1

PM-1 3½"x4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT

PRECISIONSM RESOLUTION TARGETS

**Towards an Architecture
for
Open Distributed Management**

by

Colin Ashford, B.Sc., C. Eng.

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements of the degree of
Master of Computer Science
Ottawa-Carleton Institute for Computer Science

School of Computer Science

Carleton University

Ottawa, Ontario

10 April 1996

© Copyright

1996, Colin Ashford

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-612-13820-8

Name

COLIN ASHES

Dissertation Abstracts International is arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation. Enter the corresponding four-digit code in the spaces provided.

"Towards an Architecture for Open Distributed

SUBJECT TERM

Management

0984

U-M-I

SUBJECT CODE

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS

Architecture 0729
Art History 0377
Cinema 0900
Dance 0378
Fine Arts 0357
Information Science 0723
Journalism 0391
Library Science 0399
Mass Communications 0708
Music 0413
Speech Communication 0459
Theater 0465

EDUCATION

General 0515
Administration 0514
Adult and Continuing 0516
Agricultural 0517
Art 0273
Bilingual and Multicultural 0282
Business 0688
Community College 0275
Curriculum and Instruction 0727
Early Childhood 0518
Elementary 0524
Finance 0277
Guidance and Counseling 0519
Health 0680
Higher 0745
History of 0520
Home Economics 0278
Industrial 0521
Language and Literature 0279
Mathematics 0280
Music 0522
Philosophy of 0998
Physical 0523

Psychology 0525
Reading 0535
Religious 0527
Sciences 0714
Secondary 0533
Social Sciences 0534
Sociology of 0340
Special 0529
Teacher Training 0530
Technology 0710
Test and Measurements 0288
Vocational 0747

LANGUAGE, LITERATURE AND LINGUISTICS

Language 0679
Ancient 0289
Linguistics 0290
Modern 0291
Literature 0401
Classical 0294
Comparative 0295
Medieval 0297
Modern 0298
African 0316
American 0591
Asian 0305
Canadian (English) 0352
Canadian (French) 0355
English 0593
Germanic 0311
Latin American 0312
Middle Eastern 0315
Romance 0313
Slavic and East European 0314

PHILOSOPHY, RELIGION AND THEOLOGY

Philosophy 0422
Religion 0318
General 0321
Biblical Studies 0319
Clergy 0320
History of 0322
Philosophy of 0469
Theology 0323

SOCIAL SCIENCES

American Studies 0323
Anthropology 0324
Archaeology 0326
Cultural 0327
Physical 0310
Business Administration 0272
General 0770
Accounting 0454
Banking 0338
Management 0385
Marketing 0501
Canadian Studies 0503
Economics 0505
General 0508
Agricultural 0509
Commerce-Business 0510
Finance 0511
History 0358
Labor 0366
Theory 0351
Folklore 0578
Geography 0578
Gerontology 0578
History 0578
General 0578

Ancient 0579
Medieval 0581
Modern 0582
Black 0328
African 0331
Asia, Australia and Oceania 0332
Canadian 0334
European 0335
Latin American 0336
Middle Eastern 0337
United States 0585
History of Science 0398
Law 0615
Political Science 0616
General 0617
International Law and Relations 0814
Public Administration 0452
Recreation 0626
Social Work 0627
Sociology 0938
General 0631
Criminology and Penology 0628
Demography 0629
Ethnic and Racial Studies 0630
Individual and Family Studies 0700
Industrial and Labor Relations 0344
Public and Social Welfare 0709
Social Structure and Development 0999
Theory and Methods 0453
Transportation 0453
Urban and Regional Planning 0453
Women's Studies 0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES

Agriculture 0473
General 0285
Agronomy 0475
Animal Culture and Nutrition 0476
Animal Pathology 0359
Food Science and Technology 0478
Forestry and Wildlife 0479
Plant Culture 0480
Plant Pathology 0817
Plant Physiology 0777
Range Management 0746
Wood Technology 0306
Biology 0287
General 0308
Anatomy 0309
Bacteriology 0379
Botany 0329
Cell 0353
Ecology 0369
Entomology 0793
Genetics 0410
Limnology 0307
Microbiology 0317
Molecular 0416
Neuroscience 0433
Oceanography 0821
Physiology 0778
Radiation 0472
Veterinary Science 0786
Zoology 0760
Biophysics 0425
General 0996
Medical 0996

Geodesy 0370
Geology 0372
Geophysics 0373
Hydrology 0388
Mineralogy 0411
Paleobotany 0345
Paleocology 0426
Paleontology 0418
Paleozoology 0985
Polynology 0427
Physical Geography 0368
Physical Oceanography 0415

HEALTH AND ENVIRONMENTAL SCIENCES

Environmental Sciences 0768
Health Sciences 0566
General 0300
Audiology 0992
Chemotherapy 0567
Dentistry 0350
Education 0769
Hospital Management 0758
Human Development 0982
Immunology 0564
Medicine and Surgery 0347
Mental Health 0569
Nursing 0570
Nutrition 0380
Obstetric and Gynecology 0354
Occupational Health and Therapy 0381
Ophthalmology 0571
Pathology 0419
Pharmacology 0572
Pharmacy 0382
Physical Therapy 0573
Public Health 0574
Radiology 0575
Recreation 0575

