horowitz. Model-based testing in practice. In *Software Engineering, 1999. Proceedings of the 1999 International Conference on*, pages 285–294, May 1999.

[68] J.J. Marciniak, editor. *Encyclopedia on Software Engineering.* Wiley Publishing, 2001.

[69] Jeff Offutt and Aynur Abdurazik. Generating tests from uml specifications. In Robert France and Bernhard Rumpe, editors, *The Unified Modeling Language*, pages 416–429. Springer, 1999.

[70] Jean Hartmann, Claudio Imoberdorf, and Michael Meisinger. Uml-based integration testing. *SIGSOFT Softw. Eng. Notes*, 25(5):60–70, August 2000.

[71] Fredrik Abbors, Veli-Matti Aho, Jani Koivulainen, Risto Teittinen, and Dragos Truscan. *Applying Model-Based Testing in the Telecommunication Domain*, pages 487–524. CRC Press, 2011.

[72] Jan Peleska. Industrial-strength model-based testing - state of the art and current challenges. In *Proceedings Eighth Workshop on Model-Based Testing (MBT 2013)*, pages 3–28, March 2013.

[73] Michael R. Donat. Automating formal specification-based testing. In *TAPSOFT '97 Proceedings of the 7th International Joint Conference CAAP/FASE on Theory and Practice of Software Development*, pages 833–847, April 1997.

[74] G. Bernot, M.-C. Gaudel, and B. Marre. Software testing based on formal specifications: a theory and a tool. *Software Engineering Journal*, 6(6):387–405, November 1991.

[75] Jeremy Dick and Alain Faivre. Automating the generation and sequencing of test cases from model-based specifications. In *Proceedings of the First International Symposium of Formal Methods Europe on Industrial-Strength Formal Methods*, pages 268–284, April 1993.

[76] Hyoung Seok Hong, Sung-Deok Cha, Insup Lee, O. Sokolsky, and H. Ural. Data flow testing as model checking. In *Software Engineering, 2003. Proceedings. 25th International Conference on*, pages 232–242, May 2003.

[77] Shaoying Liu and Wuwei Shen. A formal approach to testing programs in practice. In *Systems and Informatics (ICSAI), 2012 International Conference on*, pages 2509–2515, May 2012.

[78] Financial Times. Commercial products. URL http://lexicon.ft.com/term?term=commercial-product.

[79] Hans Langmaack, Willem-Paul de Roever, and Jan Vytopil, editors. *Formal Techniques in Real-Time and Fault-Tolerant Systems: Third International Symposium Organized Jointly with the Working Group Provably Correct Systems, ProCoS, Lubeck, Germany, September 19-23, 1994 Proceedings*. Springer, 1994.

[80] J.A Clark and D.K. Pradhan. Fault injection: a method for validating computer-system dependability. *Computer*, 28(6):47–56, June 1995.

[81] Dimiter Avresky, Jean Arlat, Jean-Claude Laprie, and Yves Crouzet. Fault injection for formal testing of fault tolerance. *Reliability, IEEE Transactions on*, 45 (3):443–455, September 1996.

[82] Alfredo Benso and Paolo Prinetto, editors. *Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation*. Springer, 2003.

[83] C. Bolchini, A. Miele, and D. Sciuto. Fault models and injection strategies in systemc specifications. In *Digital System Design Architectures, Methods and Tools, 2008. DSD '08. 11th EUROMICRO Conference on*, pages 88–95, September 2008.

[84] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J.-C. Fabre, J.-C. Laprie, E. Martins, and D. Powell. Fault injection for dependability validation: a methodology and some applications. *Software Engineering, IEEE Transactions on*, 16(2):166–182, February 1990.

[85] D. Appello, P. Bernardi, S. Gerardin, M. Grosso, A. Paccagnella, P. Rech, and M.S. Reorda. Dft reuse for low-cost radiation testing of socs: A case study. In *VLSI Test Symposium, 2009. VTS '09. 27th IEEE*, pages 276–281, May 2009.

