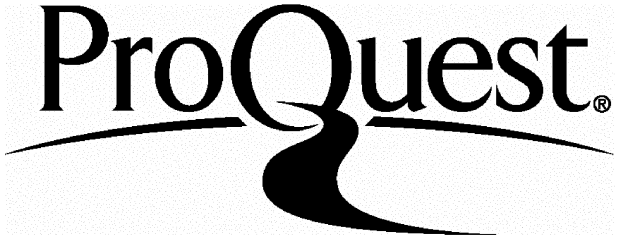
Development of Integrity Policies for Network Management

Thesis submitted for the degree of Doctor of Philosophy in Electronic and Electrical Engineering

Ognjen Prnjat

University College London



Abstract

Modern telecommunications systems are becoming complex due to technological advances, increased interconnection, and market demands. In this context, the crucial issue is the ability of systems to retain high integrity and low risk. Currently, there are no established methodological approaches to managing integrity, and there is a lack of integrity-preserving techniques.

Here, we pin-point the integrity attributes of telecommunications systems, and develop a complete methodological framework for integrity management throughout the system lifecycle. The methodology is based on our original ODP-UML model, and focuses on the analysis of integrity requirements, and specification of integrity-preserving techniques - policies. This methodology is appropriate for any telecommunications system, while its application was explored in the context of management systems developed by ACTS projects TRUMPET and FlowThru.

In this context, we focus on the development of two integrity policies.

The first policy is based on object-oriented software metrics, which yield the complexity/coupling measurements of system classes. These measurements are related to the integrity/risk status of the classes, thus being able to pin-point potential risk areas in the design. We demonstrate the applicability of this policy through three experiments involving TRUMPET and FlowThru. Experiments show that a specific suite of seven object-oriented metrics can be used as the integrity indicator early in the development lifecycle; and that the highest risk for management systems' operation is exhibited at the interconnection points between either administrative domains or stand-alone components. Moreover, we uncover a strong ordinal relationship between the individual metrics within the suite.

The second policy focuses on the testing of integrity aspects of management systems' interconnection across domains and is based on the concept of the Xuser Test-MIB. The applicability of this policy is demonstrated through a brief case study of the TRUMPET Xuser interface, the outcome indicating how complex inter-domain interactions might appear sensitive to the introduction of additional sophisticated security features.

Acknowledgements

The research work discussed in this thesis was supported by the British Council's Overseas Research Studentship and the EPSRC grant M26091. The additional source of funding was the ACTS project TRUMPET (DGXIIIB, AC112).

I would like to thank the following people.

My supervisor, Dr. Lionel E. Sacks, for constant professional and personal support.

Prof. Chris Todd of University College London, for introducing me to the field and for the much needed early support.

Prof. George Pavlou of University of Surrey for constructive comments given regarding my transfer thesis, as well as the final thesis.

Prof. Laurie Cuthbert of Queen Mary College for the suggestions given regarding the final thesis.

David Griffin of University College London for revisions of network and service management technical background.

Marcus Wittig and Oliver Schitko (formerly of Forschungszentrum Informationstechnik GmbH - GMD), for the help in the implementation of the support testing infrastructure.

A number of anonymous reviewers for comments regarding the research work published.

Céline for all the personal support.

My family.

To my parents, for all their love, understanding and support.

Table of Contents

Table of Figures

Table of Tables

1 INTRODUCTION

In the introductory chapter, we discuss the motivation for this work, outline the problem field and describe the approach taken in this thesis to address the issues identified. Section 1.1 presents the motivation: it describes the current telecommunications environment, in terms of its market dynamics, regulatory initiatives and the technological advances; factors which all contribute to the core problem that the thesis is focusing on: network and systems integrity. Section 1.2 elaborates on the integrity problem field; describes the approach to the research work undertaken; and gives the overview of the thesis.

1.1 MOTIVATION

The current telecommunications environment is characterised by a number of factors that are profoundly changing its historical role and structure.

Primary factor is the growing business and residential demand for **sophisticated services**. These services range from the classic narrow-band voice telephony, through broadband streams supported by the Synchronous Digital Hierarchy (SDH) and the Asynchronous Transfer Mode (ATM) technologies, to highly interactive dynamic services encompassing multi-party, multimedia and mobile features, increasingly supported by Internet Protocol (IP) technology. Demands for services are diversifying, and the requirements on performance and functionality intensify as the telecommunications services become mission critical for business and industrial processes.

In order to provide and operate these sophisticated services, the individual components in the core telecommunications networks have to interact more closely with each other, as well as with a number of support systems for advanced call control, Intelligent Network (IN), network and service management, *etc.*

In the Broadband Integrated Services Digital Network (B-ISDN) Reference model [I.321], as described in [Pavl98], the operations needed for service provision and maintenance are divided into three planes. The *User Plane* is effectively the core network, supporting the transfer of user information. The *Control Plane* is responsible for the establishment, operation and termination of the calls/connections. For the bearer services (such as basic telephony), the control plane provides this support through the Signalling System Number 7 (SS7) [Q.700]. The support for the enhanced services such as IN - based telephony and the Telecommunications Information Network Architecture (TINA) multimedia services is provided for through the intelligence located outside the core network. Finally, the B-ISDN

Reference model defines the *Management Plane,* responsible for the management of both user and control planes. The management plane provides support for planning, installation, operations and maintenance of the infrastructure of the other two planes (as quoted in [Pavl98]).

Complex interactions are thus occurring between the network elements and the computational resources residing in the control and management planes. With the advance of IT-inspired applications in the customer premises, the degree of interaction increases even more. The convergence of IT and telecommunications, introduction of the sophisticated management, control and support systems, and the advances in the Common Channel Signalling (CCS) [Mant91] / SS7, make the telecommunications network resemble a large, **complex distributed software system**.

This complexity is further increased under the pressure of regulatory forces [Walk97]. Initiatives such as the European Commission's Open Network Provision (ONP) are calling for the established Public Network Operators (PNOs) to open their networks to Other Licensed Operators (OLOs) and third-party, Value Added Service Providers (VASPs). These initiatives are targeted to stimulate fair competition and to increase market dynamics. However, they greatly add to the complexity of the telecommunications environment by forcing the autonomous players to interwork without having the full assurance that the invulnerability of their domain will not be compromised as a consequence of the **interconnection**.

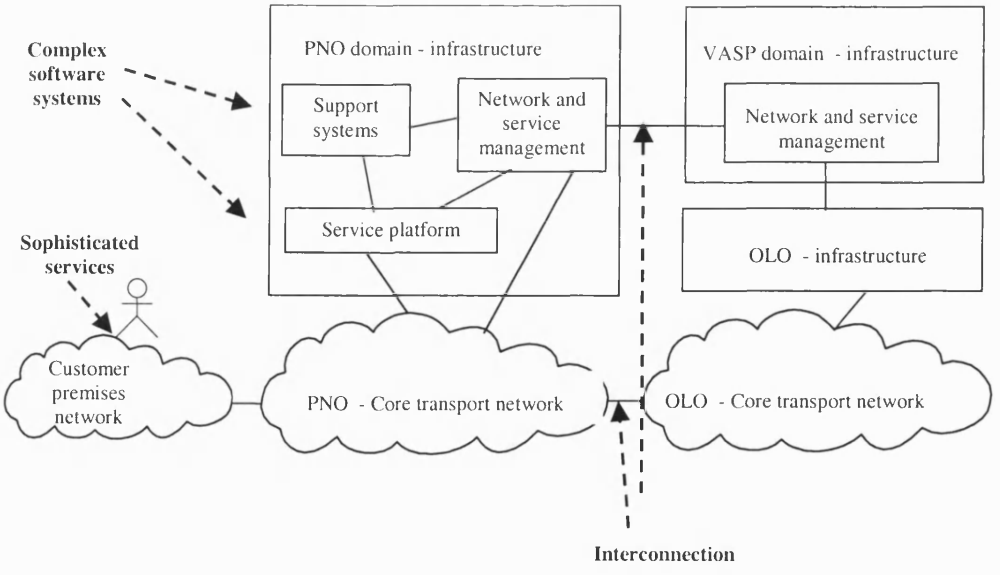


Figure 1 - The environment

The three key shaping factors in the modern telecommunications world, as highlighted in the above discussion, are depicted in the schematic diagram of Figure 1.

As a result of the above factors, an environment of highly complex telecommunications systems, where a number of software-based components from different domains interact closely together to provide and maintain the end-user services, is emerging. In such an environment, it is becoming increasingly difficult to specify, develop, test and interconnect these complex, heterogeneous distributed telecommunications systems. Moreover, it is becoming almost unattainable to guarantee the correct and proper functioning of these systems: *i.e.*, their **integral** operation.

Integrity was initially focused on the core transport network, and it was defined in the context of the public network operation as: "the ability of the network to retain its specified attributes in terms of performance and functionality" [UCL94] [Ward95]. However, in the emerging environment we described above, where a network is effectively a conglomeration of software systems, network integrity becomes inseparable from **telecommunication system integrity**. Any flaw in the integrity of a telecommunication system has a potential of impacting the integrity of the network, which the system is part of.

The classical and most serious example of things going wrong is reflected in the events of January 1990 brownout of the AT&T American network [McDo94][Hatt97]. A control mutation, originating in the switching system (and due to a single software error) propagated through the signalling (SS7) network causing degradation of the operation and ending in a total shutdown. The whole eastern seaboard of the US lost telephone connections for nine hours, the financial loss amounting to 1 billion dollars. A number of similar integrity breaches followed in Pacific Bell and Bell Atlantic networks in the two subsequent years [Hoan93]. Another recent example is the 1998 integrity breach in the AT&T Chicago frame relay network, due to a few faulty lines of code, which effected thousands of corporate customers across the US [Meht98]. The latest example is the brownout in the BT network in February 2000, which blocked millions of calls [BBC00].

These examples illustrate how vulnerable the telecommunications networks can be to initially small and isolated failures. They consequently highlight the need for serious consideration of the integrity issues by all players in the telecommunications market: the sheer scale of the financial loss incurred by an integrity breach is convincing enough. In the modern wired world, increasingly reliant on effective high-speed communications, any malfunction in telecommunications systems operation can have dire consequences. A serious failure can, in the present competitive environment, cause financial threats to both network operators and service providers - ranging from the loss of revenue to the decrease in their customer base. Similarly, customers who depend on the availability of telecommunications services can be at risk as well.

The issues of integrity will increasingly be of crucial importance in future telecommunications scenarios, where multiple operators, service providers, third party retailers and other players on the market will undoubtedly have to collaborate and interwork, on the core transport as well as control and management levels, while having to be confident that this interworking will not jeopardise the correct and proper functioning of their domain. The concept of integrity has to be well understood; and methods, tools and techniques for managing integrity throughout the system and service lifecycle have to be developed. Finally, factors influencing integrity need to be measured, because: "what you cannot measure, you cannot control" [DeMa82].

1.2 THE PROBLEM FIELD AND THE APPROACH

The **problem field** that this research work focuses on is that of ***telecommunications system integrity.*** This field is still nascent, with the majority of research and industrial initiatives focusing on the integrity of the traditional telephony network with limited capabilities and support systems. Understanding of the term integrity is similarly out-of-date. The industrial approach for managing integrity is focusing on extensive system pre-launch testing to ensure correct and proper functioning, while some attempts were made recently in academia to develop a broader framework for tackling the integrity issues. Moreover, to the author's knowledge, there is no evidence of the research into integrity of management systems as such.

The **problem statement** can be summarised as follows: "*The field of integrity assurance is heavily under-researched, the two main flaws being the lack of a structured framework for tackling the integrity issues throughout the telecommunications system lifecycle, and the lack of pre-emptive techniques for ensuring integrity*".

Our research recognises the need for an up-to-date analysis of the concept of integrity, influenced by the factors discussed in section 1.1. Moreover, it acknowledges the lack of and need for a coherent framework for understanding the integrity issues, and techniques for managing the integrity issues, throughout the telecommunications system lifecycle. Finally, it advocates the measurement of factors influencing integrity, for the purpose of deeper understanding, effective comparison, and better management.

Our **approach** represents a mixture of theoretical and practical work. The theoretical work focused on the study of the concept of integrity, its analysis in the context of the modern software-oriented distributed telecommunications systems, and its decomposition in a number of related issues. In parallel, a methodology for management of integrity issues throughout the telecommunications system development lifecycle was developed.

The practical research platform was provided by two European Commission (EC) - sponsored "Advanced Communications Technologies and Services" (ACTS) projects in the field of network and service management: TRUMPET [TRUMPET] and FlowThru [FlowThru]. Management systems developed by these projects provided an ideal research context, since they are strongly focusing on the three key factors influencing the modern telecommunications environment, identified in section 1.1: *sophisticated service provision and management*, accomplished through the *inter-domain collaboration* of *distributed software-based* management systems. These projects were used to some extent as case studies for the application of the integrity methodology, but primarily for the development and validation of integrity management techniques - integrity policies. Two distinct integrity policies were developed, and they are presented in this thesis. The flow of the research work is depicted in Figure 2.

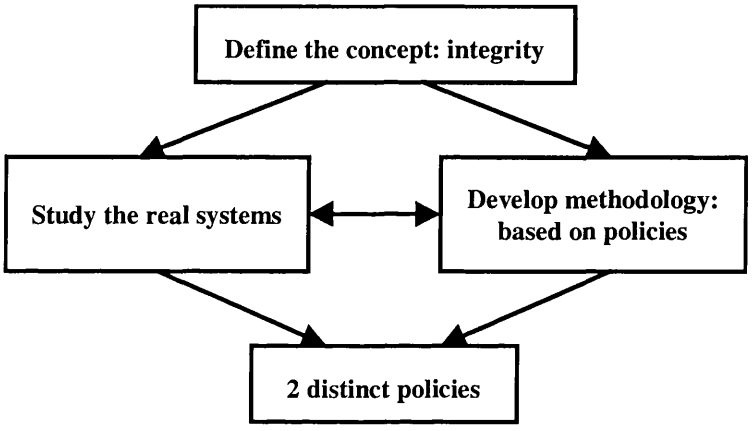


Figure 2 - Flow of research work

Although the initial theoretical research is applicable to a wide range of distributed telecommunications systems, the later stages of the research are strongly focusing on network and service management systems, which provided the main practical research platform. This is also reflected in the **structure of the thesis**.

Chapter 2 presents the background in the three key areas relevant for the research. First, the current understanding of the term integrity is discussed, the existing integrity measures are presented, and the computing concepts related to integrity are introduced. Next, the state of the art in integrity assurance is discussed - this includes both the industrial and research initiatives. Finally, the need for the integrity of the management plane is stressed, and the current trends in network and service management in terms of architectures, technologies, development methodologies and notations are discussed. The aim of this chapter is not just to present the state of the art, but also to pin-point the lack of expertise in the relevant areas and thus stress the motivation and give the justification for the novel research work conducted for this thesis.

In Chapter 3, the core theoretical research work is presented. First, our interpretation of integrity is given, represented as a set of lower-level attributes. Next, we present our methodology for management of integrity throughout the telecommunications system development lifecycle. This methodology is geared towards the network and service management systems; however, it is applicable to any distributed telecommunications software system of a given level of complexity. This chapter also introduces the concept of the integrity assurance technique - *i.e.*, integrity policy, and discusses some candidate integrity policies for network and service management systems.

In Chapter 4, the ACTS projects TRUMPET and FlowThru are presented, with the emphasis on TRUMPET, which represented the main research platform.

Chapter 5 presents the **first integrity policy** developed during this research: that of the **complexity and risk reduction** early in the telecommunications system development lifecycle, based on software metrics. It also presents three case studies of this policy: one of the TRUMPET management system and two of the FlowThru management system.

Chapter 6 presents the **second integrity policy** developed during the research: the **inter-domain interconnection testing policy**. A brief example of the application of this policy in the context of the TRUMPET management system is also given in this chapter.

In Chapter 7, we give the discussion and conclusions concerning the research work undertaken, and briefly reflect on the potential future work.

Chapter 8 contains author's publications, Chapter 9 the references and Chapter 10 the acronyms.

At the end of each chapter discussing novel research, a summary of our contributions to the research area is given.

Note that throughout the thesis, "we" is used instead of "I" - I just prefer the style.

2 STATE OF THE ART

The aim of this chapter is to give the background for the research work presented in the rest of this thesis. Moreover, it aims to highlight the lacks of expertise in the research areas dealt with, and thus to provide both the motivation and justification for the novel research conducted.

First, in section 2.1 we discuss the understanding of the concept of integrity, and highlight the need for an up-to-date definition of the term. Next, in section 2.2, we critically reflect on the existing integrity research and current industrial techniques for ensuring and maintaining integrity. Following this, in section 2.3 we elaborate on the relationship between the network and service management systems and the underlying control and core network planes, in the integrity context, and accentuate the need for integrity of management systems. Moreover, in this section we discuss the current state of the art in the area of network and service management.

2.1 INTEGRITY: THE CONCEPT

This section deals with the interpretation of the concept of integrity. Section 2.1.1 briefly critically reflects on the two main prose definitions of integrity. In section 2.1.2, the existing integrity measures are described, and the related concept of risk is reflected on. The related computing concepts of dependability and safety are presented in section 2.1.3, followed by the discussion in section 2.1.4.

2.1.1 INTEGRITY DEFINITIONS

The understanding of the term integrity is typically very nebulous. There are no unified concepts, agreed definitions or recommendations. Integrity is traditionally related to data: in the realm of data systems, data integrity refers to the incorruptibility of stored data; in the realm of security, data integrity refers to the ability to avoid modification, insertion or duplication of data both stored and in transit.

Two principal definitions of network integrity are coming from the US and the UK; from the industry and academia, respectively. As reported in [McDo94], the definition of integrity as understood in Bellcore is "the ability of a network provider to deliver high-quality, continuous service while gracefully absorbing, with little or no customer impact, failures of or intrusions into the hardware or software of network elements". The alternative definition, given in [UCL94] [Ward95], is "the ability of the network to retain its specified attributes in terms of performance and functionality".

The problem of defining integrity is already visible in the first definition. A number of issues are identified: high quality of service delivery, continuity of the service, robustness to failure, ability to prevent malicious human intervention (intrusion). Both customer perception and the responsibility of the network provider are also mentioned. Second definition possibly narrows down the problem: integrity is a set of attributes, either functional or performance-related, that need to be retained at their satisfactory levels.

Both definitions are vague. Both imply the existence of a certain subset of concepts. Both focus on the integrity of the core transport network. And both assume the two main pre-regulatory players on the telecommunications market: network operators, and their customers.

These definitions do not state clearly what the exact integrity attributes are, what are the phenomena effecting integrity or what are the desirable features of a high-integrity network (apart from "the ability to gracefully absorb failures"). These definitions acknowledge neither the new view of the telecommunications network: the "network of systems", nor the implications of the deregulated market. However, by pointing out that integrity is a higher level measure, and by mentioning the attributes, they do automatically imply the need for a framework: an issue that will be further discussed in section 2.2.

2.1.2 INTEGRITY MEASURES

This section presents the three existing integrity outage measures (section 2.1.2.1), discusses a pre-emptive integrity measure (section 2.1.2.2), and presents the related concept of risk (section 2.1.2.3).

2.1.2.1 OUTAGE MEASURES

The research into the integrity concept to date, prompted by the 1990-92 US network integrity breaches (discussed in chapter 1, section 1.1), resulted in a number of measures that are targeted to quantify the outages. Service outage is defined by the Network Reliability Council (established by the US Federal Communications Commission - FCC, in order to obtain technical advice on a variety of telecommunications issues) as "the state of a service when network failures impair the initiation of new requests for service and/or continuous use of the service, and the service outage parameters exceed their corresponding thresholds" [TA1A-93]. This definition is effectively a *post-mortem* integrity definition. The following measures (already discussed in [UCL94] and [Mont98]) were proposed to quantify the outages.

2.1.2.1.1 Cochrane Richter scale

This definition quantifies the network integrity breaches in customer-effected terms: loss of traffic. This definition categorises the outages like earthquakes, on the logarithmic, Richter scale. Total network information capacity outage, D, is thus:

$$D = \log_{10} (N \times T) \quad (1)$$

Where N is the total number of customer circuits effected, and T the total down time, in hours.

2.1.2.1.2 User Lost Erlang

A similar measure, proposed in [McDo92] is the User Lost Erlang (ULE). This measure is also based on the logarithmic scale:

$$ULE = \log_{10} (E \times H) \quad (2)$$

Where E is estimated average user traffic lost during the time of the integrity breach in Erlangs, and H is the outage duration, in hours.

2.1.2.1.3 ATIS outage index

The US Alliance for Telecommunications Industry Solutions (ATIS) T1 Standards Committee (T1A1) proposed a general framework for quantifying outages, from the user perspective. This framework is composed of three main parameters:

- Unservability (U), defined as a fraction of service that cannot be provided, as a result of a failure.
- Duration (D)[1], the length of time of a failure, during which the unservability was above a certain limit.
- Extent (E), the measure of the geographic area and population effected by the outage, when the unservability was above a certain limit.

The severity of an outage can be represented as a (U, D, E) triplet: the outages can be categorised as falling into minor, major or catastrophic regions.

Further work by the T1A1 targeted the need to enable a meaningful summation of the individual outage index values over time periods, for comparison and monitoring purposes. The second requirement was the ability to take into account the relative importance of a

[1] Note that Duration D is different from total network information capacity outage, D, of equation (1).

particular service effected by the outage. A more sophisticated outage index was thus proposed, with the basic property that the index of an outage is the sum of the indices of multiple services. Weighting functions in the shape of s-curves were introduced, and for any single service the outage index is the product of the three weighting functions. The total outage index is:

$$I(O) = \Sigma^{n}_{j=1} (W_{s(j)} W_{d(j)} W_{m(j)}) \quad (3)$$

Total outage is thus the aggregation of services j = 1,.....N; W_s is the service weight, W_d is the duration weight, and W_m is the magnitude weight. Both the duration weight and the magnitude weight are calculated with the help of the s-curves, while the service weight is allocated depending on the type of the service.

2.1.2.2 PRE-EMPTIVE MEASURES

All of the measures presented up to now are *post-mortem* measures, targeting to assess the impact of an integrity breach that already occurred. The Cochrane Richter Scale and the ULE are both understandable, but relatively simple measures: they take into account only the outage duration, and the number of customers effected or traffic lost, respectively. Cochrane scale does not take into account the customer usage of the service: business or residential, daytime or night-time. ATIS outage index is much more sophisticated, targeting to overcome the above deficiencies. In order to perform a calculation of the ATIS weighing functions for the three basic parameters, a number of other factors have to be taken into account: time of outage, lines effected/blocked calls, duration, types of services effected, *etc.* Here, an important benefit of the ATIS index can be seen: that of the decomposition of issues, and of a structured approach to address them. Still, however, all of the above measures are *post-mortem*, with no possibility of determining the current level of integrity, or quantifying a probability, threat or risk of an outage taking place.

The first attempt to formulate a "*pre-mortem*" integrity measure is given in [Mont97][Mont98]. Integrity is perceived here as a "high-level measure, influenced by a number of factors". Network integrity is then represented as a set of integrity bands, or integrity states, that the network can assume.

Figure 3 shows the variation of integrity from 0% (any malfunction resulting in complete failure) to 100% (network absolutely robust to failure), through a number integrity bands. This work envisages that each operator will define their own criteria for integrity bands. This would involve the identification of parameters/attributes that influence integrity, and the

thresholds of these parameters would imply the transition points between the bands. However, no concrete parameters/attributes or thresholds are identified.

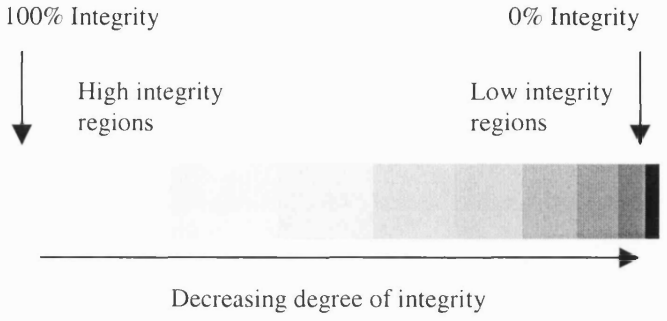


Figure 3 - Integrity bands [Mont97]

This integrity definition, as well as the two established ones presented in section 2.1.1, stresses that integrity is a higher-level measure, encompassing a number of attributes and influenced by a number of factors.

2.1.2.3 RISK

A concept closely related to the previously described outage measures is that of risk. An established definition of risk, in the computing arena (see further section 2.1.3), is: "a measure that combines both the likelihood that the system hazard will cause an accident and the severity of that accident" [IEEEStd1228-94] (while a hazard is "a system/software condition that is a prerequisite to an accident"). Similarly, risk = (probability of unsatisfactory outcome) x (loss if the outcome is unsatisfactory) [Boeh91].

2.1.3 COMPUTING LEGACY: DEPENDABILITY AND SAFETY

Two terms used in the field of computing bear a strong resemblance with the telecommunications-related term integrity: dependability and safety. Due to the convergence between telecommunications and computing, these two terms must be taken into account when considering integrity.

Dependability is a broad field of computing research, encompassing reliability assessment, security measures, *etc.* The first similarity between dependability and integrity is the lack of non-unified definitions, concepts and standards. A number of definitions of dependability exist [Pras95]:

- Trustworthiness of a computer system such that reliance can be justifiably placed on the service it delivers [Lapr92].
- The loss arising from using a system in a particular context: dependable systems are targeting to minimise risk [McDe94].

- The collective term used to describe the availability performance and its influencing factors [Musa87].
- Ability of an entity to perform one or several required functions under given conditions [Vill92].

The above indicates that no unifying definition of dependability is established in the literature. However, all the key authors in the field hint that dependability is a collective term for a number of desirable system attributes: this is another overlap with the understanding of the term integrity.

Safety, defined as "freedom from hazards" [IEEEStd1228-94], or "degree of freedom from risk in any environment" [Leve95], is another collective term for desirable system attributes. Safety research is mainly related to deduction of the possible causes of system hazards. The areas of interest are fault identification, backtracking and prevention.

The important thing to note is that, when discussing dependability and safety, it is often identified that *the concept is inseparable from the management process*: *i.e.*, dependability/safety is not just a term or an attribute: it is a notion that encompasses decomposition in lower-level attributes, and frameworks and methodologies to understand and manage them.

2.1.4 SUMMARY

As we have seen, the traditional definitions of the term integrity suffer from a number of drawbacks. The definitions are vague, in the sense that they do not sign-post to measurable quantities; and are generally concerned with the integrity of the core transport network, not taking into account the impact of the proliferation of the complex computational software systems supporting the sophisticated services. The existing integrity measures focus on the assessment of the integrity breaches, and as such are strictly *post-mortem*.

On the other hand, the research in the concept of integrity does imply that it is a complex measure, influencing and influenced by a number of factors. However, the existing research **does not clearly identify these factors.** We believe that there is a strong need to pin-point the integrity attributes. This needs to be done by considering both the integrity aspects of the core network, but also taking into account the software and system science, which are one of the leading shaping factors in the new telecommunications world. Our contribution to this aspect of the research is given in chapter 3, section 3.1.

Existence of a number of integrity factors also implies a need for a structured framework in which to understand and manage these factors. Our contribution to this aspect of the research is given in chapter 3, section 3.2.

The following section discusses the state of the art in the approaches for tackling the integrity of networks and systems.

2.2 INTEGRITY ASSURANCE AND TECHNIQUES

This section discusses the existing research and industrial initiatives in the field of integrity, and summarises the current techniques for assuring and maintaining integrity in telecommunications networks and systems.

2.2.1 STATE OF THE ART

The first wave of integrity research, taking place in the US, was triggered off by the 1990-91 SS7 brownouts (see chapter 1, section 1.1). This research focused on the definition of the outage measures, resulting with the ATIS outage index (section 2.1.2.1.3), and the integrity analysis limited to the SS7 features: a number of SS7 integrity threats were identified. However, after this first wave of research, there were no advances.

In the meantime, the established operators started to take practical measures for intra-domain integrity preservation, as well as for the protection of their networks from the outside threat. The first issue is addressed through the extensive pre-launch system/service testing, which itself is preceded by a detailed technical design overview. In the industry the usual practice is that the tests are performed manually: the selection of test suites is done by the test engineer who is well acquainted with the system under test. The most thorough testing approach is taken by Bellcore, which operates the Network Services Test System (NSTS). This test-bed can be used for auditing and testing the stand-alone systems, as well as for testing multi-supplier equipment interoperability, and inter-network interoperability. Bellcore, together with a number of other players in the US industry, took another step towards the inter-network testing: the Inter-network Interoperability Test Plan (IITP) [Lewi94]. The IITP is focusing on the interoperability testing for interconnected Common-Channel Signalling (CCS) networks, and is especially geared towards the integrity problems. In the UK, BT conducts similarly rigorous interconnect testing [Maso97]. The first phase encompasses system (software or hardware) conformance testing, to SS7 standards. Second phase is the interworking/interoperability testing in the test environment: the BT integration facility (test network) features switches which are not connected to the live network. Finally, the commission testing focuses on testing of new routes/circuits as they are introduced, and

encompasses functional testing of software/hardware. The test-beds such as the BT test network and the Bellcore NSTS are of crucial importance for the integrity assurance. Apart from catering for a wide range of test scenarios, these test-beds also remove the risks of testing the new systems/services against a "live" network.

In the US, the approach for the Plain Old Telephone Service (POTS) inter-network integrity protection during runtime is the use of screening at the interconnect: the incoming messages are checked for validity. Likewise, mediation devices are envisaged to be used in the future complex IN-based services, where the interconnection is taking place at the Service Control Point (SCP) level. Mediation devices adapt the messages at the interconnect so as to achieve compatibility. Similar measures, but to a far lesser extent, are taken by the operators in Europe: in the UK, some screening at the signalling level is taking place at the interconnect [UCL94].

However, the current approaches have a number of drawbacks. Pre-launch testing tends to be too exhaustive, and is thus often assessed as being expensive. In this context, it is also seen as a factor slowing down the system/service rollout. Moreover, it is impossible to test the combinations of all the interactions that might impact the integrity of the system/service. Screening and mediation solve some of the run-time problems that might arise at the interconnect, but are relatively tedious.

The research initiatives in Europe were encouraged by the European Commission, resulting in a comprehensive study of network integrity in the Open Network Provision (ONP) environment: given in [UCL94]. This study contains a set of recommendations to the EC. A number of open issues were identified, such as the need for:

- *System/service complexity measures.*
- *Integrity assessment and risk-oriented predictive models.*
- Studies of the *impact of collaborative network management on network integrity.*

However, there are no recent research initiatives and calls for improvement recorded.

Overall, the main shortfall of the research work and the industrial initiatives to date is the heavy focus on testing, and the lack of a structured approach for tackling integrity issues, especially in the context of the system/service pre-launch integrity management. One piece of work to address this drawback was conducted in academia: [Mont97]. This approach recognised the need for a coherent framework, and divided the integrity management actions into two basic groups: static and dynamic actions. Static actions are done prior to service launch. A number of pre-launch issues that need to be considered are discussed. These include *improvement of development methodologies* for better design coherence and

understanding; *use of formal methods* for rigorous system development; *modelling* as a tool for understanding, detecting and resolving possible integrity problems; *risk analysis*, which analyses, categorises and documents risks to integrity; as well as *testing*; *use of previous and expert experience* and the *use of knowledge-based systems*. Dynamic actions are taken after the system launch, and include *monitoring* of system integrity parameters, *risk analysis*, *restoration actions* and *documentation of information*. However, this approach does not identify the flow of these actions, and a clear way of demonstrating how they integrate in the system/service development lifecycle.

2.2.2 SUMMARY

As we have seen, both the industrial and research initiatives tackling the problems of integrity management suffer from a number of drawbacks. The main shortfall in the industrial context is the strong focus on testing: there is a lack of pre-testing techniques for integrity assurance. In the context of academic research, some advances were made in the direction of the provision of a structured framework for tackling the integrity issues throughout the system/service development lifecycle. However, no clear flow of integrity-preserving actions is defined, and the relationship with the development lifecycle is not strong enough.

Thus, there are two main key issues in the current approach to integrity management: **lack of a structured framework**, and the **lack of pre-emptive techniques for producing highly integral systems**. This is particularly true in the field of network and service management, where, to author's knowledge, there are no reports on the consideration of integrity issues.

Our contribution to the framework-oriented aspects of the research are given in chapter 3, while in chapters 5 and 6 we present two pre-emptive integrity assurance techniques.

In the next section, we highlight the need for the consideration of integrity issues in network and service management systems, and discuss the state of the art in the field of network and service management.

2.3 NETWORK AND SERVICE MANAGEMENT

As we have seen (sections 2.1, 2.2), the majority of the integrity research to date focused on the integrity of the core transport network. Some issues concerning the integrity of the control plane were also tackled (see section 2.2.1): namely, the integrity features of the SS7. The test-beds for pre-launch integrity assurance exist in both UK and the US, focusing on the core and control networks. In contrast, integrity of management systems is an under-

researched area. Management was mentioned only as a solution to the integrity problem [UCL94], not as a source of integrity problems in its own right. However, we argue that the integrity of the management systems is tightly coupled to the integrity of the underlying control plane and the core transport network. With the growing sophistication of management services, increasing level of interconnection between management systems in separate administrative domains, and diversity of management architectures and technologies, integrity threats on the management level are becoming a reality.

The relationship between management systems and the underlying managed resources, in the integrity context, is discussed in section 2.3.1. Next, the current state of the art in network and service management, characterised by the diversity of architectures, frameworks, technologies, development methodologies and modelling notations, is summarised in sections 2.3.2 - 2.3.4. This state of the art is given to some extent in the context of the European Commission - sponsored "Advanced Communications Technologies and Services" (ACTS) and "Research in Advanced Communications in Europe" (RACE) projects, which are seen as the flagship research initiatives in the field. Moreover, this state of the art does not aim to be exhaustive, but rather to cover the background necessary for the research work described in the rest of the thesis. Section 2.3.5 then discusses the future directions in network and service management.

2.3.1 RELATIONSHIP BETWEEN MANAGEMENT AND INTEGRITY

Management refers, as stated in the Open Systems Interconnection Systems Management (OSI-SM) framework document [X.700], to the activities which control, co-ordinate and monitor the use of resources. Network and service management is thus concerned with planning, installation, operations, administration, provisioning, and maintenance of networks and services (as quoted in [Pavl98]). As discussed in chapter 1, section 1.1, in the context of the B-ISDN Reference model [I.321], the management plane manages both the core network resources, and the support control systems outside the network.

OSI Systems Management (OSI-SM) [X.700][X.701] defines five distinct functional areas of management: fault, configuration, accounting, performance and security - referred to as FCAPS. Fault management deals with detection of faults in network operation, and their correction. The basic activities are alarm correlation, fault identification and testing. Accounting management deals with the identification of the costs of the use of network resources and services, and with charging for their use. Configuration management is responsible for planning, initialisation, continuous provision, and termination of communication services. Performance management is responsible for gathering and storing statistical data concerning network/service performance for the purpose of evaluation of the

network/service effectiveness. Security management deals with both securing the management applications and their communication, and provision of security for the managed resources.

Network and service management can be essentially envisaged as a control (in the traditional sense of the word, not in the B-ISDN Reference model terminology) problem, as depicted in Figure 4 (expanded from [Mamd96]). The management system is effectively a set of software components that manage the network, which can be seen as a set of switches, routers, connections, signalling, and control equipment. Note that in Figure 4 the control and the user planes, as defined in the B-ISDN model, are merged into one entity - labelled as "the managed system (network)".

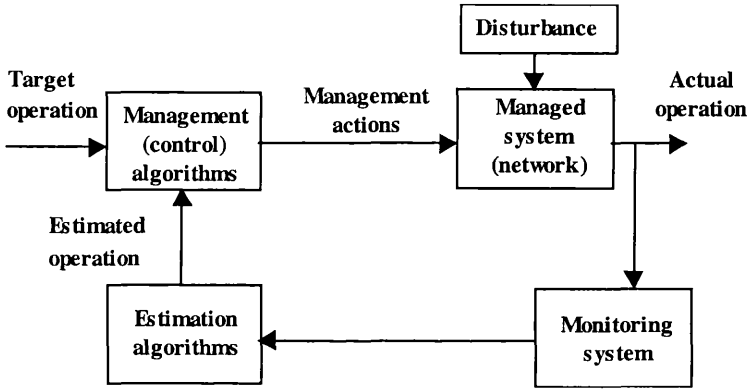


Figure 4 - Network management as a control problem

The management system performs a set of management actions, defined via management (control) algorithms, which are applied to the network so as to achieve and preserve the desired operation. The actual operation of the network is monitored, and the monitored operation is used to estimate the current operation of the network. The estimated operation is compared to the target operation and then required management actions are applied.

The control view of network management that we present here is applicable to all the FCAPS areas, with the exception of accounting management. In the other four areas, the interaction between the management system and the network being managed can be represented as in Figure 4. For example, considering configuration and fault management, if the link fails, the fault management system will diagnose the fault, choose the appropriate remedy action, and apply it, for example by notifying the configuration management to change the topology as required. Accounting management cannot be viewed as a direct control loop; however, the higher-level control-like relationships will appear, due to the impact of the operator's long-term business policies.

This "control" view (of Figure 4) implies that the problem space of the relationship between *integrity* and *network and service management* is two-fold. First, there is the issue of **management of network integrity** (where network integrity is as defined in section 2.1.1, related to the integrity of the core network and the control for bearer services). This issue relates to the provision of management functions that deal with management of network integrity as such. The second issue is that of the **integrity of management systems**. This concerns actions focusing on the development and maintenance of highly integral management systems.

The relationship between network integrity in the traditional sense and management systems integrity can be seen from Figure 4. Any disturbance in network integrity, *i.e.* the actual state of switching and signalling in the network layer, can influence the operation of the management system. This can happen if the disturbances in network operation cause the monitoring system to provide the management system with incoherent information and thus jeopardise its integrity status. Similarly, any disruption in the operation of the management system, *i.e.* any breach of management system integrity, can propagate, through management actions, to the actual network. This problem is magnified when multiple management systems are expected to share management information and interwork. As mentioned before (chapter 1, section 1.1), the Open Network Provision (ONP) stimulates the interconnection between management systems within different domains. However, this interconnection can pose an extra threat to the stability, *i.e.* integrity of the whole system if it is not carried out in a fully integral way or if separate systems themselves have a low degree of integrity. Thus, the high level of integrity of the management systems can be seen as being of paramount importance in the correct operation of the actual network.

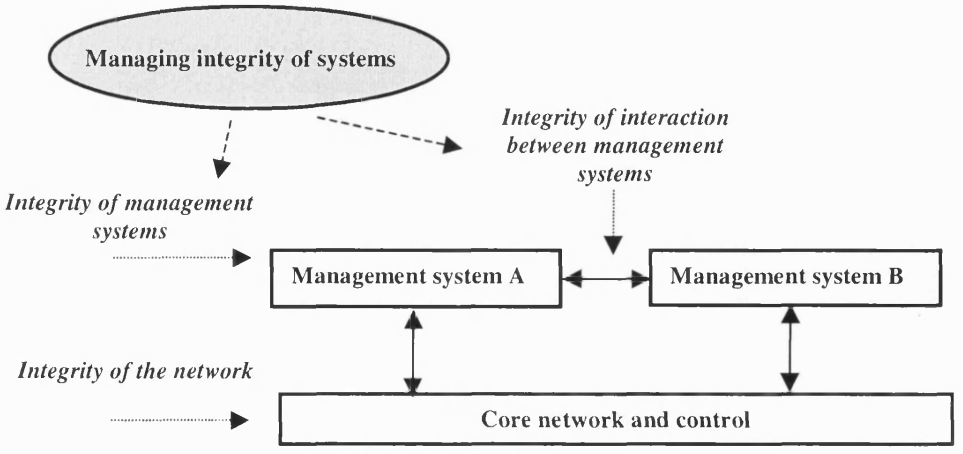


Figure 5 - Management and integrity

The issues of network integrity, management system integrity and the integrity of the management systems interconnection are depicted in Figure 5. In this thesis, we focus on the

issues of **management of integrity of systems, more specifically network and service management systems.**

Consideration of integrity issues in network and service management is also becoming critical as the highly heterogeneous management plane is taking shape. The following sections thus discuss the current trends in network and service management, in terms of architectures, technologies, development methodologies and modelling notations.

2.3.2 ARCHITECTURES

Architecture can be defined as: "a term applied to both the process and the outcome of thinking out and specifying the overall structure, logical components, and the logical interrelationships of a system" [What-is].

The most well established architectural standard in the problem field of telecommunications management is the Telecommunications Management Network (TMN) architecture [M.3010], promoted by the International Telecommunications Union (ITU). TMN management principles are based on OSI Systems Management (OSI-SM) [X.700][X.701], while recently CORBA [CORBA] is increasingly promoted as the base technology.

TMN is a logically separate network, which is effectively overlaid over the core network being managed, interfacing it at a number of points. TMN covers five distinct functional areas: fault, configuration, accounting, performance, and security management, as initially defined in [X.700] (for more detail refer to section 2.3.1). Although TMN primarily aims at management of networks and services, it is also to some extent used in service provision, namely for Virtual Private Network (VPN) - an example of service provision which was exploited by a number of RACE and ACTS projects.

TMN is a hierarchical, distributed management structure, where the management applications are organised in four distinct layers: element, network, service and business management layers. The element management layer is concerned with managing the vendor-specific functionality of individual network elements. The network management layer manages the interaction/communication between multiple network elements; providing the network view. The service management layer manages the aspects directly relevant to the users, such as Quality of Service (QoS), user subscription to services, *etc.* The business management layer manages the telecommunications enterprise: it is concerned with the strategic management, rather then with the technical/operational management as the other layers are. The management applications residing in any of these management layers can assume, using the OSI-SM terminology, both manager and agent roles.

As such, TMN aims to provide an architecture which enables the co-operation between management applications, as well as interactions with the resources they manage. This is done by specifying three dimensions of the architecture [M.3010]: functional, information and physical. Functional architecture decomposes the management functionality into functional blocks, which communicate through reference points. Physical architecture gives a lower level of abstraction by specifying how functional blocks can be mapped to the physical ones, implemented in pieces of equipment. Functional blocks thus become building blocks (physical equipment), and reference points become interfaces: for example, the x reference point between the two autonomous TMN Operations System Functions (OSFs) maps to the TMN X interface between two autonomous Operations Systems (OSs) in two separate administrative domains. Information architecture models the managed resources as managed objects, and specifies the ways of managing them, based on OSI-SM. A managed object is characterised by its attributes (properties - effectively, state), operations that can be performed on it, behaviour exhibited in response to the operations, and notifications that the object emits.

As reported in [Pavl98], the first hierarchical TMN system with fully compliant interfaces was constructed in the RACE NEMESYS (Network Management using Expert Systems) [Pavl91] project, while some of the most complex systems were developed in the RACE ICM (Integrated Communications Management) [Grif96] and PREPARE (Pre-Pilot in Advanced Resource Management) [Hall96] projects. Following the RACE projects, the ACTS projects MISA (Management of Integrated SDH and ATM Networks) [Gali00] and TRUMPET (Inter-domain Management with Integrity) [Sack98], amongst others, continued the research into TMN systems.

For more detail about TMN see [Pavl98] [Slom94a].

Another architecture for telecommunications management is the Telecommunications Information Network Architecture (TINA), developed by the TINA-Consortium. TINA is based on the existing concepts of TMN and IN. TINA aims to integrate management and control into a unified logical open distributed software architecture. TINA-C envisaged that each TINA function should be represented as a generic software object with its own integrated management and control mechanisms included. TINA architecture focuses on four distinct areas: service, network, management and computing. The TINA modelling approach is a layered one. The TINA business model [TINA-BM] presents a general business model and reference points between the autonomous players. The TINA service architecture [TINA-SA] aims to support for a wide range of services (communication, broadband, multi-party, multi-media, mobile) through a set of generic re-usable components. By defining the

session model, TINA service architecture fully separates the service support functionality and the physical connectivity. The TINA computing architecture is focused on structuring the software and providing for the distribution. The services/applications are represented as a set of distributed objects, or Computational Objects (CO), with well-defined interfaces, communicating over a Distributed Processing Environment (DPE). The network architecture aims to provide the connectivity service needed to support the TINA applications, by defining the network resource information model [TINA-NRIM]. The management architecture deals with service, connection, FCAPS and DPE management.

TINA is strongly object-oriented (OO) and largely based on the general Open Distributed Processing (ODP) [ODP] principles. ODP provides a broad architectural framework that distributed systems aiming to operate in the multi-provider environment must conform to throughout their development. ODP is not focusing solely on the telecommunications systems, but on the general distributed systems. For more details on ODP, see sections 3.2.1.1.1 (chapter 3) and 4.1.2.2 (chapter 4).

ACTS projects VITAL (Validation of Integrated Telecommunications Architecture for the Long-term) [Pavo97] and REFORM (Resource and Fault Restoration and Management) [Geor99], amongst others, conducted telecommunications network management research work using the TINA architectural principles.

An alternative approach to management is the policy-driven management [Slom94b], where the manager performs activities, specified through management policies, on target objects. The policies are interpreted by automated agents. The policies can be either authorisation policies, specifying what activities a manager is permitted/forbidden to perform, and obligation policies, specifying what a manager must/must not do on target objects.

2.3.3 TECHNOLOGIES

Technology can be understood as a mechanism enabling the functioning of a particular system. Network and service management systems are inherently distributed, and the distribution is supported by computing platforms based on a range of technologies. The TeleManagement Forum's (TMF; formerly the Network Management Forum, NMF) Technology Map [NMF-TM] captures a number of technologies: CMIS/P [X.710] [X.711], SNMP [RFC1157], JAVA [JAVA], CORBA [CORBA], Web-based technologies, *etc.* These technologies provide access to management information and enable communication between management applications.

Common Management Information Service / Protocol (CMIS/P) and Simple Network Management Protocol (SNMP) are the established protocol-based technologies for management. Both are based on the manager-agent model. The OSI-SM and the supporting CMIP/S are used as the base technology for TMN. SNMP is a commonly used technology for Internet and private network management.

In the recent years, the general distributed object technologies, such as CORBA, JAVA, *etc.* are emerging as an alternative. These are based on the client-server relationships between distributed objects, where the communications is taking place through a well-defined Applications Programming Interface (API). CORBA defines a system that caters for the interoperability between objects in a heterogeneous distributed environment, in a manner transparent to the applications user/programmer. CORBA objects are specified in an abstract language that can be mapped to a number of object-oriented programming languages such as JAVA, C++, *etc.* CORBA is increasingly used for the TMN service level applications, while the TINA computing architecture is based on CORBA. JAVA has a Remote Method Invocation (RMI) facility that supports distribution: however, only JAVA objects are supported.

Object Management Architecture (OMA) [OMA] is a general software architecture proposed by the Object Management Group (OMG). OMA is a high-level view of a distributed software environment. OMA is based on four sets of components: Object Request Brokers (ORBs), Object Services, Application Objects and Common Facilities. ORB is the heart of OMA, and in the OMG interpretation the ORB is effectively Common Object Request Broker Architecture (CORBA). ORBs enable the communication between distributed objects, independent on the implementation techniques and the platforms on which these objects reside. The Object Services support the object lifecycle management, object relocation, access control, *etc.* Common Facilities provide for generic application functionality, such as database access, printing, and email. Application Objects perform the user-specific functionality. OMA is a general distributed software architecture, rather then a specific management architecture, and as such will not be further discussed in this thesis.

The WWW technologies can also be used for management. In this scenario, the browser-based Graphical User Interface (GUI) is offered to the human manager, which can send the CMIP/SNMP messages in a string format over Hypertext Transfer Protocol (HTTP) [RFC2068] to the agent, which is capable of translating these messages, and is located next to the resource.

The diversity of technologies and the need for interworking drive the development of the technology gateway solutions, also discussed in [NMF-TM], such as CORBA-CMIS and CORBA-SNMP gateways. In the majority of the network and service management ACTS projects, more than one management technology was used. The examples of technology gateways are the ICM CMIP/SNMP [McCa95] and VITAL/REFORM CORBA/CMIS [Pavl97] gateways.

2.3.4 DEVELOPMENT METHODOLOGIES AND MODELLING NOTATIONS

Another dimension of diversity apparent in the area of network and service management is that of the system development methodologies and notations.

Development methodology is a set of rules of how to group the design information and refine the system from its specification to the actual implementation. Notation is a way, either textual or graphical, of describing the information about the system structure and functionality, throughout the development process.

TMN recommendation M.3020 [M.3020] describes the TMN interface specification methodology. Starting from the target management services that need to be developed, management service components are defined, followed by management functions, and finally resulting in the definition of managed objects needed to support the management services. RACE projects ICM and PREPARE followed and contributed to this methodology.

The development of TINA systems is typically driven by the ODP viewpoint-based methodologies. ODP specifies a way of grouping the system development information into five distinct viewpoints: enterprise, information, computational, engineering and technology viewpoints (for more detail refer to chapter 3, section 3.2.1.1.1). An example of the ODP-influenced approach is the modelling of the management systems in the RACE project PRISM (Pan-European Reference Configurations for IBC Services Management) [Berq96].

The TeleManagement Forum (TMF) has specified guidelines for developing agreements on management interfaces [Vinc97]: the approach is use-case driven. Similarly, some ACTS projects, such as PROSPECT (A Prospect of Multi-Domain Management in the Expected Open Services Market) [Lewi97], adopted the use-case driven management system development methodology [Wade98], based on the Object-Oriented Software Engineering (OOSE) [Jaco92] approach. ACTS project TRUMPET also adopted a use-case driven methodology, structured around the ODP viewpoints [Kand98].

A variety of documentation styles are used to capture the functionality and structure of management system being developed, although the natural language does prevail in some cases.

Guidelines for the Definition of Managed Objects (GDMO) [X.722] is typically used in the context of TMN to describe the managed object classes, while the General Relationship Model (GRM) [X.725] is used to capture the relationships. TINA modelling is conducted via ODP viewpoints, notation used being sequence diagrams and Object Modelling Technique (OMT) [Rumb91] class diagrams; although natural language is used for the large part of documentation. CORBA computational object interfaces are specified in Interface Definition Language (IDL), while the TINA computational object interfaces are specified in Object Definition Language (ODL) [TINA-ODL], superset of IDL. The TMF approach relies on the OMT class and sequence diagrams.

Recently, Unified Modelling Language (UML) [UML] is emerging as the notation of choice, becoming a de-facto standard notation for software modelling in general. UML was used in a number of ACTS projects such as FlowThru, TRUMPET, PROSPECT, *etc.* Working groups such as ITU, TMF and OMG are considering its adoption.

2.3.5 DISCUSSION AND FUTURE DIRECTIONS

As we have seen, there is a breadth of standards and technology frameworks that are applicable to open management system development. This situation suggests the future direction towards a fairly loose, unifying management system architecture. Likewise, in [Pavl98], it was identified that one of the main future challenges is the integration of all distributed telecommunications software, such as TMN, IN and TINA: both management and service control.

Similarly, there is an increasing need for a common methodology, or guidelines, as well as unifying notation, for development of management systems. An adoption of such an unifying approach would aid a wide range of parties involved in the development of management systems to understand and exchange ideas and documentation. Moreover, it was argued [Lew99a] [Lew99b] that such common development and modelling technique could greatly aid reuse, which is seen as one of the key requirements of future management components.

A first step towards a unifying, loose architecture / technology framework was done in [Pavl98]. Here, an approach to marry the TMN and ODP concepts was presented, and CORBA was suggested as a base technology for TMN, instead of OSI-SM. This might be seen as a step towards common middleware: undoubtedly, CORBA, as an implementation of

the ODP distributed object framework, is becoming the alternative management technology. However, implications are wider: network management is going in the direction of lax architectures, with the architectural concepts of TINA and TMN overlapping through a common distributed object framework - ODP - where CORBA is the technology of choice.

The second key future issue is that of the common management system development approach and the corresponding notation. The object-oriented modelling is a strong pre-requirement. Telecommunications management nowadays cannot be perceived without object-oriented information modelling - one of the key requirements for TMN - as well as object-oriented, interface-based computational models in the context of TINA and ODP. The emerging unifying notation seems to be UML. UML is a third-generation object-oriented modelling language with considerable expressive power, which is becoming the de-facto standard in the software industry. UML, however, lacks a clear development methodology, especially in the context of management systems. A strong candidate framework for structuring the UML modelling information in the context of management systems is ODP - considering its applicability in both TMN and TINA scenarios.

The overall indication is that the transition from protocol-based technologies towards distributed system technologies is taking place. We thus believe that the future is likely to see a use-case driven management system development methodology, where the UML models will be structured in an ODP viewpoint framework, describing lax architectures, in the form of a melange of TMN and TINA concepts.

2.4 CHAPTER SUMMARY AND DISCUSSION

This chapter dealt with three key background areas relevant for our research: definition and understanding of the term integrity; existing integrity-preserving techniques and frameworks; and state of the art in network and service management - on which our integrity research is focusing.

First, we tackled the concept and understanding of the term integrity. We discussed and critically reflected on the two main prose definitions of network integrity. Both defined integrity as a higher-level notion, encompassing a number of attributes relating to network operation and performance, but without defining what these attributes are. We then presented a survey of the existing integrity measures. The outage measures were discussed, and assessed as *post-mortem* measures lacking capability to determine neither the current level of integrity, nor the risk of a future outage. A single existing *pre-mortem* definition of integrity was then discussed. This definition also identified the existence of a number of attributes and factors effecting network integrity, however, without clearly pin-pointing what they are.