Speech Pathology 0460
Taxology 0383
Home Economics 0386

PHYSICAL SCIENCES

Pure Sciences 0485
Chemistry 0749
General 0486
Agricultural 0487
Analytical 0488
Biochemistry 0738
Inorganic 0490
Nuclear 0491
Organic 0494
Pharmaceutical 0495
Physical 0754
Polymer 0405
Radiation 0605
Mathematics 0986
Physics 0606
General 0608
Acoustics 0748
Astronomy and Astrophysics 0607
Atmospheric Science 0798
Atomic 0759
Electronics and Electricity 0609
Elementary Particles and High Energy 0610
Fluid and Plasma 0752
Molecular 0611
Nuclear 0756
Optics 0611
Radiation 0463
Solid State 0346
Statistics 0984
Applied Sciences 0346
Applied Mechanics 0984
Computer Science 0984

Engineering 0537
General 0538
Aerospace 0539
Agricultural 0540
Automotive 0541
Biomedical 0542
Chemical 0543
Civil 0544
Electronics and Electrical 0348
Heat and Thermodynamics 0545
Hydraulic 0546
Industrial 0547
Marine 0794
Materials Science 0548
Mechanical 0743
Metallurgy 0551
Mining 0552
Nuclear 0549
Packaging 0765
Petroleum 0554
Sanitary and Municipal 0790
System Science 0428
Geotechnology 0796
Operations Research 0795
Plastics Technology 0994
Textile Technology 0994

PSYCHOLOGY

General 0621
Behavioral 0384
Clinical 0622
Developmental 0620
Experimental 0623
Industrial 0624
Personality 0625
Physiological 0989
Psychobiology 0349
Psychometrics 0632
Social 0451

The undersigned hereby recommend to
the Faculty of Graduate Studies and Research
acceptance of this thesis

Towards an Architecture for Open Distributed Management

submitted by

Colin Ashford B.Sc., C. Eng.

in partial fulfilment of the requirements for
the degree of Master of Computer Science

A handwritten signature in black ink, appearing to read 'T. R. P.', is written over a horizontal line.

Thesis Supervisor

A handwritten signature in black ink, appearing to read 'E. H. ...', is written over a horizontal line.

Director, School of Computer Science

Carleton University

10 May 1996

Abstract

Over the last decade, common carriers have been able to successfully deploy high-bandwidth, seamless, end-to-end services in an increasingly fragmented marketplace because of national and international consensus on open interface standards for new technologies such as ATM and SONET. However, the deployment of seamless systems to manage the new technologies has been less than successful, partly because of the “stove-pipe” mentality of many equipment suppliers and partly because of the inherent complexity and distributed nature of network management. Thus whilst service offerings are increasingly integrated and technology-neutral, the supporting management systems remain isolated and technology-specific. This forces common carriers to either develop their own, closed solutions to network-management integration or rely on a “rolling-chair” manual approach.

The development of integrated network-management systems for large networks cannot rely on a “one-size-fits all” standard design but calls for a combination of topographical, organisational, and technological approaches—in fact it calls for an *architecture* for open distributed management. We offer a taxonomy, encompassing both architectural design and architectural style, to help reason about software architectures and we place the Reference Model for Open Distributed Processing—an architectural reference model developed by ISO—within that taxonomy. We use the reference model and the taxonomy to develop an architectural design framework for Open Distributed Management which directly supports the design of open, interoperable network-management systems.

Acknowledgements

I would like to thank Prof. John Pugh, my supervisor, for his help, advice, and criticism during the research and to my colleague, Stéphane Poirier, for his critique of the Z specifications.

I would also like to dedicate this thesis to my wife, Jo-ann, in appreciation of her encouragement and patience during its preparation.

Table of Contents

Abstract.....	iii
Acknowledgements.....	iv
Table of Contents.....	v
List of Figures.....	vii
1 Introduction <i>or, What's all about?</i>	1
1.1 The Network Management Environment	1
1.2 Managing Large Telecommunications Networks.....	2
1.3 Standards and Consortia	4
1.4 The Way Forward	4
1.5 Layout of the Paper.....	5
1.6 Original Aspects of this Research.....	5
2 Abstractions, Models, and Reality <i>or, Thinking about it</i>	6
2.1 Dealing with Complexity.....	6
2.2 Modelling.....	8
3 A Software Reference Architecture <i>or, Categorising it</i>	12
3.1 Software Architectures	12
3.2 Architectural Design Frameworks.....	15
3.3 Architectural Style	18
4 Open Distributed Systems <i>or, Thinking about it in different places</i>	20
4.1 Characteristics of Distributed Systems.....	21
4.2 The Purpose of an Architecture for Open Distributed Systems	21
4.3 The Stakeholders in an Architecture for Open Distributed Systems	22
4.4 The Qualities of an Architecture for Open Distributed Systems	23
4.5 The Reference Model for Open Distributed Processing	24
4.6 Some Observations on Open Distributed Processing	28