[86] J. Carreira, H. Madeira, and J. Silva. Xception: A technique for the experimental evaluation of dependability in modern computers. *IEEE Trans. Softw. Eng.*, 24 (2):125–136, February 1998.

[87] M. Portela-Garcia, C. Lopez-Ongil, M.G. Valderas, and L. Entrena. Fault injection in modern microprocessors using on-chip debugging infrastructures. *Dependable and Secure Computing, IEEE Transactions on*, 8(2):308–314, March 2011.

[88] D. de Andres, J.-C. Ruiz, D. Gil, and P. Gil. Fault emulation for dependability evaluation of vlsi systems. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 16(4):422–431, April 2008.

[89] Dongwoo Lee and Jongwhoa Na. A novel simulation fault injection method for dependability analysis. *Design Test of Computers, IEEE*, 26(6):50–61, Nov 2009.

[90] L. Entrena, M. Garcia-Valderas, R. Fernandez-Cardenal, A. Lindoso, M. Portela, and C. Lopez-Ongil. Soft error sensitivity evaluation of microprocessors by multilevel emulation-based fault injection. *Computers, IEEE Transactions on*, 61(3): 313–322, March 2012.

[91] Eric Bauer and Randee Adams. *Reliability and Availability of Cloud Computing.* Wiley-IEEE Press, 1st edition, 2012.

[92] S. Reiter, M. Pressler, A. Viehl, O. Bringmann, and W. Rosenstiel. Reliability assessment of safety-relevant automotive systems in a model-based design flow. In *Design Automation Conference (ASP-DAC), 2013 18th Asia and South Pacific*, pages 417–422, January 2013.

[93] Waseem Ahmed and Yong Wei Wu. A survey on reliability in distributed systems. *J. Comput. Syst. Sci.*, 79(8):1243–1255, December 2013.

[94] Eduardo Huedo, Rubaon S. Montero, and Ignacio M. Llorente. Evaluating the reliability of computational grids from the end users point of view. *Journal of Systems Architecture*, 52(12):727–736, December 2006.

[95] Vittorio Cortellessa and Vincenzo Grassi. Reliability modeling and analysis of service-oriented architectures. In Luciano Baresi and Elisabetta Di Nitto, editors, *Test and Analysis of Web Services*, pages 339–362. Springer, 2007.

[96] Zibin Zheng and Michael R. Lyu. Collaborative reliability prediction of service-oriented systems. In *ICSE '10 Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, pages 35–44, May 2010.

[97] Deng-Jyi Chen, M.-C. Sheng, and M.-S. Horng. Real-time distributed program reliability analysis. In *Parallel and Distributed Processing, 1993. Proceedings of the Fifth IEEE Symposium on*, pages 771–778, December 1993.

[98] J.-C. Laprie and K. Kanoun. X-ware reliability and availability modeling. *Software Engineering, IEEE Transactions on*, 18(2):130–147, February 1992.

[99] Y.S. Dai, M. Xie, K.L. Poh, and S.H. Ng. A model for correlated failures in n-version programming. *IIE Transactions*, 36(12):1183–1192, December 2004.

[100] Min Xie, Kim-Leng Poh, and Yuan-Shun Dai. *Computing System Reliability: Models and Analysis*. Springer, 2004.

[101] Martin L. Shooman. *Reliability of Computer Systems and Networks: Fault Tolerance, Analysis, and Design*. Wiley Publishing, 2002.

[102] Mohammad Modarres, Mark Kaminskiy, and Vasiliy Krivtsov. *Reliability Engineering And Risk Analysis: A Practical Guide*. CRC Press, 2rd edition, 2010.

[103] Mark L. Ayers. *Telecommunications System Reliability Engineering, Theory, and Practice*. Wiley-IEEE Press, 1st edition, 2012.

[104] Deng-Jyi Chen and Tien-Hsiang Huang. Reliability analysis of distributed systems based on a fast reliability algorithm. *Parallel and Distributed Systems, IEEE Transactions on*, 3(2):139–154, March 1992.