Finally, two terms were reflected upon, used in the field of computing, and bearing a strong resemblance with integrity: dependability and safety.

Considering the above, we pin-pointed the **lack of a clear definition of integrity and of the identification of integrity attributes** as one of the obstacles to effective dealing with the issue. Our contribution to this aspect of the research is given in chapter 3, section 3.1.

The second topic considered was that of the existing research and industrial activities in the field of integrity assurance. Most of the initiatives focus on the integrity of the core transport network, while some steps were also taken towards tackling the integrity of the control plane. There is no reported research in the integrity of the management plane.

In practice, the operators rely on the exhaustive pre-launch system/service testing as a means of integrity assurance. In this context, the existence of the test-beds for the core/control networks is of high importance. During runtime, the only integrity-preserving mechanisms in place are screening and mediation devices at the interconnect between autonomous domains. On the other hand, there is a shortfall of the pre-emptive integrity preserving techniques.

The academic research to date was generally high-level and analytical, pin-pointing the under-researched areas and producing recommendations for issues to be considered. Some attempts were made to provide a structured framework for tackling the integrity issues throughout the system/service development lifecycle: however, no obvious flow of integrity-preserving actions is defined, and the relationship with the system development lifecycle is not strong.

Considering the state of the art in integrity management, we identified two shortfalls: **lack of a structured framework**, and **the scarcity of pre-emptive techniques for producing highly integral systems**.

Next, we highlighted the need for integral operation of the management plane. To argument this, we captured the integrity relationship between the management plane and the underlying managed network in the form of the control problem. We identified the need for ***managing the integrity of telecommunications systems, specifically management systems***, as the key focus of our work.

Thus, we then presented a brief overview of the state of the art in network and service management, in terms of architectures, technologies, development methodologies and notations. This state of the art represents a necessary background in a number of ways. First,

any consideration of integrity of management systems needs to take into account both the architectural aspects of the management system, as well as technologies used to implement it. Moreover, integrity issues need to be considered throughout the management system development lifecycle: thus, the development methodology deployed, and the modelling notation used, will both be of paramount importance when tackling the pre-launch integrity issues. Since we already identified the lack of *pre-emptive* techniques for providing highly integral systems as one of the shortfalls of the current approach to integrity assurance, the issues of development methodologies and modelling notations must be considered even more closely.

The open issues we identified in this chapter are tackled throughout the thesis.

First, in chapter 3, section 3.1, we introduce our understanding of the term integrity, represented as a set of lower-level attributes. In chapter 3, we then present our **integrity management methodology** (section 3.2), which aims to structure the integrity-preserving actions, and overcome the deficiencies and unify the expertise of both industrial and academic approaches that we discussed in this chapter. Moreover, we consider a number of pre-emptive techniques, or **integrity policies**, that can be used as a tool for producing highly integral systems. Some candidate policies are discussed in section 3.4, while the two central policies in this thesis are presented in chapters 5 and 6. These two policies were developed using the two ACTS network and service management projects, TRUMPET and FlowThru, as research platforms. The overview of these projects will thus be given in chapter 4.

3 INTEGRITY: DEFINITION AND METHODOLOGY

This chapter presents the core theoretical work conducted for the thesis: the novel interpretation of the concept of integrity; and the development of the framework for management of integrity issues throughout the system development and operational lifecycle.

Our analysis of the concept of integrity, which is represented as a set of lower-level attributes, is given in section 3.1. In the central section of this chapter, section 3.2, we present our integrity management methodology and discuss its pre-requirements and phases in detail. We also discuss practical issues concerning the cost-effectiveness of the methodology and its applicability in the inter-domain environments, in section 3.3. In section 3.4, we elaborate on some integrity-preserving policies that can be considered in the context of developing highly integral management systems. We conclude this chapter with the summary and overview of research contributions in section 3.5. Some of the material in this chapter has been published in [Prnj99a] and [Prnj00a].

In this chapter the integrity features of distributed telecommunications systems are discussed in general terms. Nevertheless, all of the integrity concepts, strategies, and policies discussed apply to the management systems as well, since management systems are just a specialisation of a classical distributed system.

3.1 DEFINITION OF THE CONCEPT: INTEGRITY ATTRIBUTES

In this thesis, we adopt the established prose definition of integrity as given in [UCL94] [Ward95] (see chapter 2, section 2.1.1), and adapt it to define **telecommunications system integrity** as: "*the ability of the system to retain its specified attributes in terms of performance and functionality*".

Next, we recognise the need to identify these "specified attributes". This is not a simple task since integrity is a broad term, encompassing a variety of issues concerning system structure, functionality and behaviour. What follows is a breakdown of integrity attributes, and issues that need to be considered when managing integrity. These attributes are identified through consideration of both the dependability and safety concepts (chapter 2, section 2.1.3), and the issues specific to the integrity of telecommunications systems (see chapter 2, section 2.1.1, 2.1.2).

Robustness, which can be defined as "the ability of the system to handle unexpected events" is proportional to integrity - the more robust the system is, the more likely it is to retain a high level of integrity within its operational environment. A system that is robust can cope with all eventualities and continue operations (doesn't 'halt' except when required to). The opposite of robust is **brittle**. A system is brittle if it is likely to fail when the operational environment (*e.g.* accessible data, requested commands, timing constraints *etc.*) is very narrowly defined; more narrowly than is likely to be true in all circumstances.

Availability, defined as "percentage of time during which the system is operational and conforms to its specification" [Vill92], is proportional to the integrity of the system - a system that loses its integrity will suffer a loss of availability as well. Availability means that a system can always respond to all requests made on it, within a required or specified time window. Availability is sometimes expressed as A=MTTF/(MTTF+MTTR), where MTTF is the mean-time-to-failure and MTTR is the mean-time-to-repair.

Performance: The throughput of the system. This is often traded off against functionality since the more a system tries to do, the lower its throughput. Any degradation of system performance can, if magnified, significantly effect system's overall integrity status.

Data Coherence: Information copied or distributed through the system needs to remain consistent through time and change of circumstances. ACID requirements: Atomicity, Consistency, Isolation and Durability have to be fulfilled. Atomicity requires that a transaction be either executed fully, or not at all. Consistency means that a data transaction should take the system from one internally self-consistent state to the other. Isolation ensures that an incomplete data transaction can never reveal its partial changes or internal state to other transactions before its fully executed. Failure to ensure isolation can jeopardise the system operation by providing it with inconsistent or false data. Durability refers to the ability of the system to ensure that the result of a successful data manipulation can never be lost. Any corruption of data manipulated by the system can endanger its operation. If data coherence is lost, a system can gradually lose its integrity.

Liveness: Ability of the system to stay live at all times. A system might not remain live because it is in a state of either deadlock or livelock. **Deadlock** is the state of a system in which it is expecting a message or an event, which will not or can not occur. **Livelock** is the state of the system where it oscillates between a closed set of states that it cannot leave.

Complexity: There are several established notions of complexity. It may be an assessment of how long (how many iterations or cycles) an operation takes - known as ***time*** **complexity.** ***Computational*** **complexity** relates to how well a given procedure can be analytically

described or determined. ***Data*** **complexity** refers to the complexity of data structures and their interdependencies. With the advent of object-oriented culture, understanding of complexity has acquired a somewhat different meaning, being understood as "a characteristic of software that requires effort to design, understand, or code" [Hend96]. A high level of complexity, unless it is there to increase robustness, poses more threats to system operation and thus to integrity.

A high level of **coupling** (sometimes considered as a form of complexity) between system components indicates a high level of interdependence: a change in a system component will ripple through the system via the coupling paths. Similarly, a failure, or an integrity breach, may propagate in such a way through the system, effecting its integrity.

Feature Interaction: When two or more systems/services, each with well-defined and understood behaviour, result in unforeseen (and possibly unforeseeable) behaviour when operated together. This is a well-known phenomenon, especially in the context of IN services [Came93]: a classical example is the interaction between Call Screening and Call Forwarding where the screened number X can be obtained if user A calls user B and B forwards the call to X. The feature interaction issue can arise due to the other factors influencing the integrity status of the systems, such as increased computational complexity exhibited when systems are interconnected, or lack of data coherence.

Scalability: The impact on *performance* as more entities (processes, devices *etc.*) are added to the system. The way a system scales is, in great part, a function of its computational, data and time complexity.

Resilience: That a system can recover from faults. This term is often used, for example, in networks where a resilient network can recover from link faults.

Reliability is defined as [Reib91]: "probability of a system performing its purpose adequately for the period of time intended under the operation conditions encountered". Reliability estimation is focusing on prediction of mean-time-to-failure (MTTF), based on testing experience and operational profile of the system in use. For a detailed survey of reliability prediction, see [Dens98].

Security: Secure systems are more likely to stay in the correct operational state, since they are able to detect and avoid intentional external (human) attack. Typical security sub-requirements are authentication, access control, data integrity, confidentiality and non-repudiation. Authentication refers to the mutual recognition of the communicating parties.

Access control ensures that an external party can access just a certain subset of functionality/data of the system being secured, according to the contract. Data integrity means that data in transit must be protected against modification, insertion, and repetition. Confidentiality means that data content must not be disclosed, while in transit, to unauthorised parties. Non-repudiation refers to the resolution of the dispute where one party denies that communication took place.

A distinct sub-attribute of integrity is **risk**. Definitions of risk, as established in the computing arena, were introduced in chapter 2, section 2.1.2.3. In our framework, we view risk essentially as inverse to integrity, *i.e.* the higher the risk, the lower integrity.

The topics described above are all interrelated. For example, a system with poor scalability will lose performance as the number of entities it involves grows. This would reduce availability. Loss of performance and availability can impact on the timing of dependent systems, for example in impacting on the data coherence or liveness constraints. Also, a high level of security in the system ensures that the correct operation can not be jeopardised through intentional misuse, but the computational overhead introduced by security mechanisms can lower system availability and performance.

The number of integrity attributes and a high level of their interdependence imply a need for a coherent framework for tackling these issues throughout the telecommunications system lifecycle. A second step in the theoretical work discussed in this thesis is thus concerned with developing an integrity management methodology, and is presented in the following section.

3.2 METHODOLOGY

An efficient integrity methodology must embrace all stages of the system life span: the development process, testing, integration, and maintenance while operational. Without a top-down integrity methodology that encompasses all stages of system development, integration in the environment and its real-time operational features the problem cannot be fully understood and managed, and threats to integrity cannot be identified and removed. Three basic steps of the integrity methodology developed here are prediction, testing, and maintenance.

Prediction is a pre-emptive activity, assessing the relevant integrity features and the overall system integrity status prior to its introduction in the environment. It is aimed at locating and removing integrity risk areas - hotspots during system development, thus producing a robust system; and conducting actions, *i.e.* integrity policies, that ensure integrity preservation.

Testing is conducted not only during system development, but also after the implementation of the system and before its introduction in the environment. The aim of this phase is to test the correct operation of the system prior to its deployment in the operational environment. Testing here thus refers to the final validation and integration tests performed during system integration in the environment and prior to its full operational launching.

Maintenance is conducted after the system deployment and aims at detecting any malfunction or degradation in system operation that might pose a risk to system integrity. Maintenance encompasses the measurement of integrity-relevant features of the operational system, diagnostics of the cause of degradation of integrity, and the application of the relevant response. Response is a reaction to degradation of system integrity so as to preserve the highly integral operation of the active system.

The emphasis in this thesis is given to system development and integration - the prediction and testing stages of the integrity methodology. It is expected that by introducing appropriate integrity design and testing issues early in the development lifecycle the threats to integrity in later stages of the system life-span will be minimised.

Note that since the system can be perceived as a set of distributed objects providing a certain service, the following methodology is applicable to systems as well as services: both can be seen as a set of collaborating distributed components.

3.2.1 PREDICTION

As an illustration of how to start to synthesise integrity strategy into system development an example system engineering lifecycle is taken from the Hierarchical Object-Oriented Design (HOOD) [Robi92] approach: this is illustrated in Figure 6. Many other formulations of lifecycles exist, but this one is considered for its clarity. This model should not be confused with the 'waterfall' model as it does not dictate how each phase should be managed with respect to the next: instead, it describes how each phase should be mapped to the development activities. In the HOOD lifecycle, five basic development phases are defined and these are mapped onto a testing activity which matches each level in an appropriate way. Figure 6 describes the engineering lifecycle and shows how the testing and integrity activities to be considered are mapped to the development phases. Thus, for example, during user requirements development phase, the focus is on the operational integrity and the necessary integrity-related activities are conducted at this level.

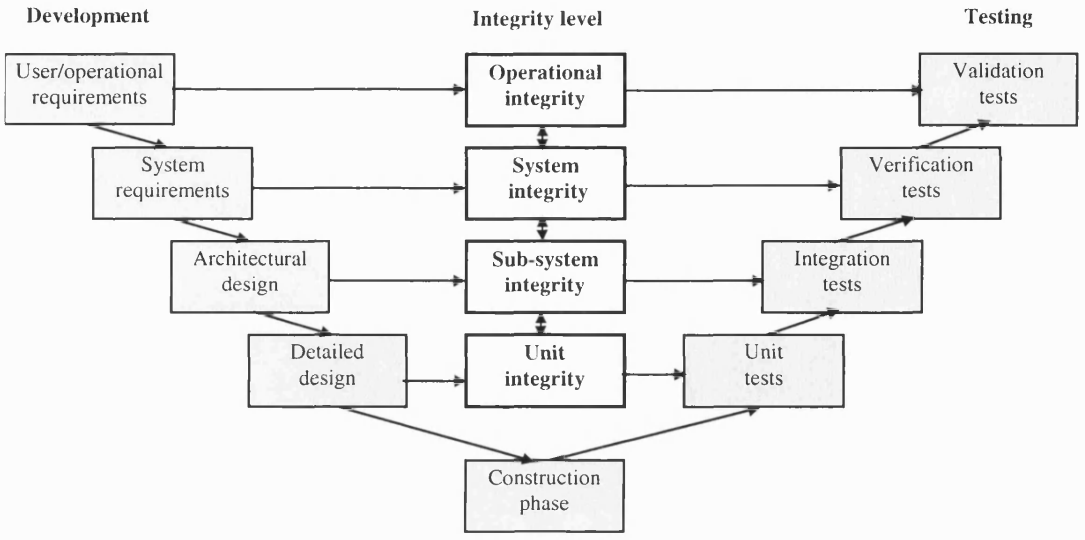


Figure 6 - Integrity strategy and system development lifecycle

In the testing hierarchy, the activities at each phase are clearly distinct and support the phase above. The integrity hierarchy is different, not only do the activities at each phase support those above; but also they *inform* those above. For example, at the unit level, the timing of any action can be specified and verified. This then defines timing tolerances and dependencies at the sub-system level.

As illustrated in Figure 6, the integrity-related activities are correlated to and performed throughout the conventional system development process. These integrity development activities can be perceived as a development lifecycle of its own, a lifecycle to be re-iterated throughout the conventional lifecycle. Thus, the integrity sub-cycle has three basic stages: **integrity analysis**, **integrity design** and **integrity implementation**. These three integrity stages are iterated throughout system development.

First, integrity analysis of the system under development is conducted: the integrity-related requirements are identified throughout the system development lifecycle. In order to accomplish this, there is a need for an analysis framework in which to focus on different kinds of integrity issues: this is discussed in section 3.2.1.2. Each of the integrity concepts outlined in section 3.1 can be located within different levels of integrity analysis. At this point it should be noted that the results of integrity analysis, which is being conducted throughout the system development (Figure 7) are correlated within different levels of development and thus feedback and overlaps will almost certainly occur.

According to the integrity requirement classification conducted during analysis, the integrity design is specified, *i.e.*, integrity can be modelled into systems by defining integrity-preserving policies that should be deployed during the system development lifecycle at the

relevant stages. The final bit of the predictive phase of the integrity methodology is *how* to actually apply these policies during system development - integrity implementation.

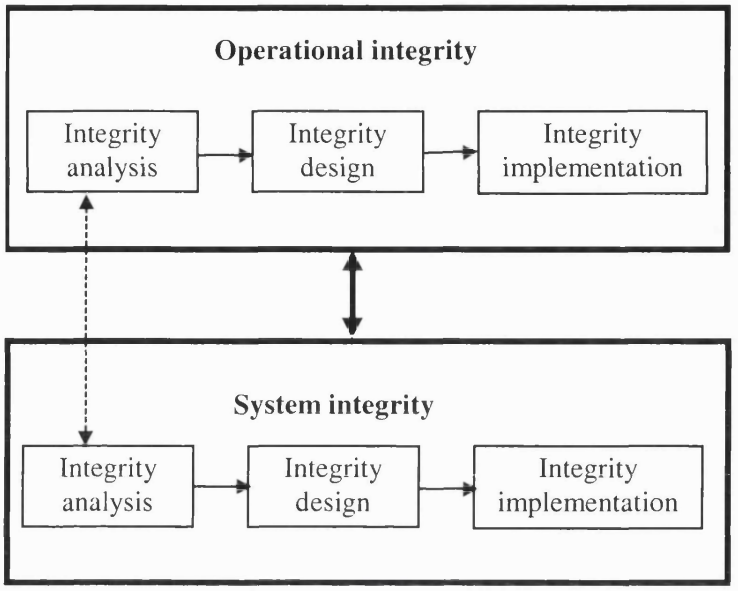


Figure 7 - Integrity lifecycle

Prior to considering the prediction phase of the integrity methodology in detail, a suitable system development process supporting basic requirements of a coherent development approach must be defined. Moreover, the integrity-related attributes should ideally be measured so as to allow comparison, assessment and improvement of the integrity features of the system under development. The next section discusses these two issues.

3.2.1.1 METHODOLOGY REQUIREMENTS

There are two requirements for the integrity management methodology to be effective: a coherent system development approach, and measurement of the integrity-related attributes.

3.2.1.1.1 System development approach

There are three basic issues concerning the system development process:

- **Specification - System Analysis and Design:** The baseline definition of integrity (correct and proper functioning) requires that it is possible to understand what the correct functioning of a system is. Thus the start of the consideration of integrity attributes is in the process of system requirements capture. A system has some chance of realising its original requirements if there is a reasonably well-defined procedure for translating the original requirements into a system design and then into a working system. Thus it can be seen that the subject of system and software analysis and design is material to the understanding of integrity issues.

- **Tracing and Mapping:** This requirement is based on the need that the entities (software classes, modules, components) occurring in one domain of system development can be identified/mapped when they occur in another, and that they do so correctly (*e.g.* a software module defined in the detailed architecture is implemented correctly in the code). The use of CASE tools and (semi) formal techniques is often used to ensure that the identity of an object can be traced through the different phases of development. Traceability is a classic requirement of good practice - forming, more or less, the backbone of quality control systems such as ISO9000 [ISO9000].
- **Integrity-friendly**: The development process must be able to support the integrity analysis and assessment of the system under development.

No definitive way of fulfilling these requirements exists as yet in the conventional distributed processing world. The following describes one approach [Kand98] which combines the Open Distributed Processing (ODP) [ODP] framework with the Unified Modelling Language (UML) [UML] notation. This approach was adopted here due to its general applicability for distributed telecommunications systems, and specifically for its suitability for network and service management system development, as discussed in chapter 2, section 2.3.5. Moreover, this approach was co-invented by the author of this thesis, and was successfully applied in the TRUMPET project, which was the main research platform (see chapter 4, section 4.1) for the investigation of integrity issues in this thesis.

ODP provides a general architectural framework that distributed systems aiming to operate in the multi-provider environment must conform to throughout their development. The bases of this architectural framework are the five distinct viewpoints, which allow different participants in system development to observe the system from a different perspective and from a different level of abstraction. The ODP methodology incorporates five distinct viewpoints: enterprise, information, computational, engineering and technology viewpoints. The *Enterprise Viewpoint* represents an overview of the system and its aims and functionality as seen by the enterprise and the user. This viewpoint describes the required system capabilities, models the basic system decomposition into components, identifies actors, policies and domains, and describes the general scenarios of the system use. The *Information Viewpoint* provides a consistent and common view of all the information handled by the system. Both the static view - the information objects, their structure and relationships; and the dynamic view - how this information evolves - are given. The *Computational Viewpoint* focuses on algorithms and data flows within the system. It identifies system components, or computational objects, that provide the functionality of the distributed system. The *Engineering Viewpoint* describes the actual realisation of the mechanisms used to support the distribution of the components in the system. The

Technology Viewpoint describes the choice of implementation technologies used to bring the design accomplished through the four previous viewpoints to life. This viewpoint depicts the configuration of the hardware and software on which the distributed system relies.

The viewpoints are partial views of the complete system specification, and the description of the same component can exist in different viewpoints. This gives rise to the viewpoint consistency issue, referring to the consistency of specifications across different viewpoints and consistency of different languages (notations) used in different viewpoints [Bowm96]. ODP recommendations do not advocate which languages to use in each viewpoint. Thus it is possible to describe some data entities in the information viewpoint, and some processing on those entities in the computational viewpoint, without being necessarily able to know that the information being used in each case is identical. Thus part of the mapping requirement in this context is that objects in each viewpoint can be clearly identified and related to each other as required, as illustrated[2] in Figure 8.

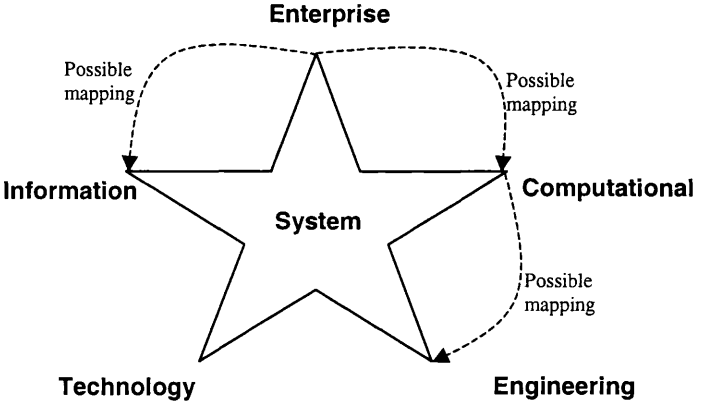


Figure 8 - ODP viewpoints with mappings

It is worth noting that the general ODP "star" diagram of Figure 8 does not effectively capture the relationships between the viewpoints from the system development perspective. Typically, system development would start with the specification of the enterprise viewpoint, then the information and computational viewpoints would be elaborated in parallel, followed by the engineering viewpoint and finally the technology viewpoint.

Different semi-formal and formal languages may be used for specifying different ODP viewpoints. Formal descriptions are deployed in the ODP framework with the aim to enable precise, unambiguous, abstract definition and interpretation of ODP standards. However, the approaches to languages and notations used nowadays have many drawbacks. Usually,

[2] This is just and illustration of the possible mappings between the viewpoints, not aimed at detailed elaboration of the mappings.

different languages are used for different viewpoints, causing poor viewpoint consistency and mapping/traceability between system components through viewpoints. Many languages lack the ODP-inherent object-orientation, as well as the tools support. Here, one single viewpoint language is suggested for the description of all the ODP viewpoints: the Unified Modelling Language, UML.

UML is a third generation object-oriented language that builds up on the established techniques such as OMT [Rumb91], the Booch technique (Object-Oriented Design - OOD) [Booc94], and the Jacobson technique [Jaco92], by offering some extensions to the notation so as to provide a richer semantics and higher coherency of models. It is envisaged that it will become a standardised object-oriented modelling language, and it is supported by a variety of CASE (Computer Aided Software Engineering) tools.

UML provides a set of diagrams, each depicting a different perspective of the model of the system under development. *Class Diagrams* describe the static structure of the object classes in the system. They can depict the stand-alone object classes; relationships between classes, such as association, aggregation (containment relationships) and inheritance (parent-child relationships); as well as class interfaces. *Use Case Diagrams* describe how the system is to be used - they depict the high-level functionality of the system. *Collaboration Diagrams* depict how the scenarios of the system use are realised through interactions between object instances. A realisation of a particular scenario is conducted via message exchange between object instances: a client object, requesting an operation to be performed by a server object, initiates operation invocation. Precedence rules are used to define the sequence of operations performed. *Sequence Diagrams* complement collaboration diagrams. Sequence diagrams depict the same scenarios as collaboration diagrams in a time dimension - object interactions are arranged in a time sequence. *State Diagrams* depict dynamic behaviour of object classes. They describe a set of states that an object goes through its lifetime. *Component Diagrams* model the development view of system components and their relationships. *Activity Diagrams* describe the order in which activities are performed, depicting parallelism and synchronisation. *Deployment Diagrams* show the organisation of the hardware devices and their particular interfaces, and software as related to the physical devices.

It is not necessarily possible to use the same notation for all the viewpoints, however, the ODP enterprise, computational, information and to some degree engineering viewpoints can all be described using UML. Since UML defines, to some extent, how different kinds of descriptions should relate to each other, there is some possibility of producing the required mapping. A possible usage and mapping [Kand98] is shown in Figure 9. Note that the technology viewpoint is not described using UML, and appears in the diagram purely for

completeness and to illustrate its relationship with other viewpoints. For the details of the ODP-UML mappings and relationships, refer to chapter 4, section 4.1.2.2.

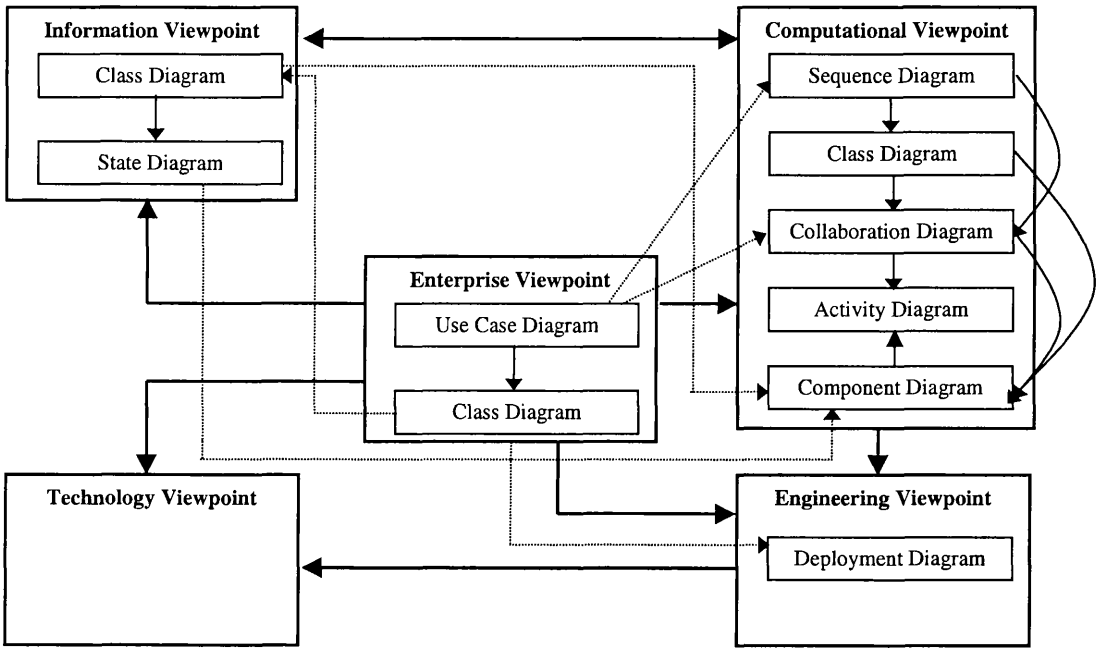


Figure 9 - ODP ↔ UML mappings [Kand98]

UML, as a single and unifying viewpoint language, eases the migration between ODP viewpoints, enabling viewpoint consistency and thus the consistency, coherence and completeness of the design itself. Tracing of the classes/components through the design and mapping between the different viewpoints is made possible [Kand98][Prnj97][Prnj98b]. Finally, this development gives a sound basis for deploying the integrity methodology.

For the details of the ODP-UML mappings of Figure 9, the description of the application of this approach for the development of a real-life management system, as well as for the practical assessment of this approach, refer to the overview of the TRUMPET project - chapter 4, section 4.1.2.2.

3.2.1.1.2 Measurement

The second requirement on the integrity methodology is the need to quantitatively assess the integrity attributes of the system under development, *i.e.* the need to measure different aspects of systems' structure and operation that can have an impact on the integrity of that system. This can be seen more as a desirable feature of the methodology, than as a requirement, since not all of the integrity attributes (defined in section 3.1) are quantifiable.

Without quantitative information, *i.e.* without measurement, the entity under observation cannot be fully assessed, managed, and improved. Without measurement, we cannot make an objective statement concerning quality of the product that we engineer. Finally, a

quantitative measure of integrity-related parameters is crucial so as to allow a wide range of parties that take part in the development of heterogeneous systems and in inter-domain interconnections to understand and demonstrate to each other that their systems operate correctly and in highly integral fashion.

Measurement can be defined as "the process by which the numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules" [Fent91]. By entity we refer to the object under observation, in our case, the telecommunications system and its components. The attribute is the property of the entity under observation.

When measuring the attributes of things, measurement requires the identification of the intuitively understood attributes of clearly defined entities. Measurement is then the assignment of numbers or symbols to these entities in a way that captures our intuitive understanding of the attribute [Fent91]. To measure the attribute we need to have a corresponding relation in a number system, and measurement is then the assignment of numbers to entities so that the empirical relationships between these entities are preserved [Kran71]. This requirement, that the relationships which exist in the empirical system must be preserved in the numerical system, is called the representation condition. There are a number of ways in which the representation condition can be satisfied, depending on the way in which we assign numbers to entities. Hence, there are different scales, depending on the approach to number assignment. For different scale types, there are different statistics applicable to measurements. There are five types of scales [Kran71]: these are listed below, together with the corresponding statistics that can be applied to measurements, and the examples of scales:

- Nominal scale, like simple labelling. The valid statistics are mode and frequency.
- Ordinal scale, like preference. The additional valid statistic is ranking: only the ordering is implied.
- Interval scale, like degrees Celsius. The valid statistics are ranking, addition and average.
- Ratio scale, like degrees Kelvin. The valid statistics are as for the above, plus ratio.
- Absolute scale, like simple counts (number of days in a year). Full range of statistics is applicable.

The attributes measured can be internal and external [Fent94]. Internal attributes are those, which can be measured objectively and directly and only in terms of the actual entity on which measurement is focused. External attributes are those which can only be measured with respect to how the entity under observation relates to its environment. Integrity, as stated before, is a subjective measure, and also an external attribute, since it is assessed not

only via internal attributes but also through its relationship with the environment. External attributes are notoriously difficult to measure, they are often not well defined, and thus we are forced to make contrived definitions of external attributes in terms of some other attributes that are measurable. This is especially valid in our case, where the external attribute is integrity, which is a complex, higher level attribute, and thus not directly measurable. In other words, we are forced to measure the internal attributes to support indirect measurement of the external one - integrity. The need for measurement of internal attributes was similarly pointed out in [Fent94].

Thus, focus throughout the integrity management process should be to measure internal attributes (those which are measurable) of systems and relate them to the target external attribute, *i.e.* integrity. The aim is to use a metric, or a defined measurement technique yielding comparable measurements (on which meaningful statistics can be applied) of internal attributes, and relate them to integrity.

This can be done in two ways (Figure 10): by measuring an internal attribute and then estimating the external characteristics by correlation (route A), or measuring the internal attribute and using a model to estimate the external attribute (route B) [Hend96]. The first approach is more feasible and considerably more widespread in the various branches of research. The second approach is more complex since there is a need for a strong scientific rationale for expecting that there is a functional relationship between the two variables. Using either of the two approaches we can claim that there is an estimate of the external attribute gained from the measurement of the internal attribute.

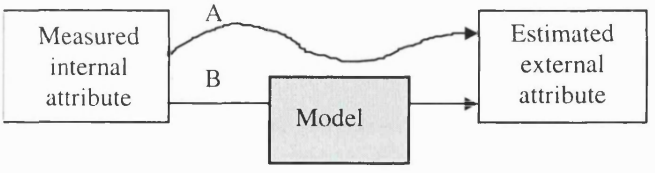


Figure 10 - Correlation versus functional representation [Hend96]

Measurement, in the context of the integrity management methodology, is of particular importance during both prediction and maintenance phases. In the prediction phase, there is a need to measure attributes of the system structure so as to allow estimation of its integrity features, both for pre-launch integrity management and for overall integrity assessment. During maintenance, there is a need to measure the aspects of system operation and relate them to system integrity status so as to be able to decide which remedial policies need to be applied and when.

3.2.1.2 INTEGRITY ANALYSIS

As stated in the preceding discussion, integrity analysis deals with identifying integrity-related issues and requirements throughout the conventional system development lifecycle. In order to be able to effectively define policies for integrity design, there is a need for a coherent framework for integrity requirements classification. Here a three-dimensional framework to accomplish this is presented. The three dimensions are:

- Integrity attributes (as defined in section 3.1).
- Integrity levels: operational, system, sub-system and unit (as defined in section 3.2.1).
- System development viewpoints, as defined by the ODP model (as discussed in section 3.2.1.1.1).

First, the wide range of attributes influencing or being influenced by the integrity status of a system, as defined in 3.1, must be considered. Once the relevant attributes are identified, it must be established which aspect of system structure and operation, *i.e.*, which viewpoint, they apply to. Each of the integrity attributes discussed above can be identified within a certain ODP viewpoint of the system development. Some of the attributes can appear within multiple viewpoints, with different meanings. Classifying integrity attributes according to viewpoints would narrow down the problem and help focus on the attribute within the viewpoint of interest. Finally, these attributes can also be classified according to the system's integrity level through which the problem is perceived. Thus, an attribute can be identified within the system operational integrity level, within system integrity, sub-system integrity or unit integrity. The attribute can also be defined within more than one level: it can be perceived from a different perspective.

Thus, a three-dimensional space (Figure 11) is defined, where each integrity attribute can find its mapping in an ODP viewpoint and in the integrity level.

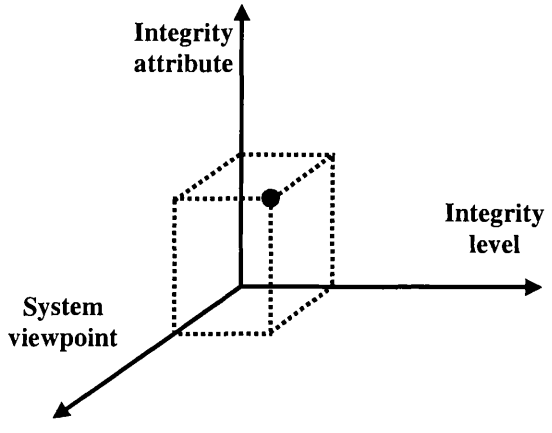


Figure 11 - 3D integrity analysis

Thus, an attribute can for example be found in the ODP information viewpoint on the system level, as well as on the unit level. Also, an attribute can appear in two different integrity levels, but the context of the attribute can be different according to the ODP viewpoint. The whole set of attributes, viewpoints and integrity levels is shown in Figure 12.

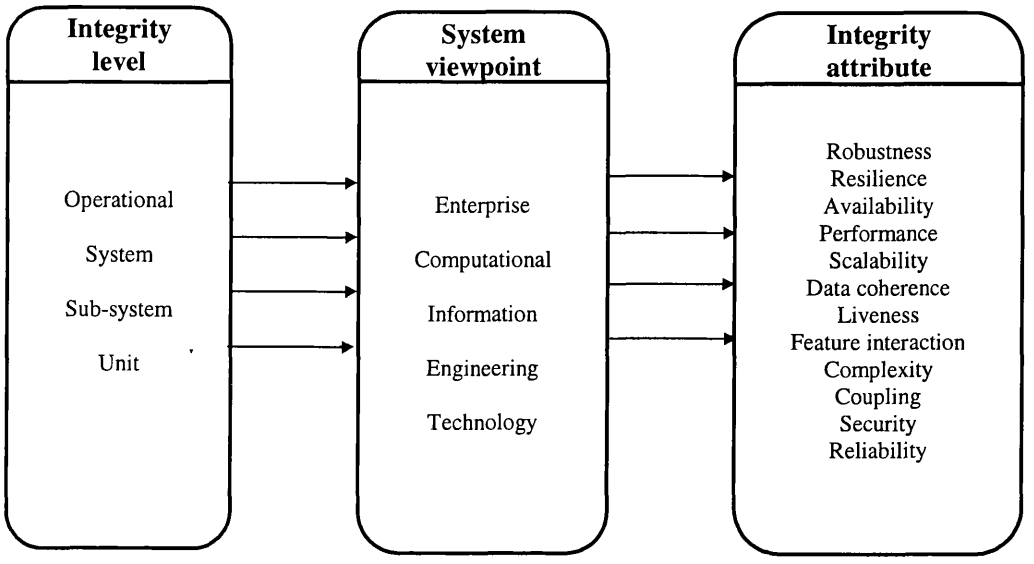


Figure 12 - Integrity requirements classification

From this diagram, starting with an attribute, it can be decided, prior or during system development, what are the integrity requirements, in which viewpoint they should be considered, and in which integrity level they belong. Different integrity requirements will have different weight, depending on the kind of the system being developed.

In the following, some typical integrity requirements are discussed, taking for the reference axis system's integrity level. Note that the integrity requirements on the higher integrity levels are directly supported by the ones identified in the lower levels. For this support to be effective, a clear development methodology must be adopted, supporting mapping and traceability (as discussed in section 3.2.1.1.1).

Operational Level:

This concerns the specific details of what is expected of the system from the user and overall operational point of view. It also takes into account how the system under consideration should work with its environment - *i.e.* other systems already in existence. This level of perception maps to the ODP ***enterprise*** viewpoint, and should capture the operational integrity requirements which must be supported through the lower level system, sub-system and unit requirements. Some typical integrity requirements on the operational level are depicted in Figure 13.

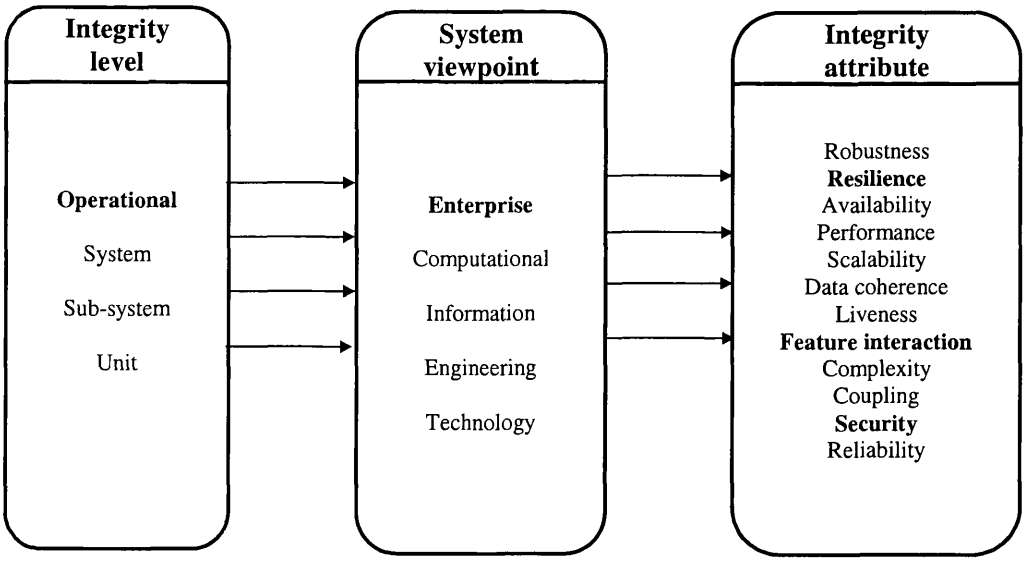


Figure 13 - Typical integrity requirements on the operational level

- **Security**: The integration of a system into an open environment may require that a certain set of security measures be in use.
- **Resilience**: The ability of a system to retain integral operation depends, to some degree, on the threat level from its operational environment, and system's ability to recover.
- **Feature Interaction**: A perfectly well-constructed and operating system may produce unexpected and unwanted effects in its operational environment due to its interactions with other systems: feature interaction issues must be analysed in detail.

System Level:

At the system level, the concern is the characteristics of the system developed in a stand-alone context; and with the parameters as defined by the operational environment. These issues can be taken into account within the ODP ***computational, information and engineering viewpoint.*** Some typical integrity requirements on the system level are depicted in Figure 14.

- **Performance**: This defines how well the system performs both in its base line operation (how quickly, for example, is one interaction processed) and as the system is extended. Thus this area includes consideration of scalability factors. **Scalability** itself can be understood through the more theoretical concepts, including timing complexity analysis. Performance issues support the resilience and feature interaction requirements on the operational level.
- **Liveness**: As the system is built up, the interaction between components becomes increasingly more complicated. This can jeopardise the liveness of the system due to possible occurrences of livelock or deadlock. It may also be the case that messages and commands occur in the system which are mis-handled or not handled at all. That all messages and commands which occur within the system are well handled is a

robustness feature. The liveness and robustness requirements at the system level have an impact on the higher level, operational requirements of resilience and feature interaction.

- **Availability**: The combined effect of good performance and liveness contribute to the overall availability of the system. On the system level some considerations must be understood such as network capacity (for communications) and platform processing capacity.
- **Data Coherence**: In a distributed system, information is gathered from many sources before decisions can be made. This takes time and there is often a possibility that not all the data is valid by the time it has all been gathered and a decision made. Equally, actions which need to be performed over a distributed system may take effect at various time intervals (depending on the degree to which the system is asynchronous) so that ordering of actions or the storage of data can be reasonably difficult.
- **Complexity** of the system and individual components has to be kept to a minimum: highly complex modules/classes within the system have to be decomposed and redesigned so as to avoid high dependency on the availability of a component. Also, **coupling** between system components/sub-systems has to be cut down to a minimum so as to decrease the probability of failure propagation.

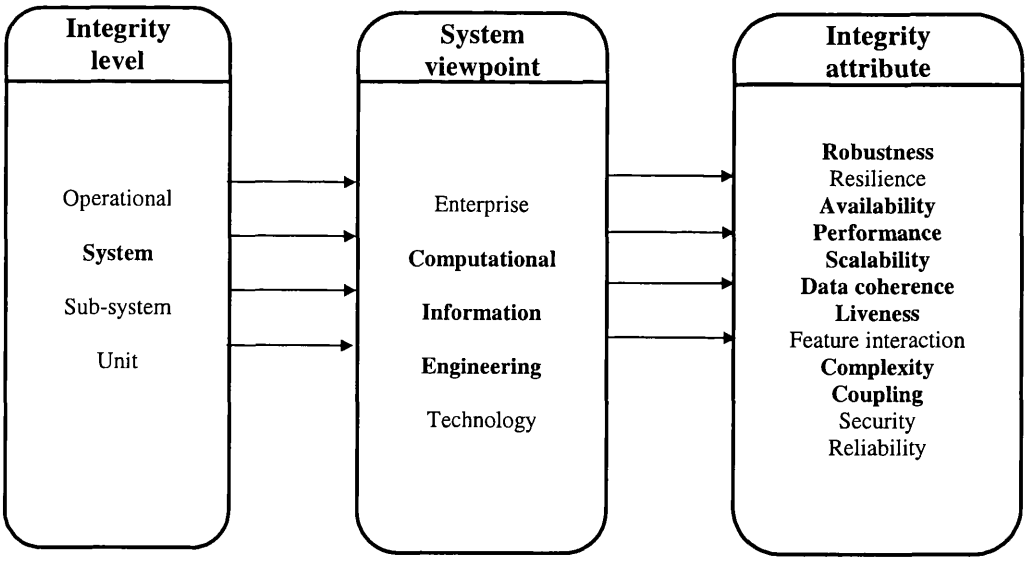


Figure 14 - Typical integrity requirements on the system level

Sub-system Level:

The sub-system level concerns sub-systems of the total system, composed of collections of units, which have the validity as a self-standing system. Thus many of the system level considerations above have a degree of validity at this level. As with testing, ensuring that things work at the sub-system/integrated level reduces the complexity of testing at the

verification level; but does not replace it. This level of perception maps to the ODP ***computational, information and engineering viewpoints.*** Some typical integrity requirements on the sub-system level are depicted in Figure 15.

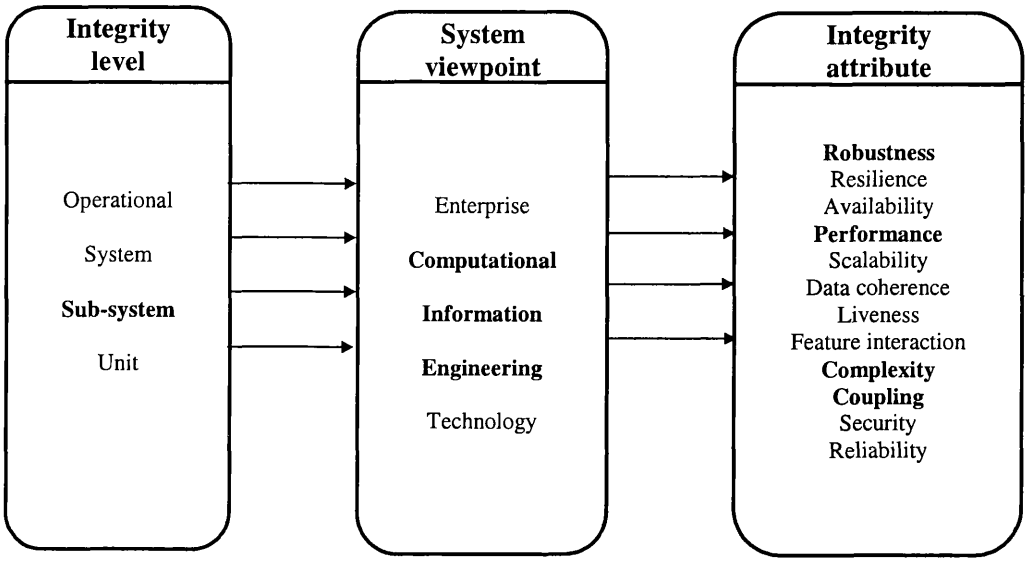


Figure 15 - Typical integrity requirements on the sub-system level

- **Performance**: How the individual units are composed into a sub-system will effect the performance of that sub-system. For example, the same units can be combined into a synchronous or asynchronous system, may be in a ring or in a tree. Even though the performance characteristics of the individual units remains constant, the sub-systems' performance - in all respects - will vary. Also, performance of a sub-system will effect the overall performance on the system level.
- **Robustness**: Robustness at this level is more than the sum of the unit elements' robustness. The sub-system must be able to cope with degrees of failure of individual units. Robustness at the sub-system level effects the performance and liveness integrity requirements on the system level.
- **Complexity**: The complexity issues considered on the system level are still of interest on the sub-system level. Understanding the complexity on the level of the sub-system supports the complexity assessment on the system level. For example, a high level of data and coupling complexity on the sub-system level might effect data and state coherence at the system level. Also, a high level of timing complexity at this level will influence the system performance at the higher integrity level.

Unit Level:

The system is, at the end of the day, composed of individual functional components, connected with a communications network (itself a set of elemental components). If these have poor characteristics, then the higher level has little chance of performing well. Some

typical integrity requirements on the unit level are depicted in Figure 16. The typical integrity requirements on the unit level match closely to those on the sub-system level - however, their meaning is dependent on the integrity level on which they are perceived, as explained below.

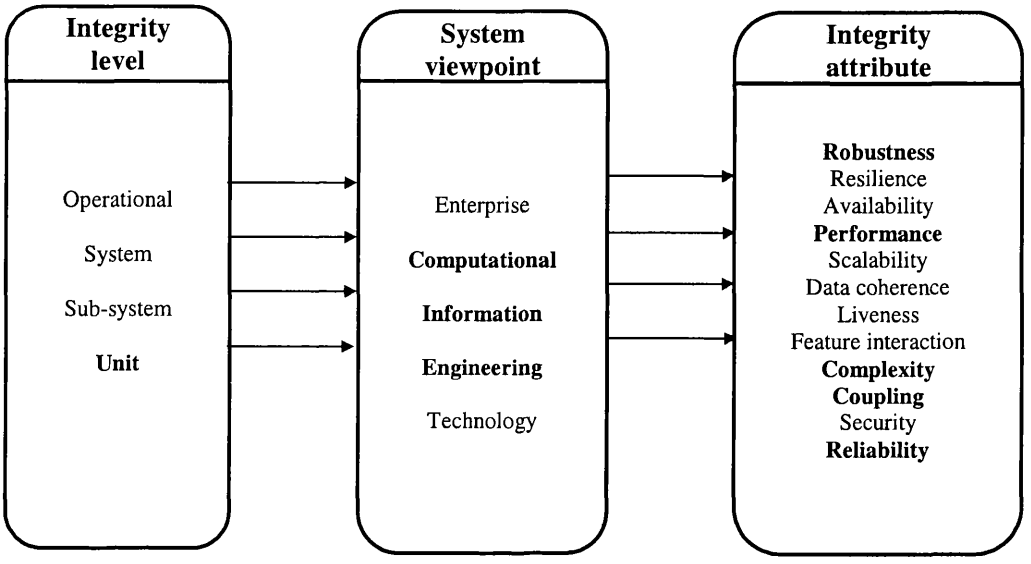


Figure 16 - Typical integrity requirements on the unit level

- **Performance**: As mentioned above, unit performance directly effects sub-system and thus system performance. If a unit is in an inner loop in some sense (*i.e.* is used very often) small changes in performance characteristics can be amplified greatly in the integrated (sub) system.
- **Robustness**: Equally to performance, the robustness of individual components can effect the robustness of the integrated (sub) system; and can be amplified.
- **Complexity** issues on the unit level equally effect the complexity, performance, data coherence and robustness at the sub-system level.
- **Reliability** of the system components, in terms of the mean-time-between-failures, is directly influencing the overall system integrity status.

The above discussed some typical integrity requirements on different levels of system development, and the interrelations between them. These will depend on the kind of system that is being developed.

This section has outlined a high-level integrity analysis framework, without suggesting any particular techniques or approaches with which to tackle these issues. The important thing at this stage is to form an awareness of the issues that exist and to be able to identify risk areas within the system development context. Once the integrity requirements are defined and located in the three-dimensional space, the focus can be shifted to developing the integrity

design policies, according to defined requirements and knowing where to concentrate. The integrity design policies should ensure that integrity requirements are met.

3.2.1.3 INTEGRITY DESIGN: INTEGRITY POLICIES

Once the integrity requirements are identified within different viewpoints and within different integrity levels, the integrity design has to be accomplished. Integrity design encompasses the definition of the policies to be carried out during system development so as to be able to meet the integrity requirements.

The term policy here is used to depict an integrity-preserving action. The policies can take the form of either: *integrity-focused design recommendations*; specification of the *integrity-preserving mechanisms* that need to be implemented; or they can give rise to the definition of *testing recommendations*. It should be noted that the term "policy" it is not used in the context of policy-based management (details of which can be found in [Slom94b]). The term policy is used in its dictionary meaning: "definite course or method of action selected from among alternatives and in light of given conditions to guide and determine present and future decisions" [MW-http].

According to the integrity requirements, the integrity policies are developed. Some example policies, addressing a subset of typical integrity requirements, identified in section 3.2.1.2 and classified according to system's integrity level, are given in the following.

Operational Level:

- If the integrity requirement is *security* on the operational level (enterprise viewpoint), this requirement will be further analysed at lower levels of system development. For example, the confidentiality requirement will be met by designing a required encryption policy within the engineering viewpoint.
- A high level of *resilience* and *robustness* can be achieved by performing extensive test in the **test-beds** that simulate the possible behaviour of the environment.
- If the integrity requirement is to avoid the *feature interactions*, thorough **interconnection and interoperability testing** should be conducted, based on a defined set of scenarios.
- A number of ways to resolve feature interactions in IN systems were proposed, a summary of which can be found in [Keck98].

System Level:

- If the integrity requirement is the *liveness* on the system level, *i.e.*, avoidance of *livelock* and *deadlock*, the UML notation schemes depicting the behaviour of the system can be

complemented with more sophisticated **behavioural modelling** techniques such as SDL [Z.100] and others. On the basis of these, **reachability analysis** and **livelock/deadlock detection techniques** can be conducted in the computational viewpoint, not just on the system level, but also on the sub-system level (see section 3.4.2 for more detail).

- *Data Coherence* requirement might be fulfilled by implementing rigorous **data coherence policies** supporting atomicity, consistency, isolation and durability of transactions on data. Example of such a policy is the Transaction Service for network management applications [Ranc98].
- If the requirement for system development methodology is the ability of the rigorous system development and formal proofing between the phases, UML notation schemes can be expanded and **formal methods** (see section 3.4.1 for more detail) can be applied as a design integrity policy. There are, however, issues of mapping between UML and fully formal notations.

Sub-system Level:

- If the aim is to minimise *complexity* and *coupling*, the adapted **software metrics** [Prnj99b] can be used as a tool to even out the complexity and thus risk level of individual components. This can be done not only on sub-system but also on unit and system levels, considering the computational and information viewpoint.

Unit Level:

- Extensive **performance measurements** can be carried out so as to assess whether the desired level of performance is achieved. This can be done on the sub-system and system levels as well.

The above briefly discussed some integrity policies that can be applied during system development. We see the development of novel integrity policies as one of the crucial issues in integrity management. Thus, some of the policies are further discussed in detail in section 3.4, and chapters 5 and 6 of the thesis focus on two distinct policies.

Once an integrity design policy is developed, it has to be carried out, *i.e.* implemented.

3.2.1.4 INTEGRITY IMPLEMENTATION

The policies identified in the integrity design stage must now be implemented and applied during system development. Some policies can be implemented during the "conventional" system development. For example, developing and coding a module that performs authentication, so as to support the security requirement, or a transaction manager [Ranc98] to ensure the distributed data coherence.

Other policies need to be implemented more carefully. If, for example, the requirement is a low level of coupling, this level must be measured so that it can be shown that it fits the requirements. In such cases, there is a need to have a closed loop whereby the integrity features of the design of the system under development can be measured, the system operation can be predicted, and, according to the design integrity policy, a response, *i.e.*, redesign can be applied. Thus, a control loop must be introduced, implementing a particular policy, as shown in Figure 17.

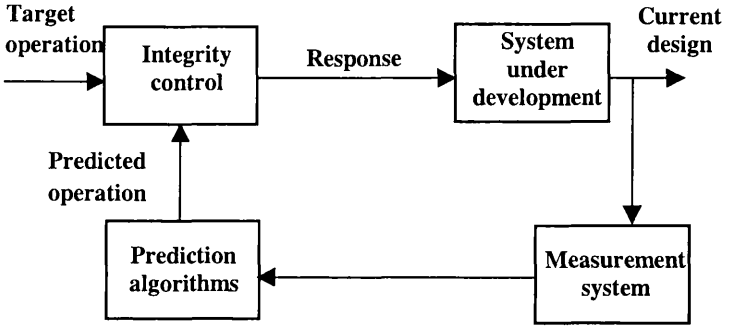


Figure 17 - Integrity policy implementation

The closed, control loop represents the high-integrity development step in the overall system development lifecycle. The relevant integrity requirements, identified during integrity analysis, are measured and then used as a basis for prediction of system operation, which is in turn used as feedback information for applying the relevant response. This is the "reactive" loop of the prediction phase. On the other hand, all of the information gathered in this phase is analysed and documented so as to assess the overall system integrity and risk status. All the information gathered needs to be well-documented and used for defining and implementing the maintenance policies.

As seen from Figure 17, there are three constitutive parts of the "integrity implementation" system:

- **Measurement system**, by means of which quantifiable information about system under development can be gathered. Without a quantitative insight, this approach can not be automated. Moreover, the measurements must be comparable and understandable.
- **Prediction algorithms**, which process these measurements so as to predict system operation.
- **Integrity control algorithms**, which analyse predicted operation and apply necessary response (in form of re-design activities), as defined by the integrity policy, to the system under development.

3.2.2 TESTING

Testing, although an integral part of the development process, is considered here as a separate phase of integrity methodology since it can be perceived as the final verification of the integral system operation. This is particularly relevant in the heterogeneous multi-domain environments where testing is crucial prior to interconnection.

Stand-alone system testing takes place throughout the system development and implementation. As separate components are developed, they are tested independently to verify their integral operation. Once the whole set of components is developed, they are integrated so as to make up the whole system. The stand-alone system testing is then performed, so as to verify correct system operation. Finally, the validation tests are performed so as to test whether the system conforms to the user requirements. The testing phases described above form a testing methodology depicted in the software engineering lifecycle derived from Hierarchical Object-Oriented Design [Robi92] (Figure 6, page 44).

Once the stand-alone system testing (that depicted in Figure 6) is performed, it is crucial to carry out the testing of the system when integrated in the operating environment. This stage takes place on two levels - the 'local', intra-domain testing and 'global', inter-domain testing, including interconnects [Ward95]. The aim of intra-domain testing is to test how well the system integrates in the environment within its own operational domain. The aim of inter-domain testing is to exercise the behaviour of the operational system when interconnected with the systems in other autonomous domains. This can be seen as the crucial part of the testing phase in the context of the Open Network Provisioning (ONP). At this stage, a certain level of performance tests can be carried out so as to predict and measure any possible degradation of performance that might cause the loss of integrity.

Testing is the most widespread integrity management activity that is currently deployed in the industry (see "state of the art" chapter 2, section 2.2, page 24). Although testing is sometimes heavy-handed, the initiatives such as NSTS and IITP are a very significant step forward in the integrity-related testing.

Testing itself could introduce integrity risks if the system is tested against another "live" system. However, test-beds for core and control networks exist, such as the Bellcore test-bed (NSTS), and the BT test network, and which include a wide range of elements encountered in the real environments, and a set of test scenarios including the failure ones. In chapter 6 of this thesis we present an approach for testing the inter-domain interconnection between autonomous management systems over the TMN Xuser interface, and introduce the concept of the management test-bed.

3.2.3 MAINTENANCE

The maintenance phase of the integrity management methodology starts after the system is launched into operation. An efficient integrity maintenance process requires monitoring and measurement of system integrity features throughout the system operation, and actions that help the preservation of highly integral operation.

Approaching the problem of integrity management from the point of view of control systems, the maintenance step can be depicted as in Figure 18 (effectively analogous to Figure 4, page 28).

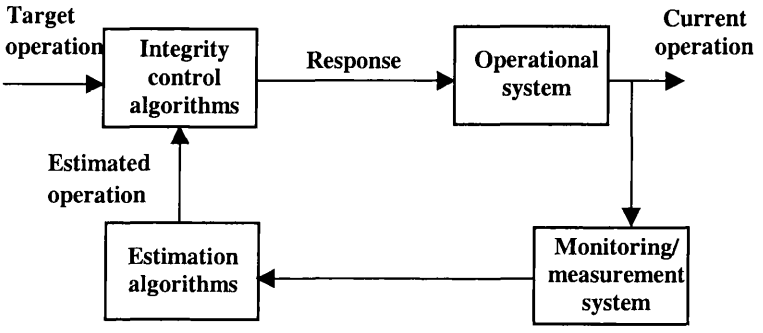


Figure 18 - Maintenance

The closed, control loop of Figure 18 depicts dynamic integrity-preserving actions conducted during system operation. The relevant integrity features are monitored and measured during system operation and then used as a basis for estimation of system operation which is in turn used as feedback information for performing the relevant integrity-preserving control actions, or response. This is the "reactive" loop of the maintenance phase. On the other hand, all of the information gathered from measurement, estimation algorithms and response is analysed and documented so as to assess the overall system integrity and risk status. This information also needs to be documented so that it can be exploited for the future use and integrity research.

As seen from Figure 18, there are three constitutive parts of the integrity maintenance system:

- **Monitoring/measurement system**, by means of which quantifiable information about system operation can be gathered.
- **Estimation algorithms**, which process these measurements so as to estimate system operational features relevant to the integrity status.
- **Integrity control algorithms**, which analyse the estimated operation and apply the necessary response.

The aim of the monitoring/measurement system is to detect any malfunction or degradation in system operation that might pose a risk to the integral system operation. Thus, both the types of measurements that have to be taken and how they can be taken must be clearly specified. These measurements must also quantitatively relate to the integrity status of the system and as such be used for diagnostics (the exact definition of the malfunction in question), which is performed by the estimation algorithms. In order to correctly configure both the measurement and estimation systems, all the data gathered during the prediction phase of the integrity methodology (such as information on integrity hotspots), as well as the testing phase, must be used. Also, measurement mechanisms could be dynamically configured so as to adapt to the actual system operation.

The data collected by the monitoring/measurement system is used as the input to estimation algorithms. There are a number of possible integrity policies that these algorithms might be based on. The functionality of the estimation algorithms might be based on whether the integrity parameters measured exceed certain values - the approach similar to the one proposed in [Mont98]. Alternatively, the measurements might be processed in the deductive fashion by the FMECA - like algorithms (see section 3.4.3 for details of FMECA) so as to anticipate the occurrence of a possible catastrophic failure. On the other hand, if the measurements instantly indicate a catastrophic failure, the estimation algorithms might be based on the inductive FTA-like technique (see section 3.4.3 for details of FTA) so as to diagnose the set of causes that led to it and thus suggest the appropriate remedial action.

A complementary policy during this phase could be based on the periodical re-iteration of tests established during the testing phase so as to verify that system performance, availability and integrity are not degrading.

Finally, an additional, purely experimental activity can be conducted in parallel with maintenance. This activity would comprise the verification of correctness and efficiency of all the measurements and integrity preserving policies carried out during the prediction phase, against the working system. The predictive integrity policies can thus be proven correct through experimentation.

3.3 PRACTICAL ISSUES

There are two key practical issues concerning the integrity management methodology. First, the application of the methodology must be proven to be financially justifiable, through the use of the cost-benefit analysis. Second, the issues of how to apply the integrity methodology in multi-domain environments have to be tackled.

3.3.1 INTEGRITY METHODOLOGY AND COST-BENEFIT ANALYSIS

Integrity methodology covers all phases in the system lifecycle: from specification through to testing, followed by system operation monitoring and remedial policies. The aim of the methodology is to produce highly robust systems, which have a high tolerance to unexpected perturbations and minimal possibility of failure - *i.e.*, high integrity. The reason for employing a rigid integrity methodology is simple - any loss of integrity might cause a significant financial loss to the telecommunications company operating this system. On the other hand, the application of integrity policies during the prediction and maintenance steps must be financially justifiable. If a company is in deficit just for the reason of building fractionally small amount of extra robustness into a system whose failure rate is small, there is no financial justification in doing so. Thus, every policy must be assessed by a cost-benefit analysis, as shown in Figure 19.

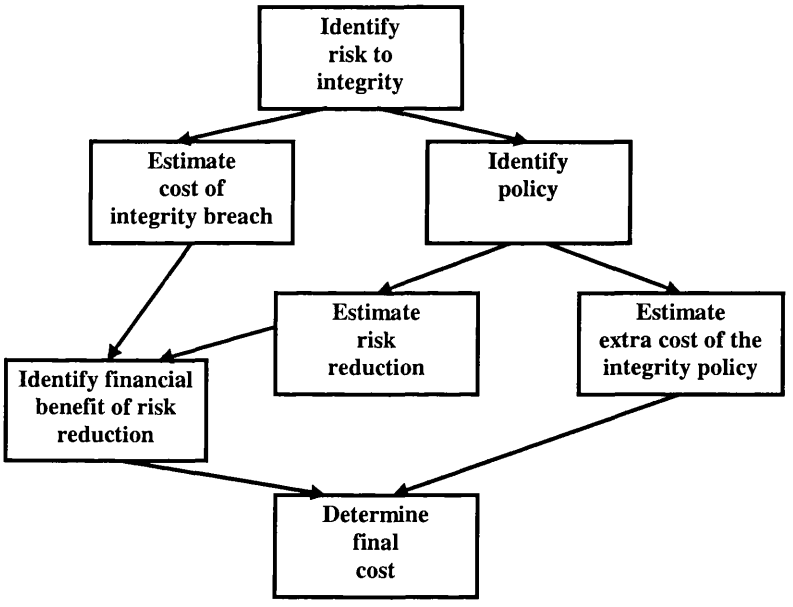


Figure 19 - Cost-benefit analysis process

Hence, if a risk to integrity and a corresponding policy were identified, a financial justification must be given for applying that policy. The financial benefit of applying the policy so as to avoid a breach in system integrity must be compared to the cost of applying the policy. If this final cost is financially viable, the policy is applied.

There are number of operational trade-offs involved in the cost-benefit analysis. Factors that need to be considered encompass the following: increase in reliability versus decrease in performance; increase in robustness versus decrease in availability; decrease in complexity versus decrease in robustness, *etc.* Note that all these attributes should be described quantitatively, so as to be able to make meaningful judgements concerning the trade-offs.

Even if it is shown that applying a certain integrity policy is financially beneficial, there is a second issue to consider: that of how to apply the integrity policy in a multi-domain environment.

3.3.2 APPLYING INTEGRITY METHODOLOGY IN MULTI-DOMAIN ENVIRONMENTS

Enforcing integrity policies in multi-domain environments is both necessary and complex. There is a need to establish a certain level of trust between network operators, service providers, and other parties which might want to interconnect their systems. Parties need to be assured that they are not exposed to integrity risk due to interconnection. However, rigorous integrity policies for system development and interconnection might discourage new service providers and system manufacturers from introducing their services or systems in the market. This would have a negative effect from that envisaged by the ONP. Integrity policies need to keep the highly integral operation of interconnected services and systems, while still stimulating interconnection.

As a baseline for potential interconnection parties involved would need to formulate their integrity requirements during system development. The output of the integrity analysis performed by parties involved in the interconnection could be consolidated in terms of these requirements. The (quantitative) output of the autonomous domains' integrity implementation would then determine the level of risk and threshold criteria determining the systems' acceptability for the interconnection. The threshold criteria would then have the form of: "if a certain integrity-related parameter exceeds x, than it is considered as a threat and thus not acceptable" [Ward95]. If a party involved in the interconnection is re-using the system for this purpose, then a thorough review and integrity assessment of design specifications and extensive stand-alone testing, according to the integrity methodology presented here, would determine the acceptability for the integration/interconnection.
Thus, it is envisaged that an effective way of applying integrity policies in multi-domain, heterogeneous environments is to establish service-level integrity agreements (SLAs), or integrity contracts, between parties in autonomous domains. In federated environments consisting of multiple autonomous players, common integrity policies can be established during development and for testing, and integrity contracts can be exercised prior to the interconnection. This approach is more lax in the sense that the integrity policies and contracts can be established solely by parties involved in the interconnection. Alternatively, global, regulatory conditions based on quantitative integrity/risk notions can be established by standardisation bodies and imposed by regulatory bodies - similar to the approach envisaged in [UCL94].

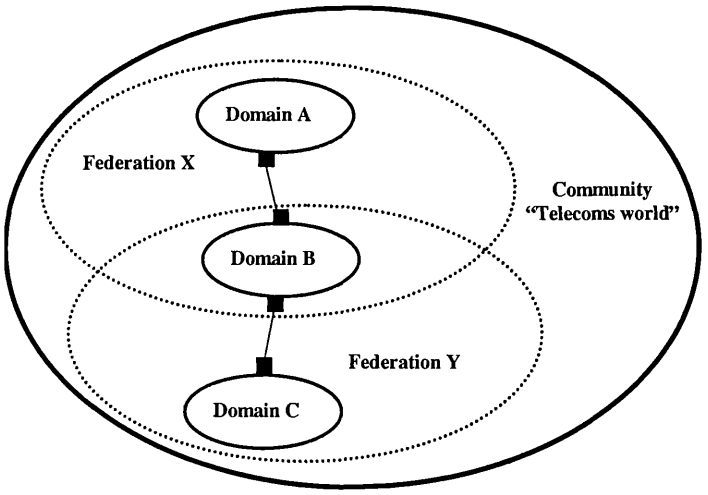


Figure 20 - Integrity contracts in federated environments

The two approaches discussed above are represented in Figure 20, using the ODP terminology. Here, domains are represented by small ellipses, depicting autonomous players. Federations, the medium ellipses, represent groupings of autonomous players with a common objective. The community representing the whole telecommunications community is marked by the large ellipse. The first approach discussed above advocates the use of different service-level integrity contracts (marked by black lines) between domains that form a federation. Federation is based on a certain integrity contract - a set of rules that implement the agreed integrity policies and acceptability criteria. Alternatively, global regulatory conditions based on integrity/risk notions, and governing the integrity-related interconnection contracts between any two parties, can be established by standardisation bodies and imposed by regulatory bodies.

3.4 FOCUS ON POLICY DEVELOPMENT

So far we presented the integrity management methodology, aimed at development and maintenance of highly integral telecommunications systems. This was seen as a necessary step towards improving the current approach to integrity management. The methodology advocated the need for the development of pre-emptive integrity policies, another open area of integrity research (as discussed in chapter 2, section 2.2). Chapters 5 and 6 of the thesis focus on the development of two distinct integrity policies. However, a number of other policies were considered during the theoretical research (see also section 3.2.1.3), and three of those are presented in this section in more detail.

3.4.1 FORMAL METHODS: VERIFICATION, VALIDATION AND FAULT REMOVAL

The first policy described here is based on the elaboration of the UML model of the system under development with fully formal models. Formal Description Techniques (FDTs), or

formal languages, are notations and descriptions of the system with a sound basis in mathematics. They utilise mathematical concepts and notations to precisely define theories and models of system structure, functionality and behaviour. Examples are Vienna Development Method (VDM) [Jone90], Z [ZArch], LOTOS [LOTOS] and others.

Formal languages capture system functionality with accuracy, as compared to semi formal languages such as UML, which show structure only. Benefits of such a rigorous formalism are multi-fold: precision, abstraction, clarity and conciseness [Barr93]. All of these benefits allow the main feature of the formal languages to be exploited - that of manipulability. Formal methods allow different levels of manipulation of the system under development throughout the lifecycle.

First, some formal languages (like Z) can be used for rigorous high-level specification and requirements analysis. Before indulging in system design, specifications can be checked for global coherence and consistency - *i.e.*, validation can be performed. Formal methods at this stage provide mechanisms for performing validation and derivation of properties using rigorous mathematical proofs. Proofs are conducted to confirm the required properties of the system. Many fundamental incoherences and bugs in system specification can thus be removed early in the lifecycle, preventing them to propagate further through the design. If bugs are not removed at this early stage, they can cause catastrophic faults culminating in the complete loss of service.

Next, formal languages such as VDM or LOTOS can be applied to the development process, using a set of rules - a design calculus - that allows stepwise refinement of the operations and data structures in the specification to an executable program. By using rigorous proofs, validation is possible at all steps. Also, verification, in the sense of formal checks of consistency between successive development stages, is possible. Some formal methods, such as LOTOS, also allow generation and testing of prototypes at all stages of system development, as well as inclusion of rigorous timing conditions.

Finally, formal methods allow, at the most rigorous level, verification of the system implementation versus the formal specification of the system (which can be understood as a conformance check). Verification is a formal proof of structural and behavioural properties expected from the implementation - a proof of correctness of the implementation.

Thus, in the context of the predictive integrity policy, formal methods can be used as an integrity-focused design recommendation to increase confidence in the correctness, completeness and overall integrity of the system under development. They can prevent fault

occurrence and fault introduction during system construction, and hence can be used for reduction of risk. Formal methods are essentially predictive, fault-avoidance techniques, enabling early error detection and fault removal in the form of re-design actions. In this form, formal methods do not allow any direct measurement of the relevant integrity attributes - they are only used for documentation, step-wise refinement and as fault removal mechanisms.

Formal methods are not one hundred percent reliable and effective. As all methods, they are subject to human error, they require excessive workforce training due to their mathematical complexity, and were shown to slow down the development process significantly. Thus, the use of formal methods incurs very high cost which might prove unnecessary when weighted against the improvements they might offer to the overall system integrity status (see section 3.3.1).

Although formal languages are being extensively standardised, and some initiatives [Bate96] [Pick97] were taken to integrate them with semi-formal description techniques so as to make them more approachable, there is not much evidence of their use in the industry [Barr93]. A rare field of application is within the safety-critical system design (systems such as space shuttle) where the extra cost of their use is acceptable [Bowe92]. On the other hand, it was shown that it is actually not possible to measure how much exactly the formal methods add to safety improvement [Bowe92], and that they are not particularly effective unless combined with testing [Hatt97] and fault tolerance metrics (discussed in section 3.4.3).

It seems that the above drawbacks of formal methods, with an emphasis on high complexity and cost, overweigh their benefits as an integrity policy. Also, in the context of the development of a distributed application such as a network management system, formal description techniques are not particularly effective since they are based on closed-world assumptions.

On the other hand, behaviour-oriented formal methods, such as Specification and Description Language (SDL) and Message Sequence Charts (MSCs), provide an ability to simulate and exercise behaviour of a distributed system during design in a more lightweight, effective, real-time fashion. These are considered as a separate integrity policy and discussed in the following section.

3.4.2 FORMAL METHODS AND BEHAVIOUR SIMULATIONS

A number of formal languages, such as Specification and Description Language (SDL) [Z.100] and Message Sequence Charts (MSCs) [Z.120], provide the description of solely

behavioural aspects of system operation. These languages allow specification and design of a concurrent, distributed system to be represented in terms of real-time asynchronous communication between independent distributed entities. Behaviour is a vital aspect of the operation of distributed telecommunications systems; it is also complex and difficult to describe.

SDL provides a clear and comprehensive behaviour description by representing the system as a set of communicating Finite State Machines (FSMs), or processes. SDL can be used throughout the development lifecycle: for specification, design, and as a model used as a basis for implementation and testing. Similarly to the formal methods described in the previous section, the system specification described in SDL can be validated, it can be used as a basis for stepwise refinement that can be subjected to verification of successive designs, and the final implementation can be verified against the specification. The main advantage of SDL as compared to the other Formal Description Techniques (FDTs) described in the previous section is that it can also be used for a full-blown simulation of system operation in order to extensively exercise system behaviour. Behaviour can be exercised so as to check whether the system stays within its specified behavioural envelope; to perform reachability analysis in order to see whether all specified behavioural paths are traversed; and to conduct deadlock and livelock detection. Moreover, it can be used for multi-system interaction simulation.

MSCs are essentially a complementary technique to SDL (and to some extent LOTOS), although they are a standardised FDT [Z.120] and can be used independently. MSCs are a trace language which in its graphical form provides a particularly intuitive representation of system runs (where one MSC depicts one scenario of system use), focusing on message interchange between communicating entities and their environment [Ekka95]. MSCs essentially depict communications between processes (FSMs) defined by SDL. They can be used for system specification and then for automatic generation of the SDL specification, and its simulation and consistency checks. As a stand-alone modelling technique, MSCs support top-down system development from specification and design through to implementation and selection of test cases [Grab93]. Since they focus on the message interchange between system components and depict scenarios of system use, they can be used for efficient real-time modelling of time-critical scenarios and system properties. Thus, they can prove useful in identifying time-related integrity hotspots in system operation.

As discussed above, the behaviour-oriented FDTs allow a wide range of integrity-critical aspects of system operation to be explored: they tackle liveness, time complexity and other issues. Thus, they have a strong potential as a predictive integrity policy. Simulation

techniques offered by these FDTs could act as a powerful integrity policy during system development: various aspects of the real-time system behaviour can be analysed. Faults and inconsistencies in system operation can be detected throughout the development and integrity control actions (see section 3.2.1.4) can be applied in the form of redesign activities. Also, a set of measures could be possibly developed concerning the time-critical operation, thus introducing a quantitative aspect in the policy. Moreover, the information gathered during simulation can be used to define the maintenance policies and pin-point aspects of system operation to be closely monitored.

The distributed, real-time model that can be developed using the behaviour-oriented FDTs is suitable for describing and analysing distributed telecommunications systems, such as network and service management systems. The drawback again is the scale of the management applications, where the large inter-domain applications would be hard and time-consuming to develop, thus proving the use of such techniques financially unjustifiable (see section 3.3.1).

The use of SDL during this research work was narrowed down to aid the specification and development of test suites designed to be applied over the inter-domain management system's interconnection points (see chapter 6).

3.4.3 SAFETY-CRITICAL APPROACH

Safety-critical analysis and design is an established research area in the production of safety critical systems, predominantly hardware [Redm91]. It is used throughout system development so as to remove faults, to ensure the implementation of a highly safe system, and to aid argumentation about system safety. Basic safety-critical approaches are both based on fault trees.

First approach is the FMECA (Failure Modes, Effects and Criticality Analysis) technique. This type of analysis starts with a known set of causes, and attempts to extrapolate toward their ultimate effect upon the system by building a failure propagation, or fault, tree. The failure modes (*i.e.*, mean-time-between-failures, MTBF) of individual components need to be known before the analysis is performed.

Fault Tree Analysis (FTA) approach complements the FMECA technique - it determines causes of hazardous events. Starting from the undesired top event (system failure) it derives one or more sets of potential causes. Model (fault tree) is thus built from top down in deductive fashion - *and/or* combinations are used to structure the event space down to the leaf events. Probabilistic methods are then used to determine the probability of the top,

catastrophic event occurring. As in FMECA, failure modes of components need to be known to determine the probabilities of leaf events and thus that of the catastrophic top event.

The Failure Propagation and Transformation Notation (FPTN) [Fene93] offers a unified framework for using FTA and FMECA. It provides a convenient way for representing and abstracting the notations and structures used by FTA and FMECA. It can be deployed as a separate, "safety modelling" tool which can be used as an aid to design and implementation of safe systems.

The use of the safety-critical approach as an integrity policy can be as follows. The internal attributes such as MTBF can be gathered, *i.e.*, measured, during system development and then subjected to the prediction algorithms derived from FMECA, FTA or FPTN. The predicted operation has the form of a set of catastrophic faults and their probabilities. If failures are many, and probabilities are high, the response needs to be applied to the system under development in form of redesign. If the failure scenarios and their probabilities are acceptable, there is no response applied and the gathered information is used to define the maintenance policies and the corresponding run-time integrity-preserving response. FMECA technique can be also used during maintenance (see section 3.2.3). In case of a failure, this technique can be used for building the failure tree so as to pin-point the cause of the catastrophic failure.

The major drawback of the safety-critical approach is that the mean-time-between-failures, mean-time-to-failure and similar reliability-related internal attributes, which are crucial for prediction, cannot be easily measured early in the software development lifecycle. It was also pointed out that it is notoriously difficult to measure failure rate of *software components* [Conn92]. These failure probabilities can be only measured when conducted in parallel with experimentation on the real-time, running system. Some authors [Butl95] argue that even in this case it is impossible to accurately quantify software reliability metrics. Similarly, in the telecommunications world, big players such as Bellcore do not recognise software reliability prediction as being a mature technique [Bell93] (quoted from [Male98]).

Thus, the safety-critical approach can be at best used for argumentation about safety of the systems built out of well-defined components with failure rates already measured and known early in the development lifecycle. For the relatively new software systems such as service provisioning and management systems, this approach can not be useful during design. Moreover, this approach is not fully effective for such large distributed systems where failures can be conspicuous in the sense that a failure of a component can be manifested as a continuous degradation in system operation and performance, rather then being a cause of a

full-blown integrity failure. However, as stated above, this candidate integrity policy can be used to a certain degree during maintenance (see section 3.2.3).

3.5 CHAPTER SUMMARY AND RESEARCH CONTRIBUTIONS

This chapter dealt with the closer definition/analysis of the concept of integrity, and with the development of a framework for management of integrity issues throughout the telecommunications system development lifecycle.

The first main research contribution of this chapter is **the novel understanding of the term integrity**. As the analysis of the state of the art (chapter 2, section 2.1) has pointed out, the current understanding and definition of integrity is dated and incomplete. Although we adapted an existing definition of integrity to formulate telecommunications system integrity as "*the ability of the system to retain its specified attributes in terms of performance and functionality*", we identified what these attributes - or issues - are. In our interpretation, integrity is represented as a set of lower-level attributes encompassing a variety of issues concerning system structure, functionality and behaviour. These attributes were defined taking into account the understanding of the term integrity as established in the telecommunications sector, but also considering the dependability and safety concepts (see chapter 2, section 2.1.3) originating in software and system science.

This number of attributes come together to build a set of concerns to be tackled while building a telecommunications system, integrating it in the environment and during the system operational lifetime. These attributes have various manifestations and degrees of importance depending on how and why the system is being constructed. Thus an integrity strategy, or methodology, must be formulated to guide the designers and implementers of systems; the integrity issues and integrity methodology should be incorporated into the engineering process. A second step in the theoretical work discussed in this chapter was concerned with developing such an integrity management methodology.

The integrity management methodology presented here aims to structure the integrity actions, and overcome the deficiencies and unify the expertise of both industrial and academic approaches that we discussed in the "state of the art" - chapter 2, section 2.2. The methodology presented here is envisaged to be applicable to any distributed telecommunications system, however here the focus is on the network and service management systems.

Three basic phases of the integrity methodology developed here are prediction, testing, and maintenance. Emphasis was on prediction, the system pre-launch integrity methodology

phase. We demonstrated how to integrate the integrity-related actions in the system development lifecycle. This is accomplished through iterative carrying out of the integrity analysis, design and implementation throughout the lifecycle. Integrity analysis identifies the integrity-related requirements: each of the integrity attributes can be located within different levels of integrity analysis. According to the integrity analysis, integrity design is specified, *i.e.*, integrity can be modelled into systems by defining integrity-preserving policies that should be deployed during the system development lifecycle at the relevant stages. The final bit of the predictive phase is how to actually apply these policies during system development - *i.e.* integrity implementation.

The integrity methodology is based on two key requirements: existence of a coherent system development approach, and measurement of integrity-related attributes. We presented a development approach based on ODP and UML, and discussed the issues related to the measurement of integrity attributes. Moreover, we reflected on the practical issues concerning the integrity methodology, in terms of the cost-benefit analysis and methodology's applicability in multi-domain environments. Finally, we considered a number of candidate integrity policies in the context of development of network and service management systems.

We see as the central contribution of this chapter the **development of the integrity methodology** - a framework for categorising and tackling integrity issues throughout the telecommunications system lifetime. The lack of such a framework is seen as a gap in the integrity research, as well as in the industrial practice (see chapter 2, sections 2.1.1, 2.1.2.2, 2.2). We do not envisage this methodology to be the ultimate answer to management of all integrity problems, but rather see it as an example of how to start categorising the integrity issues, analysing them and formulating techniques for preservation of integrity during system development. This methodology, although aimed at the systems to be developed in the future, also provides a framework for the assessment of existing systems. This can be accomplished by the thorough review of the design documents of the existing system, whereby the integrity requirements could be assessed and thus the system operator can be provided with an insight to the integrity level of the system.

Integrity issues are inevitably related to the system development approach and the supporting notation. A number of system development methodologies and notations exist (see state of the art in chapter 2, section 2.3.4) in the field of distributed systems and network and service management system development. Our integrity methodology is founded on the development approach based on ODP and UML. The author was one of the core creators of the **ODP-UML management system development approach** [Kand98], which is seen as a strong

and distinct research contribution. We believe that ODP is emerging as a structured framework for development of network and service management systems: apart from being a general framework for development of distributed systems, TINA systems are based on ODP, and recent advances to marry TMN and ODP concepts have been made [Pavl98]. Similarly, UML is becoming a de-facto standard for system development, including the management systems [Lew99a] [Lew99b]. Thus, we see the ODP-UML framework as a suitable base for integration of the integrity methodology, especially in the case of management systems. If an alternative system development approach is adopted, the integrity methodology presented here, being relatively lightweight, could possibly be adapted to other system development frameworks and notations. Another advantage of using a specific system development framework as a basis for the integrity methodology, is that of modelling. As discussed in chapter 2, section 2.2, a number of authors [Mont97][UCL94] prescribed modelling and analysis as a useful tool in understanding and managing integrity issues. In our methodology, these activities are not performed stand-alone, but as a part of the development process itself.

The fourth key contribution of this chapter is the **introduction of the concept of integrity policy**. Integrity policy is essentially an integrity-preserving action that can take the form of either *integrity-focused design recommendation*, specification of the *integrity-preserving mechanism* that needs to be implemented, or definition of a *testing strategy/approach.* Currently, very few pre-emptive techniques are used to tackle the integrity issues prior to system testing and integration (current techniques are discussed in chapter 2, section 2.2). In this chapter, we discussed a number of candidate integrity policies, including use of formal methods, behavioural simulators, fault-location and prevention techniques, and other.

In the remainder of this thesis, we focus on two distinct integrity policies, dealing with the development and testing of highly integral network and service management systems. The policies are the metrics policy for risk control early in the development lifecycle (chapter 5), and the policy for testing the inter-domain interconnection between autonomous management systems (chapter 6).

But first, in the next chapter we discuss the network and service management systems developed within two ACTS projects: TRUMPET and FlowThru. These projects were used as case studies for exploration of integrity issues, defined in this chapter, and provided a ground for further research.

4 RESEARCH PLATFORM: ACTS PROJECTS

This chapter presents an overview of the two European Commission - sponsored "Advanced Communications Technologies and Services" (ACTS) network and service management projects, TRUMPET and FlowThru, which were used as research platforms for the investigation of the integrity issues in distributed telecommunications systems, more specifically management systems. The focus is on TRUMPET, a project which the author was working on for more than two years and which was consequently the main research platform.

4.1 TRUMPET

The TRUMPET Technical Annex states that TRUMPET "aims to develop and verify, by means of trials in real environments, mechanisms which will ensure the required integrity of inter-domain access to TMN based management systems" [TRUMPET-TA]. Although initially concerned with integrity issues, the TRUMPET consortium narrowed down its research to the inter-domain security for management systems. As stated in chapter 3, section 3.1, security is a sub-attribute of integrity: a wide range of security threats such as unauthorised access to resources or data corruption and modification via intentional external attack, can jeopardise the correct and proper operation and thus the integrity of the system. On the other hand, the threat to integrity posed by systematic faults in systems operation and communications between system components was neglected by the consortium, which took the stand that all the potential integrity problems due to the inter-domain interactions can be solved by introducing security mechanisms.

The focus of the TRUMPET project thus was the development of the security architecture for the TMN X interfaces (for a short introduction to TMN refer to chapter 2, section 2.3.2). The beginning of the project saw the specification of the particular security policies for the TMN X interfaces [Mail96], and the development of the security architecture needed to support these policies [Ølne97]. After the initial security development, a need arose to develop a management system that would be a suitable and realistic platform to implement the security policies on. Thus, the service management architecture [Prnj97][Sack98] was developed, using an original development methodology [Prnj97][Kand98], to support the implementation of the TRUMPET security architecture. Moreover, a set of trials was established so as to evaluate both the management and security architectures in real operational environments.

The author's personal interests in the project focused at bringing back some of the pure integrity issues in network and service management into light. However, a number of contributions were made in direct line with the project, such as contributions to the security architecture, the management architecture, and the management systems development methodologies - these three aspects are further discussed in the following sections. The material in these sections is to some extent based on [Prnj97] [Prnj98b] [Sack98] and [Kand98] - material that author strongly contributed to.

4.1.1 SECURITY ARCHITECTURE

As stated above, the key goal of the project was to secure the inter-domain management communication between autonomous TMN domains - *i.e.*, the X interfaces. For the basic definition of the security concepts see section 3.1. The security services in the context of inter-domain management are:

- **Authentication**, referring to the mutual recognition of the communicating management parties.
- **Access control,** ensuring that an external party (manager) can access only a certain subset of management functionality/data of the system being secured, according to the contract.
- **Data integrity,** meaning that the management data (stored or in transit) must be protected against modification, insertion, and repetition.
- **Confidentiality,** meaning that management data content (stored or in transit) must not be disclosed to unauthorised parties.
- **Non-repudiation,** referring to the ability to resolve the dispute when one management party denies that the communication took place.

Two additional security services are those providing for the **security context negotiation** and **security audit and alarm.**

First, an in-depth security analysis [Mail96] was conducted, taking into account the various types of interactions between the autonomous TMN domains: X interface between two Public Network Operators (PNOs) or two Value Added Service Providers (VASPs), the X interface between a PNO and a VASP, and the interface between the customer premises and a VASP or a PNO. This analysis resulted in the definition of the suitable security policies, or so-called security Functional Classes (FCs) for different types of inter-domain interactions. A Functional Class is effectively a set of specified security services. Four kinds of FCs were identified.

The nil-security class, FC0, is defined simply for the sake of completeness: it does not specify any security services. FC1 provides for the integrity and confidentiality of stored management data. The security services provided include the authentication of the initiating management application, management resource and management association access control, and security alarm and audit. FC2 adds to FC1 the services ensuring the integrity and confidentiality of the data in transfer: security mechanisms need to be in place, ensuring that the data in transit cannot be modified, inserted, or disclosed. FC3 adds to FC2 requirements for advanced security services such as non-repudiation of origin and non-repudiation of delivery.

The next step was to develop the security architecture suitable for supporting the services defined through the FCs. The project considered two distinct cases: that of the open, and that of the closed OSI-based TMN management platform. The open management platform is a platform where the lower layers of the supporting OSI stack are open to manipulation. In the closed management platform, only the interface to the CMIS [X.710] can be accessed. The commercial management platforms are typically closed-stack.

Initially, both cases were considered. Some security services, namely data integrity, data confidentiality, non-repudiation, and security negotiations can be entirely provided only by adding extra security functionality to the supporting OSI stack. In the case of the open management platform, Transport Layer Security Protocol (TLSP) [X.274] (OSI layer 4) was recommended for provision of management data integrity and confidentiality in transit. Moreover, the Security Exchange Service Element (SESE) (OSI layer 7) [X.831][X.832] was recommended for security context negotiations - *i.e.*, for the establishment of session keys which are used for the calculation of the cryptographic seals and / or for encryption. The reference security architecture for the management platform with an open stack is shown in Figure 21 [Ølne97]. The security-specific components and interfaces are shown in dotted lines: the main component providing the security services is the Security Support Component (SSC). The full lines indicate the components/interfaces available in a typical TMN management platform.

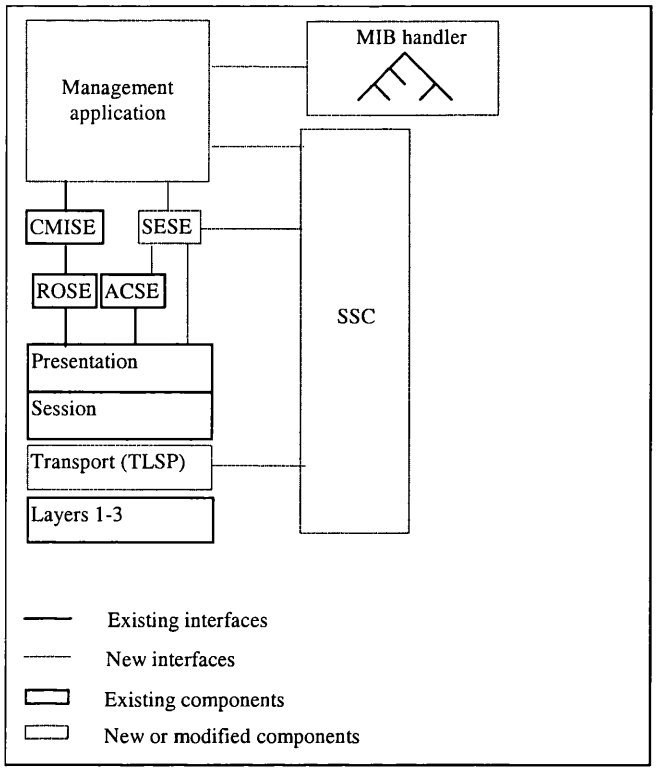


Figure 21 - Reference security architecture, open management platform [Ølne97]

Since TRUMPET opted for the use of a commercial TMN management platform - HP OpenView (HP-OV) - the architecture of Figure 21 had to be adapted to the closed management platform, where no modifications of the OSI stack can be done. Security transformations can thus be performed only on the application data. The reference security architecture for the closed management platform is shown in Figure 22. Figure 23 conveys the same information in more detail. The main security module within the SSC is the SMASC - the Secure Management Association Support Component, which performs security transformations on the application data and establishes the security context between the two communicating management parties. There are two basic requirements for the successful transfer of security data in the case of this architecture. First, the authentication field of the Association Control Service Element (ACSE) [X.227] must be supported, in order to establish the security context and for authentication. Second, the access control field of CMIP [X.711] operations must be supported to transfer the security-related information.

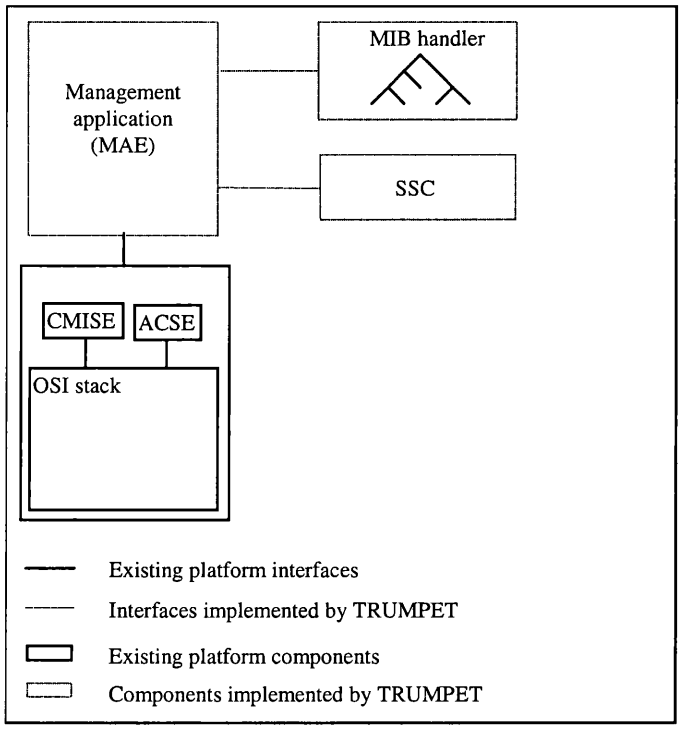


Figure 22 - Reference security architecture, closed management platform [Ølne97]

The security architecture developed for a commercial management platform cannot fully support some of the security services. Integrity and non-repudiation of data introduced by ACSE, CMIS and the lower layers cannot be guaranteed: only the application data is secured. According to this architecture, encryption for confidentiality takes place above CMIS, and this encrypted data must be inserted in one of the CMIS management operation fields: however, most of these fields do not support the encrypted data type, with the exception of the access control field - the approach which was not considered reasonable. Hence, this architecture does not fully support integrity and confidentiality of data in transit - thus effectively implementing a security functional class between FC1 and FC2.

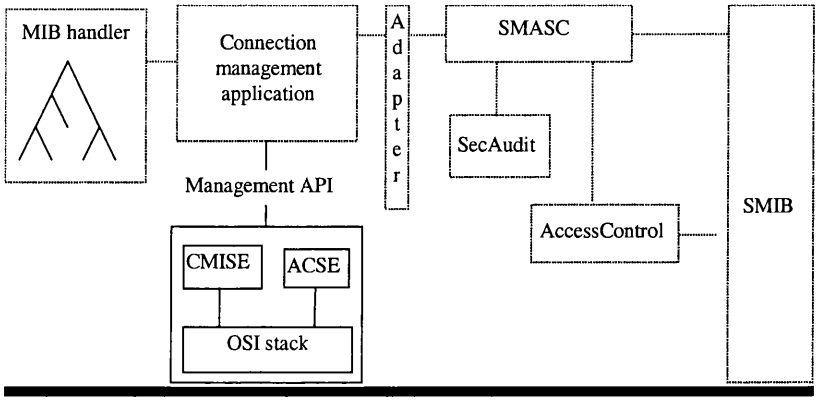


Figure 23 - Reference security architecture: detail [Ølne97]

4.1.2 MANAGEMENT ARCHITECTURE

4.1.2.1 OVERVIEW

The security architecture discussed above needs a realistic management architecture on which it can be deployed and scenarios through which its functionality can be demonstrated. There are three basic requirements on the management architecture.

First, in order to fully investigate the security requirements and to deploy the security policies (functional classes) defined, a suitable inter-domain scenario, in the framework of the ONP, needs to be defined. The architecture in this context should encompass a range of autonomous players, namely a Customer Premises Network (CPN), Value Added Service Provider (VASP) and a Public Network Operator (PNO), operating within distinct administrative domains. The interfaces between these players, aiming to be secured, must be clearly defined.

The second requirement on the management architecture is that it should be visibly functional, *i.e.* be able to support, in the operational environment, a set of user requirements focusing on the establishment and maintenance of the end-to-end broadband connections between two end users.

The third requirement is to develop a realistic architecture: the aim was to construct not only an administratively distributed environment, but also a technologically heterogeneous environment, focusing not solely on CMIS - based TMN implementations, but also on the other emerging management technologies, such as JAVA and CORBA. Although CMIP/S is the prevailing network management protocol technology, evolving price and versatility requirements of the new entrants into the market drive the need for exploration of alternative communications technologies. Both consumers and providers may be smaller and less willing to invest in large-scale high-end platforms. Also, with more competitors in the market, product differentiation becomes critical to a company's survival: to maintain a differentiated product, it is necessary for a company to be able to implement new services quickly.

These three aims were accomplished through the definition of the TRUMPET reference management architecture, shown in Figure 24. The architecture involves several administratively separate players (marked as boxes in bold): two (or more) PNOs, a VASP, and a number of customers at various sites - CPNs.

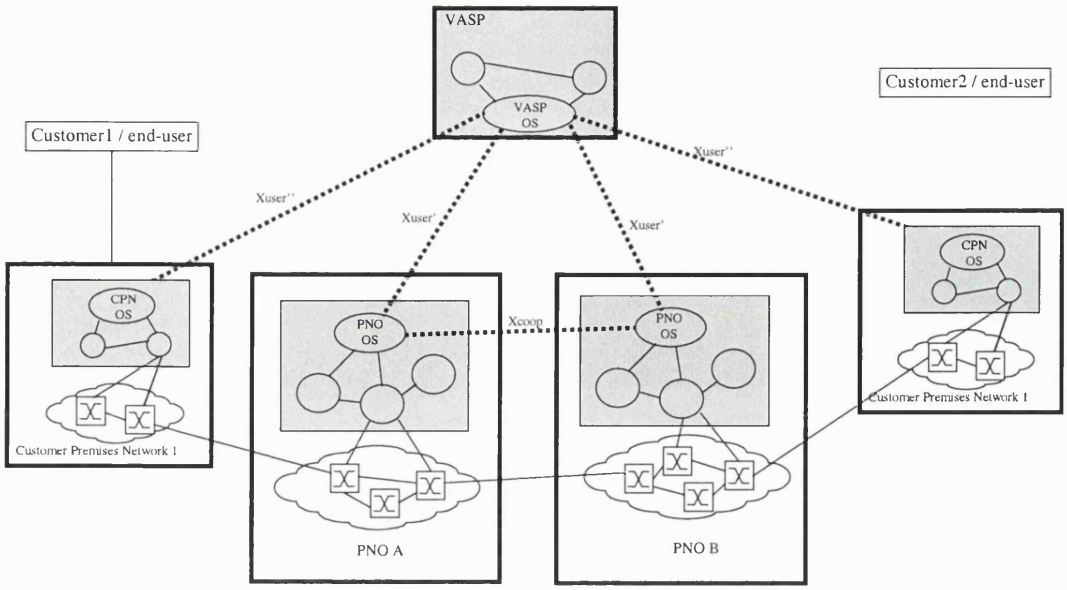


Figure 24 - Reference management architecture [Auta97]

The interfaces between the autonomous players are the TMN X interfaces [M.3010]: the physical realisations of the x reference points located between Operations Systems Functions (OSF) of different TMN domains. There are two types of TMN X interfaces: the Xcoop and the Xuser interface. The Xcoop interface is the interface between the management systems with a symmetric relationship: both management systems can take the role of the consumer and the provider. This is the interface between either two PNOs or two VASPs. Xuser is the interface between the two management parties whose communication is hierarchical: one party is providing the management service, while the other is consuming it. This consumer-provider relationship can take place either between a VASP and a PNO, CPN and a VASP, or CPN and a PNO. Further, as previously defined by the ACTS project MISA [Gali00], the Xuser' interface is the Xuser interface between a VASP and a PNO, and the Xuser'' interface is the Xuser interface between a CPN and a VASP or a PNO. These interfaces are depicted in Figure 24 as bold dashed lines between the autonomous domains. In the context of deploying the security policies, however, the particular interfaces used in the TRUMPET architecture (VASP-PNO, PNO-PNO and Customer-VASP) are essentially the same. Any of the parties involved in the interconnection over these interfaces may require any level of security (*i.e.*, any security FC as defined in section 4.1.1). Thus, the TRUMPET security policies apply to any kind of X interface. In practice, they were applied only to the Xuser interface between the VASP and the PNO.

The management systems of the above players form a TMN-based service management system for provision and maintenance of broadband Asynchronous Transfer Mode (ATM) network connections between two customers/end users. Each of the players has an independent management system (marked as a grey box in Figure 24) under its control. CPN is an actor that has a contract agreement with the VASP regarding the use of the service by

one or more authorised end-users. The VASP management system provides network connectivity to customers, on contractual basis, by utilising the resources of one or more Public Network Operators. The VASP is responsible for the service offered, and allows customers to create, modify and delete end-to-end connections, thus providing the Virtual Private Network (VPN) service to the customers. PNOs provide the VASP with the physical infrastructure and connectivity capabilities, by operating basic switching and transmission capabilities.

Within the TMN layered framework, TRUMPET management system is operating on the service management layer (where all the X interfaces are located), and to some extent the network management layer (Figure 25).

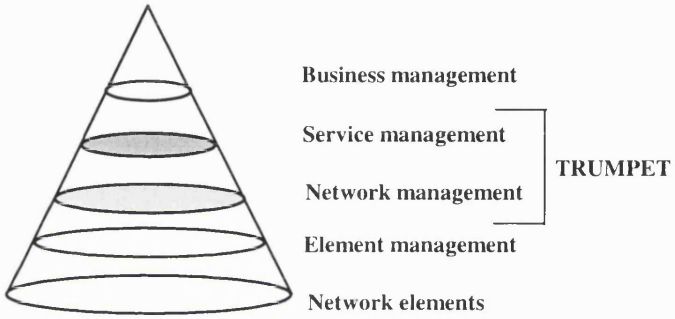


Figure 25 - TMN layers and TRUMPET

The customer is presented, by the VASP service, with a connection between the end-points of his various CPNs. VASP provides this end-to-end service by using the services from many independent PNOs. VASP is presented only with the end-to-end connection within the PNO domain (*i.e.*, PNO service view) by each respective PNO service management system. PNOs have the full knowledge of the structure of the network within their domains, which is controlled by their network layer management OSs (Figure 26).

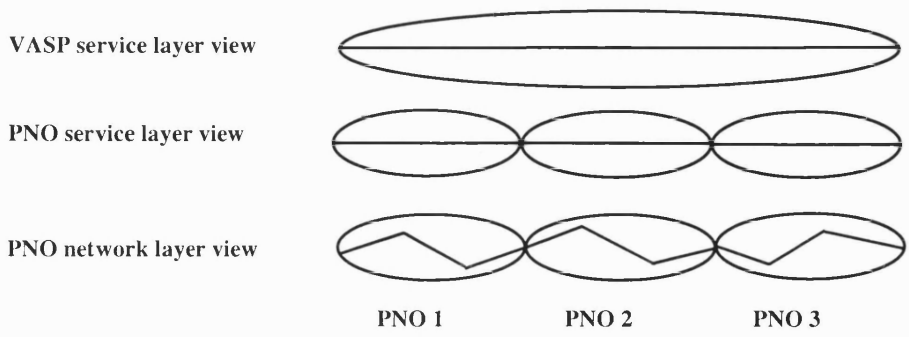


Figure 26 - TRUMPET service views

A number of technologies were used to implement the TRUMPET management architecture. The interface between the CPN and the VASP was developed using the JAVA based ORB -

Voyager [Obje97]. On the other hand, the interface between the VASP and the PNOs is CMIS based. With this mixture of technologies, it was also proven necessary that a light-weight CORBA gateway between the JAVA and the CMIS worlds should be deployed (effectively, JAVA binding to CORBA).

4.1.2.2 DEVELOPMENT METHODOLOGY

As discussed in the previous section, the TRUMPET service management and provisioning system is a complex, inter-domain, technologically heterogeneous system. Motivations for specifying a suitable development methodology for TRUMPET were many-fold. In general, designing and implementing complex distributed systems in large international consortia is complicated. The main issue in this context is that all the developers need to understand the scope of their work, the work of their partners, and the relationship between the two. Thus, a coherent methodology and notation scheme has to be adopted. This would also yield the in-depth documentation about system structure and design decisions. This documentation supports not only the maintenance of the system, but also captures the functionality and roles of the system that need to be presented to the users in the context of their business practice. Another important drive for a coherent development approach is the need to enable close analysis of the design to ensure, early in the development lifecycle, that the system is complete in meeting its requirements and consistent in its operation. This should render coherent and consistent designs and robust, highly engineered products.

The development of the TRUMPET system considered the criteria for the convergence (see chapter 2, section 2.3.5) of the TMN-based and distributed object models and methodologies, resulting in the "three-dimensional" approach, which took into account: the TMN architecture models, the ODP Viewpoint framework, and the UML notation schemes. The methodology was essentially use-case driven.

The only strong architectural requirement on the system was the full-featured CMIS-based TMN X interface between the VASP and the PNO, since the security policies were applied to this interface. The rest of the system did not need to adhere to any particular standards: however, the TMN architecture was central to the system development since the system is essentially targeted at the world of large-scale commercial public network management.

ODP framework was used to structure the development information, which was itself depicted through the UML diagrams. The fusion of ODP and UML was already presented in chapter 3, section 3.2.1.1.1. To briefly reiterate, ODP provides a general architectural framework that distributed systems aiming to operate in the multi-provider environment should conform to throughout their development. The bases of this architectural framework

are the five distinct viewpoints (enterprise, information, computational, engineering and technology), which allow different participants in system development to observe the system from a different perspective and from a different level of abstraction. ODP recommendations do not prescribe any particular notation to be used to describe the information captured in the viewpoints. However, since the description of the same component can exist in different viewpoints, there is a strong requirement that these specifications are consistent. Similarly, components in each viewpoint must be clearly identified and related to each other as required. Thus, it is favourable to use one single language for all the viewpoints. In chapter 3, section 3.2.1.1.1, we depicted how UML can be used to describe the ODP enterprise, computational, information and to some degree engineering viewpoints. UML provides a set of diagrams, has rich semantics and defines, to some extent, how different kinds of diagrams should relate to each other. Thus, it offers a possibility for providing for consistency and coherence of ODP specifications [Kand98]. The usage of the UML in the ODP framework, and the required mappings, adopted in TRUMPET, are shown in Figure 9, page 49.