[105] Kent G. Stephens. *A Fault Tree Approach to Analysis of Behavioral Systems: An Overview*. Distributed by ERIC Clearinghouse [Washington, D.C.], 1974.

[106] V.K.Prasanna Kumar, Salim Hariri, and C.S. Raghavendra. Distributed program reliability analysis. *Software Engineering, IEEE Transactions on*, SE-12(1):42–50, Janury 1986.

[107] Yuan-Shun Dai, Min Xie, and Kim-Leng Poh. Reliability analysis of grid computing systems. In *Dependable Computing, 2002. Proceedings. 2002 Pacific Rim International Symposium on*, pages 97–104, December 2002.

[108] Yuan-Shun Dai, Min Xie, Kim-Leng Poh, and G. Q. Liu. A study of service reliability and availability for distributed systems. *Rel. Eng. & Sys. Safety*, 79(1): 103–112, January 2003.

[109] Yuan-Shun Dai, Min Xie, and Kim-Leng Poh. Reliability of grid service systems. *Computers & Industrial Engineering*, 50(1-2):130–147, May 2006.

[110] Yuan-Shun Dai, Yi Pan, and Xukai Zou. A hierarchical modeling and analysis for grid service reliability. *IEEE Trans. Computers*, 56(5):681–691, May 2007.

[111] Yuan-Shun Dai, Bo Yang, Jack Dongarra, and Gewei Zhang. Cloud service reliability: Modeling and analysis, 2010. URL http://www.techrepublic.com/resource-library/whitepapers/cloud-service-reliability-modeling-and-analysis.

[112] Mohamed-Larbi Rebaiaia and Daoud Ait-Kadi. Network reliability evaluation and optimization: Methods, algorithms and software tools, December 2013. URL https://www.cirrelt.ca/DocumentsTravail/CIRRELT-2013-79.pdf. CIRRELT-2013-79.

[113] ITU-T Rec Z.110-Z.119 - Application of formal description techniques, 2008.

[114] ITU-T Rec Z.100-Z.109 - Specification and Description Language (SDL), 2011.

[115] ITU-T Rec Z.120-Z.129 - Message Sequence Chart (MSC), 2011.

[116] ITU-T Rec Z.150-Z.159 - User Requirements Notation (URN), 2011.

[117] ITU-T Rec Z.160-Z.179 - Testing and Test Control Notation (TTCN), 2014.

[118] IEEE Std 1362-1998 (R2007) - IEEE Guide for Information Technology - System Definition - Concept of Operations (ConOps) Document, 2007.

[119] ISO/IEC/IEEE Std 15288:2015 - Systems and software engineering - System life cycle processes, 2015.

[120] ISO/IEC/IEEE Std 15289:2011 - Systems and software engineering - Content of life-cycle information products (documentation), 2011.

[121] ISO/IEC/IEEE Std 42010:2011 - Systems and software engineering - Architecture description, 2013.

[122] IEEE Std 1233-1998 - IEEE Guide for Developing System Requirements Specifications, 1998.

[123] ISO/IEC/IEEE Std 29148:2011(E) - Systems and software engineering - Life cycle processes - Requirements engineering, 2011.

[124] Andrew S. Tanenbaum and Todd Austin. *Structured Computer Organization.* Prentice Hall Press, 6th edition, 2012.

[125] Andrey A. Shchurov. A formal model of distributed systems for test generation missions. *International Journal of Computer Trends and Technology*, 15(3):128–133, September 2014.

[126] Andrey A. Shchurov. A multilayer model of computer networks. *International Journal of Computer Trends and Technology*, 26(1):12–16, August 2015.

[127] IEEE Std 802.1AX-2014 - IEEE Standard for Local and metropolitan area networks - Link Aggregation, 2014.

[128] Maarten van Steen. *Graph Theory and Complex Networks: An Introduction.* Maarten van Steen, 1st edition, 2010.