Thus, the TRUMPET development methodology [Prnj97] is based on this ODP-UML framework and is essentially use-case driven (similar to that of the RACE project PRISM [Berq96]), where the development of the system starts with defining the management services as use cases, on the basis of which the core system components and their functionalities are defined. The enterprise viewpoint is presented first, providing the high-level specifications, and defining the core players (CPN, VASP and the PNOs). Next, the high-level overview of the implementation - the technology viewpoint - is given, with respect to the target technologies to be used, software and hardware. Following the high-level design, information and computational viewpoints are reiterated for each component defined in the enterprise viewpoint so as to give the low-level, detailed designs. Finally, the engineering viewpoint is given which elaborates on the underlying infrastructure for communication of the components, and their distribution.

The enterprise viewpoint is described using the UML use case diagram that depicts the desired functionality of the system through scenarios of the management system use; and the high-level class diagram presenting the main actors as packages within autonomous domains. The TRUMPET system incorporates three domains (or enterprise objects in the ODP terminology): the CPN, the VASP, and the PNO. The PNO domain is further subdivided into PNO Service Layer and PNO Network Layer. These four entities were modelled as UML packages with interdependencies, using UML class diagram notation as shown in Figure 27.

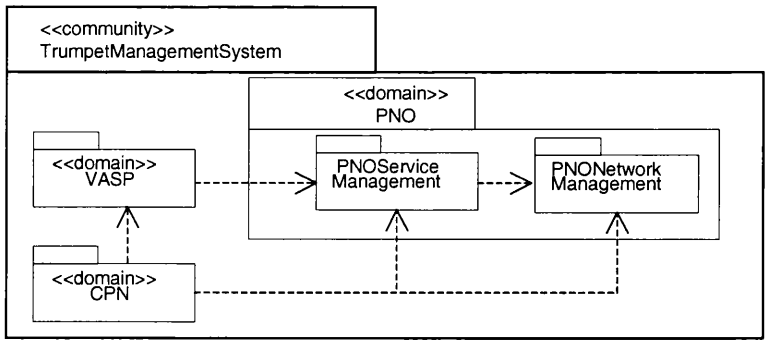


Figure 27 - TRUMPET class diagram enterprise package [Kand98]

The TRUMPET scenarios were specified using the UML use case diagram, depicting the actors, sets of use cases (ellipses) within a system, and associations between actors and use cases - illustrated in Figure 28. Note that the UML stereotype <<*community*>> is used to classify the high-level enterprise object *"TrumpetManagementSystem"* as community in the sense of ODP. Figure 28 depicts the functionality (or Management Functions) identified in the VASP management service as use cases, and the interaction of the different users with the use case package. As shown, there are six use cases. Customers/end users are capable of reserving end-to-end connections (of a given duration and desired Quality of Service, QoS), modifying them (changing duration, QoS, or both), and releasing, *i.e.*, deleting these connections. PNOs are capable of notifying the users via the VASP of connection activation, or notifying the users of the connection release due to a segment/link failure. The use cases, or scenarios, can be further elaborated on using the high-level sequence charts.

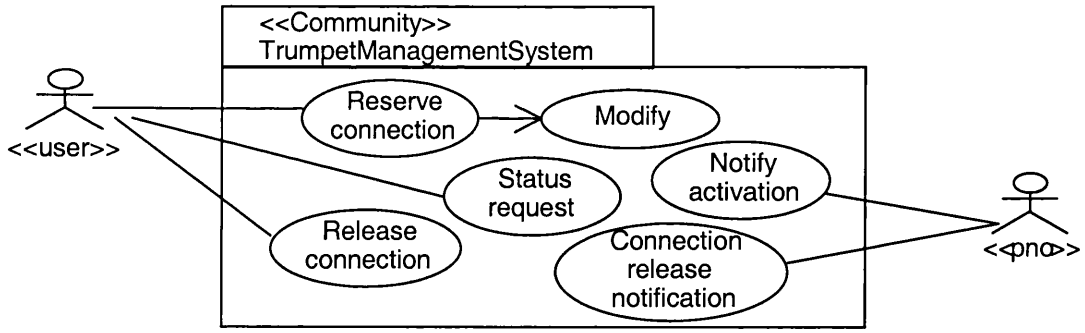


Figure 28 - Use case diagram [Prnj97]

The information object classes within the information viewpoint are described using the class diagrams, depicting the structure of object classes and their relationships (Figure 29). The class relationships include inheritance (parent-child relationships), associations (general relationships) and aggregations (containment relationships).

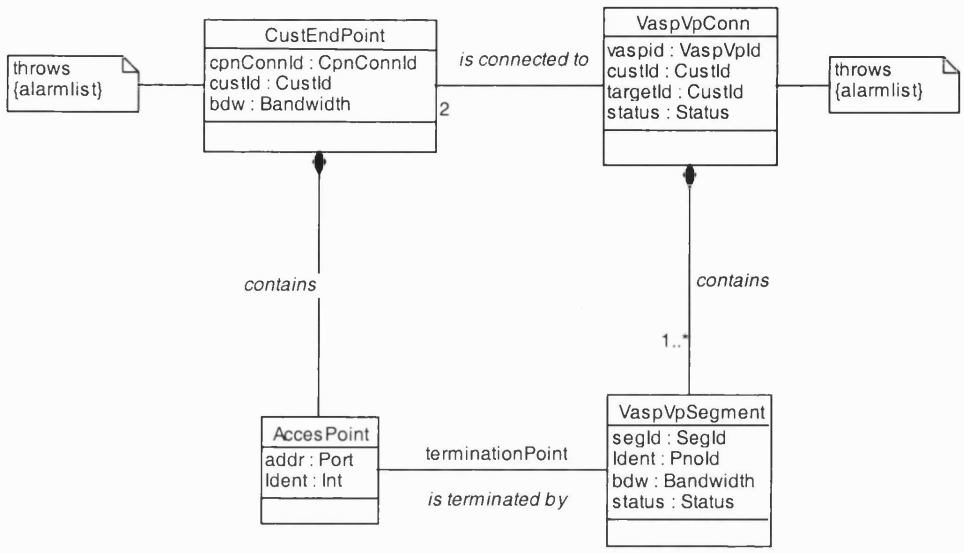


Figure 29 - Class diagram [Prnj97]

The computational viewpoint describes how the management functions, identified via enterprise use cases, are performed by the management system. Each management function is described in terms of computational objects and computational activities, the latter representing sequences of operations invoked on computational objects. As a starting point for the computational design, the components identified in the enterprise viewpoint can be mapped to computational objects which provide an abstract, course grain computational view of the management system. Each component can then be broken further down into a set of computational objects representing the detailed computational object model. At this level, the UML class diagrams were used to describe the structure of computational objects, their interrelationships and interfaces (Figure 30).

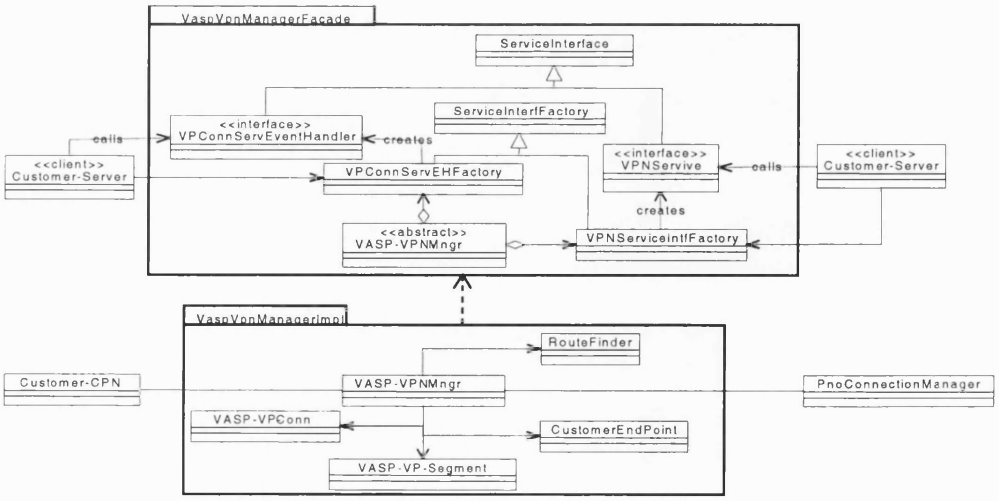


Figure 30 - Class diagram of computational objects [Kand98]

At the higher level of abstraction, class diagrams were used to describe the interfaces of the stand-alone object classes (Figure 31). Class diagrams describe computational objects'

external interfaces, which offer a set of services. Using these computational object-like diagrams as a basis, Interface Definition Language (IDL) files can be written easily.

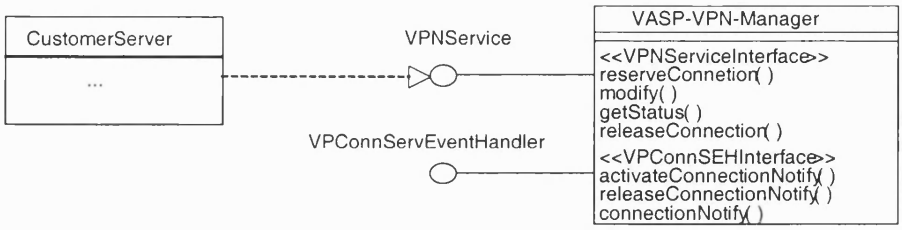


Figure 31 - Class diagram describing interfaces [Prnj97]

Next, the computational activities are described. Computational activities are the interactions between the computational objects in order to perform the management functions defined through use cases in the enterprise viewpoint. Interaction between computational objects is described in terms of an operation invocation initiated by a client object requesting an operation to be performed by a server object. Precedence rules are used to define the sequence of operations performed when an interaction takes place. To describe the computational objects' interactions UML collaboration diagrams (Figure 32) and sequence diagrams (Figure 33) are used.

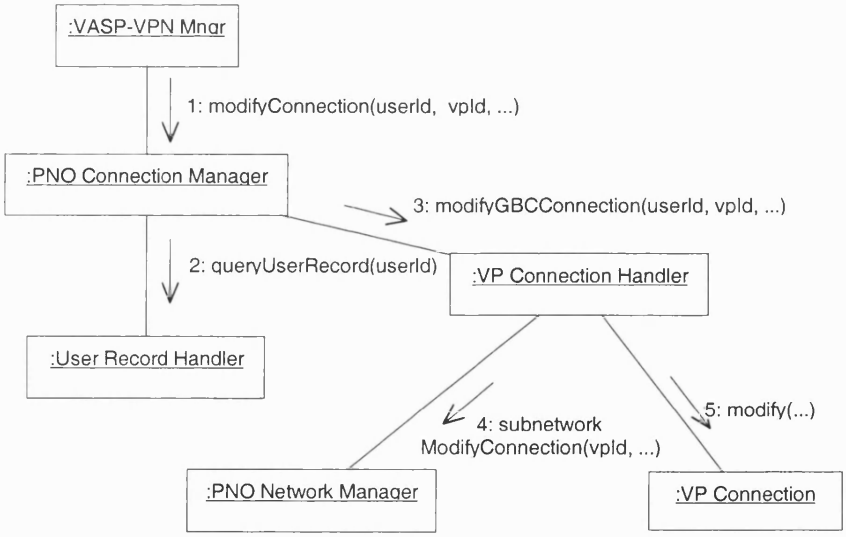


Figure 32 - Collaboration diagram [Prnj97]

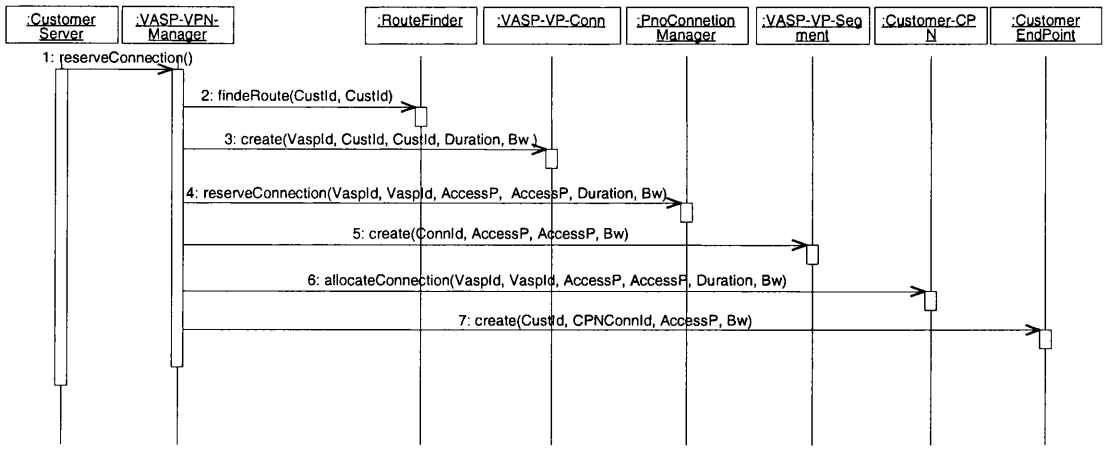


Figure 33 - Sequence diagram [Kand98]

The engineering viewpoint was elaborated through the component and deployment diagrams: the component diagram shows the organisations and dependencies among runtime modules, while the deployment diagram shows how components and objects are distributed and moved around the system. No UML diagram proved to be suitable to describe the technology viewpoint.

The combination of ODP and UML proved to be effective in many ways.

This approach inherently supports the object-oriented design process (both UML and ODP), and the design of reusable components which make up a distributed system (ODP). UML and ODP fit naturally, both being object-oriented in their essence, and their symbiosis proved to be efficient for modelling distributed, object-oriented systems, such as the TRUMPET management system. The outcome was a coherent design of a distributed management system, and the supporting developers', implementers' and users' documentation.

As discussed in chapter 3, section 3.2.1.1.1, UML, as a single and unifying viewpoint language, eases the migration between ODP viewpoints, enabling viewpoint consistency and thus the consistency, coherence and completeness of the design itself. Tracing of the components/classes through the design and mapping between the different component viewpoints is made possible [Kand98][Prnj97][Prnj98b]. Conversely, ODP proved to be an efficient way to manage the potential complexity of a wide range of UML diagrams.

Also, this approach had the power to embrace some TMN concepts [Sack98], and proved efficient in mapping and implementing some ODP functionalities in IDL.

Although the UML notation is an attractive choice for use in the design of a distributed system, it also has some drawbacks. Some of the ODP concepts are not directly supported by

UML. In such situations, UML introduces the concept of stereotypes to provide for extensibility. In the example diagrams of this section, stereotypes were used extensively to map ODP concepts that did not have a direct counterpart in the UML notation, such as enterprise objects, communities, *etc.* Moreover, although the concept of an interface is part of UML, its description uses the same notation as for a class. Again a stereotype, *<<interface>>*, was used to differentiate between the class and the interface descriptions. This use of the same notation to express different concepts leads to a certain level of ambiguity. In a pictorial notation the core entities of a model (ODP) should have individual representations as to be readily distinguished from one another. Also, UML did not prove to have enough power to fully describe the ODP concept of the computational object. During the design, only the external interfaces provided by a component were specified, and concepts like binding rules and lifetime aspects were not included.

From the practical point of view, adopting the ODP-UML approach in TRUMPET proved to be an efficient development, documentation and collaboration tool. After the initial methodology was established (which did require the consortium consensus), the work assignment was agreed on and understood within an afternoon of discussions. After the labour division was made and the approach was agreed and understood by all the partners, the consortium undertook to design the system according to the approach defined. The approach adopted in the system development provided the developers with a clear documentation tool, where all the developers could reach a common understanding and use others' design documents and work together efficiently. The design was developed within the contractual deadline, the time amounting to three months. There were 15 individuals involved in producing the design document (150 pages) [Prnj97]. Considering the size of the TRUMPET system, the development efforts and the quality of the output can be judged as optimal. The documents produced in this phase were also extensively used by the implementers as well as the trials team.

The next section concludes the description of the TRUMPET project by giving the design and implementation details of the three principal domains in the TRUMPET management architecture.

4.1.2.3 DESIGN AND IMPLEMENTATION DETAILS OF THE DOMAINS

4.1.2.3.1 CPN

The design of the management system on the customer premises focuses not only on the basic interface to the VASP for the purpose of service provision, but also on the automation of the interactions between the VASP and other elements within the customers premises -

such as local network management, local database management of accounts or usage, *etc.* This requirement is based on the assumption that in realistic situations the manager at the customer site may have to manage a bulk of connections - an operation that is much easier to perform through a database operation rather than through a simple Graphical User Interface (GUI). The UML class diagram of the computational objects necessary to support this is shown in Figure 34.

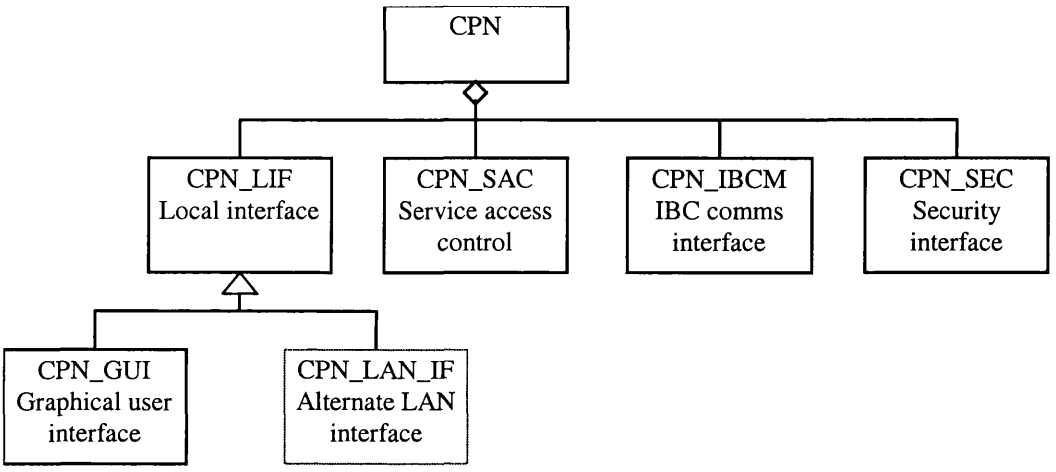


Figure 34 - CPN computational objects [Prnj97]

The technology chosen for the CPN management is JAVA. JAVA provides three critical facilities: Local Area Network (LAN) management interfaces (SNMP) (some aspects discussed in [Yama97]), distributed data transport, and versatile user interface capabilities using its built-in GUI libraries or the WWW. The CPN equipment essentially consists of a JAVA Virtual Machine (JVM) containing an interface with the VASP and local displays for user interfaces and for displaying events generated at the service level from the VASP and of relevance to the particular customer.

4.1.2.3.2 VASP

VASP's main role is to support the provision and maintenance of the number of end-to-end connections for a number of customers, by using the resources of one or more PNOs. This is achieved by designing the three main components of the VASP: Customer Server, Control Server (or VASP-VPN-Manager), and the VASP Management Information Base (MIB) - these are depicted in Figure 35.

The Customer Server provides for customer access to the VASP, and the Control Server (VASP-VPN-Manager) provides for VASP access to the Public Network Operators. The MIB-like component supports the required data models. As discussed above, the CPN is JAVA-based, and the PNO management system is CMIS-based TMN. Thus, the MIB-like component needs to support an information model that maintains interactions both with the

CPN and the PNO. This is achieved by constructing a management object model in JAVA, based on the TMN principles, which effectively provides a standard interface both to the PNOs and the CPN as well as giving a uniform information model across several players.

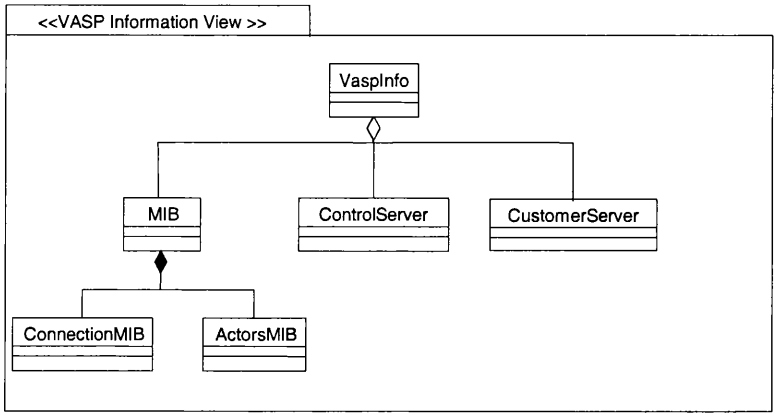


Figure 35 - VASP basic structure [Prnj97]

The VASP MIB contains managed objects that hold all the information about the resources that VASP needs to manage. It has the facility for selection of managed objects based on their properties and ensures that these objects are persistent. This is provided for by having the structure of the TMN-like MIB reflected by a Directory structure stored in a LDAP Directory Server, accessed through the Lightweight Directory Access Protocol, LDAP [RFC1777]. The LDAP entries hold the Distinguished Names (DNs) of the managed objects they represent, as well holding attributes for use in filtering and scoping operations. Persistence is provided for through making the managed objects Serializable (JAVA terminology) and storing the information in a database.

The structure of the MIB is as follows. The *Customer MIB* contains information pertaining to the VASP customers, their respective service profiles, and the terms of their subscriptions. A corresponding MIB exists for the PNOs whom the VASP is dealing with. These MIBs are rather static, in the sense that the information they contain is seldom updated. The *Connection MIB* contains information about all the connections that the VASP is currently supporting. Furthermore, its structure reflects the view that the VASP has of a connection, *i.e.*, a connection consisting of segments individually supported by a PNO. This MIB is being constantly updated (therefore dynamic) as requests for new Virtual Private Connections (VPCs) and change/release of the existing ones are received from the customers.

The communication between the CPN and the VASP is done through ObjectSpace's Voyager ORB package [Obje97]. First, the CPN establishes the association with the Customer Server, after which it can call CMIS-like operations (GET, SET, DELETE,

ACTION) on the managed objects selected. The main managed object class in the VASP domain is the VASP VPConnection. The objects of this class are created to represent an end to end connection between two Customer Premises Networks. Attributes contained represent connection information such as bandwidth, schedule and Quality of Service. For more details of the relationships between VASP managed object classes, see Figure 29, page 86.

Hence, by using the TMN interoperability notions, based on manager-agent interaction and the MIB concept, coupled with Voyager and LDAP technology, the TRUMPET management architecture provides integration and flexibility between the customer (CPN) and VASP domains.

VASP is fully implemented in JAVA: however, the interface between VASP and the PNO is CMIS-based. Thus, the management architecture also encompasses a lightweight CORBA gateway that interfaces the Control Server (VASP-VPN-Manager) to CMIS.

4.1.2.3.3 PNO

The interface between the VASP CORBA gateway and the PNO service management layer (the PNO_Connection_Manager) component is the Xuser interface initially defined by the ACTS project MISA collaboration [MISA-X]. The TRUMPET Xuser interface implementation is based on the MISA implementation, so as to ease interworking and joint trials. The information model for the Virtual Path (VP) connection management, supported at each PNO site, is shown in Figure 36.

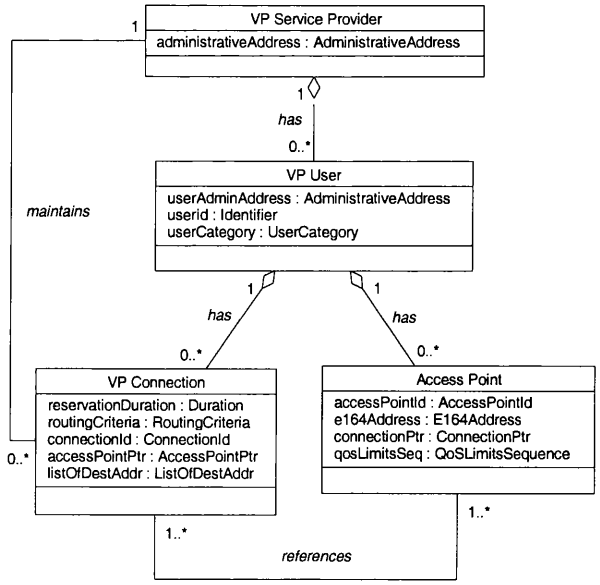


Figure 36 - PNO information model [Prnj97]

The VP Service Provider is the entity within the PNO domain, which is responsible for the provisioning of the VP connectivity service. This service is provided to many customers represented by instances of class VP User: in the TRUMPET case, the VP user is the VASP. The VP user has one or more VP connections, provided by the VP Service Provider. The VP user is also associated with a number of access points representing network access points of the public network which provide interfaces to adjacent network domains.

The network layer functionality in the PNO domains is not in full scope of TRUMPET: it is expected it would be provided at the project trial sites. In the trials, the implementation of the ATM-Forum M4 [ATM-M4] interface for ATM network management was used to provide for the element support.

The overall model of the main elements discussed in this and the VASP section is shown in Figure 37, this time depicted in the TMN fashion as a set of OSs and interfaces. The full manager-agent chain thus consists of the Customer Server - Control Server (VASP_VPN_Manager) - CORBA Gateway - Xuser OS (PNO_Connection_Manager) - M4 Gateway - NM Function - Sub-NM Function - Element Function.

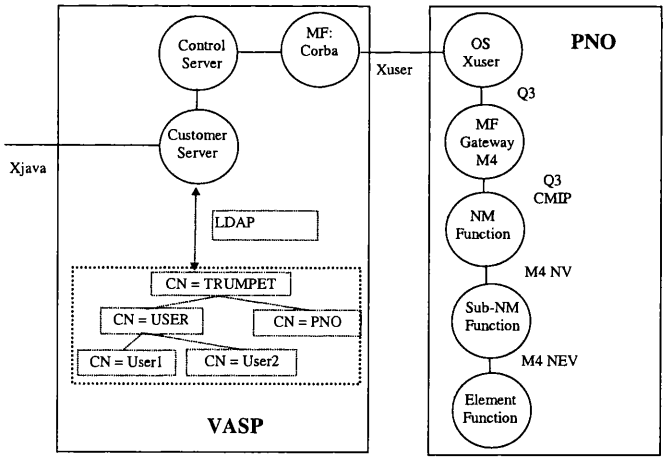


Figure 37 - VASP / PNO OSs and interfaces [Sack98]

4.2 FLOWTHRU

The FlowThru project focuses on the reuse and integration of a number of components developed by the other ACTS projects, including PROSPECT, REFORM, and VITAL, while MISA, TRUMPET and RETINA (An Industrial-quality TINA-compliant realtime DPE) [Dang96] projects were initially considered but later excluded from the scope of FlowThru. The aim is to demonstrate integrated multi-domain service and network management using a

number of re-used components, demonstrate integration technology at work, and specify development guidelines for reusable management components.

First, a development methodology, focused on building the reusable management components, and building systems from reusable components, was specified [Lew99a][Lew99b]. The reuse was incorporated in the development lifecycle through specification of reusable components not only on the basis of their design and software, but also by including component's analysis model. In such a manner, flexibility is achieved whereas the component is more self-contained and as such does not necessarily have to be a part of any distinct framework. The component reuse model is that of a façade [Jaco92]. Moreover, means of mapping between the façade and the ODP viewpoint models are specified. The notation used is UML.

Components specified in this manner provide a basis for developing a management system satisfying the target business process requirements. FlowThru identified three distinct trial business systems, based on the TeleManagement Forum's (formerly the Network Management Forum, NMF) Telecom Operations Map (TOM) process areas [NMF-TOM]: fulfilment business system, assurance business system, and accounting business system. The components constituting these systems, in the FlowThru scenario, form the management system responsible for provision and maintenance of the ATM connectivity services.

The fulfilment business system aims at provision of the services to the customers. It consists of subscription management component, configuration management component and network planning management component. The subscription management component is responsible for introduction of new services available to the customer, and withdrawal of these services. Moreover, it enables both the service provider administrators and the customers to manage the end-user access to the service capabilities. Configuration management deals with network provisioning: it performs the configuration of network elements according to customer demands. Network planning component performs Virtual Path (VP) and route planning, considering the anticipated network traffic and customer demands.

The assurance business system is the in-service system, dealing with the in-service problems that are encountered: it focuses on fault management, Service Level Agreement (SLA) violations, and the like. A number of components are encompassed, including service level accounting, TINA trouble ticketing, ATM accounting, and subscription management components.

The accounting business system focuses on the accounting processes for the connectivity provider and the third party service provider. The TINA-based components include access session, service session, subscription, accounting, ATM accounting and connection management components.

In the next section, we focus on the subscription management component, which was the subject of the integrity study in this thesis (chapter 5). For more details on the FlowThru development methodology see [Lew99a][Lew99b], while for the general FlowThru system overview and details of integration consult [Lew99c].

4.2.1 SUBSCRIPTION MANAGEMENT COMPONENT

The design of the FlowThru subscription management component is based on the subscription model specified in the TINA service architecture [TINA-SA]; and was further refined, implemented and reused a number of times in the ACTS project PROSPECT.

The subscription management component is located in the service provider's domain, and its basic role is to manage the subscription aspects of the service-level interactions between the service customer and the service provider. It is accessed by both the service provider's domain administrators and the customer's administrators.

This component manages the view of the services offered to the user: *i.e.*, the definition and the list of available services. Secondly, it manages the subscribers' profile: it deals with the creation and deletion of subscribers, with the details of the subscribers, and the details of subscriber's network sites and user groups. Finally, it manages the process of customer's subscription to the services offered: creation and deletion of subscriptions, management of subscription details, and authorisation of the end-users access to the services.

Thus, the subscription management component has the full knowledge of the classes of service provided, and the SLAs and service records that are part of the service offered. Additionally, it stores the information about the service subscribers (customers). Customers can create a new subscription contract, they can modify an existing contract, modify a Service Usage Group (SUG) associated with an existing contract, and cancel an existing contract.

The analysis-level modelling of the subscription management component was conducted using the FlowThru methodology [Lew99a][Lew99b]. Since this component was already implemented, and its design already specified, the analysis model was developed *post-facto*, and from scratch. However, the existing component design was not followed in detail; rather,

the generic requirements on this component were considered when building the analysis model.

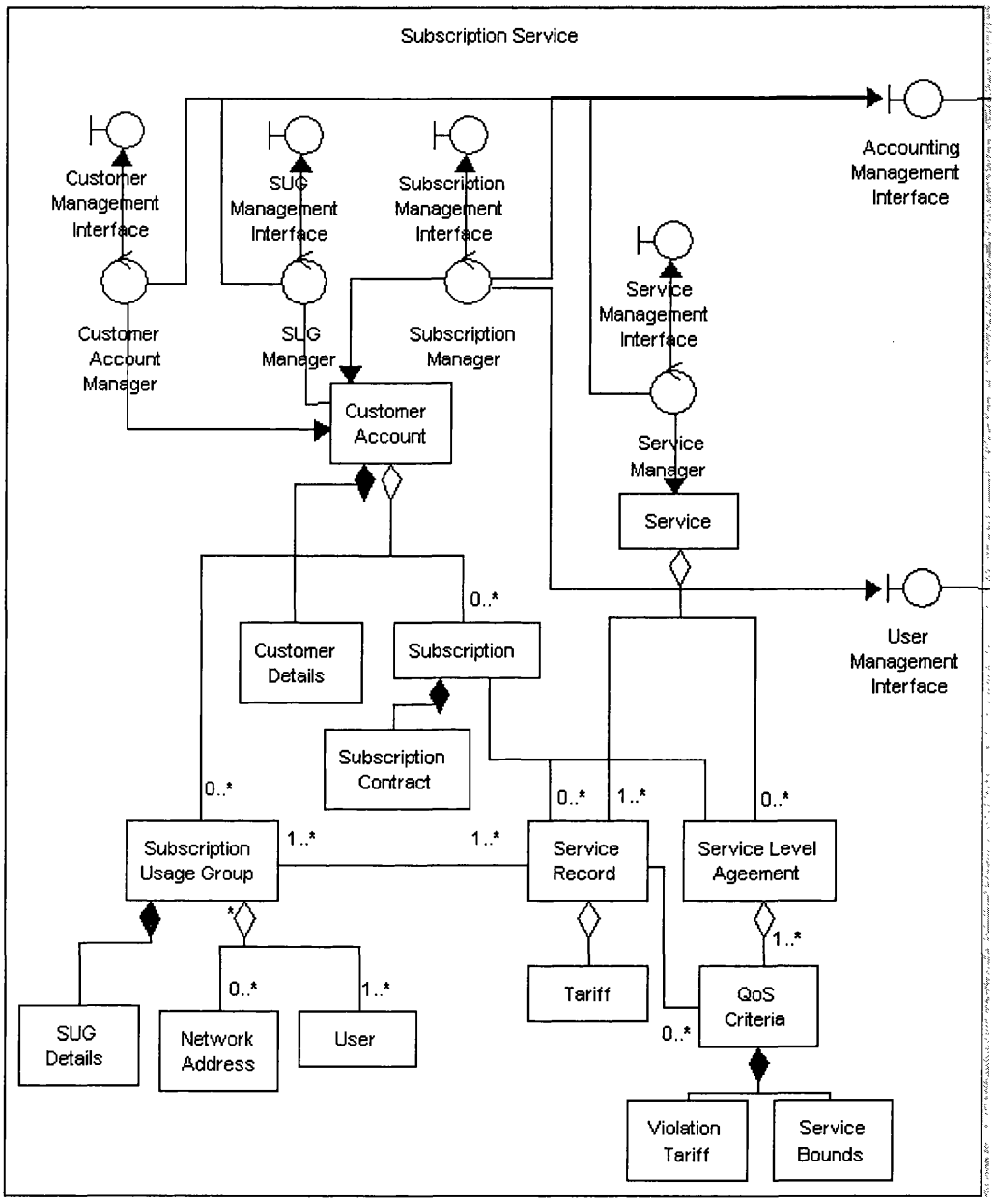


Figure 38 - The consolidated analysis class diagram [Flow-http]

The scenarios of the use of the subscription management component were modelled through the UML use case diagrams, depicting the interactions between the actors and the component. These diagrams were complemented with the UML class diagrams depicting the interrelationships between the analysis-level object classes: boundary objects (handling the communication between the component and the outside world), control objects (performing the use case specific behaviour) and entity objects (representing the information within the component) [Jaco92].

The consolidated analysis diagram is shown in Figure 38. This diagram does not depict the interactions between the Provider Administrator (PA) and the Customer Administrator (CA) with the subscription management component.

The analysis-level control and boundary objects correspond to the ODP computational objects (COs) and their interfaces, respectively. The main computational objects used to manage the services offered, the customer's subscriptions to these services, the customer profiles, and the service usage groups, are the Service Manager, Subscription Manager, Customer Account Manager and the Service Usage Group (SUG) Manager, respectively.

The most complex interactions within the component were modelled using the UML collaboration and sequence diagrams. An example collaboration diagram is shown in Figure 39.

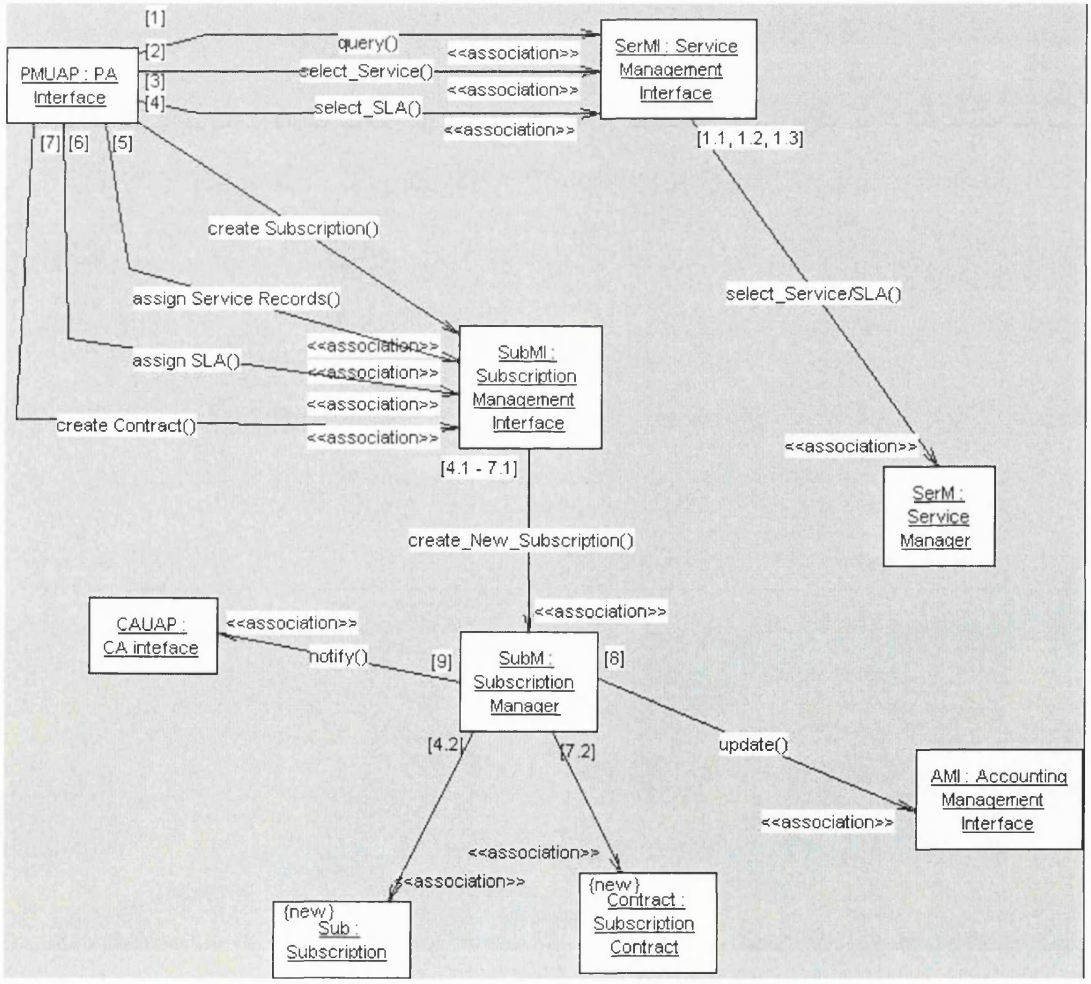


Figure 39 - Analysis collaboration diagram [Flow-http]

The design and implementation of the subscription management component already existed prior to component's use in the FlowThru scenario. Thus the design-level model of the component was simply re-documented using UML. The design model is comprised of the

"static" model and the "computational" model. The static model is a set of UML class diagrams representing in more detail the entity objects from the analysis model, and depicting their relationships. These entity objects are referred to as t-type objects in the FlowThru design. The computational model shows the core functional units - computational objects (mapped from the analysis model control objects), represented as packages which export the interface (i-type) objects (which in turn are mapped from the analysis model boundary objects). The entity objects are linked to functional units that manage and use them. The example class diagram is shown in Figure 40.

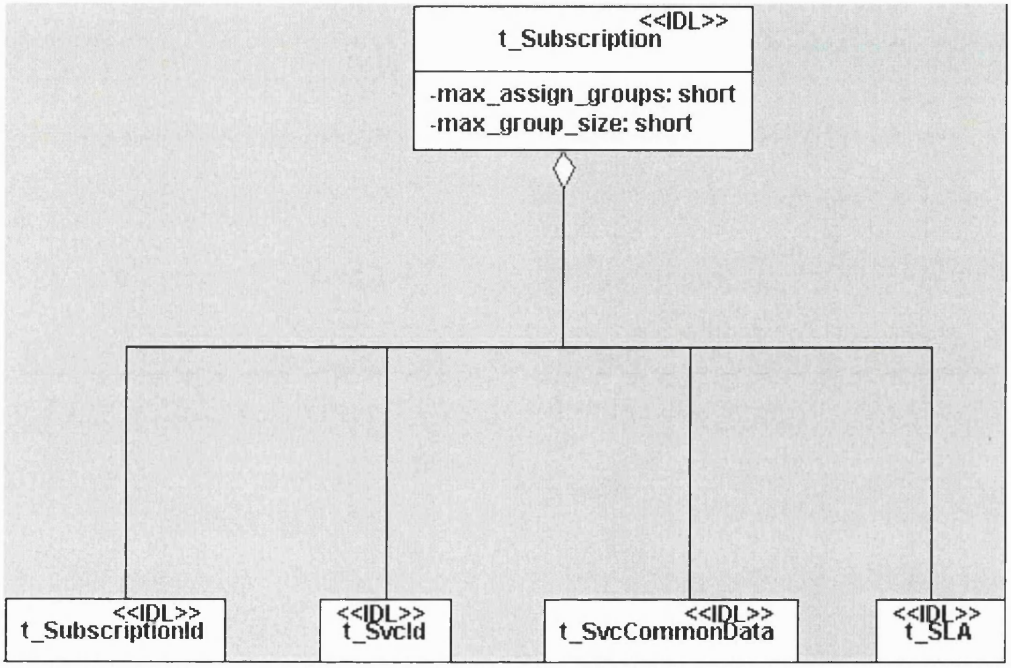


Figure 40 - Design class diagram [Flow-http]

While most of the analysis to design mappings were reported [Lew99a] to be one-to-one, in some cases the one-to-many (when an object is decomposed in the design phase) and many-to-one (when two or more objects in the analysis were identified, having similar functionality) mappings also occurred.

Although the FlowThru development methodology is strongly based on the façade concepts, we can claim, considering the above discussion, that the mapping between the FlowThru analysis/design model and the ODP viewpoint model is relatively straightforward (some of the mappings were discussed in [Lew99b]).

The analysis and design level entity objects correspond to ODP information objects. The FlowThru analysis level control objects, which can be mapped to the UML packages in the FlowThru design model, correspond to the ODP computational objects. These packages in the FlowThru design model export the interface objects, which originate from the FlowThru

analysis model boundary objects. These interface/boundary objects correspond to the ODP computational object interfaces. Thus, the FlowThru analysis and design models effectively cover both the ODP information and computational viewpoints.

The analysis and design diagrams of the FlowThru subscription management component presented in this section were taken from the publicly available version of the model [Flow-http].

4.3 CHAPTER SUMMARY AND RESEARCH CONTRIBUTIONS

This chapter gave an overview of the two ACTS projects that were used as a basis for the study of integrity issues. The TRUMPET project was the main research platform: the description of TRUMPET was thus given in full detail.

As we have seen, the TRUMPET system consists of two distinct entities: the security architecture and the management architecture. The security architecture was designed to secure the inter-domain TMN interactions, focusing on the Xuser interface between the two distinct administrative entities: the Value Added Service Provider (VASP) and the Public Network Operator (PNO). Mutual authentication, managed object access control, and integrity of data in transit are provided for. As such, the security architecture tackles a subset of integrity attributes (those related to security) defined in chapter 3, section 3.1.

The core aim of the management architecture is to support the deployment and demonstration of the security architecture. The service management architecture spans three separate administrative domains: Customer Premises Network, Value Added Service Provider and the Public Network Operator. The autonomous management systems of these players collaborate so as to provide and maintain the end-to-end ATM connections for the end-users.

Apart from just providing a platform for the application of the security architecture, a number of other questions were addressed through the construction of the TRUMPET service management and provisioning system.

Basic issue in the open network management and provisioning in the emerging market is that of the integration of the legacy TMN-based protocols and models with the emerging distributed object techniques which are aiming at standardised service provisioning. The future is more likely to see the customers moving from the heavy-weight TMN solutions to the more accessible CORBA and JAVA approaches. On the other hand, the major players will not be willing to discard their existing TMN systems. The TRUMPET management

architecture allows flexible customer access to the PNO CMIS-based management services via the third-party retailer (VASP). The customer has access to the services provided by the VASP through the managed object model developed in JAVA, which is compatible with the TMN architecture. In turn, the JAVA-based VASP accesses the CMIS-based PNO service management systems via the CORBA-based gateway.

Also, this management architecture was designed using a novel management system development approach. This approach aimed to unite the TMN and distributed object concepts, by specifying a use-case driven methodology, based on the ODP viewpoints, where the TMN architectural concepts are described using the UML notation. The approach proved to be flexible and effective.

The integrated TRUMPET system was validated in real operational environment during three trials [Prnj98b]. The first trial offered a platform for initial top-down system integration, and the TRUMPET service management system was deployed to successfully provide broadband connectivity between a number of medical sites, using the EXPERT test-bed in Basle. The second trial scenario concerned the provisioning, through the service provider, of connections between end-users across the network of an established provider - Scottish Telecom. The last trial took place within the "Network Society" event in the Eurescom premises in Sophia Antipolis, and similarly involved the establishment of ATM VPCs with specified Quality of Service parameters between the host sites. Some experiments also offered an opportunity to integrate the open interface of the TRUMPET VASP with a non-TRUMPET PNO: that developed by the MISA consortium.

The author's contributions to the concepts described above were manifold. Firstly, the contribution to the security architecture included discussions and contributions to design decisions, and document revisions. The contribution to the security aspect of the project further formed a basis for the development of the testing integrity policy for interconnected inter-domain management systems, discussed in chapter 6. Secondly, the author was one of the core creators of the ODP-UML management system development approach [Kand98], which was further refined and exploited in the context of the integrity management methodology - as discussed in chapter 3, section 3.2.1.1.1. In this context, the approach is seen as one of the key requirements forming a basis for consideration of integrity issues. Finally, the author contributed to the specification, design and implementation of the TRUMPET service management system [Prnj97].

The second project that was used as a research platform for the investigation of the integrity issues in network and service management was the ACTS project FlowThru. FlowThru

project dealt with the component reuse and integration issues in inter-domain network and service management scenarios. The issue of reuse strategy was addressed through the specification of a methodology for both the development of management systems out of reusable components, and for the development of the reusable components as such. Methodology is based on the specification of reusable components through not only their design model and software, but also the analysis model. The methodology is heavily based on UML, and exploits the concept of the façade.

A range of components from a number of other ACTS projects was used. In our description of the FlowThru project, we focused on the subscription management component, which was central in our further work concerning the investigation of integrity issues in network and service management. This component is responsible for introduction and withdrawal of new services available to the customer, and enables service provider administrators and the customers to manage the end-user access to the service capabilities.

We described the analysis and design UML models of the subscription management component. The analysis model of the component, addressing the component generic requirements, was specified using UML use case and class diagrams. The design model, based on the existing design, was reverse-documented through UML class diagrams depicting the structure and interrelationships between information objects, computational objects, and computational object interfaces. We then elaborated on the mappings between the FlowThru analysis and design models and the ODP viewpoints, concluding that the FlowThru analysis and design effectively specify the ODP information and computational viewpoints.

The background information concerning the FlowThru project is based on the FlowThru papers, deliverables and web presentations, as referred to, since the author did not participate directly in this project.

Both of the ACTS projects discussed here were used as a basis for the development of the integrity-metrics policy presented in chapter 5 (illustrated by three case studies involving these projects), while the TRUMPET project was the sole research platform for the development of the interconnection testing integrity policy discussed in chapter 6.

5 METRICS - RISK CONTROL

This chapter presents a predictive integrity-preserving policy to be applied during the development of distributed telecommunications systems, and its application in three case studies. The policy is based on the quantitative notions of system and class complexity and integrity, and is developed through exploration of the semi-formal model of the system under development and the adapted software measurement techniques (software metrics).

In section 5.1, we present this integrity-metrics policy, while in section 5.2 we demonstrate its applicability in three case studies concerning network and service management systems. One case study relates to the design of the TRUMPET management system, while the other two case studies relate to the FlowThru subscription management component: the analysis and the design model are considered separately. Section 5.3 concludes this chapter, and highlights the research contributions.

Some of the material in this chapter has been published in [Prnj96] and [Prnj99b].

5.1 METRICS: THE POLICY

This section introduces the integrity-preserving policy based on object-oriented (OO) software metrics, which yield complexity and coupling measurements of the system classes. These measurements are correlated with the integrity status of the system classes, and thus have the ability to pin-point potential risk areas in the telecommunications system design.

In section 5.1.1, we give the background concerning software metrics. In section 5.1.2, we argument the relationship between the complexity/coupling measures and the integrity/risk status. Moreover, we discuss the general applicability of this policy in the context of the integrity methodology presented in chapter 3. Section 5.1.3 elaborates on the policy by presenting our metric suite, and maps out the position of each metric in the ODP-UML framework (introduced in chapter 3, section 3.2.1.1.1) of the integrity methodology. Section 5.1.4 briefly summarises the theoretical work presented.

5.1.1 SOFTWARE METRICS BACKGROUND

Software measurement (for the fundamentals of measurement as such refer to chapter 3, section 3.2.1.1.2) is a branch of software science dealing with the measurement of various attributes of software. The main aim of software measurement is to "acquire control over software processes, products and resources" [Fent91]. Process measurement deals with measuring the attributes of any software activity that has a time factor, such as the software

analysis phase, design phase, implementation phase, *etc.* Product measurement focuses on measuring the attributes of the outputs of the processes: the actual software itself, deliverables/documentation *etc.* Resource measurement deals with the measurement of the inputs to processes, such as personnel, materials, tools, *etc.* Software measurement can be used for assessment (*e.g.*, amount of money spent during the project) or for prediction (*e.g.*, measurement of resources/personnel to predict software project duration).

As already discussed in chapter 3, section 3.2.1.1.2, the software attributes to be measured can also be grouped into two distinct sets: internal and external attributes. Internal attributes are measured only in terms of the actual entity under observation, and they are measured directly, *i.e.*, independently [Fent94]. Examples are number of bugs, time, or effort that has been spent during the software project. The external attributes are measured in terms of how the entity relates to its environment, and they are measured indirectly - *i.e.*, measures of other attributes must exist so as to obtain the measure of a particular external attribute. Examples of the external attributes are cost effectiveness, productivity, usability, *etc.* Through analogy, the internal attributes are measured in the context of assessment, and external through prediction. Note that integrity, being a complex attribute (see chapter 3, section 3.1), is also an external one.

Although a considerable amount of research was done in the field, software measurement is rarely applied in the industry. As reported in [Your96], only between 1 and 2 percent of software organisations are on the maturity level where they actually use metrics in the development process. Usually, metrics applications in the industry deal with process prediction, and strongly focus on cost, productivity and effort estimation [Well94].

Process predictions are performed early in the development lifecycle, and in this context the predictions focus on the cost-benefit feasibility analysis of the whole project. Later in the development lifecycle, cost prediction supports the project planning with respect to effort and duration. The cost is expressed in terms of duration (project elapsed time), or in terms of effort, which is normally calculated in Man-Months.

There are a number of software measurement techniques, or metrics, used in process predictions. The most established prediction model is the Constructive Cost Model (COCOMO) [Boeh81], which gives the prediction of effort as:

$$\text{Effort} = a * (\text{size})^b \qquad (4)$$

Where size is given in thousands of lines of code (KLOC) and a and b are parameters varying between 1 and 3.6, depending on the environment. COCOMO can be used during the requirements capture (referred to as the basic COCOMO); when the major components are already defined (referred to as the intermediate COCOMO); and when the details of software modules are defined (referred to as the detailed COCOMO). The problem with this model is that the size (KLOC) is difficult to measure early in the lifecycle, especially if diagrammatic techniques are used for analysis and design. The two popular alternative measures for size are Albrecht's Function Points (FPs) [Albr79], and the DeMarco's BANG measure [DeMa82].

Albrecht's FPs measure can be obtained early in the lifecycle, and it is technology independent. Thus, it is widely accepted in the industry, and many cost estimation models have been built on it. FPs can be used to measure the size of the software, as a weighted sum of external inputs, external outputs, external inquires, external files and internal files. The drawback of this measure is that it requires full system specification to be available. Also, it is a subjective measure and as such cannot be fully automated and, likewise, it is not independent of the design method used.

The DeMarco BANG measure classifies systems as function-strong and data-strong. For function-strong systems, BANG is actually the number of bubbles in a Data Flow Diagram. For data strong systems, BANG is defined as the number of entities in an Entity Relationship Diagram. BANG was not fully validated, but it is relatively widespread.

As we have seen, there a number of software metrics measuring the size of software, from which, through a suitable model, the process predictions can be made (for overview see [Kitc85] and more recently [Garm96]). Another set of software metrics focuses on measuring the internal structure of software. These metrics aim to capture the software complexity: that of the software modules (classes) and their interdependencies.

A number of the pre-object-oriented complexity measures exist [Shep93]. The control-flow family (as discussed in [Zuse90]) of software metrics is based on the graph theory. The program is represented as a set of statements, which are represented as edges (e), and which are connected through the control flow between them (vertices - v). The typical control-flow measure is the McCabe's cyclomatic complexity [McCa76], given as a difference between edges and vertices, plus 2.

The data complexity is measured through a number of simple metrics [Fent91] such as number of variables, constants, *etc.* Information flow between modules (where a module is

effectively a continuous sequence of program statements, separately compileable) can be quantified through calculating the squared product of the fan-in and the fan-out of the modules. Fan in is the number of local flows terminating at the module, plus the number of data structures from which information is retrieved by the module. Fan out is the number of local flows that emanate from the module, plus the number of data structures that are updated by the module. Local flow means that one module invokes another. This metric is referred to as the IF4 metric [Henr81].

With the evolution of the object-oriented (OO) analysis and design, a number of new complexity metrics emerged. The old metrics are not applicable to the OO paradigm, where the data and algorithms are bound closely together in a class, and a software program is really a number of collaborating objects. In the context of this new paradigm, complexity is generally considered to involve a human factor. Thus, we refer to psychological or cognitive complexity, where complexity of a program or a class relates to how difficult it is for a programmer to comprehend the problem, or to successfully develop a class [Zuse90]. Complexity can then be loosely defined as "a characteristic of software that requires effort to design, understand, or code" [Hend96]. Since in this thesis we are not directly interested in programmer characteristics, we are left with the issue of the OO *structural complexity* as the main factor influencing the psychological complexity.

The OO metrics are presumed to be collectable early in the development lifecycle [Chid98] [Kami99], considering the analysis and design documents developed through a diagrammatic notation such as OMT [Rumb91] or UML [UML]. A number of OO metrics exist (for a slightly out-of-date overview see [Hend96]).

The sheer scale of the OO system can be assessed using the number of use cases [Mink97] and the number of packages [Mink97] metrics. The inheritance complexity is measured using the Depth of Inheritance Tree (DIT) [Chid91][Chid94] and the Number of Children (NOC) [Chid91][Chid94] metrics. The complexity of the inter-class relationships can be measured using the number of relationships [Lore94] metric. Stand-alone class complexity is assessed using the Weighted Methods per Class metric (WMC) [Chid91][Chid94], and the interface complexity metric [Hend96]. The interrelationship between classes can also be measured using the Coupling Between Objects (CBO) [Chid91][Chid94], Message-Passing Coupling (MPC) [Li93][Lore94] and Response For a Class (RFC) [Chid91][Chid94] metrics. Whitmire complexity metric [Whit97] quantifies the overall relationship complexity, including associations, aggregations, inheritance and message passing. The Lack of Cohesion of Methods (LCOM) [Chid91][Chid94] measures the amount of cohesion in a class. The DIT, NOC, CBO, RFC, WMC and LCOM are collectively known as CK

(Chidamber-Kemerer) metrics. These, and some other of the OO metrics mentioned above, are discussed in detail in section 5.1.3.

The CK-metrics were suggested, in the research arena, as a basis for prediction of external process attributes, such as productivity, re-work effort and design effort [Chid98]; testing effort and reuse [Chid94]; as well as maintenance effort [Li93]. It is worth noting that for [Li93], the maintenance effort data was gathered throughout 3 years to enable the development of a model linking the maintenance effort and the CK metric values. In these three studies, it was indicated that the metrics are effective for the assessment of these economic variables. As such, these metrics are generally considered as a managerial tool, envisaged to aid project managers in effort allocation and project planning. Also, in [Chid94] these metrics were suggested as a means to identify the design flaws and areas of re-design: however, no details were given.

Currently, the studies mentioned above are the key ones that dealt with the actual practical applications for the object-oriented metrics. Another family of studies [Bria98] [Kami99] [Basi96] dealt with another aspect of the OO metrics application: their relationship with fault-proneness. The most notable study is that of [Basi96], which demonstrated that the CK metrics are more successful in predicting fault-proneness of the classes then other existing metrics. The metrics counts were related through a mathematical model to the binary value of fault-proneness: the class was detected during testing as either with a fault, or not. The metric counts were collected from the final code, while the faults were recorded during testing. The data sample was the student C++ programs.

Generally, there are few reported studies dealing with empirical OO measurements. In [Chid98], three commercial systems, containing 45, 27 and 25 classes, were considered. The metrics data source in two case studies was the source code, while in the third case study metrics were collected from the design documentation. In [Chid94], two systems were assessed: a graphical user interface (GUI) of 634 classes originating from two C++ libraries; and a piece of Smalltalk software for VLSI circuits, consisting of 1459 classes. In [Basi96], eight medium sized information systems, developed in C++, were subjected to measurements. In [Bria98], eight systems assessed consisted of a total of 180 classes. [Li93] assessed two commercial software products developed in Classic-Ada. [Kirs99] collected measurements from student programs consisting of 15 JAVA classes. In all these studies, the CK metrics were collected directly from the code. Apart from one of the three studies presented in [Chid98], the only other reported study where it was attempted to collect the metrics from the analysis and design documents is [Cart96]. Here, a telecommunications system consisting of 32 C++ classes was assessed. However, most of the metrics proved to

be difficult to collect from the analysis and design documents without having access to the system implementation, with the exception of DIT and NOC. In [Kami99], use of metrics was suggested early in the development lifecycle; however, the source code of a mail delivery system consisting of 141 classes was used for metrics collection.

5.1.2 METRICS AND INTEGRITY: OVERVIEW OF THE POLICY

This section represents a step towards designing an integrity policy based on the OO metrics, and towards the realisation of its implementation. Thus, this section also positions this policy in the methodology framework presented in chapter 3.

Since the early years of software engineering [Cons79] through to the modern days of OO software engineering [Bern93][Riel96] an axiom was established stating that good internal structure of software implies good external attributes of software.

First, good software should have low coupling between classes (or modules, in the pre-OO terminology). Coupling is a measure of the degree of interdependence between classes. Two object classes are coupled if and only if at least one of them acts upon the other: A is said to act upon B if the history of B is influenced by A [Vess84]. History is a set of chronologically ordered states that an object goes through in time. Or, the alternative way to define coupling would be: "two classes are coupled if there is evidence that methods defined in one class use methods or instance variables defined in another class" [Chid94][3].

Second, the stand-alone object classes of a good piece of software should have high cohesion and low internal complexity. Cohesion is the extent to which the class (module) is geared towards performing a coherent task: "how tightly bound or related internal module elements are to one another" [Cons79]. Internal class complexity could be concerned with either class internal structure (such as complexity of its control flow) or the complexity of the class as seen from the outside: effectively, the complexity of its interface.

To summarise, good software design has low class complexity, low coupling between classes and high cohesion: this good internal structure of software implies good external attributes. In the context of OO metrics, this point was supported by relating the complexity/coupling measures to external attributes such as maintainability [Li93], fault-proneness [Basi96][Bria98], and reuse [Chid94], as discussed before.

[3] However, note that using instance variables in another class is generally bad programming practice.

Here we suggest that the OO metrics can be used as the integrity/risk indicators early in the telecommunications system development lifecycle (note that in *system* is considered to be a set of interacting distributed objects providing a certain service). Integrity is an external attribute, and also a complex one, influenced by and influencing a number of diverse attributes, as discussed in section 3.1. As such, integrity cannot be measured directly. Thus, we cannot establish a direct functional relationship between complexity/coupling measures and integrity, *i.e.* we cannot build a full-scale mathematical model (as already discussed in chapter 3, section 3.2.1.1.2). However, we can state that there is a positive correlation between complexity/coupling measures and the integrity status of the telecommunications system.

The OO metrics single out the most complex and coupled classes in the design.

The higher the class complexity, the more difficult it is to design, implement, test, and maintain, and more likely it is that it will be designed or implemented incorrectly. As such, the class is more likely to fail, and more risk it poses to the integral system operation. Ideally, the complexity levels of individual classes should be evened out, so as to avoid the risk being focused on few points of failure.

High level of coupling between system classes indicates a high level of interdependence: a change in one of the system classes will ripple through the system via the coupling paths. Similarly, a failure, or an integrity breach, may propagate through the system, effecting its integrity.

Thus, the higher the class complexity and coupling, the higher the risk it poses to system operation, and thus lower its integrity:

$$\text{class complexity} \sim \text{class risk} \sim 1/(\text{class integrity}) \quad (5)$$

$$\text{class coupling} \sim \text{class risk} \sim 1/(\text{class integrity}) \quad (6)$$

Singling out the most complex/coupled classes is supported by the widely accepted rule of thumb, which states that if the modules/classes are ordered according to the number of faults, the top 20% of the classes will contain 80% of faults (as pointed out in [Sidd94]). Or, as was suggested from the early days of software engineering, in the context unrelated to metrics, but still relevant: "complexity ... is one of the major causes of unreliable software" [Myer76].

In the framework of the integrity methodology presented in chapter 3, the typical **integrity analysis** (discussed in chapter 3, section 3.2.1.2) of a telecommunications system would identify the integrity requirement of minimising and evening out the class complexity and coupling in the design. This can be done on the unit, sub-system and system integrity levels, and considering the information and computational viewpoints of the system under development - the viewpoints where the class semantics and communication with other classes are defined. Thus, the integrity requirements classification diagram, presented in chapter 3, section 3.2.1.2, Figure 12, page 53, would in this case have the form as depicted in Figure 41.

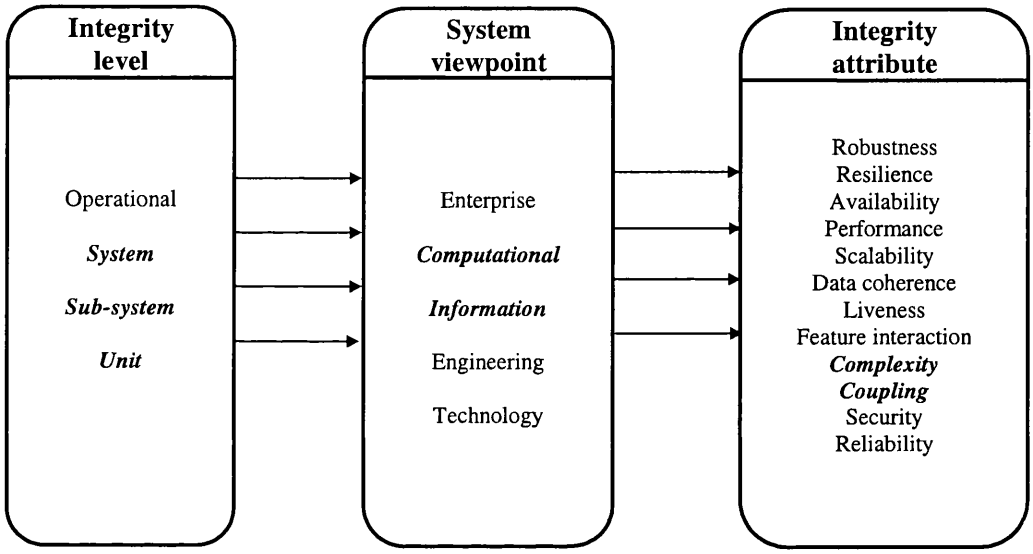


Figure 41 - Integrity requirements classification: complexity and coupling

Then the **integrity design** (discussed in section 3.2.1.3) would identify which metrics are to be used to assess the complexity and coupling levels of the classes in the system. Finally, the **integrity implementation** (discussed in section 3.2.1.4) would specify how to realise this policy. The implementation of this integrity-metrics policy is based on measurement and comparison of the complexity and coupling of system classes: these measurements indicate the high-risk areas of the design and imply the risk-reduction actions: redesign.

Referring to Figure 17, page 60, chapter 3, section 3.2.1.4, depicting the implementation of an integrity policy, the integrity-metrics policy implementation would have the form as shown in Figure 42. The measurement system is provided through OO metrics, which measure the complexity and coupling level of the classes in the system design. The system is specified in a semi-formal modelling notation such as OMT or UML. The complexity/coupling measurements, as mentioned before, are positively correlated to the integrity/risk status of the system classes. Thus, by pin-pointing the high-risk areas (hot-spots) in the design through a prediction algorithm which is effectively correlation, the

candidate redesign areas in the system under development are identified, and the response action, *i.e.* redesign, is applied. In the later stages of development, the response action can be also the extensive testing of the high-risk area in the implementation.

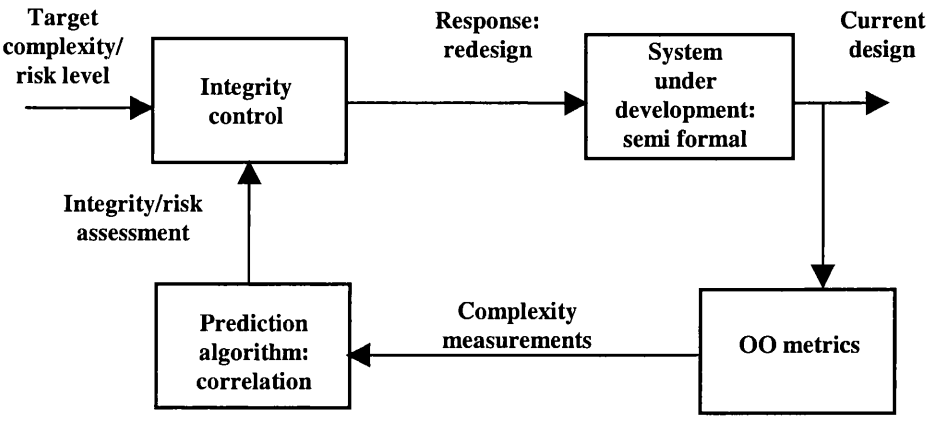


Figure 42 - Integrity-metrics policy implementation

Moreover, per class values of the metrics can be summed up to yield the system-level measurements. Thus, the overall complexity of alternative designs can be compared and hence the choice between alternative designs based on their integrity/risk levels can be made. Finally, all of the information gathered from measurement, prediction algorithms and response is analysed and documented so as to assess the final system integrity and risk status, and for the definition of maintenance (see chapter 3, section 3.2.3) policies.

Using the terminology defined in chapter 3, sections 3.2 and 3.2.1.3, this integrity policy belongs to the *prediction* phase of the integrity methodology and effectively gives rise to the *integrity-preserving design recommendations.*

5.1.3 DESIGNING THE POLICY: A METRIC SUITE

In the previous section, we gave the overview of the integrity policy based on metrics, and discussed its position in the integrity methodology, by illustrating what integrity analysis, design and implementation would consist of in the context of this policy.

This section presents the detailed design of the integrity policy based on the OO metrics. This activity involves the definition of the integrity-focused metric suite.

Our integrity metrics suite (published in [Prnj99b]) consists of seven distinct OO metrics: Depth of Inheritance Tree (DIT) [Chid91][Chid94], Number of Children (NOC) [Chid91][Chid94], Coupling Between Objects (CBO) [Chid91][Chid94], Message-Passing Coupling (MPC) [Li93], Response For a Class (RFC) [Chid91][Chid94], interface complexity metric [Hend96], and Whitmire complexity metric [Whit97]. All of the above

metrics are class-level metrics, and the measurements they yield are ranked on the interval scale [Hend96] (for the scale types see chapter 3, section 3.2.1.1.2).

DIT [Chid91][Chid94] is the depth of the inheritance tree, given by

$$\text{DIT} = \text{Inheritance level number; } (0 - N),\ N \geq 0. \tag{7}$$

In the case there is multiple inheritance, it is calculated as a maximum length from the node to the root of a tree. Deeper trees constitute greater design complexity, and the deeper a class is in the inheritance hierarchy, more methods it inherits and more complex it becomes.

NOC [Chid91][Chid94]is defined as the number of immediate sub-classes subordinated to a class in the class hierarchy:

$$\text{NOC} = \text{number of direct sub-classes; } (0 - N),\ N \geq 0 \tag{8}$$

The classes with high NOC count are more complex - they effect more classes.

CBO [Chid91][Chid94]is a count of a number of other classes that a class is coupled to. If a method in class A uses a method or an instance variable in class B, then A is coupled to B. CBO is independent of the number of references that A makes to B. There is some disagreement about how to calculate CBO. The original definition would yield that both classes A and B have the CBO count of 1 (*i.e.*, the directionality of arrows does not count). However, generally it is accepted that the calculation of CBO should be unidirectional: thus, class A would have a CBO of 1 and B a CBO of 0.

$$\text{CBO} = \text{number of collaborating classes; } (0 - N),\ N \geq 0 \tag{9}$$

As mentioned before, high coupling is undesirable: it makes a class highly dependent on other classes and thus more vulnerable to error propagation and less reliable.

MPC [Li93] is, in contrast to CBO, dependent on the number of references that class A of the above example makes to the class B. MPC is defined as

$$\text{MPC} = \text{number of send statements in a class; } (0 - N),\ N \geq 0 \tag{10}$$

where the number of send statements in a class is effectively the number of remote method invocations. Large MPC count implies large dependency on other classes. Classes with high MPC effectively have higher coupling and thus pose more risk to system operation.

RFC [Chid91][Chid94] is defined as a set of all methods that can be invoked in a response to a message received by an object of a class. In other words, this is the number of methods potentially available to the class:

$$\text{RFC} = \text{NLM} + \text{number of methods called by local methods; } (0 - N),\ N \geq 0 \qquad (11)$$

where NLM is the number of local methods. Large RFC indicates large complexity - tracing of the interdependencies becomes more difficult, and the coupling paths more intricate.

The interface complexity metric [Hend96] assesses the stand-alone complexity of the class. The interface can be specified as the set of services: queries (returning an object) and commands (not). A query can be perceived as a "get"-type operation, while command is a "set"/"do"-type operation. Interface complexity is given as the sum of weighted commands and queries, where the weight factor is the number of arguments required for the query/command. Hence, the equation is:

$$C_{total} = (c + \Sigma^{c}_{j=1} C_{arglist(j)}) + (q + \Sigma^{q}_{i=1} C_{arglist(i)}) \qquad (12)$$

where c is the number of commands, q number of queries, and $C_{arglist(j)}$ is the cardinality of the argument list for the j^{th} service. The larger the interface size, the more difficult it is to select and correctly use the service provided by the class.

The Whitmire complexity metric [Whit97] assesses the total class coupling within the design. It is a four-dimensional metric, where the four dimensions are sets of inheritance (set A_g), association (A_s), aggregation (A_a) and message passing (A_m) arrows related to the particular class. The magnitudes in each dimension are given by the cardinality of the corresponding set of arrows. The metric is additive, and thus the overall class coupling complexity is given by:

$$C_{total} = Card(A_g) + Card(A_s) + Card(A_a) + Card(A_m) \qquad (13)$$

where Card(X) denotes the cardinality of set X.

The set of OO metrics discussed here can be calculated from semi-formal analysis and design documents (if those are reasonably complete). DIT and NOC metrics can be calculated very early in the development lifecycle, considering the UML class diagrams depicting the inheritance hierarchy in the system. The CBO, MPC, RFC and Whitmire complexity can be calculated once the interrelationships between classes have been identified. To calculate these metrics, the UML class diagrams depicting associations and aggregations must be available, as well as the collaboration diagrams illustrating the message exchange between the collaborating objects. The interface complexity metric can be calculated once the stand-alone class interface has been specified, including the full set of parameters. As can be seen from the metrics formulae, all the metrics are relatively simple to calculate.

When placed in the ODP viewpoint framework (as discussed in chapter 3, section 3.2.1.1.1), the metric suite has the form as shown in Table 1. Here, each metric is presented in relation to the UML diagram it is calculated from, ODP viewpoint depicted by this UML diagram, and the integrity level (as discussed in section 3.2.1) at which the metric can be perceived.

Metric	**UML diagram**	**ODP viewpoint**	**Integrity level**
DIT	Class	Information	System
NOC	Class	Information	System
CBO	Class, Collaboration	Information, Computational	System/Sub-system
MPC	Class, Collaboration	Information, Computational	System/Sub-system
RFC	Class, Collaboration	Information, Computational	System/Sub-system/Unit
Interface	Class	Information, Computational	Unit
Whitmire	Class, Collaboration	Information, Computational	System/Sub-system

Table 1 - Metrics and system development mappings

The metrics making up our metric suite were chosen as a representative set for the assessment of the analysis/design complexity of the system under development. We believe these metrics effectively capture the complexity of the design, tackling both the stand-alone class complexity (in terms of the interface complexity), as well as different forms of inter-class coupling, ranging from inheritance coupling, through general relationship coupling such as association and aggregation, to message-passing oriented coupling which reflects the

amount of interaction on the detailed level. Moreover, this metric set is representative because it includes the key metrics suggested in the software measurement research arena. We omitted the stand-alone class cohesion measures (such as LCOM [Chid91] [Chid94]), since those strongly depend on the low-level class internal detail, available only through code-level information such as details of specification of procedures (methods) and the interdependencies between them [Biem98]. As such, these measures do not prove useful when considering the analysis and design information, which we are proposing to measure.

5.1.4 SUMMARY

In the preceding sections, we presented a predictive integrity-preserving policy based on OO software metrics.

First, we reflected on the state of the art in software measurement, pointing out that its focus nowadays is the process prediction as related to project management.

Then, we suggested the OO metrics as the risk/integrity indicators early in the telecommunications system development lifecycle. We argumented the relationship between complexity/coupling measurements and the risk/integrity status of system classes: the highly complex/coupled classes have lower integrity status.

We then discussed the position of this integrity-metrics policy in the integrity methodology presented in chapter 3. Integrity analysis in this case identifies the target to minimise and even out complexity and coupling of the stand-alone classes, integrity design involves the definition of the metric suite used for complexity/coupling measurements, while the integrity implementation conceptually automates the control loop.

Next step was the definition of the metrics making up the suite: these are the seven distinct OO metrics. We also showed how each of the metrics relates to the UML diagram it is calculated from, ODP viewpoint depicted by this UML diagram, and the integrity level (as discussed in chapter 3, section 3.2.1) at which the metric can be perceived.

The development of this integrity-metrics policy is seen as one of the main contributions of the research work, and as such will be discussed in detail in the final conclusive discussion of this chapter.

To test the efficiency of this integrity-metrics policy, we applied it to three case studies: that of the TRUMPET service management system design, the FlowThru subscription

management component analysis model and the FlowThru subscription management component design model. These case studies are discussed in the following sections.

5.2 METRICS: CASE STUDIES

In order to illustrate the application of the proposed integrity-metrics policy, three case studies were undertaken. In section 5.2.1, we report on the assessment of the TRUMPET service provisioning and management system using our proposed metric suite. In section 5.2.2, the assessment of the FlowThru subscription management component is presented: both FlowThru analysis and design models were assessed, since these were developed independently (as discussed in chapter 4, section 4.2.1). Thus, effectively, three distinct analysis/design entities were assessed. For each experiment, we first elaborate on the approach taken to metrics collection in that particular study, followed by the details of the measurements, and ending with the brief summary of the results. Then, in section 5.2.3 we discuss in detail the results of the case studies, present the statistical analysis of the results, and reflect on the lessons learnt.

5.2.1 TRUMPET SYSTEM

5.2.1.1 APPROACH

The TRUMPET service provisioning and management system was presented in detail in chapter 4, section 4.1.2. As mentioned before, the design was conducted using the ODP-UML approach. The metrics data source - the design documentation [Prnj97] - consisted of 150 pages of text and UML diagrams, drawn manually. The design was developed by 15 collaborators over 3 months.

In what follows we assessed only the classes in the Value Added Service Provider (VASP) and the Public Network Operator (PNO) domains, since the documentation for the Customer Premises Network (CPN) management system was incomplete. The design of the VASP and the PNO domains consisted of 32 classes.

Each of the metrics forming the metric suite, apart from DIT and NOC, was applied within an ODP viewpoint and on relevant UML diagram(s) describing that viewpoint, as suggested before in Table 1, page 113. The designers of the TRUMPET system did not use inheritance at all. Thus, DIT and NOC metrics, measuring the inheritance complexity, are not applicable in this case. The metrics data collection was performed manually.

5.2.1.2 ASSESSMENT

The metric values of the classes are shown in Table 2. The table is sorted with respect to decreasing Whitmire complexity (last column).

Class number	Class	CBO	MPC	RFC	Interface	Whitmire
1	VASP_VPN_Manager	6	23	30	31	27
2	PNO_Conn_Manager	3	9	14	29	10
3	PNO_VP_Conn_Handler	3	9	14	21	9
4	PNO_atm_Subnetwork	1	6	12	6	8
6	PNO_Nw_Manager	2	7	17	20	7
5	VASP_Customer_Server	1	4	8	16	7
8	VASP_VP_Connection	2	2	11	14	6
7	Vasp_Top	0	0	0	0	6
9	VASP_Connection_MIB	0	0	0	0	4
10	VASP_Customer_MIB	0	0	0	0	4
11	VASP_MIB	0	0	0	0	4
12	VASP_PNO_MIB	0	0	0	0	3
19	PNO_VP_Conn_Manager	1	2	4	7	2
15	VASP_Customer_End_Point	0	0	3	9	2
17	PNO_VP_Conn	0	0	2	10	2
13	VASP_Actors_MIB	0	0	0	0	2
14	VP_User	0	0	0	0	2
16	PNO_VP_Service_Provider	0	0	0	0	2
18	PNO_ATM_subnetworkConnection	0	0	0	0	2
21	PNO_Access_Point	0	0	3	10	1
22	VASP_VP_Segment	0	0	3	10	1
20	VASP_Access_Point	0	0	0	0	1
23	ATM_NW_Access_Point	0	0	0	0	1
24	PNO_VP_User_Record_Handler	0	0	3	5	0
25	Route_Finder	0	0	1	3	0
26	CustID	0	0	0	0	0
27	CustServProf	0	0	0	0	0
28	Pnoid	0	0	0	0	0
29	Pnoserprof	0	0	0	0	0
30	Pnostatus	0	0	0	0	0
31	Sec_Profile	0	0	0	0	0
32	Conn_Profile	0	0	0	0	0

Table 2 - TRUMPET metrics values

Figure 43 graphically depicts the metrics distributions per class. On average, the most complex classes are the VASP_VPN_Manager and the PNO_Conn_Manager. These two classes are the main computational object classes operating on the X interface between the VASP and the PNO domains. The two classes that follow are the PNO_VP_Conn_Handler, and the PNO_Nw_Manager, the other two computational object classes in the PNO domain, on the service and network levels, respectively. Next is the VASP_Customer_Server, the computational object class in the VASP domain which is operating at the interface with the CPN domain. Following these is a set of purely information object classes representing the key data entities in both the VASP and the PNO domains.

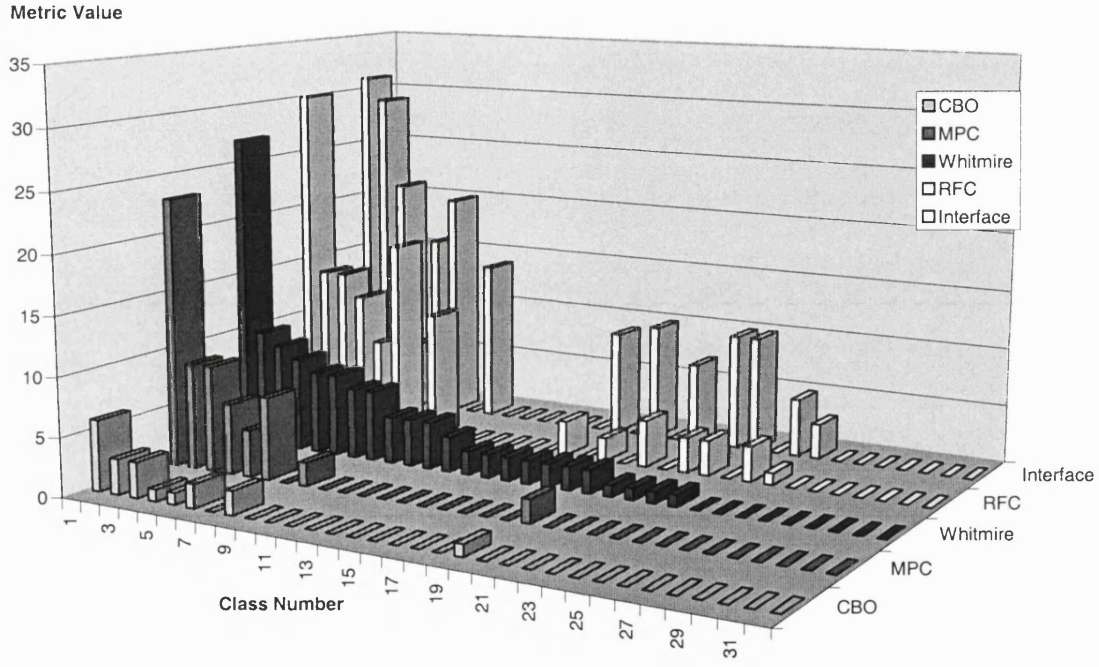


Figure 43 - Metrics distributions per class (TRUMPET)

In what follows we discuss the complexity measurements as per metric. For each metric, we present the basic summary statistics (mean, median, standard deviation, minimum, maximum), the histogram of the distribution of metric values, and the boxplot of metrics values. The boxplot essentially depicts the centre and variation of the data set, effectively marking the outliers. The boxplot is constructed from the three summary statistics: *median* (value *m* for which half the values of data set are smaller then *m* and half are bigger), the *upper fourth* (value *u* which is the median of values larger than *m*), and *lower fourth* (value *l* which is the median of values smaller than *m*). Values *m*, *u* and *l* split the data into quarters. The box length is d = u - l, and upper tail value is u + 1.5 d. The outliers are marked with a star. Thus, the boxplot shows the skewness of data, by the position of the median in the box, and by the length of the tail. A more detailed statistical analysis over the whole set of metrics is given in section 5.2.3.1.

Table 3 gives the basic descriptive statistics for the Whitmire complexity measurements, and Figure 44 depicts the histogram of the distribution of the Whitmire complexity values. Figure 45 shows the boxplot of the complexity values: in this case, the median is off-set of the centre and the tail lengths are unequal (left being non-existent); the data set is strongly skewed to the left. The Whitmire complexity metric identifies the two X interface computational object classes as the most complex.

Table 4 gives the descriptive statistics for the CBO measurements, Figure 46 the histogram of the distribution of the CBO values, and Figure 47 the CBO boxplot. CBO measures appear distinctly low, indicating that the interconnection between classes is kept at a reasonable level. As seen from the histogram, most of the classes have the CBO count between 0 and 2, with only a few classes having higher CBO (up to 6). Highest CBO is exhibited by the X interface computational object classes in both domains, and the other two main computational object classes in the PNO domain.

Table 5 gives the descriptive statistics for the MPC measurements, Figure 48 the histogram of the distribution of the MPC values, and Figure 49 the MPC boxplot. As seen from the histogram, most of the classes have the MPC count between 0 and 2, with only a few classes having higher MPC (up to 23). MPC values follow the distribution of the Whitmire complexity and the CBO values, with both X interface computational classes in the two domains distinctly standing out. MPC counts of the information object classes are 0.

Table 6 gives the descriptive statistics for the RFC measurements, Figure 50 the histogram of the distribution of the RFC values, and Figure 51 the RFC boxplot. RFC follows previously discussed complexity measurements for the computational object classes: however, now the most important information object classes exhibit a complexity increase as compared to the previous measurements. This is an expected result, since RFC measures the methods available to the class, which even in the case of a moderately interacting class can be high due to the high number of methods within a class itself.

Table 7 gives the descriptive statistics for the interface complexity, Figure 52 the histogram of the distribution of the interface complexity values, and Figure 53 the interface complexity boxplot. Interface complexity follows the CBO and MPC for the computational object classes: however, now the most important information object classes exhibit a complexity increase as compared to the CBO and MPC measurements, the rationale for this following that of the RFC.

Mean	3.531
Median	2
Standard deviation	5.187
Minimum	0
Maximum	27

Table 3 - Whitmire complexity statistics (TRUMPET)

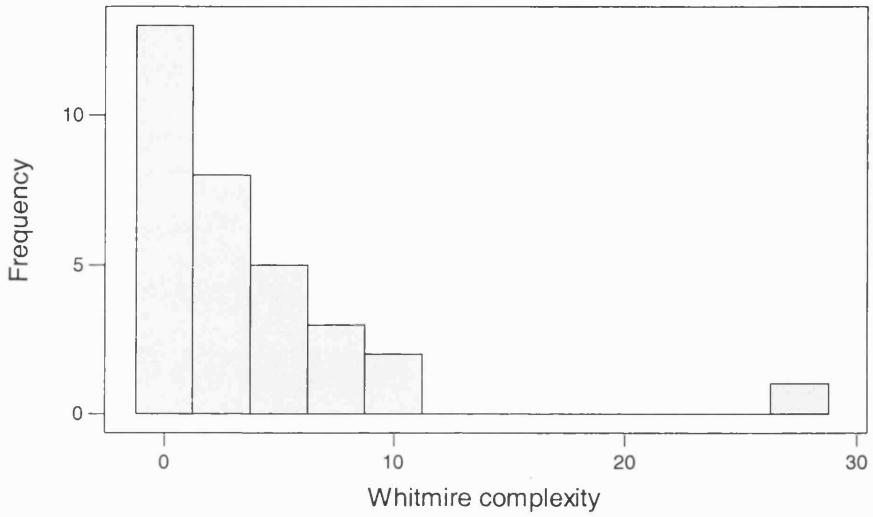


Figure 44 - Whitmire complexity histogram (TRUMPET)

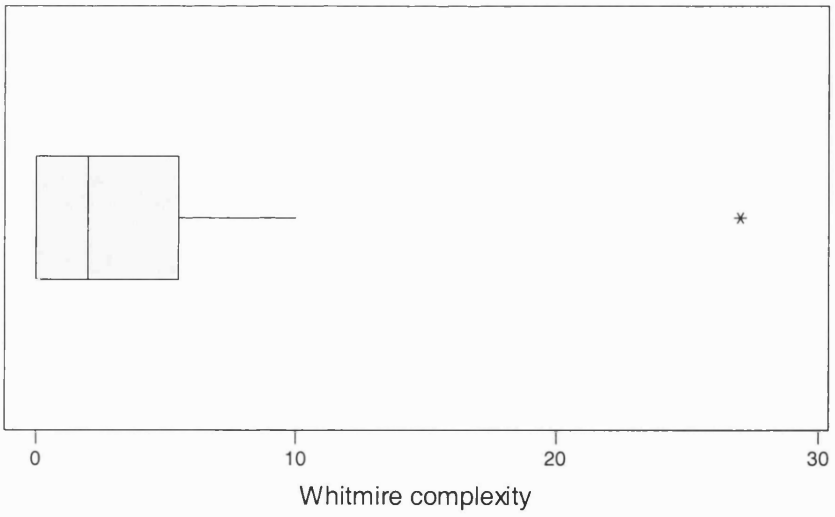


Figure 45 - Whitmire complexity boxplot (TRUMPET)

Mean	0.594
Median	0
Standard deviation	1.316
Minimum	0
Maximum	6

Table 4 - CBO statistics (TRUMPET)

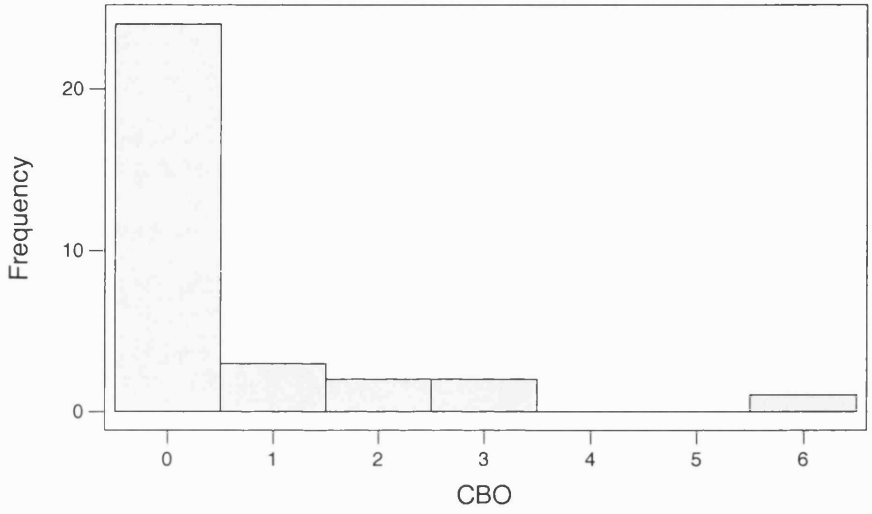


Figure 46 - CBO histogram (TRUMPET)

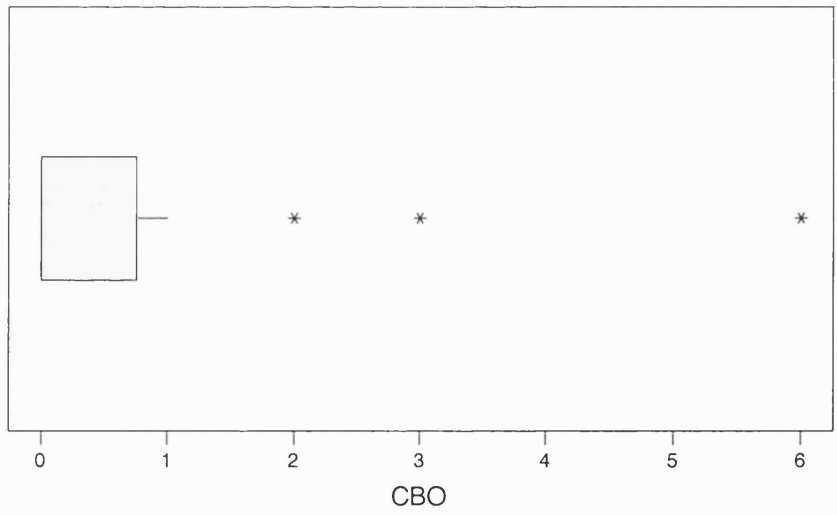


Figure 47 - CBO boxplot (TRUMPET)

Mean	1.937
Median	0
Standard deviation	4.683
Minimum	0
Maximum	23

Table 5 - MPC statistics (TRUMPET)

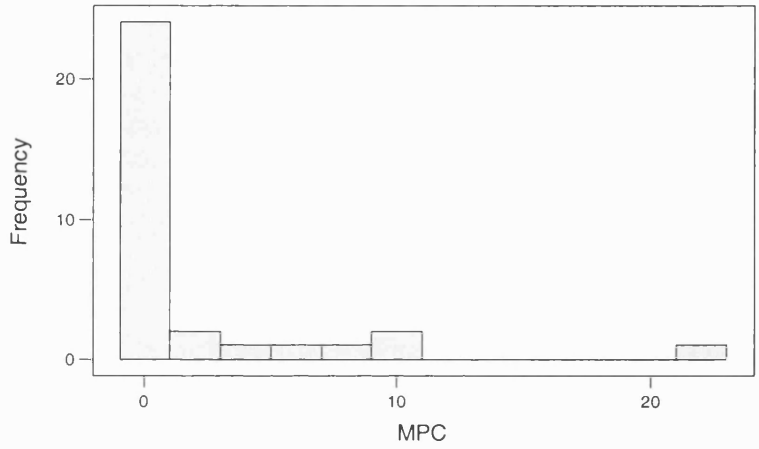


Figure 48 - MPC histogram (TRUMPET)

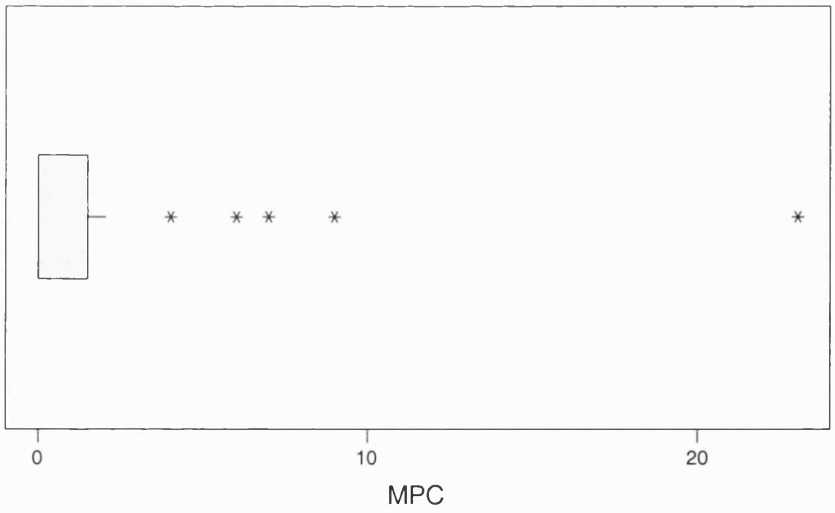


Figure 49 - MPC boxplot (TRUMPET)

Mean	3.906
Median	0
Standard deviation	6.907
Minimum	0
Maximum	30

Table 6 - RFC statistics (TRUMPET)

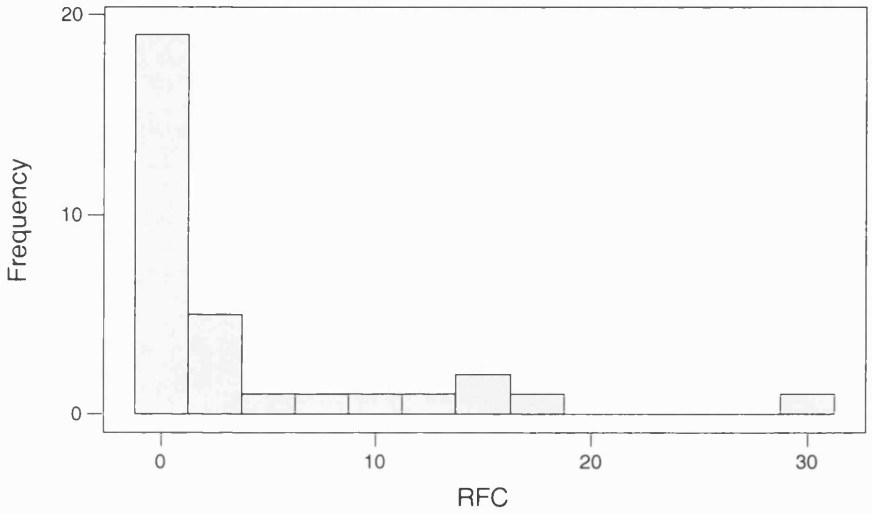


Figure 50 - RFC histogram (TRUMPET)

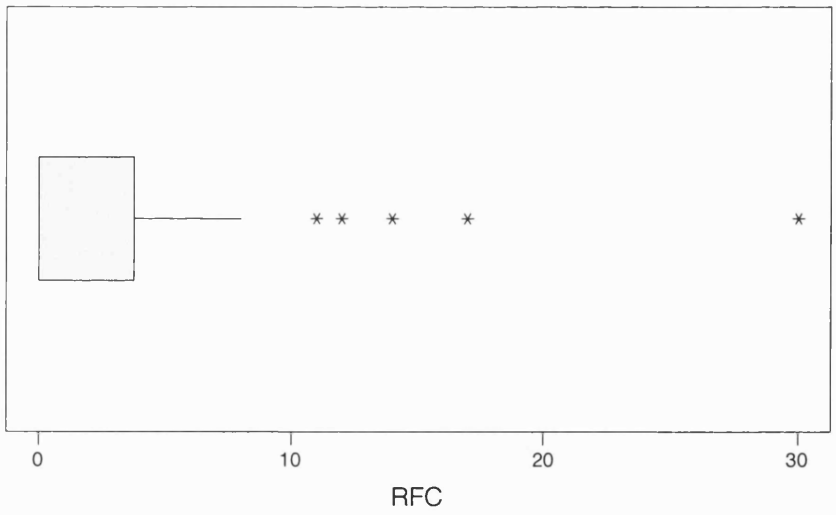


Figure 51 - RFC boxplot (TRUMPET)

Mean	5.969
Median	0
Standard deviation	8.899
Minimum	0
Maximum	31

Table 7 - Interface complexity statistics (TRUMPET)

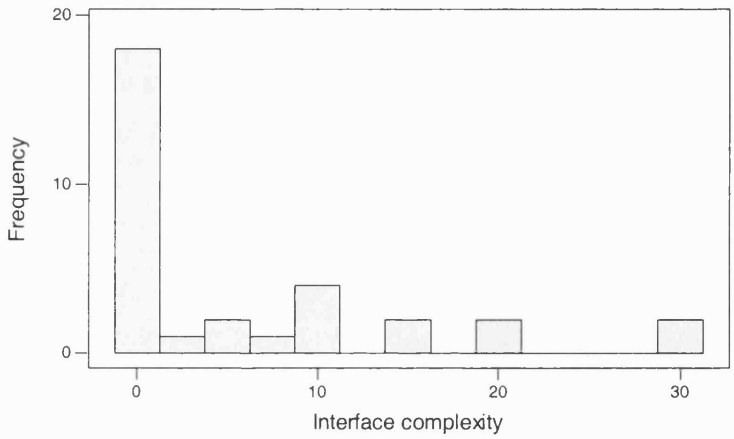


Figure 52 - Interface complexity histogram (TRUMPET)

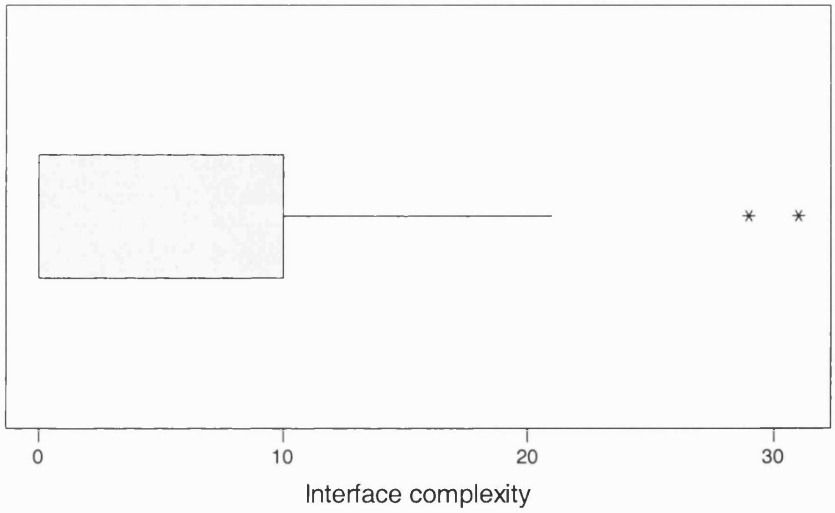


Figure 53 - Interface complexity boxplot (TRUMPET)

5.2.1.3 SUMMARY

The assessment of the TRUMPET system involved five metrics out of the seven proposed in the suite: DIT and NOC metrics were not applicable since inheritance was not used in the TRUMPET design. Although the design team consisted of professionals, this omission of inheritance suggests that the team was relatively unaccustomed to the OO design.

As already discussed in section 5.1.1, the one of the only two reported studies that involved the collection of metrics from design documents [Cart96] argued that only the DIT and NOC measurements can be obtained at this stage of the development. We showed that the CBO, MPC, RFC, interface and Whitmire complexity metrics can also be calculated early in the development lifecycle. However, we cannot claim that all of the measurements are 100% complete since the TRUMPET design, like any other, does not guarantee completeness of design information.

All of the five metrics used to assess the TRUMPET management system indicate that the computational object classes operating at the interfaces between the VASP and the PNO domains exhibit the highest level of complexity in the design. These two classes are the VASP_VPN_Manager (in the VASP domain) and the PNO_Conn_Manager (in the PNO domain). During development and testing of the TRUMPET software, no detailed testing or failure data was collected. However, this feature was confirmed during the software integration on the trial sites, where these two classes represented the main source of pitfalls [Prnj98b].

This assessment also indicates that the points of highest risk are those located at the interconnection points of autonomous management systems, as was suggested by the discussions found in [Ward95] [UCL94].

The boxplots of the metrics indicate that all the distributions of all the metrics are non-normal: they are strongly left-skewed, with a few outliers distinctly standing out. The histograms of all the metric distributions clearly show that the majority of classes exhibit low complexity counts - with a large number having 0 complexity. Moreover, the CBO counts are distinctly low, indicating that the general coupling is kept at a very low level, which is a desirable feature. The MPC counts for the information object classes are 0 - indicating that the information objects are effectively the communication sinks, as expected.

A more comprehensive statistical analysis of the results is given in section 5.2.3.1.

5.2.2 FLOWTHRU SYSTEM

5.2.2.1 ANALYSIS MODEL

5.2.2.1.1 Approach

The FlowThru subscription management component was presented in detail in chapter 4, section 4.2.1. As mentioned before, the analysis model was developed *post-facto*, considering the generic requirements for the component. The analysis model is based on [Jaco92] framework, and is represented by a set of 24 UML diagrams [Flow-http]. As discussed in section 4.2.1, the analysis model UML diagrams effectively capture the ODP computational and information viewpoints: the analysis model boundary and control objects correspond to ODP computational objects and their interfaces, while the analysis model entity objects correspond to the ODP information objects. Thus, the metric suite framework shown in Table 1, page 113, is still applicable, and each of the metrics was applied to the appropriate UML diagram(s) describing the relevant viewpoint.

There is no inheritance in the FlowThru analysis model, and thus the inheritance metrics, DIT and NOC, are not applicable. The interface complexity metric is also not applicable, since although the interface methods were defined, the parameters were not.

The FlowThru analysis model consists of 28 classes. The data collection was performed manually.

5.2.2.1.2 Assessment

The metric values of the classes are shown in Table 8. The table is sorted with respect to decreasing Whitmire complexity (last column).

Class number	Class	CBO	MPC	RFC	Whitmire
1	PA Interface	4	19	19	19
2	Subscription Manager	9	9	24	9
3	SUG Manager	5	5	5	5
4	Service Manager	4	4	12	4
5	Customer Account Manager	4	4	9	4
6	Subscription Usage Group	0	0	1	4
7	Service Record	0	0	1	4
8	Customer Account	0	0	1	4
9	Subscription Management Interface	1	3	17	3

10	QoS Criteria	0	0	0	3
11	Subscription	0	0	0	3
12	CA interface	2	2	8	2
13	Service	0	0	0	2
14	Service Level Agreement	0	0	0	2
15	Service Management Interface	1	1	9	1
16	Customer Management Interface	1	1	6	1
17	SUG Management Interface	1	1	1	1
18	Accounting Management Interface	0	0	3	0
19	User Management Interface	0	0	2	0
20	Network Address	0	0	0	0
21	User	0	0	0	0
22	Subscription Contract	0	0	0	0
23	Customer Details	0	0	0	0
24	Service Bounds	0	0	0	0
25	Violation Tariff	0	0	0	0
26	Tariff	0	0	0	0
27	SUG Details	0	0	0	0
28	SA Interface	0	0	0	0

Table 8 - FlowThru analysis metrics values

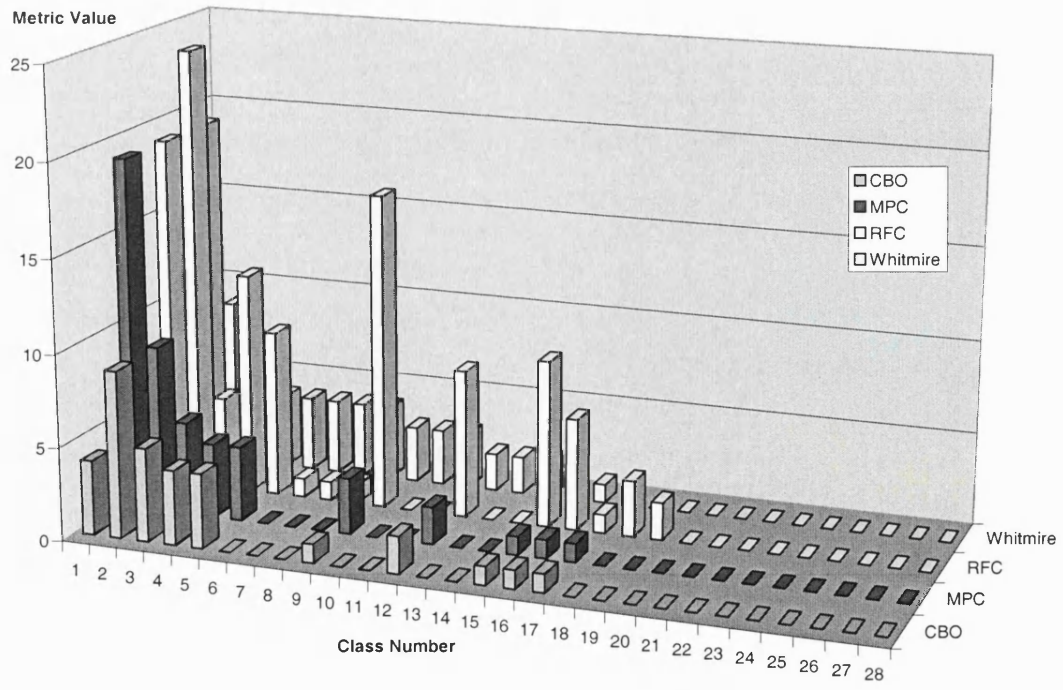


Figure 54 - Metrics distributions per class (FlowThru analysis)

Figure 54 graphically depicts the metrics distributions per class. On average, the most complex class is the Provider Administrator (PA) interface - the boundary/interface class of

the provider management application, which controls the subscription management component. Next, the four control/computational object classes follow: the Subscription Manager, Service Manager, Customer Account Manager and the SUG Manager. The boundary/interface object classes are next on the complexity scale, the Subscription Management Interface exhibiting particularly high complexity. Finally, the purely information (entity) object classes representing the key data entities follow.

In what follows we discuss the complexity measurements as per metric. For each metric, we present the basic summary statistics, the histogram of the distribution of metric values, and the boxplot of metrics values. A more detailed statistical analysis over the whole set of metrics is given in section 5.2.3.1.

Table 9 gives the descriptive statistics for the Whitmire complexity measurements, Figure 55 the histogram of the distribution of the Whitmire complexity values, and Figure 56 the Whitmire complexity boxplot. This metric identifies the PA interface as the most complex, followed by the four main control/computational classes - Subscription Manager standing out.

Table 10 gives the descriptive statistics for the CBO measurements, Figure 57 the histogram of the distribution of the CBO values, and Figure 58 the CBO boxplot. As in the case of TRUMPET, CBO counts are low: most of the classes have the CBO count between 0 and 2. Highest CBO is exhibited by the Subscription Manager class, followed by the other control/computational classes and the PA interface. Entity objects CBO counts are 0.