[129] OpenStack. URL http://www.openstack.org/.

[130] VMware vSphere. URL http://www.vmware.com/products/vsphere/.

[131] ISO/IEC Std 7498-1:1994 - Information technology - Open Systems Interconnection - Basic Reference Model: The Basic Model, 1994.

[132] Andrew S. Tanenbaum and David J. Wetherall. *Computer Networks.* Prentice Hall Press, 5th edition, 2011.

[133] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach*. Pearson, 6th edition, 2012.

[134] Douglas E. Comer. *Internetworking With TCP/IP Volume I: Principles, Protocol, And Architecture*. Pearson, 6th edition, 2015.

[135] ISO/IEC Std 27005:2011 - Information technology - Security techniques - Information security risk management, 2011.

[136] Vladimir A. Khlevnoy and Andrey A. Shchurov. A formal approach to distributed system security test generation. *International Journal of Computer Trends and Technology*, 16(3):121–127, October 2014.

[137] George A. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *The Psychological Review*, 63(2):81–97, March 1956.

[138] The Internet Assigned Numbers Authority (IANA). Service Name and Transport Protocol Port Number Registry. URL http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml.

[139] *WAN Design Summary*. Cisco Systems, August 2014.

[140] *Campus Design Summary*. Cisco Systems, August 2014.

[141] IETF RFC 791 - Internet Protocol, September 1981.

[142] IETF RFC 2460 - Internet Protocol, Version 6 (IPv6), October 1998.

[143] *Campus Wired LAN Technology Design Guide*. Cisco Systems, August 2014.

[144] *Campus Wireless LAN Technology Design Guide*. Cisco Systems, August 2014.

[145] M. Torngren, DeJiu Chen, and I. Crnkovic. Component-based vs. model-based development: a comparison in the context of vehicular embedded systems. In *Software Engineering and Advanced Applications, 2005. 31st EUROMICRO Conference on*, pages 432–440, September 2005.

[146] Nick Marshall. *Mastering VMware Vsphere 6*. Sybex, 1st edition, 2015.

[147] Dhiraj K. Pradhan, editor. *Fault-tolerant computer system design*. Prentice-Hall, 1996.

[148] Amir Ranjbar. *Troubleshooting and Maintaining Cisco IP Networks (TSHOOT) Foundation Learning Guide*. Cisco Press, 1st edition, 2014.

[149] Wikipedia. Symmetric difference, . URL https://en.wikipedia.org/wiki/Symmetric_difference.

[150] Sean Kandel, Andreas Paepcke, Joseph M. Hellerstein, and Jeffrey Heer. Enterprise data analysis and visualization: An interview study. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2917–2926, December 2012.

[151] ITU-T Rec L.72 - Databases for optical access network infrastructure, 2008.

[152] Christopher Alexander, Sara Ishikawa, Murray Silverstein, Max Jacobson, Ingrid Fiksdahl-King, and Shlomo Angel. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, 1977.

[153] Andrey A. Shchurov and Radek Marik. A presentation format of architecture description based on the concept of multilayer networks. In *ITU Kaleidoscope Academic Conference: Trust in the Information Society (K-2015)*, pages 227–232, December 2015.

[154] Andrey A. Shchurov and Radek Marik. A trusted model of complex computer networks. *Journal of ICT Standardization*, 3(3):201–230, March 2016.

[155] Karl Marx. Theses on feuerbach, 1845. URL https://www.marxists.org/archive/marx/works/1845/theses/theses.htm.

[156] SPC TRIGGER s.r.o. URL http://www.spc-trigger.com/.

[157] Apache Software Foundation. URL http://www.apache.org/.

[158] Canonical ltd. URL http://www.ubuntulinux.org/.

[159] Microsoft Corporation. URL http://www.microsoft.com.

[160] Mozilla Foundation. URL https://www.mozilla.org/.

[161] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, 3rd edition, 2009.

[162] The global state of information security, 2017. URL http://www.pwc.com/gsiss2017.