Table 11 gives the descriptive statistics for the MPC measurements, Figure 59 the histogram of the distribution of the MPC values, and Figure 60 the MPC boxplot. Similarly to the TRUMPET case study, most of the classes have the MPC count between 0 and 2, with only a few classes having higher MPC (up to 19). MPC values follow the distribution of the Whitmire complexity values, with the PA interface exhibiting the highest complexity, followed by the four main control/computational classes, Subscription Manager standing out. Entity objects MPC counts are 0.

Table 12 gives the descriptive statistics for the RFC measurements, Figure 61 the histogram of the distribution of the RFC values, and Figure 62 the RFC boxplot. RFC follows previously discussed complexity distribution for the control/computational objects: however, now the boundary/interface objects exhibit a complexity increase as compared to the previous measurements - due to the high count of local methods.

Mean	2.536
Median	1.5
Standard deviation	3.892
Minimum	0
Maximum	19

Table 9 - Whitmire complexity statistics (FlowThru analysis)

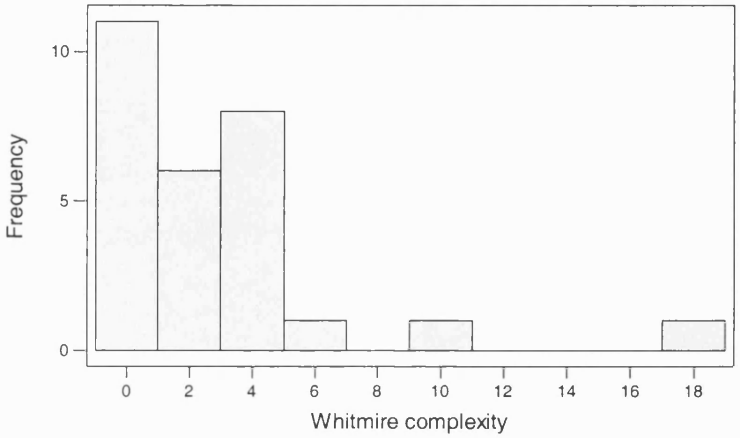


Figure 55 - Whitmire complexity histogram (FlowThru analysis)

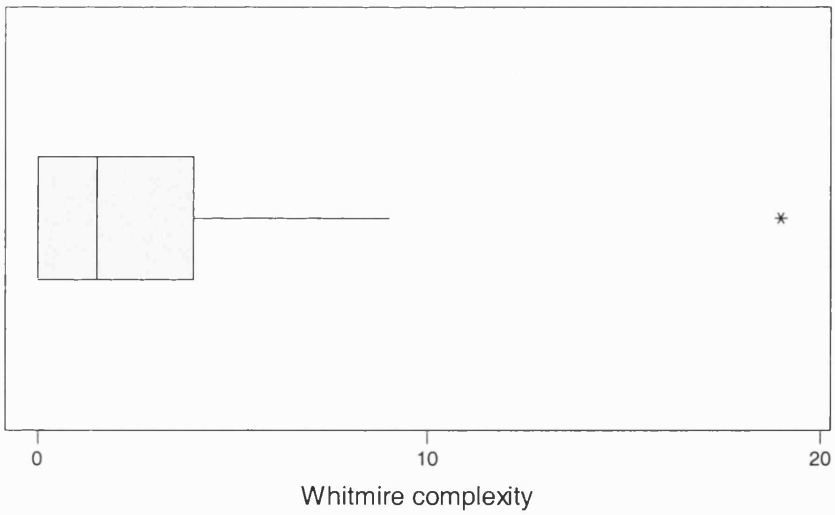


Figure 56 - Whitmire complexity boxplot (FlowThru analysis)

Mean	1.143
Median	0
Standard deviation	2.155
Minimum	0
Maximum	9

Table 10 - CBO statistics (FlowThru analysis)

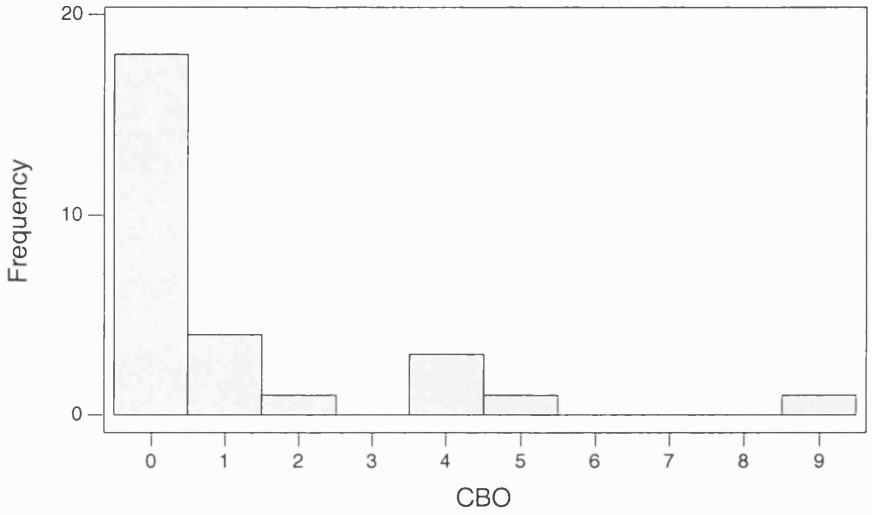


Figure 57 - CBO histogram (FlowThru analysis)

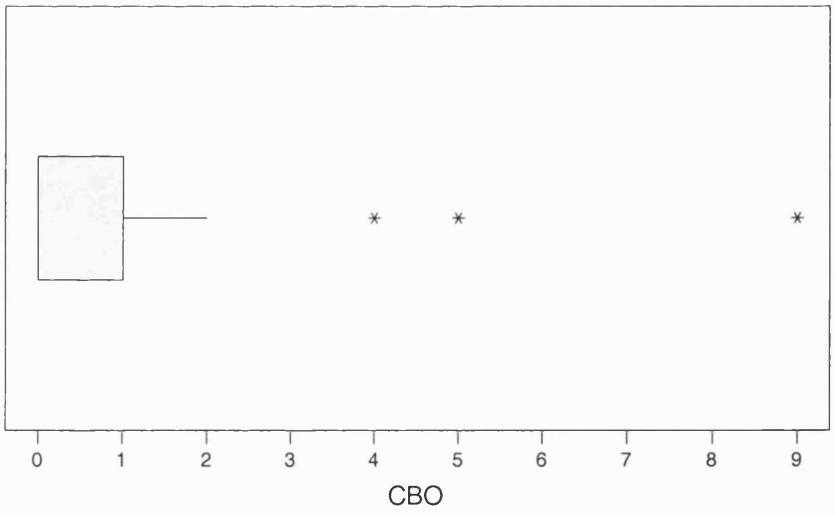


Figure 58 - CBO boxplot (FlowThru analysis)

Mean	1.750
Median	0
Standard deviation	3.987
Minimum	0
Maximum	19

Table 11 - MPC statistics (FlowThru analysis)

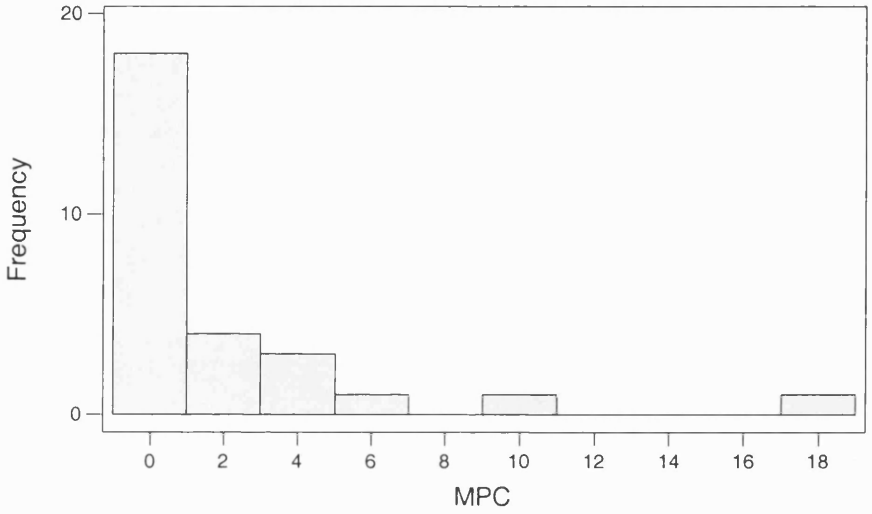


Figure 59 - MPC histogram (FlowThru analysis)

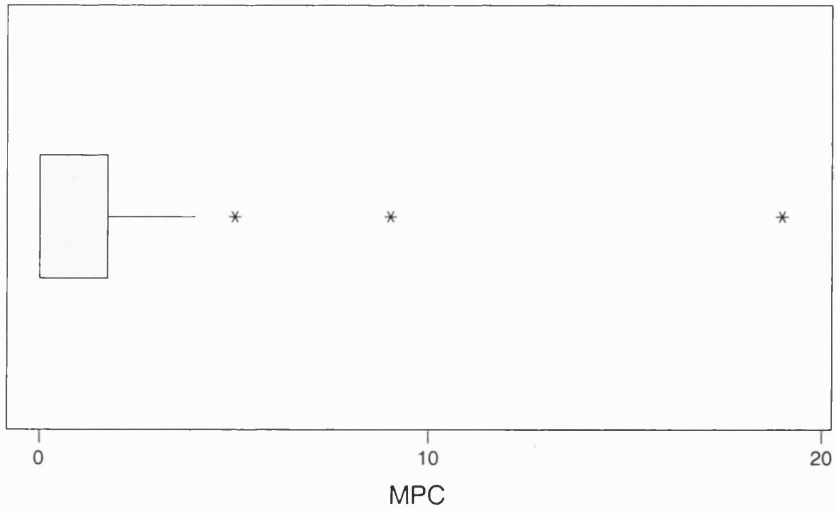


Figure 60 - MPC boxplot (FlowThru analysis)

Mean	4.214
Median	1
Standard deviation	6.602
Minimum	0
Maximum	24

Table 12 - RFC statistics (FlowThru analysis)

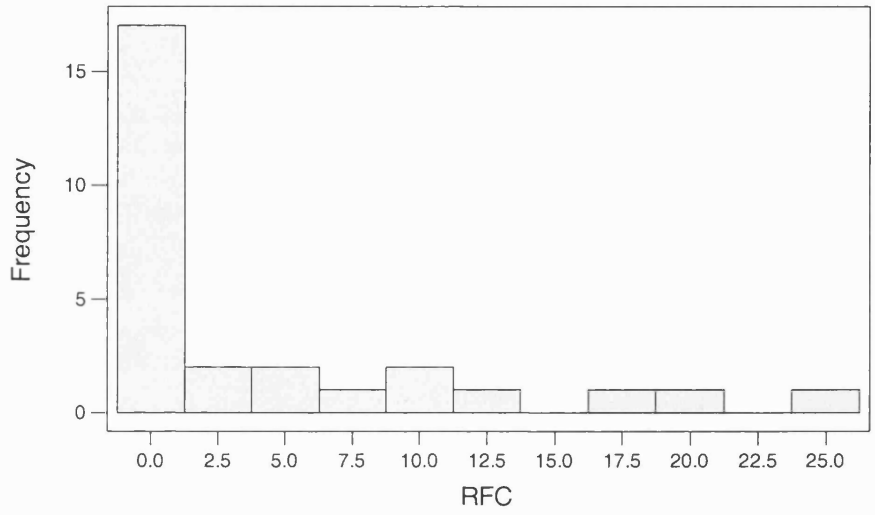


Figure 61 - RFC histogram (FlowThru analysis)

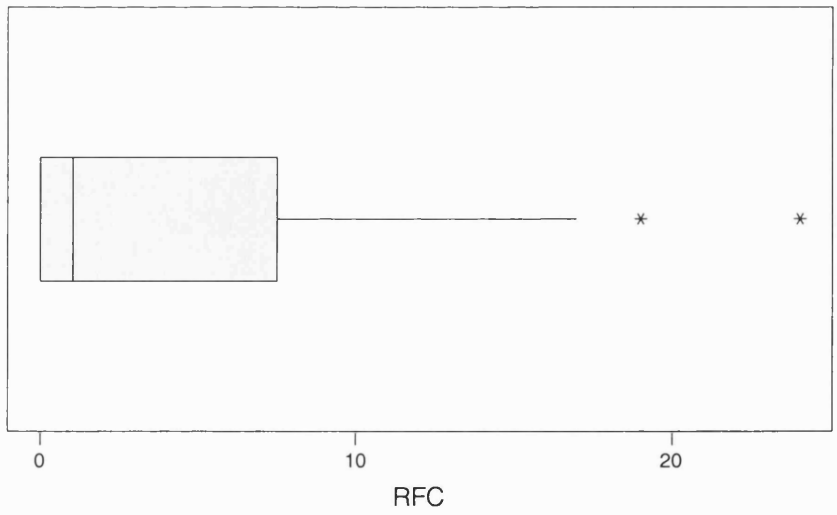


Figure 62 - RFC boxplot (FlowThru analysis)

5.2.2.1.3 Summary

The assessment of the FlowThru analysis model involved 4 metrics out of the 7 proposed in the suite: DIT and NOC metrics were not applicable since inheritance was not used; and the interface complexity was not assessed due to the undefined method parameters in the interface.

Similarly to the TRUMPET case study, this study also demonstrated that all the metrics making up our metric suite could be collected early in the development lifecycle, in contrast to the established practice (as discussed in section 5.1.1). The exception is the interface complexity metric, which is a relatively late-lifecycle measure, requiring full interface elaborations. However, we again cannot claim that all of the measurements are complete due to the inherent incomplete nature of any analysis/design. This is especially true in the case of the MPC values, due to the incompleteness of collaboration diagrams.

The four metrics used to assess the FlowThru management component indicate that the two most complex classes are the PA interface - the interface from the provider management application to the subscription management component, and the Subscription Manager - the main control/computational class in the component. This result just reinforces the TRUMPET results: the highest complexity is exhibited at the interfaces between domains, or, in the FlowThru case, stand-alone components.

As was the case in TRUMPET, the boxplots of the metrics indicate that all the metric distributions are non-normal: they are strongly left-skewed, with a few outliers distinctly standing out. The histograms of all the metric distributions clearly show that the majority of classes exhibit low complexity counts - with a large number having 0 complexity. Also, the CBO counts are again distinctly low, indicating that the general coupling is kept at the very low level, which is a desirable feature. The MPC counts for the information objects are 0 - again indicating that the information objects are effectively the communication sinks, as expected.

A more comprehensive statistical analysis of the results is given in section 5.2.3.1.

5.2.2.2 DESIGN MODEL

5.2.2.2.1 Approach

The FlowThru subscription management component was presented in detail in chapter 4, section 4.2.1. As mentioned before, the design model was simply reverse-documented using UML on the basis of the existing implementation. The design model consists of 108 UML diagrams [Flow-http]. As discussed in section 4.2.1, the design model UML diagrams effectively capture the ODP computational and information viewpoints: design-model packages exporting the interface (i-type) objects depict the computational viewpoint, while the entity (t-type) objects depict the information viewpoint. Thus, the metric suite framework shown in Table 1, page 113, is still applicable, and each of the relevant metrics was applied to the appropriate UML diagram(s) describing the suitable viewpoint.

There is no inheritance in the FlowThru design model, and thus the inheritance metrics, DIT and NOC, are not applicable. The MPC metric is also not applicable, since collaboration diagrams illustrating the message exchange between the collaborating objects are not included in the design documents.

The FlowThru design model consists of 103 classes. The data collection was performed manually.

5.2.2.2.2 Assessment

The metric values of the classes are shown in Table 13. The table is sorted with respect to decreasing Whitmire complexity.

Class number	Class	CBO	RFC	Whitmire	Interface
1	i_DB_sag	16	20	16	36
2	i_subscrnInfoQuery	16	7	16	13
3	i_subscrnCntrl	10	6	10	12
4	i_DB_profile	10	5	10	9
5	i_DB_subs	9	6	9	11
6	i_subscription	9	3	9	4
7	i_setReference	9	2	9	4
8	i_sagInfoQuery	8	3	8	6
9	i_DB_subscriber	7	6	7	10
10	i_sagMgmt	7	5	7	10
11	i_subscrnMgmt	7	5	7	10

12	i_DB_subCtrct	7	4	7	8
13	i_DB_sth	7	4	7	4
14	t_AssignGroupSelection	0	0	6	0
15	t_SvcTemplate	0	0	6	0
16	i_DB_portfolio	5	3	5	6
17	i_sbrInfoQuery	5	3	5	5
18	i_svcTmpltInfoQuery	5	3	5	4
19	i_portfolioMgmt	4	3	4	6
20	i_subscrnNotif	4	2	4	4
21	i_svcTmpltMgmt	4	2	4	2
22	i_svcFctryRefQuery	4	1	4	1
23	t_Subscription	0	0	4	0
24	t_SubscriptionContract	0	0	4	0
25	i_sbrMgmt	3	3	3	5
26	i_saInit	3	2	3	2
27	i_DB_accountList	3	1	3	2
28	i_DB_serviceList	3	1	3	2
29	t_SvcProfile	0	0	3	0
30	t_Configuration	0	0	3	0
31	t_Subscriber	0	0	3	0
32	t_SagItem	0	0	3	0
33	i_srInit	2	2	2	4
34	i_saMgmt	2	1	2	1
35	i_subscrnVerify	2	1	2	1
36	t_SLA	0	0	2	0
37	t_SelectionKey	0	0	2	0
38	t_SvcSelection	0	0	2	0
39	t_SubscriptionPortfolio	0	0	2	0
40	t_AvailableSvc	0	0	2	0
41	t_DBAssignInfo	0	0	2	0
42	t_QoSCriterion	0	0	2	0
43	i_smInit	1	1	1	1
44	t_SvcIdList	0	0	1	0
45	t_SvcCommonData	0	0	1	0
46	t_AccountList	0	0	1	0
47	t_IntRefList	0	0	1	0
48	t_SagList	0	0	1	0
49	t_SvcProfileIdList	0	0	1	0
50	t_SubscriptionAssignmentGroup	0	0	1	0
51	t_AvailableSvcList	0	0	1	0
52	t_DBAssignList	0	0	1	0

53	t_DbSagUserList	0	0	1	0
54	t_AuthLimit	0	0	1	0
55	t_DbSagUser	0	0	1	0
56	t_NapIdList	0	0	1	0
57	t_QoSCriteriaList	0	0	1	0
58	t_SLAList	0	0	1	0
59	t_SvcRecord	0	0	1	0
60	t_SvcRecordList	0	0	1	0
61	t_SvcRefList	0	0	1	0
62	t_TermIdList	0	0	1	0
63	t_UserIdList	0	0	1	0
64	t_QoSCriterionId	0	0	1	0
65	i_smMgmt	0	0	0	0
66	i_srMgmt	0	0	0	0
67	i_sthInit	0	0	0	0
68	i_sthMgmt	0	0	0	0
69	e_subAccessDenied	0	0	0	0
70	t_SvcId	0	0	0	0
71	e_subInvalidAccountNo	0	0	0	0
72	e_subDBcorrupted	0	0	0	0
73	e_subInvalidSvcTemplate	0	0	0	0
74	t_SagId	0	0	0	0
75	t_SvcProfileId	0	0	0	0
76	e_subDBkeyMissing	0	0	0	0
77	e_subDBParsingFailed	0	0	0	0
78	e_subDBkeyExists	0	0	0	0
79	e_subInvalidSAG	0	0	0	0
80	t_UserId	0	0	0	0
81	e_subInvalidSvcProfile	0	0	0	0
82	t_IntRef	0	0	0	0
83	t_SubscriptionId	0	0	0	0
84	t_SvcType	0	0	0	0
85	t_NapId	0	0	0	0
86	t_TermId	0	0	0	0
87	t_SubscriberDetails	0	0	0	0
88	t_DateTime	0	0	0	0
89	t_NapType	0	0	0	0
90	t_TariffId	0	0	0	0
91	t_TermType	0	0	0	0
92	t_EndUserDomain	0	0	0	0
93	e_subInvalidItem	0	0	0	0

94	e_subInvalidPortfolio	0	0	0	0
95	e_subInvalidUser	0	0	0	0
96	t_Account	0	0	0	0
97	t_Person	0	0	0	0
98	t_PresentationSupport	0	0	0	0
99	t_SLAId	0	0	0	0
100	t_SvcProviderId	0	0	0	0
101	t_SvcRef	0	0	0	0
102	t_Credit	0	0	0	0
103	t_DbUserStatus	0	0	0	0

Table 13 - FlowThru design metrics values

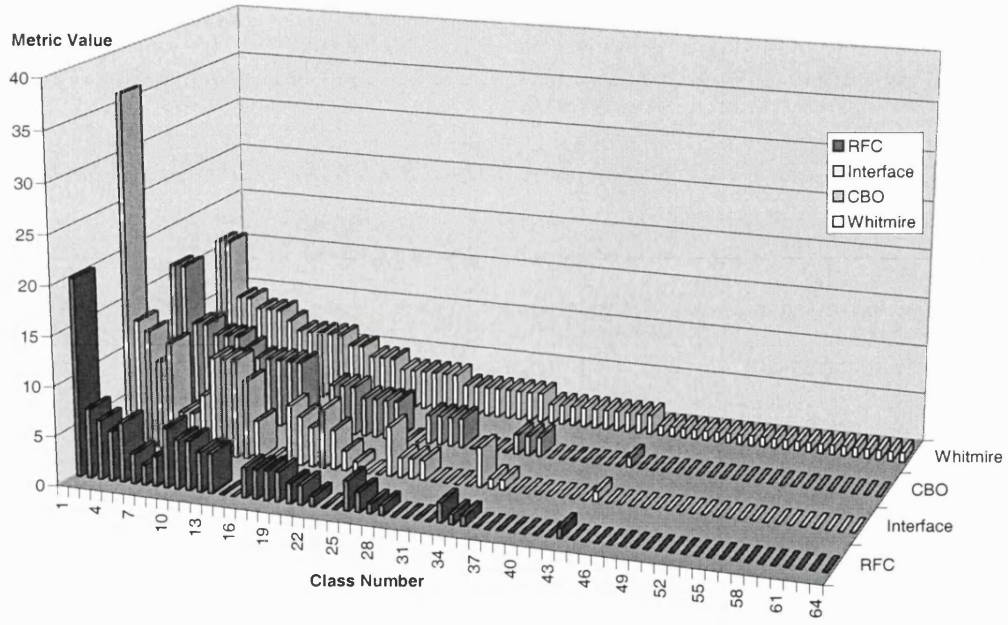


Figure 63 - Metrics distributions per class (FlowThru design)

Figure 63 graphically depicts the metrics distributions per class. Classes with all the metric values of 0 are not included in the diagram. On average, the most complex class is the i_DB_sag, the main Database interface class. However, in the following discussion we will omit all the Database (DB) interface classes, and concentrate on the classes directly relevant for the realisation of the design target functionality. Now, the most complex class on average is the i_subscrnInfoQuery, the interface class derived from the analysis-level Subscription Management Interface. Next on the average complexity scale are a number of other interface classes derived from analysis-level Subscription Management Interface, followed by the interface classes derived from the SUG Management Interface and the Service Management Interface analysis classes. Finally, the purely information (t-type) object classes representing the key data entities follow.

In what follows we discuss the complexity measurements as per metric. For each metric, we present the basic summary statistics, the histogram of the distribution of metric values, and the boxplot of metrics values. A more detailed statistical analysis over the whole set of metrics is given in section 5.2.3.1.

Table 14 gives the descriptive statistics for the Whitmire complexity measurements, Figure 64 the histogram of the distribution of the Whitmire complexity values, and Figure 65 the Whitmire complexity boxplot. This metric identifies i_subscrnInfoQuery, the interface class derived from the analysis-level Subscription Management Interface, as the most complex one. Next on the Whitmire complexity scale are a number of other interface classes derived from analysis-level Subscription Management Interface, followed by the interface classes derived from the SUG Management Interface. Following these are the information objects corresponding to the analysis-level SUG and Service information objects.

Table 15 gives the descriptive statistics for the CBO measurements, Figure 66 the histogram of the distribution of the CBO values, and Figure 67 the CBO boxplot. The CBO counts, as compared to the FlowThru analysis and the TRUMPET case studies, are marginally higher on average, while still most of the classes have the CBO count between 0 and 2. Highest CBO is exhibited by the interface classes derived from the analysis-level Subscription Management Interface, SUG Management Interface and the Service Management Interface, with the i_subscrnInfoQuery still being the most complex class.

Table 16 gives the descriptive statistics for the RFC measurements, Figure 68 the histogram of the distribution of the RFC values, and Figure 69 the RFC boxplot. Highest RFC is exhibited by the interface classes derived from the analysis-level Subscription Management Interface, SUG Management Interface and the Service Management Interface, with the i_subscrnInfoQuery still being the most complex class.

Table 17 gives the descriptive statistics for the interface complexity measurements, Figure 70 the histogram of the distribution of the interface complexity values, and Figure 71 the interface complexity boxplot. Highest interface complexity is exhibited by the interface classes derived from the analysis-level Subscription Management Interface, SUG Management Interface and the Service Management Interface, with the i_subscrnInfoQuery still being the most complex class.

Mean	2.320
Median	1
Standard deviation	3.264
Minimum	0
Maximum	16

Table 14 - Whitmire complexity statistics (FlowThru design)

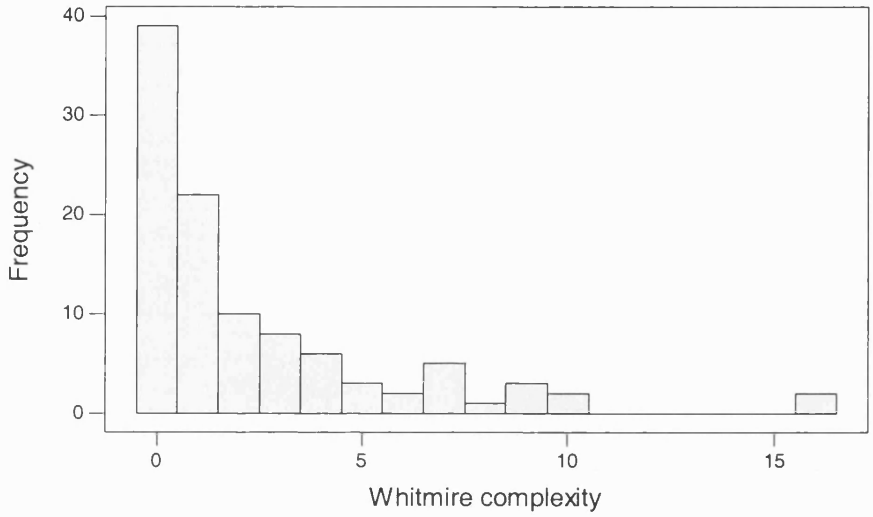


Figure 64 - Whitmire complexity histogram (FlowThru design)

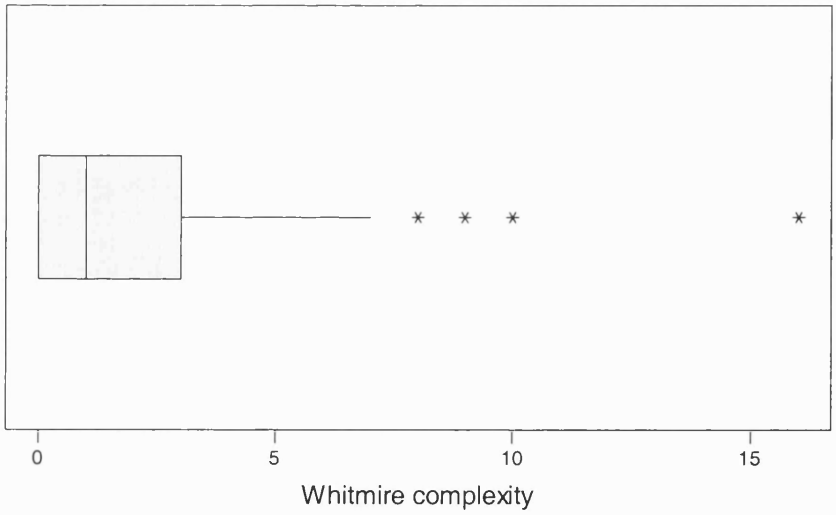


Figure 65 - Whitmire complexity boxplot (FlowThru design)

Mean	1.670
Median	0
Standard deviation	3.379
Minimum	0
Maximum	16

Table 15 - CBO statistics (FlowThru design)

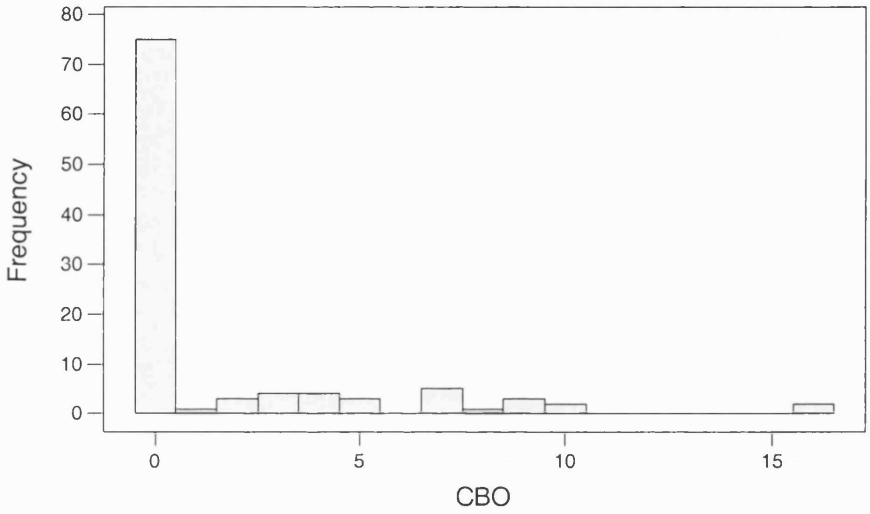


Figure 66 - CBO histogram (FlowThru design)

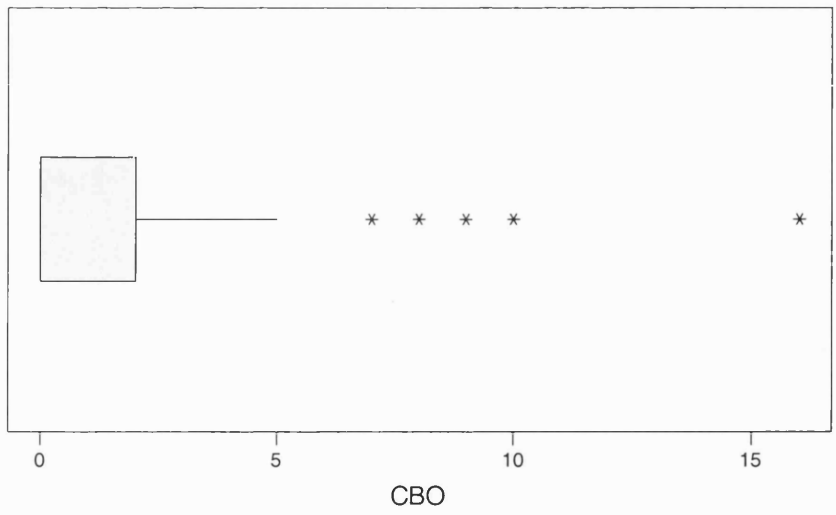


Figure 67 - CBO boxplot (FlowThru design)

Mean	1.019
Median	0
Standard deviation	2.516
Minimum	0
Maximum	20

Table 16 - RFC statistics (FlowThru design)

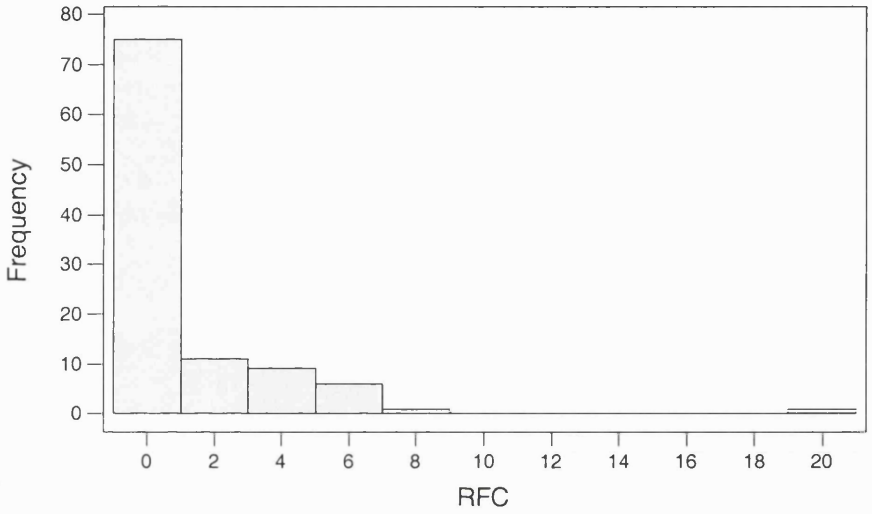


Figure 68 - RFC histogram (FlowThru design)

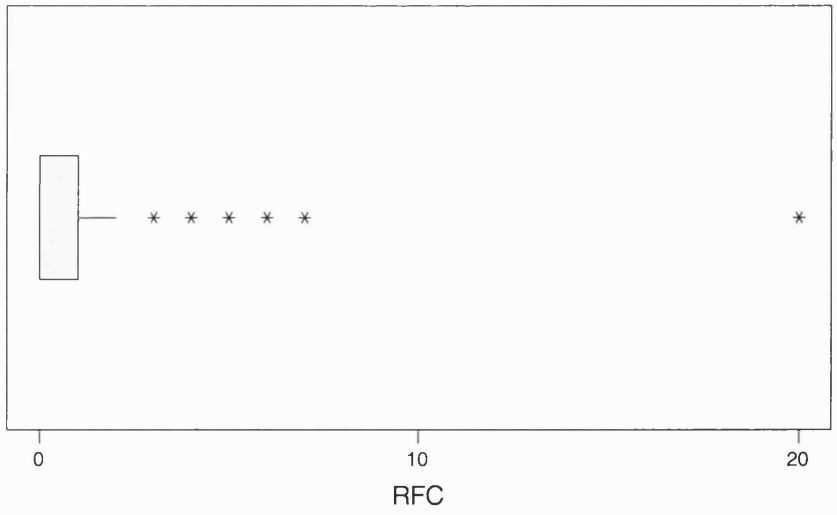


Figure 69 - RFC boxplot (FlowThru design)

Mean	1.777
Median	0
Standard deviation	4.563
Minimum	0
Maximum	36

Table 17 - Interface complexity statistics (FlowThru design)

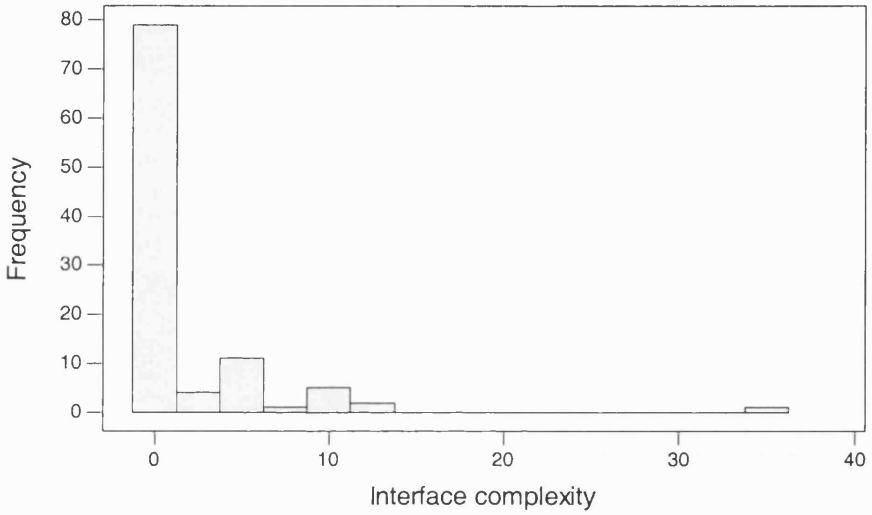


Figure 70 - Interface complexity histogram (FlowThru design)

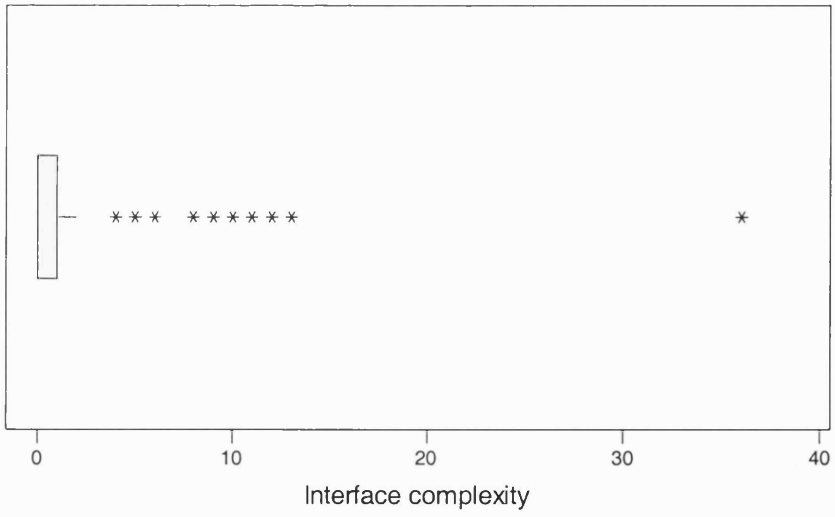


Figure 71 - Interface complexity boxplot (FlowThru design)

5.2.2.2.3 Summary

The assessment of the FlowThru design model involved four metrics out of the seven proposed in the suite: DIT and NOC metrics were not applicable since inheritance was not used; and MPC was not assessed since the collaboration and message passing was not depicted in the FlowThru design. As before, we do not claim that all of the measurements are complete due to the inherent incomplete nature of any design.

All four metrics used to assess the FlowThru management component design indicate that the most complex class is the i_subscrnInfoQuery, the interface class derived from the analysis-level Subscription Management Interface, as the most complex one. All of the metrics next point out the other interface classes derived from analysis-level Subscription Management Interface, followed by the interface classes derived from the SUG Management Interface and the Service Management Interface analysis classes. The purely information (t-type) object classes representing the key data entities exhibit much lower complexity counts.

Considering this result, we can note that the complexity of the design-level packages, derived from the analysis-level control/computational classes, is hidden behind the interfaces they export. This was not the case in the analysis model, where the interface complexities were trailing after their corresponding control/computational class counterparts. However, the complexity measurements do propagate evenly through the development phases. In the design, the interface classes derived from the analysis Subscription Management Interface are the most complex. This is analogous to the analysis model situation, where the most complex class, apart from the PA interface which is not described in the design, was the Subscription Manager. Similarly, the most complex design-level information object class (or t-type class in the FlowThru design terminology) is the t_AssignGroupSelection, derived from the most complex information object class in the analysis: the Subscription Usage Group.

The most complex class, i_subscrnInfoQuery, is the outside interface to the main control/computational class in the component. Thus, the complexity measurements of the FlowThru design again re-iterate the point raised in the TRUMPET and FlowThru analysis studies: the highest complexity is exhibited at the interfaces between domains (as is case in TRUMPET), or, in this case, the stand-alone components.

Again, the boxplots of the metrics indicate that all the metric distributions are strongly left-skewed, with a few outliers distinctly standing out. Similarly, the histograms of all the metric

distributions clearly show that the majority of classes exhibit low complexity counts, large number of them having the complexity value of 0.

A more comprehensive statistical analysis of the results is given in section 5.2.3.1.

5.2.3 DISCUSSION

In the preceding sections, we presented three case studies dealing with the assessment of management systems using the integrity-focused metric suite defined in section 5.1.3. The FlowThru case study involved the assessment of the analysis and design models separately, and the TRUMPET case study focused on the assessment of the design model. The metrics data was collected manually from the UML analysis and design documents.

The metrics data presented in the preceding sections exhibits a range of distinctive features, including non-normal distributions and apparent multicollinearity. Thus, we conducted a statistical analysis of measurements, presented in section 5.2.3.1, so as to trace interesting trends in the data. Section 5.2.3.2 includes a more general discussion of the experiments.

5.2.3.1 STATISTICAL ANALYSIS

As we have seen, all the metrics distributions, in all three experiments, are strongly left-skewed, with a few outliers distinctly standing out. This feature can be seen from both the boxplot diagrams and the histograms. This left-skewed outlier-heavy distribution appears to be the typical metrics distribution.

Since the metrics distributions are non-normal, the basics statistics such as mean and the use of parametric statistical methods do not accurately capture the distribution features and the interrelationships between the distributions. In the case of non-normal distributions the use of robust statistics (such as median and ranks) and nonparametric statistical methods is advocated [Schn92][Fent91]. The assumptions for the use of nonparametric statistical methods are much less restrictive then for the parametric methods (which usually assume normal distributions, equality of variances across samples, *etc.*). However, the nonparametric methods are as rigorous, and allow the analysis of order relations [Schn92].

The metrics distributions for all the metrics are very similar, and there seems to be a strong relationship between the metrics in all three experiments. This strong relationship between the whole set of metrics is a problem referred to as *multicollinearity*. Thus, we concentrated on investigating the associations between the metrics.

First, we investigated the linear association between the metric values, by calculating r, the *linear (Pearson) correlation coefficient* between each pair of metrics. The linear correlation coefficient is a descriptive measure of the linear (straight-line) relationship between two variables. This is the typical approach to testing for metrics interrelationships, commonly reported in literature [Chid98] [Li93]. The linear correlation coefficients for TRUMPET, FlowThru analysis, and FlowThru design measurements are high for each pair of metrics within one isolated experiment. The correlation coefficients fall in the range between 0.631 and 0.993. The majority of coefficients are higher than 0.8. Similar result was reported in [Chid98], where the correlation between the metrics was mostly higher than 0.8.

These results should indicate that the regression equations linking each pair of metrics are highly suitable for making predictions of one metric on the basis of the other (*i.e.*, one metric should be a good linear predictor of the other). Then, the *coefficient of determination*, or r^2 (where r is the linear correlation coefficient), would give the quantitative measure (percentage) of the amount of variation of one metric (*response variable*) explained by the variations in the other metric (*predictor variable*).

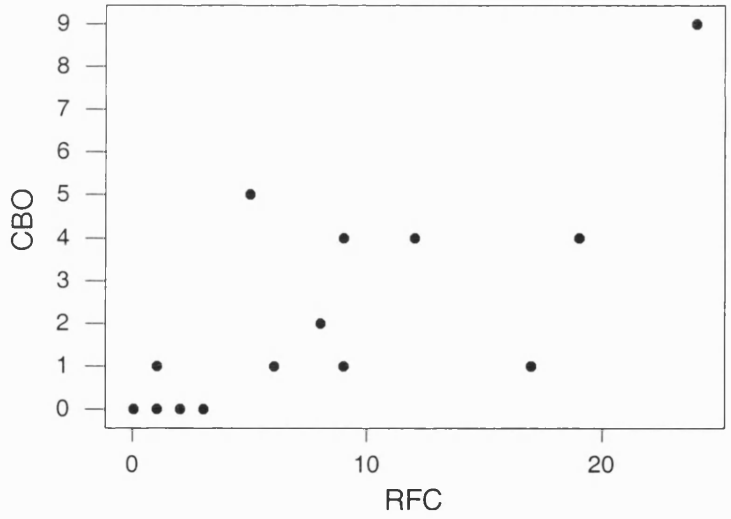


Figure 72 - Scatter plot of CBO versus RFC (FlowThru analysis)

However, by examining the scatter plots we concluded that the data points are actually very weakly scattered about a straight line: one of the scatter plots is shown in Figure 72 (CBO plotted against RFC for the FlowThru analysis experiment). Once this is the case, we cannot make definitive statements concerning the usefulness of one metric as a linear predictor of the other. The assumption for both the linear correlation coefficient and for finding the regression equation is that the data points are clearly scattered about the straight line [Weis99]. Also, regression is sensitive to the presence of outliers, which appear to be a distinctive feature (legitimate data points) of metrics distributions (as discussed above) and as such can not be justifiably removed.

Moreover, we can say that we definitely can not use regression inferences because the two basic conditions supporting the assumptions for regression inferences are not met. The first condition is that the plot of the residuals against the values of the predictor variable (*residual plot*) should fall in a horizontal band centred and symmetric about the x-axis. The second condition is that the normal probability plot of the residuals should be linear. *Residual* is the difference between the observed and predicted value of the response variable. The plot of the FlowThru analysis CBO residuals against the RFC values is shown in Figure 73. Clearly, the residuals do not fall in a horizontal band. The normal probability plot of the residuals is shown in Figure 74: the plot is not straightforwardly linear. Thus, we conclude that the assumptions for regression inferences are violated.

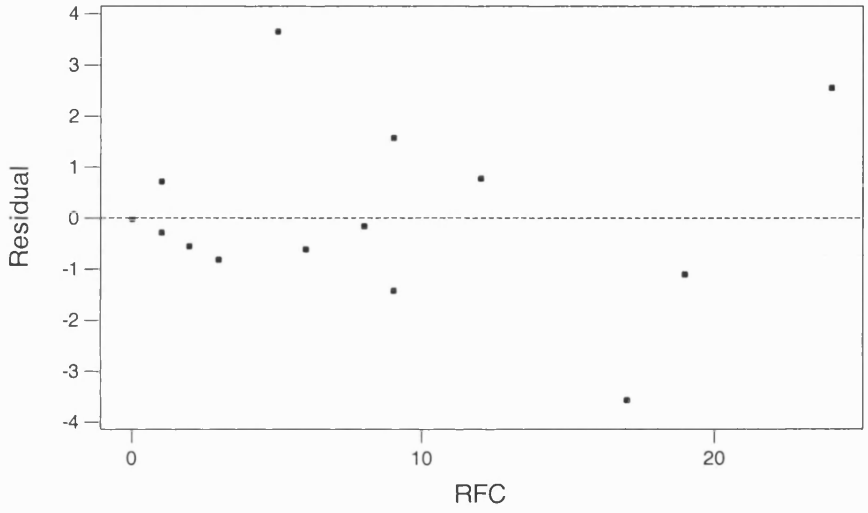


Figure 73 - Residuals versus RFC (response is CBO)

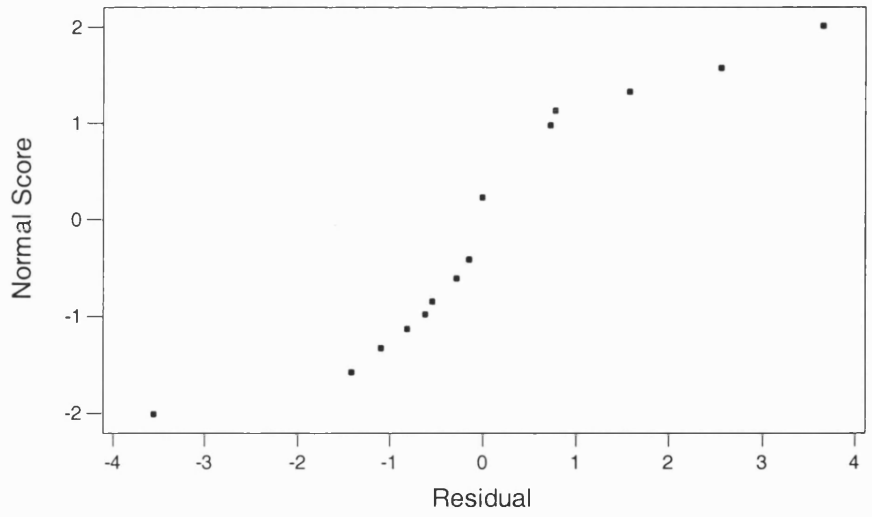


Figure 74 - Normal probability plot of the residuals

Considering the weak linearity of scatter plots, and the violations of the regression inferences assumptions, we conclude that **we cannot claim that any pair of metrics can be used to determine the regression equation (linking the two metrics) from which meaningful predictions can be made.** In other words, we can say that there is enough statistical evidence to doubt the possibility that the magnitude ordering of one metric directly implies the linear magnitude ordering of the other, and that the intervals between the two values of one metric are proportional to the intervals between the values of the other metric, for any pair of adjacent classes that these measurements refer to.

An alternative approach to determine the relationship between the metrics is through the use of nonparametric statistical methods. As discussed before, the nonparametric methods can be used to avoid rigorous assumptions such as linearity.

The nonparametric method equivalent to linear correlation is the calculation of the *rank (Spearman's) correlation coefficient* between paired values (from same classes) of the two metrics. This procedure effectively lowers the metrics scale from interval to ordinal, avoiding the magnitude-related relationships between metrics [Schn92]. This procedure uses ranks of metrics rather than the metrics values themselves. This loosens up assumptions about data relationships (linearity), while still giving a valid measure - ranking of classes according to the metrics value. The rank correlation coefficients for TRUMPET, FlowThru analysis, and FlowThru design measurements are shown in Table 18, Table 19, Table 20, respectively. All the rank correlation coefficients are significant at the 95% confidence level.

	CBO	**MPC**	**RFC**	**Interface**	**Whitmire**
CBO	1				
MPC	0.995	1			
RFC	0.835	0.837	1		
Interface	0.779	0.775	0.975	1	
Whitmire	0.712	0.719	0.555	0.532	1

Table 18 - Metrics rank correlation coefficients: TRUMPET

	CBO	**MPC**	**RFC**	**Whitmire**
CBO	1			
MPC	0.993	1		
RFC	0.846	0.859	1	
Whitmire	0.614	0.626	0.626	1

Table 19 - Metrics rank correlation coefficients: FlowThru analysis

	CBO	RFC	Whitmire	Interface
CBO	1			
RFC	0.994	1		
Whitmire	0.749	0.742	1	
Interface	0.993	0.998	0.741	1

Table 20 - Metrics rank correlation coefficients: FlowThru design

The metrics rank correlation coefficients are high. This indicates that there is a strong ranking relationship between the metrics, for the same set of classes. Thus, effectively, we can say that there is enough statistical evidence to say that **each metric could be useful in predicting the ranks of the other metrics** (more or less, depending on the value of the rank correlation coefficient).

Consistently, through all three studies, exceptionally high (>0.95) rank correlation coefficient is exhibited between CBO and MPC, and between RFC and interface complexity. This indicates that any combination of: CBO or MPC; RFC or interface complexity; and Whitmire complexity metrics would form a set of three complementary metrics. Thus, in any hypothetical model linking metrics and integrity, one of these alternative combinations of three metrics should be used instead of all five metrics together. However, as noted before (section 5.1.2), such a model cannot be built since integrity is a complex attribute which is not directly measurable.

The identification of the underlying dimensionality of the set of correlated metrics could have been also done using the principal component analysis: however, we regarded one method as being sufficient. We also considered assessing the equality of metrics distributions using the chi-squared test. However, the nature of data - small, discrete values, and many null values, rendered the tests invalid. Finally, the robust, nonparametric regression approaches (discussed in [Spre89]) could have been used to develop the regression equations linking the metrics. However, this was not done since there is a shortfall of tools that are capable of performing such nonparametric statistical calculations. The calculations presented here were performed using the Minitab statistical package [Minitab].

5.2.3.2 LESSONS LEARNT

We consider the TRUMPET and FlowThru metrics experiments to be an empirical research contribution in their own right. As discussed in section 5.1.1, there are few reported studies dealing with the practical system evaluation using the OO metrics. In the majority of reported studies, the metrics (CK) were collected from code, despite the fact that they were originally envisaged [Chid94], and later advertised [Kami99], as earlier lifecycle measures.

The one of the only two reported studies involving the design documents [Cart96] reported problems with metrics collection - only DIT and NOC were collected successfully. Moreover, metrics were never applied to network and service management systems.

Our three experiments are the first to assess management systems using metrics. Although the systems assessed originate from the research projects, we consider them as representative since a number of professional organisations participated in the design. The systems were chosen due to the public availability of design documentation, which is not the case in the industrial organisations.

The experiments demonstrated that the metrics collection from the analysis and design documents is possible. However, not all of the metrics can be collected early in the development lifecycle: the analysis-level documentation allows collection of Whitmire complexity, CBO, MPC, RFC, NOC and DIT; while the interface complexity can be collected only at the detailed design stage - as demonstrated by the FlowThru case study. It is also generally believed that the ability of metric collection to some extent depends on the development framework and the notation used. However, we demonstrated that through thorough use of established diagrammatic techniques such as UML metrics collection becomes easy. Moreover, by assessing two different systems - one developed in the ODP [ODP] framework and the other through the Jacobson [Jaco92] framework - we showed the independence of metrics from the development technique.

A number of general observations can be made concerning the results of our case studies.

First, none of the systems contained any inheritance. This might be due to either the fact that the design teams were not accustomed to the OO design philosophy, or it could be a reflection of the size of the systems, which can be considered as small-scale. However, surprisingly low inheritance measures (DIT and NOC) were also reported in a number of earlier studies [Chid98] [Cart96] [Basi96].

Second observation is that the CBO counts appear distinctly low, as compared to the MPC, RFC and Whitmire complexity counts, which also depict the coupling between objects. This is mainly due to the fact that these other three coupling measures actually include the amount of collaboration between classes, while the CBO accounts for the number of collaborating classes. However, the CBO counts are still much lower than in previous studies reported in the literature, which can be due either to the size of our systems, or the fact that the designers, using the design heuristics, aimed at minimising the number of collaborating classes.

Next, as already discussed in the previous section, all the metrics, in all three experiments, have a typical distribution: strongly left-skewed, with a few outliers distinctly standing out. Also, metrics are highly correlated in terms or ranking of the classes. Particularly high rank correlations are exhibited between CBO and MPC; and RFC and interface complexity.

The final observation is that the majority of classes have distinctly low metric counts - between 0 and 2. In case of TRUMPET, 61% of the classes have the average complexity between 0 and 2, in FlowThru analysis 53%, while in the FlowThru design this number rises to 76%.

All of the metrics used thus identify a subset of classes as distinct from others in terms of complexity. These classes are most usually the main computational object classes in the system, performing a manger-control task. Also, the highly complex classes can be the most important information object classes in the system.

The most complex classes singled out in our case studies were the classes operating at the interfaces between administrative domains (as in TRUMPET), or at the interfaces between the stand-alone management components (as in FlowThru). These most complex classes were singled out by all of the metrics making up our metrics suite. Considering our integrity viewpoint, these measurements indicate that the highest risk for the systems' integral operation is exhibited at the major interconnection points. This argument in the integrity context was already pointed out in [UCL94] - however, it was not empirically justified. In the case of TRUMPET, the two classes operating at the X interface between the VASP and the PNO domains - that were singled out as the most complex in design - did prove to be the most difficult to design, implement and test, and were the main source of pitfalls during the project trials.

In the context of the integrity framework, we demonstrated, in our previous discussion (section 5.1.2), how the integrity-metrics policy fits conceptually in the integrity methodology presented in chapter 3. The case studies did not demonstrate this in practice, since the re-design, or the integrity-preserving response, was not applied during the systems development. This was not done because the systems were assessed *post-facto*. Thus, the metric suite, in the integrity context, was used simply as an analysis tool for assessment, *i.e.* diagnostics of the most complex classes. These classes are then labelled as the classes with lowest integrity, or highest risk, according to our prediction rationale described in section 5.1.2.

5.3 CHAPTER SUMMARY AND RESEARCH CONTRIBUTIONS

In this chapter, we presented a predictive integrity-preserving policy based on the OO software metrics. The theoretical bases of this policy were further empirically demonstrated through three distinct experiments. In what follows, we summarise the research work, and highlight our achievements in bold.

The theoretical part of the work focused on the development of the integrity-metrics policy. This activity included:

- Assessment of the current use of OO metrics.
- Elaboration on the **alternative use of OO metrics - in the integrity context.**
- Positioning of the **integrity-metrics policy in the integrity methodology framework.**
- Definition of the **integrity-oriented metric suite.**

First, we discussed the current state of the art in software measurement, presenting a number of traditional software measures, as well as the emerging OO metrics - the most representative of which are the five measures collectively known as CK metrics. We pointed out that software measurement is a nascent field: its focus nowadays is the prediction of economic variables such as project effort, duration, re-work and maintenance effort. Software metrics are generally considered as a managerial tool, envisaged to aid project managers in effort allocation and project planning. Few studies suggested metrics for an alternative purpose. In [Chid94] an additional feature of CK metrics was mentioned - identification of the design flaws - however, no details were given. Amongst other studies in fault-proneness, in [Basi96] CK metrics were shown to be more successful in predicting fault-proneness of the classes then other existing metrics.

We then suggested that the **OO metrics could be used as the integrity/risk indicators *early in the telecommunications system development lifecycle*** by pointing out the most complex/coupled classes. We elaborated on the positive correlation between the class stand-alone complexity, as well as class coupling, and the risk level of the class. Classes of higher complexity and coupling are more difficult to develop and test correctly, and as such pose more risk to integral system operation, *i.e.*, these classes have lower integrity. Further, the strong coupling paths give way to the propagation of the integrity breach through the system.

This positive correlation is the sole relationship that can be formed between the complexity/coupling measurements and the integrity/risk levels, since integrity is a complex external attribute that cannot be measured. Thus, a direct functional relationship between

complexity/coupling measures and integrity cannot be established: *i.e.* a full-scale mathematical model cannot be built.

Next, we **positioned this integrity-metrics policy in the framework of the integrity methodology** presented in chapter 3. The typical integrity analysis in this context would have as the requirement the need to minimise and even out the complexity/coupling levels of individual classes so as to avoid the risk being focused on a few points of failure. The integrity design would involve the definition of the metric suite used to measure the complexity/coupling levels of the system classes. Integrity implementation focuses on the actual realisation of the policy, by conceptually automating a control loop. In this loop, the complexity/coupling measurements are taken from the semi-formal model of the system under development, correlated with the integrity/risk status of the system classes, and then on the basis of this the adequate redesign actions are taken. Using the terminology defined in chapter 3, the integrity-metrics policy belongs to the *prediction* phase of the methodology and has effectively the form of an *integrity-focused design recommendation.*

We then elaborated on the design of this integrity policy through the **specification of a metric suite consisting of seven distinct metrics**: four out of five CK metrics (DIT, NOC, CBO and RFC), the MPC, the interface complexity and the Whitmire complexity metrics. Finally, in the context of the integrity methodology presented in chapter 3, we **illustrated how each of these metrics relates** to the UML diagram it is calculated from, ODP viewpoint depicted by this UML diagram, and the integrity level (as discussed in chapter 3, section 3.2.1.2) at which the metric can be perceived.

To test the efficiency of this integrity policy, we assessed two distinct management systems using our integrity-metrics suite: the TRUMPET service provisioning system and the FlowThru subscription management component. The FlowThru case study involved separate assessment of analysis and design documents, thus the total number of experiments was three. As we mentioned earlier in section 5.1.1, there are few reported studies dealing with empirical OO measurements. In all but two [Cart96][Chid98] of the reported studies the OO (*i.e.* CK) metrics were collected directly from the code, and in [Cart96] the measurements were highly incomplete (only inheritance measures were collected). Moreover, metrics were never applied to network and service management systems.

Thus, we consider the experiments presented in this chapter not only as a vehicle for validation of our integrity-metrics policy, but also as an **empirical research contribution in its own right**. In this context, as already discussed in section 5.2.3, the research contributions are:

- Our case studies are **the first to assess management systems using metrics**. These systems were chosen due to the public availability of the documentation, and are considered as representative since they were developed by a number of professional organisations.
- We demonstrated that **metrics collection from the analysis and design documents is possible**, despite the usual practice. Some metrics can be collected early in the analysis, while some only on the level of detailed design.
- We demonstrated that **metric collection is easy through use of UML**, and is independent of the development framework used: the FlowThru system was developed in the Jacobson framework, while the TRUMPET system in the ODP framework.

During the assessment of the systems using our metric suite, we also came across a number of interesting observations. First, the systems did not use any inheritance: this was also reported in a number of previous studies in the literature, and in our case can be explained through the inexperience of design teams and relative small scale of the projects.

Next, the CBO counts are low: this is due both to the nature of this metric (which assesses simply the number of collaborating classes) as well as the size of the system.

From the statistical point of view, all the metrics, across all the experiments, exhibit the same typical distribution: strongly left-skewed, with a few outliers distinctly standing out. To assess the relationship between the metrics, we considered a number of statistical tests. We **demonstrated that the metrics interrelationships cannot be assessed through linear correlation, and that no meaningful linear regression equations linking any two pair of metrics can be developed.** The nonparametric statistical test for determining the rank correlation coefficient between the metrics proved to be more suitable, since no linearity is assumed. All the metrics rank correlations are high, especially the CBO-MPC and RFC-interface complexity correlation coefficients. Thus, we concluded that **any combination of: CBO or MPC; RFC or interface complexity; and Whitmire complexity can be used as a three-metric set instead of using all of these five metrics.**

Final observation is that the majority of classes have distinctly low metric counts - between 0 and 2: in TRUMPET this number is 61%, FlowThru analysis 53%, and FlowThru design 76%.

Thus, our metric suite identifies a subset of classes, which have high complexity/coupling counts. In our case studies, these classes are the most important computational object classes. Also, these classes can be the most important information object classes in the system. The

single most complex classes in our experiments were the classes located either at the interfaces between the administrative domains (as in TRUMPET), or at the interfaces between the stand-alone management components (as in FlowThru). These classes were singled out by all of the metrics making up our metrics suite.

Hence, our case studies **empirically demonstrated that the highest risk for the systems' integral operation is exhibited at the major interconnection points**. In the integrity context, this argument was already raised in [UCL94] - however, it was not empirically justified. In the case of TRUMPET, the two classes operating at the X interface between the VASP and the PNO domains - that were singled out as the most complex in design - did prove to be the most difficult to design, implement and test, and were the main source of pitfalls during the project trials.

The shortfall of our case studies is that they do not fully illustrate how the integrity-metrics policy fits, in practice, in the integrity methodology framework. Since the trial systems were assessed *post-facto*, no integrity-preserving actions could be taken during development, as recommended by the integrity methodology framework. The metric suite was thus used simply as an analysis tool for assessment *i.e.* diagnostics of the most complex classes. The discriminative power of the metrics was used to pin-point the highly complex/coupled classes, which were then labelled as the classes with lowest integrity, or highest risk, according to our prediction rationale described in section 5.1.2.

We believe that the integrity-metrics policy developed in this chapter has a number of attractive features. The main advantage of this policy lies in the fact that minimising the risk to integral operation early in the telecommunications system development lifecycle (analysis/design stage) would greatly reduce the cost of removing risks at the later stages of system implementation, testing, interconnection and maintenance. Through the case studies involving management systems, we demonstrated that the metrics can be used as early risk/integrity indicators. We also demonstrated that the metrics are easy to calculate, on the basis of established diagrammatic techniques (UML) and system development methodologies. Finally, the integrity policy developed in this chapter fits smoothly in the overall integrity methodology developed in chapter 3, and strongly supports the measurement requirement as prescribed in section 3.2.1.1.2.

6 MANAGEMENT INTERCONNECTION TESTING

This chapter presents the second integrity policy developed in the course of the research work: the inter-domain management system interconnection testing policy. This policy was developed specifically in relation to the TRUMPET project - however, some concepts are envisaged to be applicable in other scenarios of management system interactions as well. Effectively, the work carried out for this policy encompasses review and development of testing principles for the management interconnections, focusing on the integrity issues.

In section 6.1, we present the testing policy, in the context of the TRUMPET project. In section 6.2, we present the application of this testing policy to the TRUMPET inter-domain management system. This case study was restricted due to limited time and resources, and thus has a form of a brief illustration of the implementation. Section 6.3 concludes the chapter, and highlights the research contributions.

Some of the material in this chapter has been published in [Prnj98a], [Prnj99c] and [Prnj00a].

6.1 TESTING: THE POLICY

This section introduces the testing policy aimed at the integrity aspects of the management systems' interconnection across administrative domains. This policy was developed in the context of the TRUMPET project: thus, the core theoretical discussion is also focused on TRUMPET.

Since this policy is tightly coupled with the TRUMPET project, the structure of this section slightly differs from that of our first integrity policy description in the previous chapter. Section 6.1.1 re-visits the background in integrity-related testing, and addresses the open issues in the management systems' interconnection testing. Section 6.1.2 addresses the integrity requirements as related to the inter-domain management system interconnection: in the context of the integrity methodology presented in chapter 3, this is the integrity analysis. Section 6.1.3 then elaborates on the testing approach/policy developed to support these integrity requirements: in the context of the integrity methodology, this is the integrity design. Section 6.1.4 compares our testing policy to the OSI-SM recommendations which focus on testing. Section 6.1.5 briefly summarises the theoretical work presented.

6.1.1 TESTING BACKGROUND

This section gives a brief overview of the testing background. Section 6.1.1.1 summarises the state of the art in core network and control plane testing, while section 6.1.1.2 addresses the issues in management systems testing, with the emphasis on the OSI-SM testing principles.

6.1.1.1 CORE NETWORK AND CONTROL PLANE TESTING

It is generally accepted [Ward95] that the network interconnection increases the threat to network integrity (see chapter 1, section 1.1). As discussed in chapter 2, section 2.2.1, the main focus of the industrial integrity assurance is thorough testing of the system/service prior to operational launching. Regarding the integrity-oriented testing of the core transport network and the control plane, Bellcore leads the way in the US with its Network Services Test System (NSTS). This test bed is used for auditing and testing the stand-alone systems, as well as for testing multi-supplier equipment interoperability, and inter-network interoperability. Another industry-led initiative in the US, the Inter-network Interoperability Test Plan (IITP) [Lewi94], focuses on the interoperability testing for interconnected Common-Channel Signalling (CCS) networks. The interconnect testing as carried out in BT encompasses: systems conformance testing (software or hardware) to SS7 standards, interworking/interoperability testing in test environments, and commission testing (involving functional testing of software/hardware as new routes/circuits are introduced) [Maso97]. The interworking/interoperability testing takes place in the test environment: the BT integration facility (test network). Test environments for the core network and the control plane, such as NSTS and BT test network, are seen as of high importance for integrity assurance. Apart from accommodating a wide range of elements and scenarios that the new systems/services can be tested against, these test-beds effectively remove the integrity risks that can be introduced if the systems/services are tested against other "live" systems.

In our integrity methodology presented in chapter 3, testing is a distinct phase of the methodology, viewed as of high importance, particularly in the multi-domain environments where interconnection is taking place. Testing in our methodology encompasses a number of stand-alone system testing stages, followed by the intra-domain and inter-domain integration tests (chapter 3, section 3.2.2). The inter-domain testing, exercising the behaviour of the operational system when interconnected with the systems in other autonomous domains, is seen as the crucial testing stage.

6.1.1.2 MANAGEMENT INTERCONNECTION AND OSI-SM TESTING PRINCIPLES

In the context of the management systems' inter-domain communications, the critical point of the interconnection is the Telecommunications Management Network (TMN) X interface,

such as Xuser between a Value Added Service Provider (VASP) and the Public Network Operator (PNO) and Xcoop between the two PNOs (or VASPs).

As discussed above, integrity-related testing for core and control network interconnection is relatively established in industry. In contrast, there is little evidence of management system interconnection testing with respect to integrity features. Some ACTS projects, such as MISA, performed a certain level of X interface testing, but only restricted to the provision of management functionality [MISA-D9].

On the other hand, OSI - Systems Management specifies two recommendations focused on testing: [X.737] and [X.745]. [X.745], Test Management Function, specifies a model and generic managed objects for the invocation of tests on real remote resources included in an open system. Test conductor is an entity initiating the tests and the test performer is an entity which applies the tests. The Test Objects (TOs) facilitate controlled test invocation, while the Managed Objects Referring to Test (MORTs) are objects used to refer to particular functionalities, of the real resources, which are being tested. [X.737], Confidence and Diagnostic Test Categories, specifies a range of tests focused at the generic aspects of testing of the real resources. In this context, a number of test are specified, targeted to test the generic aspects of the real resources: echo, loop-back, data integrity, resource, and protocol integrity.

In the following sections we develop an integrity policy focused on testing the integrity features of the management systems' interconnection over the X interface. The policy is developed in the context of the TRUMPET project, and to some extent overlaps with the testing principles defined in [X.737] and [X.745] – however, our policy was developed independently of these existing recommendations (for details see 6.1.4).

6.1.2 X INTERFACE INTEGRITY REQUIREMENTS: ANALYSIS

As discussed in chapter 4, section 4.1, the TRUMPET project focused on the inter-domain security of management. As stated in chapter 3, section 3.1, security is a sub-attribute of integrity: security threats can jeopardise the correct and proper operation and thus the integrity of the system. In contrast, other integrity-related attributes concerning management systems' operation and communications between system components were not considered in the project. Considering this fact, we can group the integrity requirements on the X interface in two sets:

- First set comprises the *security* measures between autonomous organisations within the TRUMPET management system. Security measures must be in place to avoid malicious human intervention and illegal use of resources by a party.

- The second set comprises those requirements not considered in the main line of the project: liveness, performance, availability, *etc.* We collectively refer to these as *communications integrity requirements*, since they focus on the integrity of the communications mechanism and infrastructure supporting the interactions between the management systems in autonomous domains.

Now, we can further elaborate on these two sets of requirements via the three-dimensional analysis space defined in Figure 12, page 53, chapter 3, section 3.2.1.2. Note that the analysis of the security requirement is reverse-engineered, since our study was conducted late in the project when the security mechanisms were already developed.

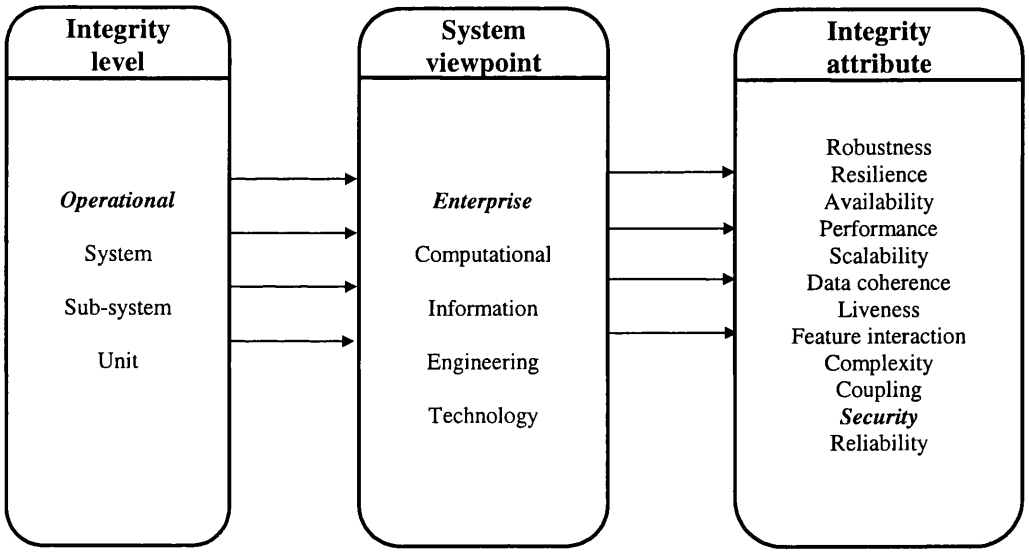


Figure 75 - Integrity requirements classification: security

The security requirement is primarily tackled at the operational level, since it concerns the system interactions within its operational environment. Following the same rationale, this attribute is located within the enterprise viewpoint: see Figure 75. The security requirement identified in the enterprise viewpoint at the operational integrity level can be further refined during the system implementation at the lower integrity levels. Five basic security sub-requirements are:

- Authentication, which refers to the mutual recognition of the communicating parties.
- Access control to managed objects, which ensures that the managing party can access only a certain set of objects on the agent party side, according to the contract.
- Data integrity, meaning that the management data must be protected against modification, insertion, and repetition.
- Confidentiality, meaning that the management data content must not be disclosed, while in transit, to unauthorised parties.

- Non-repudiation, providing mechanisms for the resolution of the dispute where one party denies that communication took place.

These security requirements can be further analysed as shown in Table 21. The detailed analysis of the security topic can be found in chapter 4, section 4.1.1.

Integrity level	System viewpoint	Security sub-requirement
Operational	Enterprise	Authentication
Sub-system	Enterprise	Access control
Sub-system	Engineering	Data integrity
Sub-system	Engineering	Non-repudiation
Sub-system	Engineering	Confidentiality

Table 21 - Security sub-requirements classification

The communications integrity requirements are considered at the operational level (since they are concerned with the interactions between autonomous domains) and, being focused on the communications mechanism itself, appear in the engineering and to some extent computational viewpoints (Figure 76). Correct and high-integrity operation is essential between the OSs over the X interface. This means that the management systems and communications infrastructure have to retain their correct attributes in terms of functionality and performance. We can further elaborate on the communications integrity requirements as follows:

- **Liveness** of the communications mechanism and infrastructure supporting the interactions between parties in autonomous domains has to be preserved. Situations such as deadlocks (system being blocked awaiting a message that cannot be emitted) and livelocks (system oscillating between a certain number of states that it cannot leave) have to be avoided, so that **availability** is maintained.
- **Robustness:** the management applications need to be able to handle all possible states of their environment: unexpected messages, duplicate messages, *etc.*; *i.e.*, the management system needs to be robust and stay operational in these circumstances.
- **Sequencing:** proper sequencing of actions has to be preserved over the management communications mechanism.
- **Data Integrity:** the data exchange between two OSs has to be correct. Data should not be corrupted or lost by the communications stack and software, and conversely, performance of the communications mechanism should not be influenced by the data content. Thus, the data integrity requirement is a performance-related requirement as well as a functional one.

- **Time sensitive performance:** timing of the operations has to stay within well-defined limits - the communications mechanism between two OSs should not jeopardise the correct timing. Increased response time may cause processes to slow down, collapse, or decrease availability.
- **Throughput/Rate:** the communications mechanism has to support a certain throughput; and the rate of signal exchange should not in any way impact the operation of the applications.

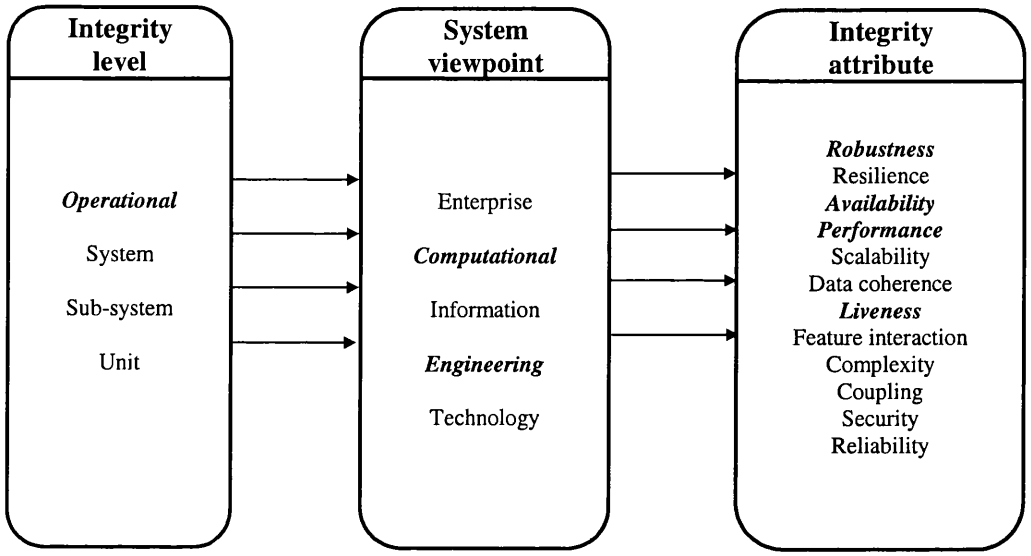


Figure 76 - Communications integrity requirements classification

The above requirements can be grouped into two sets: liveness, robustness and sequencing are focused on the functionality aspect of communications integrity, while the timing and throughput are performance aspects. Data integrity requirement is concerned with both, as outlined above.

If the management system (and the supporting communications mechanism) loses its liveness, starts performing outside its time limits, or cannot deal with unexpected messages; or if sequencing is distorted or data exchange is corrupted, then system performance is degrading and functionality might be lost. In other words, the integrity of that system is at stake.

6.1.3 INTEGRITY DESIGN: INTEGRITY POLICIES

6.1.3.1 OVERVIEW

The two sets of X interface integrity requirements identified during analysis give rise to the development of two integrity policies.

The first policy is *security* for the TMN X interface. The security policies were central to the TRUMPET project, and as such were neither central to our work, nor developed through our integrity methodology presented in this thesis - the integrity analysis given in the previous section was only an illustration. In TRUMPET, security policies [Mail96] were implemented through the security architecture consisting of a set of security modules performing mutual authentication, access control, and data integrity [Gagn97] [Ølne97]. Security mechanisms (residing in the Security Support Component, SSC) are implemented so that they are conceptually used as a part of the stack (CMIS [X.710]) by the applications, while effectively being implemented above the stack, thus being an OSI layer-7 add-on feature to the TMN applications (Figure 77). This means that security may be turned on and off during management communication.

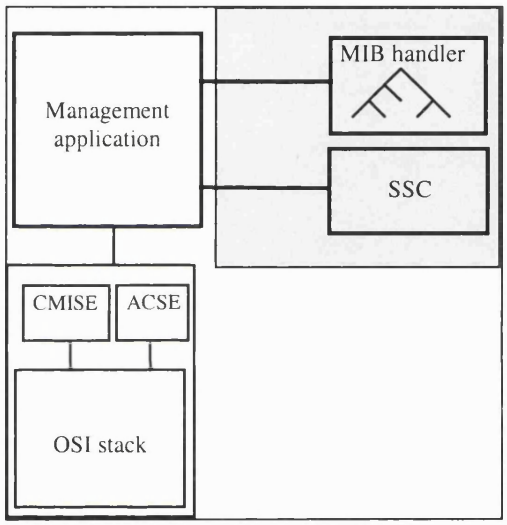


Figure 77 - TRUMPET security architecture: overview

In the framework of our integrity methodology, security policies belong to the *prediction* phase, since they were implemented during system development, and effectively give rise to the implementation of an *integrity-preserving mechanism* (using the classification given in chapter 3, sections 3.2 and 3.2.1.3, respectively). For the details of the security policies and the security architecture, refer to the TRUMPET security section (4.1.1) in chapter 4.

The second policy, based on the analysis outlined in the preceding section, was developed to check whether the communications integrity requirements over the X interface were satisfied both with and without the presence of security mechanisms. This policy is to be applied prior to the interconnection of autonomous TMN OSs, and it is based on a testing regime to be deployed as a communications integrity verification policy. Effectively being an interconnection *testing integrity policy*, in the framework of our integrity methodology (for the terminology refer to chapter 3), this policy belongs to the *testing phase* of the methodology. This policy is the second integrity policy developed in the course of the research work presented in this thesis.

The following section describes the design and implementation of this testing integrity policy in detail, while in section 6.2 we present its application in TRUMPET. In the following discussion we focus on the Xuser interface between the TRUMPET VASP and the PNO: however, all the concepts are targeted to apply to any X interface between two administrative domains.

6.1.3.2 TESTING APPROACH: DESIGN

6.1.3.2.1 Testing requirements and methodology

The interconnection testing methodology must exercise and establish that both communications integrity and security requirements are fulfilled over the Xuser interface [Prnj98a]. They might also be conflicting: some security mechanisms require a pre-transaction overhead, availability of specific information at both sides of the association, *etc.* The lack of synchronisation and coherence of the security-related information, or the introduction of significant security overhead, might disrupt the basic operation, and degrade the performance of the management system - it can jeopardise its integrity status. Even if the functionality is preserved, the possible degradation of performance of the management system can prove to be a costly drawback of the introduction of security policies. Thus, the first aim of testing is to establish that security and communications integrity requirements are not conflicting, *i.e.*, that the behaviour of the interaction has not changed after the implementation of the security policies, and that the performance is not significantly effected. This requires the system behaviour to be perceived in a concrete way so as to establish the baseline stand-alone behaviour that might be compared to the behaviour when the security policies are introduced. The second aim of testing in TRUMPET is the proof that the security policies themselves are correctly implemented, and function under both normal circumstances and security breaches. Our integrity policy focuses on testing the impact of introduction of security mechanisms on communications integrity requirements. There are three phases within this aspect of testing.

- **Phase 1:** the basic behaviour of the applications' interaction over the Xuser interface must be established, in terms of management infrastructure and support object functionality.
- **Phase 2:** this behaviour must be tested over the communications mechanism, so as to ensure proper functioning of the communications (without security) and satisfactory performance.
- **Phase 3:** it must be shown that the introduction of the security mechanisms in different domains does not jeopardise the communications integrity requirements - the proper and correct functioning of the management system communications infrastructure. It has to be shown that the introduction of security mechanisms does not push the system outside

its time limits, that it does not distort the sequencing of the messages, that it does not cause deadlock or livelock situations, *etc.* The basic *behaviour* of the interaction over the Xuser interface established in phase 1 must be *preserved.* Also, the performance must be satisfactory. These two factors would imply that the basic level of communications integrity is preserved.

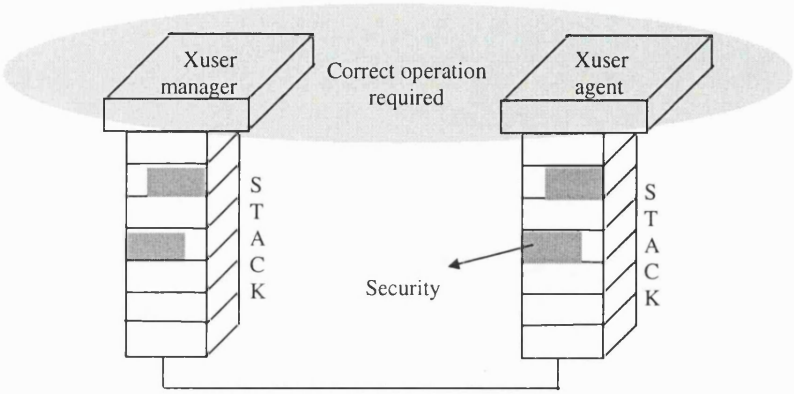


Figure 78 - Correct operation to be preserved

Figure 78 illustrates this. The aim of this phase of testing is to demonstrate that the functionality of the management infrastructure (stack) has not changed after the introduction of security (shaded boxes), which is conceptually done within the stack. If the behaviour of the interaction of two management entities can be established, and if it can be shown that this behaviour has not changed after deploying security in the stack, the task is accomplished. Also, performance of the communications mechanism, recorded during phases 2 and 3, would give a level of understanding of the impact of introducing security in the Xuser interface implementation.

6.1.3.2.2 Phase 1 - specifying the Xuser interface behaviour

The behaviour of the Xuser interface is specified through the Xuser Test-MIB (Management Information Base), a set of managed test-objects on the agent side. The Xuser Test-MIB can be used to implement basic, abstract, and finite behaviour visible to a party acting through a Xuser interface. This name was chosen since the focus is on the manager/agent interaction that can be seen, in the true management sense, as a manager controlling a set of objects through an agent - this set being a part of the shared management knowledge. The Xuser Test-MIB can thus be implemented as a set of managed object classes representing some typical behaviour, accommodating the integrity requirements defined in section 6.1.2. Such a behavioural envelope can be presented to an application as a simulation of the basic Xuser interface behaviour.

The place of the Xuser Test-MIB in the TRUMPET manager-agent chain (for the TRUMPET details refer to chapter 4, section 4.1.2) is shown in Figure 79 (extended from [M.3010]). The hollow objects represent the objects of the Xuser MIB, *i.e.*, the information model provided by system B (PNO) to system A (VASP). In the real world, some operations performed by system B on the managed objects in its MIB (on behalf of the system A) will involve further operations on the objects in system C, while others will not. This will depend on the actual MIB configuration in the agent model. The idea of the Xuser Test-MIB (dark objects) is to implement a few objects that can simply imitate the behaviour of the system B's information model, instead of representing the actual MIB. The manager calls on the Xuser Test-MIB do not propagate further as those on the Xuser do (Figure 79). These calls are processed in the "behavioural" OS. The behaviour of the Xuser Test-MIB objects is a superset of the possible behaviour of the Xuser. Thus, Xuser Test-MIB can be used without interfering with the operation of the "live" system. In this sense, the Xuser Test-MIB can be seen as the management-level equivalent of the established interconnection test-beds (such as the Bellcore NSTS or the BT test network) for the core network and the control plane.

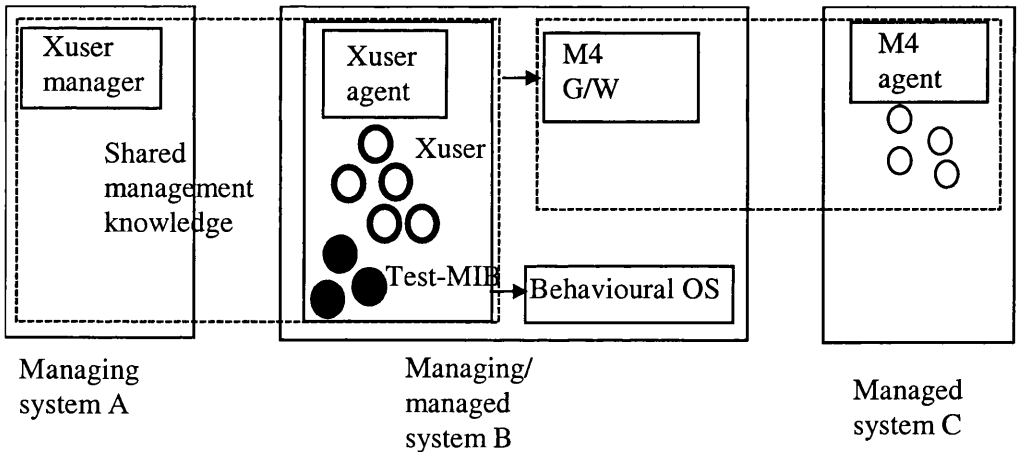


Figure 79 - Xuser Test-MIB and the manager-agent chain

The typical way of representing behaviour is by modelling it using the Finite State Machine (FSM) approach. This approach is based on modelling behaviour as a set of states that the system can be in. The transitions between these states can occur as a response to external stimuli, such as the FSM receiving a particular signal (in our case, the CMIS M-Get, M-Set, M-Action calls) while being in a correct state, or as a response to internal stimuli, such as a particular operation within a FSM being completed or an internal timer timing out.

Possible inconsistencies in the operation of a system modelled by a FSM can occur when, for example, a system is in a state where it cannot respond to a particular input signal - an external stimulus. This can lead to the malfunction of the system, since no behaviour is specified as a response to that input signal. Now, the sender of that signal might be expecting

the receiving system to be in a state in which it would be if it did respond to that input signal. Also, if a system is in a state, or oscillating between a number of states, awaiting a signal that can never be received, its operation is corrupt - the system is in a state of deadlock or livelock, respectively. If a system is designed in a such way that all possible inconsistencies are avoided, it is robust to failure. This is optimistic, especially for the time-dependent system where the transitions between states may depend on correct timing of external and internal stimuli. In any real-time system of considerable level of complexity, a wide range of problems will arise and the correct modelling of behaviour will be of paramount importance.

As pointed out in the above discussion, the two main issues in behavioural modelling are:

- Ability of a system to deal correctly with input signals while being in a particular state.
- Timing of operations performed by the system and timing of transitions between the states.

These two factors were taken as a baseline for the abstraction of the Xuser Test-MIB objects' behaviour. Five basic behavioural patterns were identified. These are shown in Table 22, compared to the integrity requirements they address and the corresponding concrete behaviour of the Xuser.

Behavioural pattern	**Integrity requirement**	**Example corresponding Xuser behaviour**
Various time delays	Time-sensitive performance	Modification of the connection parameters which takes variable amounts of time
Rate-critical behaviour	Throughput – rate	Set of successive operations on the Xuser
Various values of input signals	Data integrity	Various values of parameters of the Xuser data
Time-critical behaviour	Timing-sequencing-liveness	Set of concurrent operations on the Xuser
Sequencing of operations performed on the Xuser Test-MIB	Correct sequencing	Well-defined sequence of operations needed to reserve a connection

Table 22 - Behavioural patterns

The implementation of the Xuser Test-MIB is shown in Figure 80. The behaviour of the Xuser Test-MIB objects is provided to the Xuser agent through an Application Programming Interface (API). This behavioural code can be seen as a stand-alone TMN OS (a "behavioural" OS). The behavioural code is thus independent of the platform, and accessed through the API via defined operations. Xuser Test-MIB objects (defined in GDMO [X.722]

and ASN.1 [X.208]) implement a set of actions which represent calls to the Behaviour-API. So, if a CMIS M-Action call is received, it is translated into a call to the Behaviour-API. This can also be done for other CMIS calls, such as M-Set and M-Get.

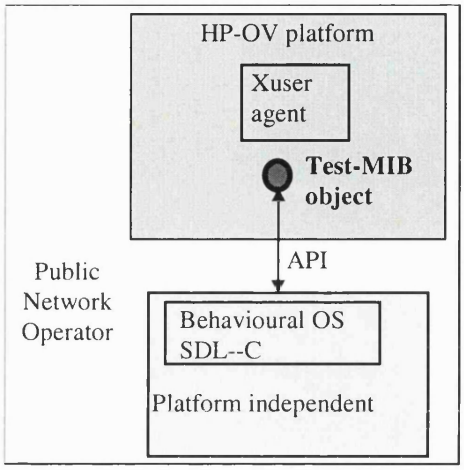


Figure 80 - Xuser Test-MIB implementation

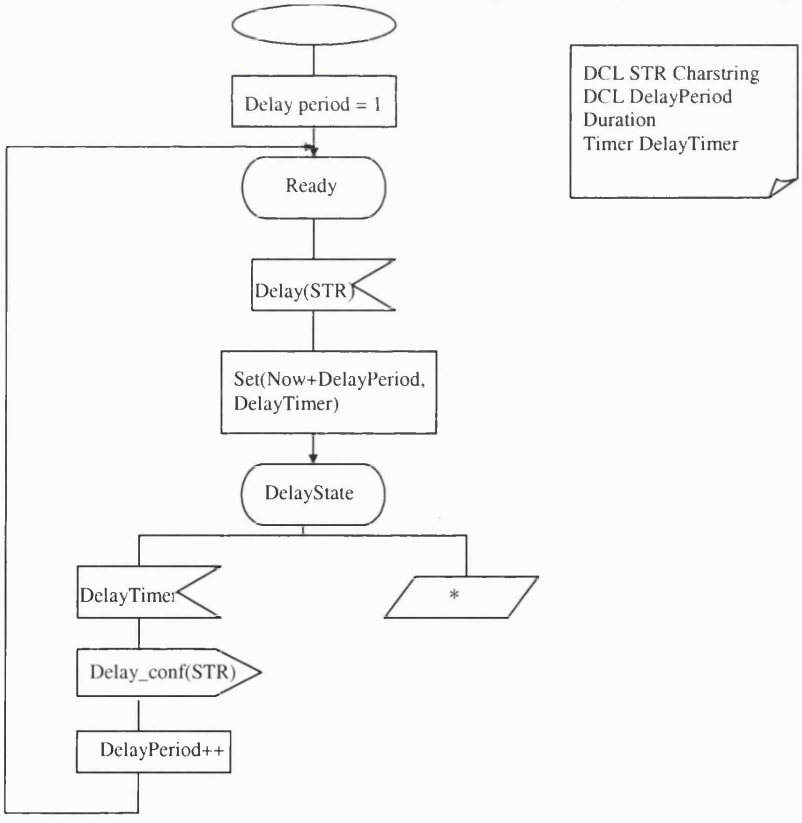


Figure 81 - SDL behavioural pattern: various time delays

The behaviour at the specification stage is defined using UML [UML] state diagrams, and the Specification and Description Language (SDL) [Z.100], while the implementation is done in the C [Kern88] programming language, since both manager and agent applications in

TRUMPET were implemented using C. An example behavioural pattern described in SDL is shown in Figure 81. This pattern describes the case where the Xuser Test-MIB exhibits incrementing time delays (the first behavioural pattern from Table 22). This situation reflects the concrete behavioural pattern of the Xuser, the case when the modification of connection parameters takes various amounts of time to be completed.

6.1.3.2.3 Phase 2 - testing without security

After establishing the basic behaviour on the agent side, the next step is to develop a set of test cases initiated from the other side of the Xuser interface to test this behaviour over the stack. The testing configuration shown in Figure 82 classifies, using the conformance testing methodology [ISO9646-1] terminology, as the remote testing method. The only Point of Control and Observation (PCO) is situated on the Xuser manager's API (marked in black). This test-API provides an interface similar to the CMIS M-Action interface, which enables the test M-Action calls to be invoked on the Xuser Test-MIB objects on the agent side.

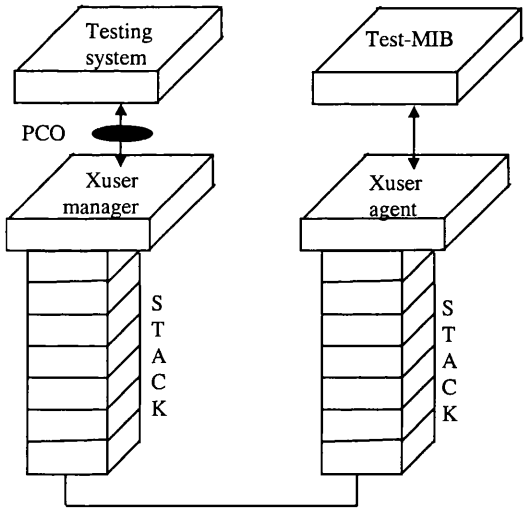


Figure 82 - Testing configuration (without security)

Knowing the behaviour of the Xuser Test-MIB (Table 22, page 164), a set of test cases targeted to exercise this behaviour over the management support machinery and the stack is specified in the Tree and Tabular Combined Notation - TTCN [ISO9646-3] language. The behavioural part of the test cases is derived from the Xuser Test-MIB specification. Hence, when running the test, both the ordering of the events to be observed at the PCO and the timing of these events can be established and measured. These two dimensions of the test cases give an observational framework for testing the behaviour and quantitatively comparing the performance when the security mechanism is active. Applying the set of test cases on the Xuser Test-MIB over the communications stack thus provides two sets of information. First, it ensures the proper and correct functionality of the communications stack (without security). Second, it allows the measurement of the time-related performance

parameters, such as delay. Thus, quantifiable integrity-related information, based on functionality and performance, can be gathered.

6.1.3.2.4 Phase 3 - testing with security

Having tested the correct functioning and measured the management infrastructure performance, the next step is to prove that the introduction of the security mechanism does not adversely effect the behaviour of the management system and significantly degrade the performance. *I.e.*, the addition of the security mechanism should not jeopardise the communication integrity requirements over the Xuser interface: it should not change the behaviour of the Xuser interface. The system should still perform within its time limits, and its behaviour should stay the same: liveness should be maintained; sequencing of particular operations should not change; data integrity should be preserved; and performance should not degrade significantly.

The TRUMPET security mechanism consists of a set of security functionalities added below the management functionality. Thus, the management calls are expanded to the level of the secure management calls. The testing configuration still has the structure as on Figure 82, only now the manager and agent applications are effectively expanded with extra security functionality.

The proof that the stack functionality and performance did not change with the introduction of security is done by executing the test cases from the phase 2 testing step (postulating that the security is transparent to the management applications, which is the case in TRUMPET), and by observing the verdicts and analysing the results. The behaviour (ordering of events at the PCO) has to stay the same as in phase 2, and the system should not block. The timing, however, can be different, since the security mechanism is expected to introduce some delay. Only if all the test cases established in the phase 2 are passed successfully, can it be stated that the behaviour of the Xuser did not change with the introduction of the security mechanism.

6.1.4 COMPARISON WITH OSI-SM TESTING PRINCIPLES

Some of our principles developed in the preceding sections are related to the testing concepts specified in [X.737] and [X.745] (discussed in 6.1.1.2): however, our testing policy was developed independently.

These recommendations specify a testing framework and a range of tests focused at the generic aspects of testing of the real resources. The tests, including echo, loop-back, data integrity, resource, and protocol integrity, are targeted at testing the real resources in the

open environments – to quote, "a suspected cable break" [X.737]. A set of generic test objects is specified to allow for testing of these particular functionalities of real resources. In contrast, in our testing regime we aim, through the Xuser Test-MIB, to abstract out the behaviour of the Xuser interface and test it over the stack so as to verify the correct operation both with and without security.

Our communications integrity requirements, identified in 6.1.2, to some extent match the generic tests on real resources, specified in [X.737]. Thus, instead of using the Xuser Test-MIB and our associated concepts, an alternative way for partially testing our integrity requirements over the TMN X interface would be to apply the principles specified in [X.737] and [X.745] to the X interface, which would then be seen as just another real resource.

Thus, our testing policy can be seen as an alternative approach to the OSI-SM testing principles. However, it has a number of additional features. These include abstraction of the behaviour through the Xuser Test-MIB, protection of real resources during X interface testing (as discussed in 6.1.3.2.2), relative simplicity, and platform independence (achieved through the "behavioural OS", as discussed in 6.1.3.2.2). We believe that our approach is a valid alternative to the OSI-SM testing principles, and that it adds to these a direct focus on the issues related exclusively to integrity of the most critical management interface.

6.1.5 SUMMARY

In the preceding sections we presented an approach for testing the integrity requirements over the TMN X interface - the interconnection point between two autonomous management systems. This integrity policy is developed in the context of the TRUMPET project.

First, we reviewed the current approaches for integrity-related interconnection testing. For the core/control network, the test environments enable the pre-launch integrity assurance. The management system interconnection testing is not as advanced, while recommendations exist specifying generic testing functionalities, in OSI-SM environments, targeted at real resources.

Then, following our integrity methodology, we conducted the integrity analysis and the integrity design for the TMN X interface. The integrity analysis identified two sets of requirements, both through our three-dimensional analysis space: *security* and *communications integrity* requirements. The latter encompass liveness, robustness, sequencing, timing, throughput and data integrity. The two distinct sets were formed since the security requirements were considered in detail by the TRUMPET project, and as such were not of direct interest for our research.

The integrity design involved the development of two integrity policies. The security requirements were met through the definition of the security architecture, which was developed by the TRUMPET project. This policy is thus not of the central interest for this thesis. The second integrity policy, the interconnection testing policy, was developed to exercise and establish that the communications integrity requirements are satisfied both with and without security mechanisms deployed. The policy encompasses three phases:

- Definition of the behaviour of the Xuser interface on the agent side via a number of behavioural patterns accessible through the 'test' managed objects - the Xuser Test-MIB.
- Testing of this behaviour from the manager side, without the presence of the security mechanisms, so as verify that the communications integrity requirements, in terms of performance and functionality, are satisfied.
- Repeating the tests with security switched on, to verify whether the communications integrity requirements are still satisfied and to measure any degradation of performance.

In the following we present the application of our testing policy to the TRUMPET Xuser interface.

6.2 CASE STUDY: TRUMPET

6.2.1 APPROACH

The TRUMPET management applications (VASP and PNO), have been implemented as single-threaded, and communicate in a fully blocking fashion over the Xuser interface. Hence, neither the performance integrity requirement of the throughput/rate nor the functional integrity requirement of sequencing are applicable in this case.

Thus, only two behavioural patterns out of five given in Table 22 (page 164) were implemented: the Xuser Test-MIB exhibited various time delays in the first case (Figure 81, page 165), and was made to be sensitive to the values of the input signals in the second case. These behavioural patterns were implemented in the behavioural OS accessible through the Behaviour-API of a single Xuser Test-MIB object (Figure 80, page 165). The behavioural patterns, as discussed in section 6.1.3.2.2, were specified in SDL - using the Telelogic Tau tool SDT 3.5 [Tau3.5], and implemented in the C [Kern88] programming language. Although this tool had the power of translating SDL specifications into C, this was not done, since the code produced by the tool was very intricate and proved to be difficult to integrate with the agent application. Since both SDL specification code and C code written from scratch were relatively simple, we feel that the consistency between the specifications and implementation was not compromised.

The test-cases were specified in TTCN and implemented in C. Although the testing tool, Telelogic Tau ITEX 3.5 [Tau3.5], could translate the TTCN test specifications into C, this was not done, for the same reasons as outlined above for the SDL tool. The test cases implemented had two purposes. First was the demonstration of the robust operation and functionality, in terms of data integrity and liveness. The second purpose was to get a feel of the impact of security features on the Xuser interface performance. However, since the statistical conditions were not stable, the measurements were not considered as representative. Instead, the measurements are presented simply in the graphical form, as an illustration.

Generally, the application of the testing policy to the TRUMPET system was highly restricted due to limited time and resources, and thus has a form of a brief illustration. Note that in what follows we refer collectively to the TRUMPET security mechanisms as the "security package".

6.2.2 ASSESSMENT

The functionality, in terms of liveness and data integrity, was preserved both with and without the security mechanism deployed. Data was not corrupted or lost by the communications stack and software. The system was live at all times, independent of the data size or content. Performance was captured in an illustrative way on a number of levels: the overview is given in the following.

200 samples of the time taken to perform the management association with and without mutual authentication between the test manager and the test agent were taken. Figure 83 shows two curves: the top one depicts the management association establishment delays with authentication, and the bottom one the delays without authentication. The association delays for the secured management association are not just considerably larger, but also the fluctuations seem to be remarkably more drastic.

500 samples of the time taken to perform the secured and the unsecured management operation (M-Action, with the simple one-string parameter) between the test manager and agent were taken. Figure 84 shows the delays for the secured management operations (top) and the delays for the unsecured ones (bottom). Delays for the secured management operations are higher, and the fluctuations are larger.

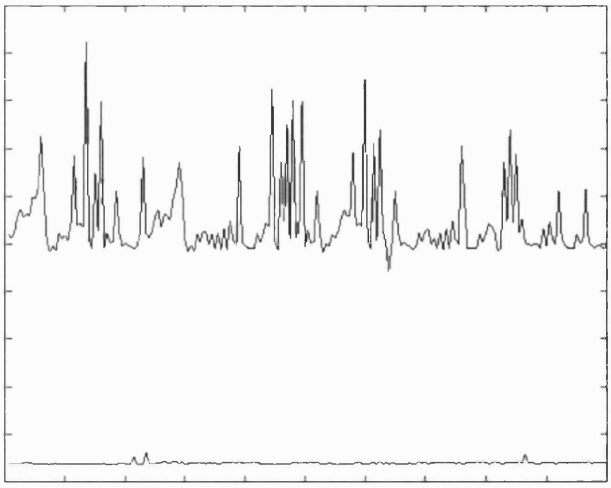


Figure 83 - Association delays with and without authentication

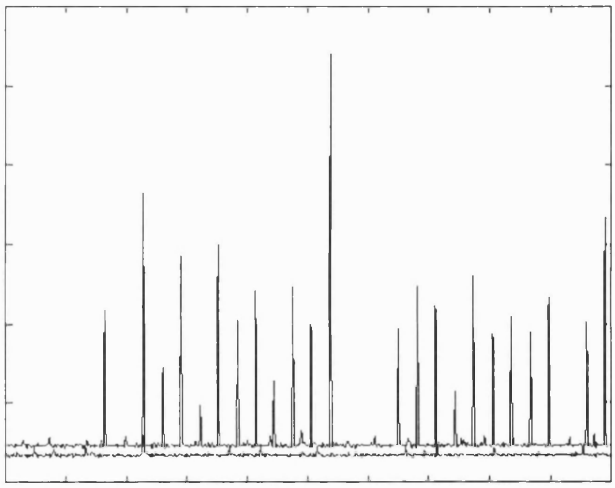


Figure 84 - Operation delays with and without security

Next, the delays exhibited when performing management operations, which take various amounts of time to be completed by the agent, were recorded. The aim was to detect possible agent-delay dependent jitter, resulting from differing processing times, both with and without security. The agent (*i.e.*, the Xuser Test-MIB) was implemented so as to exhibit 500 delays, in even increments. The overall delay was recorded on the test-manager side. If there was no agent-delay dependent jitter, the expected delay would be equal to the sum of the management operation delay and the Xuser Test-MIB in-built delay, and it would be equal to the recorded delay. The recorded delay followed the expected. Thus, the agent-simulated delay did not seem to influence the performance of the management communications mechanism, both with and without security.

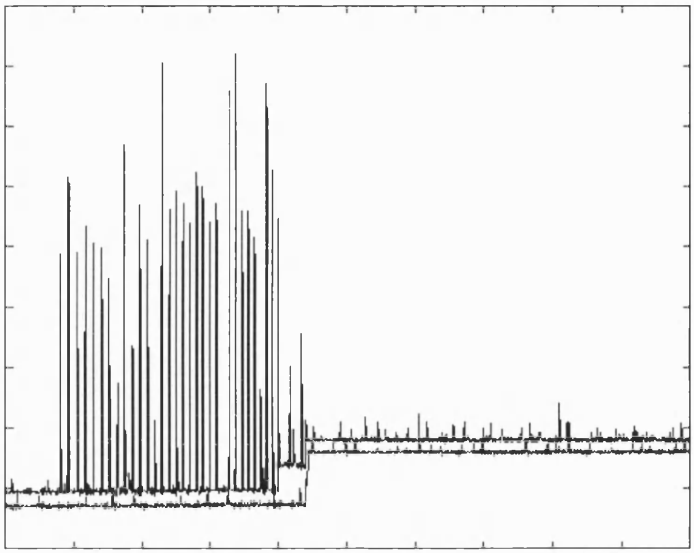


Figure 85 - String-length dependent delay for secured and unsecured communications

Finally, effect of different operation argument lengths on the performance of the communications mechanism, in terms of the string-length-dependent delays, was captured. The test case was run 1000 times, with string lengths 1 to 1000, both over the secured and unsecured management communications. Data integrity was preserved in both cases: the communications mechanism did not corrupt the data, and liveness was preserved. The delays for the secured and unsecured management operations are shown in Figure 85 (top and bottom curves, respectively). In both cases the delays increase steadily as the string length increases, and there is a step increase in the delay as the string length reaches 440. This increase is possibly due to the memory allocation on the management platform on which the implementation was running. Before the increase, the fluctuations for the secured operations are as drastic as for the fixed-length parameter secured management operation delays discussed before. However, after the increase in the delay, the fluctuations settle down, becoming comparable with the unsecured operations case.

6.2.3 DISCUSSION

In the previous sections, we presented an application of our interconnection testing policy to the TRUMPET Xuser interface. Generally, the case study was highly restricted due to limited time and resources, and thus has a form of a brief illustration.

The application of the proposed approach was also restricted due to the fully synchronous, blocking manager-agent communications between the TRUMPET domains. Thus, not a full set of behaviours was developed. The test cases implemented had two purposes. The first purpose was the proof of the robust operation and functionality, in terms of data integrity and liveness. The second purpose was the visualisation of the impact of the security mechanisms on the performance of the management communication mechanism.

The functionality, in terms of liveness and data integrity, was preserved both with and without security mechanisms deployed. Data was not corrupted or lost by the communications stack and software. The system was live at all times, independent of the data size or content. Performance was captured on different levels. However, since the statistical conditions were not stable, the measurements were not considered as representative, and were used only for comparison. The association and operation delays were compared with and without security deployed. The delays are not just comparatively higher in the secured case, but also arbitrary fluctuations in the delays seem to be appearing. Also, effect of the increasing operation argument length on the delays was recorded: in both secured and unsecured cases, there is a step delay increase when string length reaches 440. In the secured case, the fluctuations settle down after the step increase.

Overall, the results of the case study show many anomalies. Although the apparent arbitrary delays seem to be a feature solely of the security package, the step increase appears also in the off-the-shelf software (HP-OpenView). The anomalies thus could be a compound effect of the management platform, security software and the testing software. Tracing back these flows would require a lot of effort. Moreover, the design improvements could not be conducted, since the experiment was carried out during the final stages of the project. The core achievement of the case study would thus be that we demonstrated how complex inter-domain interactions might appear sensitive to the introduction of additional sophisticated features.

6.3 CHAPTER SUMMARY AND RESEARCH CONTRIBUTIONS

In this chapter, we presented an integrity policy focused on the testing of the TMN X interface integrity requirements. This policy was developed in the context of the TRUMPET project. Effectively, the work carried out for this policy encompassed review and development of testing principles for the management interconnections, focusing on the integrity issues. The application of the policy was briefly illustrated through the TRUMPET case study. In what follows, we summarise the research work, and highlight our achievements in bold.

The theoretical part of the work focused on the development of the management system interconnection testing policy. This activity included:

- Assessment of the current state of the art in interconnection testing.
- Identification of the **integrity requirements for the TMN X interface**, in the framework of the **integrity methodology.**
- **Detailed design of the testing integrity policy.**

The last two steps effectively represent integrity analysis and integrity design in the framework of our integrity methodology developed in chapter 3. The integrity policy developed, using the terminology defined in chapter 3, has the form of a *testing recommendation* and belongs to the *testing phase* of the integrity methodology.

First, we reflected on the current approaches for integrity-related interconnection testing. For the core/control network, the industrial test environments enable the pre-launch integrity assurance, by providing a range of test elements/scenarios, but also by eliminating risk of testing against "live" systems. The management system interconnection testing is not as advanced, while recommendations exist specifying generic testing functionalities, in OSI-SM environments, targeted at real resources.

Then, following our integrity methodology, we conducted the integrity analysis and the integrity design for the TMN X interface. First, we **identified the integrity requirements over the X interface**. We grouped these in two sets: *security requirements* and *communications integrity requirements*. The latter encompass liveness, robustness, sequencing, timing, throughput and data integrity. These two sets were made distinct since the security requirements were considered directly by the TRUMPET project, while the other integrity requirements were not. We demonstrated how these two sets of requirements can be **identified in the three-dimensional analysis space** defined in the context of the integrity methodology (chapter 3, section 3.2.1.2).

On the basis of these requirements two integrity policies were considered. The security policies were catered for in the TRUMPET project through construction of security mechanisms, and thus were not of central interest for our work. To cater for the communications integrity requirements, we **developed an interconnection testing policy**. This policy aims to exercise and establish that the communications integrity requirements are satisfied both with and without security mechanisms deployed. This is done in three phases. First, the behaviour of the Xuser interface is established on the agent side, through a number of abstract behavioural patterns accessible through the 'test' managed objects - the **Xuser Test-MIB**. Next, this behaviour is tested from the manager side, without the presence of the security mechanisms, to verify that the communications integrity requirements, in terms of performance and functionality, are satisfied. The third step involves running the same tests with security switched on, so as to verify whether the requirements are still satisfied and to measure performance impact of the introduction of security features in the implementation of the TMN Xuser interface and the supporting CMIS-based stack.

The central feature of this policy is the concept of the **Xuser Test-MIB**. Its implementation would allow the players involved in the interconnection to avoid exposing their real resources and particular information models during testing of the communications mechanism and the management infrastructure. Instead, they could use the Xuser Test-MIB and the "behavioural" OS as a test-bed that aims to provide a superset of the possible behaviours exhibited by the shared information model (Xuser interface). This approach distinguishes the stack and infrastructure testing from the application-specific testing (which would include the full, detailed testing of the information models and their particular behavioural aspects mapped 1:1 with these information models [Eber97]). The testing results would ensure the required level of integrity of the communications stack and infrastructure - the possible integrity-related behavioural problems would be restricted to the incorrect specification of the *particular* X interface information models. Thus, the Xuser Test-MIB can be seen as the first step towards the specification of the management-level equivalent of the established interconnection test-beds for the core network and control planes.

The testing policy developed here was applied to the TRUMPET Xuser interface. Generally, the case study was restricted due to limited time and resources, and thus had a form of a brief illustration. The case study was also restricted due to the synchronous, blocking manager-agent communications between the TRUMPET domains, and thus only a subset of proposed behaviours and test-cases was implemented.

The communication integrity requirements, in terms of liveness and data integrity, were preserved both with and without security mechanisms deployed. Different aspects of performance impacts of security mechanisms were captured. Since the statistical conditions were not stable, the measurements were not considered as representative, and were used only for comparison. The association and operation delays were compared with and without security deployed. The delays are not just higher in the secured case, but also fluctuations in delays are manifested. Also, effect of the increasing operation argument length on the delays was recorded: in both secured and unsecured cases, there is a step delay increase when string length reaches 440. In the secured case, the fluctuations settle down after the step increase. Overall, the results of the case study show a number of anomalies. Although the apparent arbitrary delays seem to be a feature solely of the security package, the step increase appears also in the off-the-shelf software (HP-OpenView). The anomalies thus could be a compound effect of the platform, security software and the testing software. Tracing back these flows would require a lot of effort, and the design improvements could not be conducted since the experiment was carried out during the final stages of the project.

The achievement of this case study was that we demonstrated how complex inter-domain management interactions might appear sensitive to the introduction of additional sophisticated features. The arbitrary delays appear to be a feature of the security package and, as such, could possibly have an impact on the integral operation of concurrent, real-life management applications communicating in an asynchronous fashion. We showed how the integrity requirements, especially security and performance, could be closely interlinked. As such, the integrity requirements must be considered throughout the system development lifecycle, starting from requirements capture down to implementation, to avoid not just possible inconsistencies in system operation, but also the need to re-engineer the applications post facto. Thus, through a very brief case study, we showed that the integrity requirements demand detailed consideration during system development, as suggested by the integrity management methodology presented in chapter 3.

7 CONCLUSION

In this chapter, we give a conclusive overview of the thesis, pointing out the novel research contributions to the field, in section 7.1. In section 7.2, we briefly discuss some advanced research carried out in the framework of potential future work.

7.1 DISCUSSION

In this thesis, we identified that the telecommunications systems are becoming increasingly complex, both in their internal construction and the degree of interconnection and interdependence between systems. This is due to technological advances characterised by the convergence of telecommunications and computing, growing demands for sophisticated services, and the pressure of regulatory forces stimulating the inter-domain interconnections. In this context, the key issue is that of the ability of systems to retain high integrity and low risk. The number of integrity issues is vast, and currently there is no coherent approach to understanding and managing these issues throughout the system lifecycle. Moreover, there is a shortfall of techniques for pre-launch integrity assurance.

Thus, in this thesis we explored the concept of telecommunications system integrity, the methodological frameworks for managing integrity throughout the telecommunications system lifecycle, and the techniques for insuring integrity during system development.

We pin-pointed the integrity attributes of modern telecommunications systems, taking into account the specific telecommunications engineering topics as well as software and system science. These attributes encompass issues regarding system functionality, structure and behaviour.

We developed a complete methodological framework for managing these integrity attributes throughout the telecommunications system lifecycle. The methodology presented encompasses three distinct phases: prediction, testing and maintenance. The focus is on prediction, the system pre-launch methodology phase. This predictive phase of the methodology is based on the ODP-UML model of the system. The author was one of the core creators of the ODP-UML management system development approach, which is seen as a strong and distinct research contribution.

In the integrity methodology developed here, the integrity-related actions are integrated in the system development lifecycle through iterative carrying out of the integrity analysis, design and implementation. Integrity analysis identifies the integrity-related requirements,

integrity design specifies the integrity-preserving action - the integrity policy, and integrity implementation focuses on the techniques needed for realising the integrity policy. Integrity policy can have the form of: an integrity-focused design recommendation; specification of an integrity-preserving mechanism to be implemented; or definition of a testing strategy.

The integrity methodology developed in this thesis is envisaged as a tool for categorising, analysing and tackling the integrity issues throughout the system lifecycle. This methodology is applicable to any distributed telecommunications system, while its application was explored through the consideration of network and service management systems, with two ACTS projects - TRUMPET and FlowThru - representing the main research platform.

In this context, we focused on the detailed development of two distinct integrity policies. Moreover, through the case studies of TRUMPET and FlowThru, we demonstrated that the theoretical integrity concepts developed in this thesis are applicable to real advanced management systems.

The first integrity policy is based on the object-oriented software metrics, and has the form of an integrity-focused design recommendation. We used seven existing software measures to form a metric suite which yields the complexity/coupling measurements of the system classes, and suggested that this metric suite can be used as the integrity/risk indicator early in the telecommunications system development. We elaborated on the positive correlation between the class stand-alone complexity, as well as class coupling, and the risk level of the class. Classes of higher complexity and coupling are more difficult to develop and test correctly, and as such pose more risk to integral system operation, *i.e.*, these classes have lower integrity. Further, the strong coupling paths give way to the propagation of the integrity breach through the system.

We demonstrated the usefulness and applicability of the metrics policy through three distinct experiments: one of the TRUMPET service management system and two of the FlowThru subscription management component. These experiments showed that the specific suite of object-oriented metrics can be used as the integrity indicator early in the telecommunications system development lifecycle (at the design stage), by pin-pointing the highly complex/coupled classes in the design. They also empirically demonstrated that the highest risk for the management systems' operation is exhibited at the major interconnection points between either administrative domains or stand-alone components. Finally, these experiments assessed the nature of the interrelationship between the individual metrics within the metrics suite, uncovering a strong ordinal relationship between the metrics.

The second integrity policy developed focused on the testing of the integrity aspects of the management systems' interconnection across administrative domains (the TMN X interface). Effectively, the work carried out for this policy encompassed review and development of testing principles for the management interconnections, focusing on the integrity issues. The policy was developed in the context of the TRUMPET project, and aims to exercise and establish that a set of integrity requirements is satisfied both with and without TRUMPET security mechanisms deployed in the TMN Xuser interface implementation. The policy is based on the concept of the Xuser Test-MIB (Management Information Base), which accommodates a set of abstract behavioural patterns that can be exercised over the interface during testing.

The applicability of the interconnection testing integrity policy was demonstrated through the case study of the TRUMPET Xuser interface. Generally, the case study was restricted due to limited time and resources, and thus had a form of a brief illustration. The approach was additionally limited due to the synchronous manager-agent communication over the interface. The outcome of the study demonstrated how the complex inter-domain interactions between management systems might appear to be sensitive to the introduction of additional sophisticated features.

Thus, through a mixture of theoretical and practical work, we investigated a range of integrity concepts of interest in the modern telecommunications systems. The theoretical research, involving the analysis of the integrity attributes and the development of the integrity methodology, was practically deployed in real case studies involving advanced network and service management systems. Moreover, two integrity assurance techniques - integrity policies - were developed and deployed on these management systems, yielding a range of results and practical experiences as discussed above.

7.2 FUTURE WORK

The integrity concepts developed in this thesis also provided a basis for the future research focusing on the integrity of the programmable networks.

Integrity issues will be of increasing importance in the future telecommunications scenarios involving programmable networks. By exploiting the programmable network technologies, third party application developers and end-users will have the access to the considerably lower level of network control and infrastructure, traditionally operated and run by the dominant network operators. Network operators will have little knowledge of the logic of the applications deployed over their control plane, the active packets travelling through their networks, or the pieces of mobile code deployed on their equipment. However, they will

require the assurance from the application developers that their applications will operate in a fully integral way and will not harm the operation of the operators' control systems or the network as a whole.

There are numerous approaches to programmable networks. The initial *active network* concepts are based on the data packets which travel through the network carrying programs in their header which can run on network devices such as switches or routers [Tenn96]. These concepts progressed through active bridging [Alex97], where packets carry only the flag indicating the desirability of running a program. The latest initiative in this area is the application-level active networking [Mars99]. *Mobile agents* are another big arena of research and development, telecommunications [Karm97] applications including IN-based service provisioning, network management [Gold98] [Grif00] and personal communication services. Finally, the *network programming interfaces* provide an open access for service developers to the service components and network control in the operator domain [Laza97]. Using these interfaces, service providers can develop their own applications using the underlying control and network infrastructure provided by the operator. Main initiatives in the area of programming interfaces are those led by the IEEE Project P1520, which aims at standardising the programming interfaces for networks [Bisw98], and the industry-led Parlay group, which is specifying and developing a new open network application programming interface [Parl00]. Parlay specification provides an object-oriented open interface, the Parlay Application Programming Interface (API), to the network generic services in the operator domain. Common feature of these interfaces is that they essentially specify the interconnection point between the two autonomous domains: network operator and service provider domains. These interfaces can be compared to the ConS and Xuser interfaces in TINA [TINA-BM] and TMN [M.3010] architectures respectively.

The integrity concepts developed in this thesis were discussed in the perspective of the network programming interfaces, more specifically the Parlay interface, in [Prnj00b]. The integrity attributes identified in this thesis and seen as relevant in the Parlay context were reflected on. The integrity methodology was suggested for managing the appropriate integrity issues in network programming interface scenarios, and some integrity preserving policies were discussed in this context. The majority of the policies were envisaged to be enforced by the *integrity gateway*, the component in the operator domain implementing the required integrity mechanisms. This work represented the core of a larger project proposal.

Moreover, the author participated in the development of the specific integrity policy for the Parlay API, published in [Kolt99][Kolt00]. This policy takes the form of an integrity-focused design recommendation, and involves the use of behavioural techniques (Specification and

Description Language - SDL) (as discussed in chapter 3, section 3.4.2) in specifying not just the logic, *i.e.* the behaviour of the Parlay API-based application, but also the behaviour of the API offered, in the operator domain. Current Parlay API specification has the form of Unified Modelling Language (UML) class diagrams and shows neither the behaviour of the objects implementing the interface, nor the relationships between objects. Some simple UML state diagrams describe the behaviour of few complex objects, and sequence diagrams are specified to guide the developers. This is not enough for specifying and enforcing the correct sequencing and timing of method invocations on the API, and cannot guarantee the preservation of liveness and the robustness of the interface. In the integrity policy developed, SDL is suggested for the in-depth modelling of the behaviour of the API. The SDL behaviour models would further allow the developers and network operators to analyse, validate and test the new applications with respect to their integrity features, with the help of widely available SDL support tools. Service verification and validation can be performed, and deadlock, livelock and feature interaction detection can be conducted prior to service launch.

This further work conducted reflects our belief that the integrity issues will need to be even more closely considered in the future telecommunications scenarios which point in the direction of open, flexible, and dynamic network and service configuration and provision. It also demonstrates the applicability of the concepts developed in this thesis to these future scenarios.

8 AUTHOR'S PUBLICATIONS

[Kand98] M. M. Kande, S. Mazaher, O. Prnjat, L. Sacks, M. Wittig, "Applying UML to Design an Inter-Domain Service Management Application", Proceedings of the UML '98 International Conference, June 1998. Also published in the LNCS volume "<<UML'98>>: Beyond the Notation", Springer-Verlag, 1998.

[Kolt99] M. Koltsidas, O. Prnjat, L. Sacks, "Design and Development of the Customer's Applications Based on the Parlay API", Proceedings of the London Communications Symposium, July 1999.

[Kolt00] M. Koltsidas, O. Prnjat, L. Sacks, "Development of Parlay-based Applications Using UML and SDL", Proceedings of the 3rd IFIP/IEEE International Conference on Management of Multimedia Networks and Services (MMNS'2000), September 2000.

[Prnj96] O. Prnjat, L. Sacks, "Adaptation of Software Metrics to Telecommunications Networks", Proceedings of the UCL Communications Research Symposium, July 1996.

[Prnj97] O. Prnjat, L. Sacks (Eds.), "Detailed Component and Scenario Designs", ACTS Project AC112 TRUMPET, Deliverable 8, June 1997.

[Prnj98a] O. Prnjat, L. Sacks, H. Hegna, "Testing the Integrity vs. Security Requirements on the TMN X Interface", EUNICE '98 Network Management and Operation Summer School, September 1998.

[Prnj98b] O. Prnjat, L. Sacks (Eds.), "Trials and Technology Assessment", ACTS Project AC112 TRUMPET, Deliverable 15, December 1998.

[Prnj99a] O. Prnjat, L. Sacks, "Integrity Methodology for Interoperable Environments", IEEE Communications, Special Issue on Network Interoperability, Vol. 37, No. 5, pp. 126-139, May 1999.

[Prnj99b] O. Prnjat, L. Sacks, "Telecommunications System Design Complexity and Risk Reduction Based on System Metrics", Proceedings of the 10th European Workshop on Dependable Computing (EWDC10), May 1999.

[Prnj99c] O. Prnjat, L. Sacks, " Impact of Security Policies on the TMN X Interface Integrity and Performance", Proceeding of the First IEEE Latin American Network Operations and Management Symposium (LANOMS'99), December 1999.

[Prnj00a] O. Prnjat, L. Sacks, "High Integrity Inter-Domain Management", in A. Galis (Ed.), "Multi-Domain Communication Management Systems", CRC Press - USA, ISBN: 084930587X, June 2000 [Gali00].

[Prnj00b] O. Prnjat, L. Sacks, "Inter-domain Integrity Management for Programmable Network Interfaces", Proceedings of the 3rd IFIP/IEEE International Conference on Management of Multimedia Networks and Services (MMNS'2000), September 2000.

[Sack98] L. Sacks, O. Prnjat, M. Wittig, M. M. Kande, B. Bhushan, C. Autant, "TRUMPET Service Management Architecture", Proceedings of the 2nd International Enterprise Distributed Object Computing Conference (EDOC '98), November 1998.

9 REFERENCES

[Albr79] A. J. Albrecht, "Measuring Application Development Productivity", IBM Application Development Joint SHARE/GUIDE Symposium, Monterey, CA, pp. 83-92, 1979.

[Alex97] Alexander et. al., "Active Bridging", Computer Communications Review, Vol. 27, No. 4, pp. 101-111, 1997.

[ATM-M4] The ATM Forum, Technical Committee, "Network Management, M4 Network View Interface, Requirements and Logical MIB", March 1996.

[Auta97] C. Autant (Ed.), "The Nil-Security Prototype", ACTS Project AC112 TRUMPET, Deliverable 6, February 1997.

[Barr93] L. M. Barroca, J. A. McDermid, "Formal Methods: Use and Relevance for the Development of Safety Critical Systems", in P. A. Bennett (Ed.), "Safety Aspects of Computer Control" pp. 96-153, Butterworth Heinemann, 1993.

[Basi96] V. R. Basili, L. C. Briand, W. L. Melo, "A Validation of Object-Oriented Design Metrics as Quality Indicators", IEEE Transactions on Software Engineering, Vol. 22, pp. 751-761, 1996.

[Bate96] B. W. Bates et. al., "Formalising Fusion Object-Oriented Analysis Models", First IFIP International Workshop on Formal Methods for Open Object-Based Distributed Systems (FMOODS), Chapman and Hall, March 1996.

[BBC00] BBC News, "BT Network Fault Fixed", 26th February 2000, http://news.bbc.co.uk

[Bell93] Bellcore, "Generic Requirements for Software Reliability Prediction", GR-2813-CORE, Issue 1, December 1993.

[Bern93] E. V. Bernard, "Essays on Object-Oriented Software Engineering", Prentice-Hall, 1993.

[Berq96] K. Berquist, A, Berquist, (Eds.), "Managing Information Highways. The PRISM Book: Principles, Methods and Case Studies for Designing Telecommunications Management Systems", Lecture Notes in Computer Science, Vol. 1164, Springer-Verlag, Berlin Heidelberg New York, 1996.

[Biem98] J. M. Bieman, B. Kang, "Measuring Design-Level Cohesion", IEEE Transactions on Software Engineering, Vol. 24, No. 2, pp. 111-124, February 1996.

[Bisw98] J. Biswas et. al., "The IEEE P1520 Standards Initiative for Programmable Interfaces", IEEE Communications Magazine, pp. 64-72, October 1998.

[Boeh81] B. W. Boehm, "Software Engineering Economics", Prentice-Hall, 1981.

[Boeh91] B. W. Boehm, "Software Risk Management: Principles and Practices", IEEE Software, Vol. 8, pp. 32-41, January 1991.

[Booc94] G. Booch, "Object-Oriented Analysis and Design with Applications", Benjamin Cummings, 1994.

[Bowe92] J. P. Bowen, V. Stavridou, "Safety-Critical Systems, Formal Methods and Standards", Technical Report PRG-TR-5-92, Programming Research Group, Oxford University Computing Laboratory, 1992.

[Bowm96] H. Bowman et. al., "Viewpoint Consistency in ODP, a General Interpretation", Proceedings of the First IFIP International Workshop on Formal Methods for Open Object-Based Distributed Systems (FMOODS), Chapman and Hall, March 1996.

[Bria98] L. C. Briand, J. Daly, V. Porter, J. Wust, "A Comprehensive Empirical Validation of Design Measures for Object-Oriented Systems", Proceedings of the Fifth International Software Metrics Symposium, pp. 246-257, 1998.

[Butl95] R. W. Butler, G. B. Finelli, "The Unfeasibility of Quantifying the Reliability of Life-Critical Real-Time Software", Software Engineering Notes, Vol. 16, No. 5, pp. 66-76, 1995.

[Came93] E. J. Cameron, H. Velthuijsen, "Feature Interactions in Telecommunications Systems", IEEE Communications Magazine, pp. 18-23, August 1993.

[Cano71] W. J. Canover, "Practical Nonparametric Statistics", John Wiley & Sons, New York, 1971.

[Cart96] M. Cartwright, M. Shepperd, "An Empirical Investigation of Object-Oriented Software in Industry", Technical Report TR96/01, Bournemouth University, 1996.

[Chid91] S. R. Chidamber, C. F. Kemerer, "Towards a Metric Suite for Object-Oriented Design", Proceedings OOPSLA'91, pp. 197-211, 1991.

[Chid94] S. R. Chidamber, C. F. Kemerer, "A Metrics Suite for Object-Oriented Design", IEEE Transactions on Software Engineering, Vol. 20, No. 6, pp. 476-493, 1994.

[Chid98] S. R. Chidamber, D. P. Darcy, C. F. Kemerer, "Managerial Use of Metrics for Object-Oriented Software: An Exploratory Analysis", IEEE Transactions on Software Engineering, Vol. 24, No. 8, August 1998.

[Conn92] J. O'Connor, D. T. Patrick, "Practical Reliability Engineering", 3rd edition, J. Wiley, 1992.

[Cons79] L. Constantine, E. Yourdon, "Structured Design", Prentice-Hall, 1979.

[CORBA] Object Management Group, "The Common Object Request Broker (CORBA) - Architecture and Specification", 1995.

[Dang96] F. Dangtran, V. Pervaskine, J. B. Stefani, B. Crawford, A. Kramer, D. Otway, "Binding and Streams: the ReTINA Approach", TINA '96 Conference, September 1996.

[DeMa82] T. DeMarco, "Controlling Software Projects: Management, Measurement and Estimation", Prentice-Hall, 1982.

[Dens98] W. Denson, "The History of Reliability Prediction", IEEE Transactions on Reliability, Vol. 47, No. 3, pp. 321-345, September 1998.

[Eber97] R. Eberhardt, S. Mazziotta, D. Sidou. "Design and Testing of Information Models in a Virtual Environment", The 5th IFIP/IEEE International Symposium on Integrated Network Management "Integrated Management in a Virtual World", May 1997.

[Ekka95] R. Ekkart et. al., "Tutorial on Message Sequence Charts", in O. Haugen (Ed.), "SDL 95 with MSC in CASE", September 1995.

[Fene93] P. J. Fenelon, J. A. McDermid, "An Integrated Toolkit for Software Safety Analysis", Journal of Systems and Software, July 1993.

[Fent91] N. E. Fenton, "Software Metrics - A Rigorous Approach", Chapman and Hall, 1991.

[Fent94] N. E. Fenton, "Software Measurement: A Necessary Scientific Basis", IEEE Transactions on Software Engineering, Vol. 20, No. 3, March 1994.

[Flow-http] http://www.cs.ucl.ac.uk/research/flowthru/models/

[FlowThru] ACTS Project FlowThru, AC335, "Co-operative Secure Management of Multi Technology and Administrative Domain Network and Service Management Systems".

[Gagn97] F. Gagnon et. al., "A Security Architecture for TMN Inter-Domain Management", Proceedings of the 4th International Conference on Intelligence in Services and Networks, Springer-Verlag, Berlin, 1997.

[Gali00] A. Galis (Ed.), "Multi-Domain Communication Management Systems", CRC Press - USA, ISBN: 084930587X, June 2000.

[Garm96] D. Garmus, D. Herron, "Effective Early Estimation", Software Development, Vol. 4, No. 7, pp. 57-65, July 1996.

[Geor99] P. Georgatsos, D. Makris, D. Griffin, G. Pavlou, S. Sartzetakis, Y. T'Joens, D. Ranc, "Technology Interoperation in ATM Networks: The REFORM System," IEEE Communications, Special Issue on Network Interoperability, Vol. 37, No. 5, pp. 112-118, May 1999

[Gibb71] J. D. Gibbons, "Nonparametric Statistical Inference", McGraw-Hill, New York, 1971.

[Gold98] G. Goldszmidt, Y. Yemini, "Delegated Agents for Network Management", IEEE Communications Magazine, Vol. 36, No. 3, pp. 66-70, March 1998.

[Grab93] J. Grabowski et. al., "Test Case Generation with Test Purpose Specification by MSC", in "SDL '93 - Using Objects", North-Holland, October 1993.

[Grif96] D. Griffin et. al., "ATM Virtual Path Connection and Routing Management", in D. Griffin (Ed.), "Integrated Communications Management of Broadband Networks", Crete University Press, pp. 73-145, 1996.

[Grif00] D. Griffin, G. Pavlou, P. Georgatsos, "Providing Customisable Network Management Services Through Mobile Agents," Proceedings of the 7th International Conference on Intelligence in Services and Networks, Springer-Verlag, Berlin, 2000.

[Hall96] J. Hall (Ed.), "Modelling and Implementing TMN-based Multi-domain Management", Springer-Verlag, 1996.

[Hatt97] L. Hatton, "Software Failures: Follies and Fallacies", IEE Review, pp. 49-52, March 1997.

[Hend96] B. Hendson-Sellers, "Object-Oriented Metrics, Measures of Complexity", Prentice-Hall, 1996.

[Henr81] S. Henry, D. Kafura, "Software Structure Metrics Based on Information Flow", IEEE Transactions on Software Engineering, Vol. 7, No. 5, pp. 510-518, 1981.

[Hoan93] B. Hoang, D. J. Bastien, B. Lewin, "Common Channel Signalling Network Integrity Experiences from the US", Proceedings of the IEEE Conference on Communications (ICC'93), Vol. 2, pp. 644 – 649, 1993.

[I.321] ITU Rec. I.321, "B-ISDN Protocol Reference Model and its Application", 1991.

[IEEEStd1228-94] IEEE Standard for Software Safety Plans, Std 1228-1994.

[ISO9000] International Standards Organisation, ISO-9000 family of standards, 1991.

[ISO9646-1] ISO/IEC 9646, "Open Systems Interconnection - Conformance Testing Methodology and Framework - Part 1: General Concepts", 1992.

[ISO9646-3] ISO/IEC 9646, "Open Systems Interconnection - Conformance Testing Methodology and Framework - Part 3: The Tree and Tabular Combined Notation (TTCN)", 1992.

[Jaco92] I. Jacobson, "Object-Oriented Software Engineering - A Use-Case Driven Approach", Addison-Wesley, 1992.

[JAVA] J. Gosling, B. Joy, G. Steele, "The Java Language Specification, Version 1.0", Addison-Wesley, http://java.sun.com/docs/books/jls/index.html

[Jone90] C. B. Jones, "Systematic Software Development using VDM", Prentice-Hall International, 1990.

[Kami99] T. Kamiya, S. Kusumoto, K. Inoue, "Prediction of Fault-proneness at Early Phase in Object-Oriented Development", Proceedings of the 2nd IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC '99), pp. 253-258, 1999.

[Kand98] M. M. Kande, S. Mazaher, O. Prnjat, L. Sacks, M. Wittig, "Applying UML to Design an Inter-Domain Service Management Application", Proceedings of the UML '98 International Conference, June 1998. Also published in the LNCS volume "<<UML'98>>: Beyond the Notation", Springer-Verlag, 1998.

[Karm97] A. Karmouch, "Mobile Software Agents for Telecommunications", IEEE Communications Magazine Guest Editorial, Vol. 36, No. 7, 1997.

[Keck98] D. O. Keck, P. J. Kuehn, "The Feature and Service Interaction Problem in Telecommunications Systems: A Survey", IEEE Transactions on Software Engineering, Vol. 24, No. 10, October 1998.

[Kern88] B. W. Kernighan, D. M. Ritchie, "The C Programming Language", Second Edition, Prentice-Hall, 1988.

[Kirs99] C. Kirsopp, M. J. Shepperd, S. Webster, "An Empirical Study Into the Use of Measurement to Support OO Design", Proceedings of the 6th IEEE International Metrics Symposium, IEEE Computer Society, 1999.

[Kitc85] B. A. Kitchenham, N. R. Taylor, "Software Development Cost Estimation", Journal of Systems and Software, Vol. 5, pp. 67-78, 1985.

[Klei78] D. G. Kleinbaum, L. L. Kupper, "Applied Regression Analysis and other Multivariable Methods", Duxbury Press, Boston, 1978.

[Kolt99] M. Koltsidas, O. Prnjat, L. Sacks, "Design and Development of the Customer's Applications Based on the Parlay API", Proceedings of the London Communications Symposium, July 1999.

[Kolt00] M. Koltsidas, O. Prnjat, L. Sacks, "Development of Parlay-based Applications Using UML and SDL", Proceedings of the 3rd IFIP/IEEE International Conference on Management of Multimedia Networks and Services (MMNS'2000), September 2000.

[Kran71] D. H. Krantz et. al., "Foundations of Measurement", Vol. 1, Academic Press, 1971.

[Lapr92] J. C. Laprie (Ed.), "Dependability: Basic Concepts and Terminology: in English, French, German, Italian and Japanese", Springer-Verlag, Wien, 1992.

[Laza97] A. Lazar, "Programming Telecommunication Networks", IEEE Network, pp. 2-12, October 1997.

[Leve95] N. G. Leveson, "Safeware: System Safety and Computers", Addison-Wesley, 1995.

[Lew99a] D. Lewis, C. Malbon, A. DaCruz, "Modelling Management Components for Reuse using UML", Proceedings of the 6th International Conference on Intelligence in Services and Networks, Springer-Verlag, Berlin, 1999.

[Lew99b] D. Lewis, "A Software Development Methodology for Service Management", Published in Telecom'99 Forum.

[Lew99c] D. Lewis, C. Malbon, G. Pavlou, C. Stathopoulos, E. Jaen, "Integrating Service and Network Management Components for Service Fulfilment", in Active Technologies for Network and Service Management: Proceedings of the 10th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM '99), R. Stadler, B. Stiller, (Eds.), pp. 49-62, Springer-Verlag, October 1999.

[Lewi94] B. Lewin, D. Branflick, "Future Directions for National Interoperability Testing for Network Integrity", IEEE Globecom'94, Vol. 1, pp. 222-226, 1994.

[Lewi97] D. Lewis, T. Tiropanis, A. McEwan, C. Redmond, V. Wade, R. Bracht, "Experiences in Integrated Multi-Domain Management", Proceedings of the IFIP/IEEE International Conference on Management of Multimedia Networks and Services, 1997.

[Li93] W. Li, S. Henry, "Object-Oriented Metrics that Predict Maintainability", Journal of Systems and Software, Vol. 23, pp. 111-122, 1993.

[Lore94] M. Lorenz, J. Kidd, "Object-Oriented Software Metrics", Prentice-Hall, 1994.

[LOTOS] LOTOS Virtual Tutorials at http://wwwtios.cs.utwente.nl/lotos/

[M.3010] ITU Rec. M3010, "Principles for a Telecommunications Management Network", 1996.

[M.3020] ITU Rec. M3020, "TMN Interface Specification Methodology", 1995.

[Mail96] D. Maillot (Ed.), "Inter-TMN Security Policies", ACTS Project AC112 TRUMPET, Deliverable 2, June 1996.

[Male98] H. A. Malec, "Communications Reliability: A Historical Perspective", IEEE Transactions on Reliability, Vol. 47, No. 3, pp. 333-345, September 1998.

[Mamd96] E. Mamdani, "Lecture Notes in Network and Service Management", University College London Telecommunications M.Sc., 1996.

[Mant91] R. J. Manterfield, "Common-Channel Signalling", Peter Peregrinus Ltd., London, 1991.

[Mars99] I. W. Marshall, et. al., "Application-level Programmable Network Environment", BT Technology Journal, Vol. 17, No. 2, April 1999.

[Maso97] P. J. Mason, "Ensuring Network Integrity", IEE Colloquium on "How to Compete and Connect: Understanding the Engineering of Telecommunications Network Interconnection", pp. 11/1 - 11/7, 1997.

[McCa76] T. J. McCabe, "A Complexity Measure", IEEE Transactions on Software Engineering, Vol. 2, No. 4, pp. 308-320, 1976.

[McCa95] K. McCarthy, G. Pavlou, S. Bhatti, J. N. DeSouza, "Exploiting the Power of OSI Management for the Control of SNMP-capable Resources Using Generic Application Level Gateways", in Integrated Network Management IV, Proceedings of the IFIP/IEEE Symposium on Integrated Network Management (ISINM '95), Santa Barbara, USA, A. Sethi, Y. Raynaud, F. Faure-Vincent, eds., pp. 440-453, Chapman & Hall, 1995.

[McDe94] J. A. McDermid, T. O. Jackson, I. C. Wand, M. A. Wilkins, "Final Report on the Project: Dependability Measurement of Safety Critical Systems", Technical Report No.1.94.116 ISEI/IE/2776/9, University of York, 1994.

[McDo92] J. C. McDonald, "Public Networks - Dependable?", IEEE Communications Magazine, April 1992.

[McDo94] J. C. McDonald, "Public Network Integrity - Avoiding a Crisis in Trust", IEEE Journal on Selected Areas in Communications, Vol. 12, No. 1, pp. 5-12, January 1994.

[Meht98] S. N. Mehta, "AT&T is Seeking Cause of a Big Outage in Data Network Used by Corporations", The Wall Street Journal, Wed, 15th April 1998, p B15.

[Minitab] Minitab Statistical Analysis Package, Version 13, http://www.minitab.com

[Mink97] A. Minkiewicz, "Objective Measures", Software Development, Vol. 5, No. 6, pp. 43-50, June 1997.

[MISA-D9] ACTS Project MISA Deliverable 9, "Implementation of the MISA Testing System", June 1998.

[MISA-X] ACTS Project MISA Deliverable 3, Annex A, "Initial MISA High Level Design, Annex A: Xuser Interface Definition", September 1996.

[Mont97] V. Monton, K. Ward, M. Wilby, R. Masson "Risk Assessment Methodology for Network Integrity", BT Technology Journal, Vol. 15, No. 1, pp. 223-234, January 1997.

[Mont98] V. Monton, "Investigation of the Maintenance of Integrity in Telecommunication Networks Using Formal and Heuristic Methodologies", Ph.D. Thesis, Dept. of Electronics and Electrical Engineering, University College London, 1998.

[Musa87] J. D. Musa, A. Iannino, K. Okumoto, "Software Reliability, Measurement, Prediction, Application", McGraw-Hill, 1987.

[MW-http] Merriam-Webster Collegiate Dictionary Online, http://www.m-w.com/cgi-bin/dictionary

[Myer76] G. J. Myers, "Software Reliability - Principles and Practices", John Wiley and Sons, New York, 1976.

[NMF-TM] Network Management Form, Technology Map, NMF GB909 Draft, July 1998.

[NMF-TOM] Network Management Form, "Telecoms Operations Map", NMF GB910, Stable Draft 0.2b, April 1998.

[Obje97] ObjectSpace, "Voyager: Agent-Enhanced Distributed Computing for Java", User Guide, Version 1.0, http://www.objectspace.com, July 1997.

[ODP] ITU Draft Recommendation X.901-X.904, "Basic Reference Model of Open Distributed Processing"; Part 1: "Overview and Guide to Use", 1995; Part 2: "Foundations", 1995; Part 3: "Architecture", 1995; Part 4: "Architectural Semantics", 1995.

[Ølne97] J. Ølnes (Ed.), "Security Policies & System Architecture Specification", ACTS Project AC112 TRUMPET, Deliverable 7, April 1997.

[OMA] Object Management Group, "A Discussion of the Object Management Architecture", 1997, http://www.omg.org/library/oma1.html

[Parl00] The Parlay Group, "Version 2.0 Parlay Specification", http://www.parlay.org/

[Pavl91] G. Pavlou, A. Mann, "Quality of Service Management in Integrated Broadband Communications: an OSI Management / TMN Based Prototype", Proceedings of the 5th International TMN Conference, November 1991.

[Pavl97] G. Pavlou, D. Griffin, "Realizing TMN-like Management Services in TINA", Journal of Network and System Management (JNSM), Special Issue on TINA, Vol. 5, No. 4, pp. 437-457, Plenum Publishing, December 1997.

[Pavl98] G. Pavlou, "Telecommunications Management Network: a Novel Approach Towards its Architecture and Realisation Through Object-Oriented Software Platforms", Ph.D. Thesis, Dept. of Computer Science, University College London, March 1998.

[Pavo97] J. Pavon, T. Mota, G. Pavlou, L. Veillat, P. Palavos, "The VITAL Network Resource Architecture", Proceedings of the TINA '97 Conference on Global Convergence of

Telecommunications and Distributed Object Computing, pp. 130-138, IEEE Computer Society, 1997.

[Pick97] S. Pickin et. al., "Introducing Formal Notations in the Development of Object-Based Distributed Applications", in E. Najm and J. B. Stefani (Eds.), "Formal Methods for Open Object-Based Distributed Systems", Chapman and Hall, 1997.

[Pras95] D. Prasad, J. McDermid, I. Wand, "Dependability Terminology: Similarities and Differences", Proceedings of the 10th Annual Conference on Computer Assurance (COMPAS'95), pp. 213-221, 1995.

[Prnj96] O. Prnjat, L. Sacks, "Adaptation of Software Metrics to Telecommunications Networks", Proceedings of the UCL Communications Research Symposium, July 1996.

[Prnj97] O. Prnjat, L. Sacks (Eds.), "Detailed Component and Scenario Designs", ACTS Project AC112 TRUMPET, Deliverable 8, June 1997.

[Prnj98a] O. Prnjat, L. Sacks, H. Hegna, "Testing the Integrity vs. Security Requirements on the TMN X Interface", EUNICE '98 Network Management and Operation Summer School, September 1998.

[Prnj98b] O. Prnjat, L. Sacks (Eds.), "Trials and Technology Assessment", ACTS Project AC112 TRUMPET, Deliverable 15, December 1998.

[Prnj99a] O. Prnjat, L. Sacks, "Integrity Methodology for Interoperable Environments", IEEE Communications, Special Issue on Network Interoperability, Vol. 37, No. 5, pp. 126-139, May 1999.

[Prnj99b] O. Prnjat, L. Sacks, "Telecommunications System Design Complexity and Risk Reduction Based on System Metrics", Proceedings of the 10th European Workshop on Dependable Computing (EWDC10), May 1999.

[Prnj99c] O. Prnjat, L. Sacks, " Impact of Security Policies on the TMN X Interface Integrity and Performance", Proceeding of the First IEEE Latin American Network Operations and Management Symposium (LANOMS'99), December 1999.

[Prnj00a] O. Prnjat, L. Sacks, "High Integrity Inter-Domain Management", in A. Galis (Ed.), "Multi-Domain Communication Management Systems", CRC Press - USA, ISBN: 084930587X, June 2000 [Gali00].

[Prnj00b] O. Prnjat, L. Sacks, "Inter-domain Integrity Management for Programmable Network Interfaces", Proceedings of the 3rd IFIP/IEEE International Conference on Management of Multimedia Networks and Services (MMNS'2000), September 2000.

[Q.700] ITU Rec. Q.700, "Introduction to CCITT Signalling System No. 7", 1993.

[Ranc98] D. Ranc, S. Sedillot, "Use of Transactions in Network Management Applications", Proceedings of 5th International Conference on Intelligence in Services and Networks, Springer-Verlag, Berlin, 1998.

[Redm91] F. Redmill, T. Anderson, "Safety-Critical Systems - Current Issues, Techniques and Standards", Chapman and Hall, 1993.

[Reib91] A. L. Reibman, M. Veeraraghavan, "Reliability Modelling: an Overview for System Designers", IEEE Computer, April 1991.

[RFC1157] RFC 1157, "Simple Network Management Protocol", 1990.

[RFC1777] RFC 1777, "Lightweight Directory Access Protocol", 1995.

[RFC2068] RFC 2068, "Hypertext Transfer Protocol - HTTP/1.1", 1997.

[RFC2078] RFC 2078, "Generic Security Service Application Program Interface", Version 2, 1997.

[Riel96] A. J. Riel, "Object-Oriented Design Heuristics", Addison-Wesley, 1996.

[Robi92] P. J. Robinson, "Hierarchical Object-Oriented Design", Prentice-Hall, 1992.

[Rumb91] J. Rumbaugh et. al., "Object-Oriented Modelling and Design", Prentice-Hall, 1991.

[Sack98] L. Sacks, O. Prnjat, M. Wittig, M. M. Kande, B. Bhushan, C. Autant, "TRUMPET Service Management Architecture", Proceedings of the 2nd International Enterprise Distributed Object Computing Conference (EDOC '98), November 1998.

[Schn92] N. F. Schneidewind, "Methodology for Validating Software Metrics", IEEE Transactions on Software Engineering, Vol. 18, No. 5, pp. 410-422, May 1992.

[Shep93] M. Shepperd, "Software Engineering Metrics, Vol. 1", McGraw-Hill, 1993.

[Sidd94] R. Siddhartha, "Controlling the Software Development Process", IEEE Journal on Selected Areas in Communications, Vol. 12, No. 1, pp. 33-39, January 1994.

[Slom94a] M. Sloman (Ed.), "Network and Distributed Systems Management", Addison-Wesley, 1994.

[Slom94b] M. Sloman, "Policy-driven Management for Distributed Systems", Journal of Network and Systems Management, Vol. 2, No. 4, 1994.

[Spre89] P. Sprent, "Applied Nonparametric Statistical Methods", Chapman and Hall, 1989.

[TA1A-93] Committee T1-Telecommunications, "A Technical Report on Network Survivability Performance", Document T1A1.2/93-001R3, Technical Report No. 24, November 1993.

[Tau3.5] Telelogic Tau 3.5, SDT and ITEX Tools, http://www.telelogic.se

[Tenn96] D. Tennenhouse, D. Wetherall, "Towards an Active Network Architecture", Computer Communications Review, Vol. 26, No. 2, 1996.

[TINA-BM] H. Mulder (Ed.), "Business Model and Reference Points", Version 4.0, http://www.tinac.com/specifications/specifications.htm

[TINA-NRIM] N. Natarajan (Ed.), "Network Resource Information Model", Version 3.0, http://www.tinac.com/specifications/specifications.htm

[TINA-ODL] A. Parhar (Ed.), "ODL Manual", Version 2.3, http://www.tinac.com/specifications/specifications.htm

[TINA-SA] L. Kristiansen (Ed.), "Service Architecture", Version 5.0, http://www.tinac.com/specifications/specifications.htm

[TRUMPET] ACTS Project TRUMPET, AC112, "TMN'S Regulations and Multiple Providers Environment; Inter-domain Management with Integrity".

[TRUMPET-TA] ACTS Project TRUMPET, Technical Annex, 1996.

[UCL94] A. Galis, C. Todd, K. Ward, M. Wilby, "Final Report of a Study Entitled Network Integrity in an ONP Environment for the Commission of European Union", November 1994.

[UML] Rational Software Corporation, Unified Modelling Language, http://www.rational.com/

[Vess84] I. Vessey, R. Weber, "Research on Structured Programming: An Empiricist's Evaluation", IEEE Transactions on Software Engineering, Vol. 10, pp. 394-407, 1984.

[Vill92] A. Villmeur, "Reliability, Availability, Maintainability and Safety Assessment, Vol. 1: Methods and Techniques", John Wiley and Sons, 1992.

[Vinc97] A. Vincent, C. Hall, "Modelling/Design Methodology and Template", NMF Internal Document, Draft 4, October 1997.

[Wade98] V. Wade et. al., "A Design Process for the Development of Multi Domain Service Management Systems", in S. Rao (Ed.), "Guidelines for ATM Deployment and Interoperability", pp. 88-103, 1998.

[Walk97] P. Walker, "How to Compete and Connect: Setting the Business and Regulatory Context", in IEE Colloquium on "How to Compete and Connect: Understanding the Engineering of Telecommunications Network Interconnection", Digest No: 1997/179, 1997.

[Ward95] K. Ward, "Impact of Network Interconnection on Network Integrity", British Telecommunications Engineering, Vol. 13, pp. 296-303, January 1995.

[Weis99] N. A. Weiss, "Introductory Statistics", Addison-Wesley, 1999.

[Well94] E. F. Weller, "Using Metrics to Manage Software Projects", IEEE Computer, Vol. 27, No. 9, pp. 27-33, 1994.

[What-is] "Whatis" Knowledge Exploration Tool, http://whatis.com

[Whit97] S. A. Whitmire, "Object-Oriented Design Measurement", John Wiley and Sons, 1997.

[X.208] ITU Rec. X.208, "Abstract Syntax Notation One - Basic Notation", 1988.

[X.217] ITU Rec. X.217, "Association Control Service Element - Service Definition", 1992.

[X.227] ITU Rec. X.227, "Association Control Service Element - Protocol Definition", 1993.

[X.274] ITU Rec. X.274, "Transport Layer Security Protocol", 1994.

[X.700] ITU Rec. X.700, "OSI Management Framework", 1992.

[X.701] ITU Rec. X.701, "System Management Overview", 1992.

[X.710] ITU Rec. X.710, "Common Management Information Service Definition (CMIS)", 1992.

[X.711] ITU Rec. X.711, "Common Management Information Protocol Specification (CMIP)", 1992.

[X.722] ITU Rec. X.722, "Structure of Management Information: Guidelines for the Definition of Managed Objects", 1992.

[X.725] ITU Rec. X.725, "Structure of Management Information: General Relationship Model", 1995.

[X.737] ITU Rec. X.737, "Systems Management: Confidence and Diagnostic Test Categories", 1995.

[X.745] ITU Rec. X.745, "Systems Management: Test Management Function", 1993.

[X.831] ITU Rec. X.831, "Generic Upper Layers Security - Part 2: Security Exchange Service Element (SESE) Service Definition", 1995.

[X.832] ITU Rec. X.832, "Generic Upper Layers Security - Part 3: Security Exchange Service Element (SESE) Protocol Definition", 1995.

[Yama97] T. Yamamura, T. Tanahashi, M. Hanaki, N. Fujii, "TMN-Based Customer Network Management for ATM Networks", IEEE Communications, Vol. 35, No. 10, pp. 46-52, October 1997.

[Your96] E. Yourdon, "Rise and Resurrection of the American Programmer", Prentice-Hall, 1996.

[Z.100] ITU Rec. Z.100, "Specification and Description Language (SDL)", 1993.

[Z.120] ITU Rec. Z.120, "Message Sequence Charts (MSC)", 1994.

[ZArch] University of Oxford Z Archive,
http://www.comlab.ox.ac.uk/archive/z.html#archive

[Zuse90] H. Zuse, "Software Complexity: Measures and Methods", Walter de Gruyter, Berlin, 1990.

10 ACRONYMS

ACID	Atomicity/Consistency/Isolation/Durability
ACSE	Association Control Service Element
ACTS	Advanced Communications Technologies and Services
API	Application Programming Interface
ASN.1	Abstract Syntax Notation 1
ASP	Abstract Service Primitive
ATIS	Alliance for Telecommunications Industry Solutions
ATM	Asynchronous Transfer Mode
ATS	Abstract Test Suite
B-ISDN	Broadband Integrated Services Digital Network
BT	British Telecom
CA	Certification Authority
CA	Customer Administrator
CASE	Computer Aided Software Engineering
CBO	Coupling Between Objects
CCS	Common Channel Signalling
CK	Chidamber-Kemerer (metric suite)
CMIP	Common Management Information Protocol
CMIS	Common Management Information Service
CMIS/P	Common Management Information Service/Protocol
CMISE	Common Management Information Service Elements
CO	Computational Object
COCOMO	Constructive Cost Model
CORBA	Common Object Request Broker Architecture
CPN	Customer Premises Network
DB	Database
DIT	Depth of Inheritance Tree
DN	Distinguished Name
DPE	Distributed Processing Environment
EC	European Commission
EPF	Ecole Polytechnique Federale
ETS	Executable Test Suite
FC	Functional Class
FCAPS	Fault - Configuration - Accounting - Performance - Security

FCC	Federal Communications Commission
FDT	Formal Description Technique
FMECA	Failure Modes, Effects and Criticality Analysis
FP	Function Point
FPTN	Failure Propagation and Transformation Notation
FSM	Finite State Machine
FTA	Fault Tree Analysis
GDMO	Guidelines for the Definition of Managed Objects
GRM	General Relationship Model
GUI	Graphical User Interface
HOOD	Hierarchical Object-Oriented Design
HP-OV	HP OpenView
HTTP	Hypertext Transfer Protocol
IDL	Interface Definition Language
IEEE	Institute of Electrical and Electronics Engineers
IFIP	International Federation for Information Processing
IITP	Inter-network Interoperability Test Plan
IN	Intelligent Network
IP	Internet Protocol
IS	Intelligent Services
ISO	International Standards Organisation
IT	Information Technology
ITU	International Telecommunications Union
JVM	Java Virtual Machine
KLOC	Thousand Lines Of Code
LAN	Local Area Network
LCOM	Lack of Cohesion of Methods
LDAP	Lightweight Directory Access Protocol
MAE	Management Application Entity
MF	Management Function
MIB	Management Information Base
MORT	Managed Object Referring to Test
MSC	Message Sequence Chart
MTBF	Mean Time Between Failures
MTTF	Mean Time To Failure
MTTR	Mean Time To Repair
NL	Network Layer
NLM	Number of Local Methods

NM	Network Management
NMF	Network Management Form
NMS	Network Management System
NOC	Number of Children
NSTS	Network Services Test System
ODL	Object Definition Language
ODP	Open Distributed Processing
OLO	Other Licensed Operator
OMA	Object Management Architecture
OMG	Object Management Group
OMT	Object Modelling Technique
ONP	Open Network Provision
OO	Object-Oriented
OOD	Object-Oriented Design
OOSE	Object-Oriented Software Engineering
ORB	Object Request Broker
OS	Operations System
OSF	Operations System Function
OSI	Open Systems Interconnection
OSI-SM	Open Systems Interconnection - Systems Management
PA	Provider Administrator
PCO	Point of Control and Observation
PDU	Protocol Data Unit
PNO	Public Network Operator
POTS	Plain Old Telephone Service
PTN	Public Telecommunications Network
QoS	Quality of Service
RACE	Research in Advance Communications in Europe
RFC	Request For Comments
RFC	Response For a Class
RMI	Remote Method Invocation
ROSE	Remote Operations Service Element
SCP	Service Control Point
SDH	Synchronous Digital Hierarchy
SDL	Specification and Description Language
SESE	Security Exchange Service Element
SL	Service Layer
SLA	Service Level Agreement

SM	Systems Management
SMASC	Secure Management Association Support Component
SMS	Service Management System
SNMP	Simple Network Management Protocol
SP	Service Provider
SP(N)	Security Profile (N)
SS7	Signalling System No.7
SSC	Security Support Component
SUG	Service Usage Group
TINA	Telecommunications Information Network Architecture
TINA-C	Telecommunications Information Network Architecture - Consortium
TLSP	Transport Layer Security Protocol
TMF	TeleManagement Forum
TMN	Telecommunications Management Network
TO	Telecommunications Operator
TO	Test Object
TOM	Telecom Operations Map
TTCN	Test and Tabular Tree Notation
TTP	Trusted Third Party
ULE	User Lost Erlang
UML	Unified Modelling Language
VASP	Value Added Service Provider
VDM	Vienna Development Method
VLSI	Very Large Scale Integration
VP	Virtual Path
VPC	Virtual Private Connection
VPN	Virtual Private Network
WMC	Weighted Methods per Class
WWW	World Wide Web