5	An Architectural Framework for Open Distributed Management <i>or, On being more formal</i>	30
5.1	Open Distributed Management Concepts	30
5.2	A Reference Model for Open Distributed Management.....	36
5.3	Correspondence between the viewpoints.....	42
5.4	Transparencies	44
6	Generic Functional Models for Open Distributed Management <i>or, On helping the designer</i>	45
6.1	The Management Object Model	45
6.2	The Naming Model	47
6.3	The Binding Model	48
6.4	The Invocation Model.....	51
6.5	The Managed Relationship Model.....	52
6.6	The Transaction Model	54
7	An Implementation Framework for Open Distributed Management, <i>or On helping the programmer</i>	56
7.1	OSI Management Components	56
7.2	RM-ODM Components	56
7.3	Commercial Frameworks.....	57
8	Summary and Conclusions <i>or, So far—so good</i>	58
8.1	Summary	58
8.2	Conclusions.....	60
8.3	Future Work.....	61
8.4	Final Observations	64
A	A Small Specification <i>or, Trying it out</i>	65
A.1	Overview of Telecommunications Application	65
A.2	Enterprise Viewpoint	66
A.3	Information and Computational Viewpoint.....	67
A.4	Engineering Viewpoint.....	81
A.5	Conclusions.....	88
AA	SmalltalkV Simulation	89
AA.1	Simulation Script	89
AA.2	Transcript of Simulation	91
	References	92
	Abbreviations	100

List of Figures

Figure 1.1 Point-to-point Organisational Model	2
Figure 1.2 Distribution Examples	3
Figure 3.1 A Taxonomy for the Software Reference Architecture	13
Figure 4.1 ODP-Reference Model Viewpoints	25
Figure 5.1 Correspondence between the RM-ODM Viewpoints	43
Figure 6.1 OSI Management Point-to-point Model	45
Figure 6.2 The Management Object Model	46
Figure 6.3 RM-ODM Global Naming Model	47
Figure 6.4 RM-ODM Point-to-point Binding Model	48
Figure 6.5 RM-ODM Point-to-multipoint Binding Model	50
Figure 6.6 RM-ODM Relay Binding Model	51
Figure 6.7 RM-ODM Relationship Model	53
Figure 8.1 Visualisation of a Software Reference Architecture	63
Figure A.1—Application Scenario	65
Figure A.2—Information Viewpoint Overview	68

The masons carried in their heads a stock, not so much of patterns as of ideas, that grew by experience as they went from one site to the next.

Jacob Bronowski
The Ascent of Man

Chapter 1

Introduction *or, What's all about?*

In this chapter we set the scene for today's network management against the backdrop of an increasingly competitive and fragmented marketplace.

1.1 The Network Management Environment

Telecommunications has rapidly become an indispensable part of the *modus operandi* of a spectrum of organisations from large, multi-national corporations to small not-for-profit community associations. This has put pressure on common carriers, traditionally the telephone companies, to offer innovative supporting services such as LAN-to-LAN interconnection, virtual private networks, personal communications systems, and video conferencing. The common carriers have recognised that the only economically feasible way of delivering such services is by investing in an infrastructure of high performance switching and transmission equipment based on new technologies such as SONET, ATM, and Frame Relay. Common carriers have been largely successful in this strategy but the rapid introduction of the new technologies has put a great strain on their ability to manage the underlying communications infrastructure.

The term "network management" has traditionally covered activities ranging from the planning of new facilities, through the day-to-day care of the network, to personnel management. For the purposes of this discussion, we will restrict the term to apply to the day-to-day care of the network—such things as service activation, fault detection and recovery, performance analysis and load balancing, and billing. The activities of service

activation and fault detection alone typically account for 70% of network management costs.

Figure 1.1 Point-to-point Organisational Model

The new services offered by common carriers are more sophisticated than traditional services such as leased lines or POTS (Plain Old Telephone Service) and are typically built of components employing different technologies and of equipment supplied by different manufacturers. Whilst the common carriers would like to manage the new services as single entities rather than aggregates of different technologies, equipment manufacturers have been slow to recognise this and have, in the main, only provided equipment-level management systems using a point-to-point organisational model, see Figure 1.1. Thus common carriers have had to rely on their own resources to integrate management systems. Network management has consequently become one of the most important, yet confusing, topics in telecommunications today [Aid94].

1.2 Managing Large Telecommunications Networks

The three most important characteristics of telecommunications networks that fundamentally determine network management requirements are:

- real-time nature;
- the inherent distributed nature of telecommunications networks; and
- customer expectation of near-perfect availability and reliability.

The management of small telecommunications networks (say less than a few dozen nodes) can be accomplished within this norm in a manual fashion with relatively low-level management tools. With large networks (say over a hundred nodes) such an approach becomes untenable.

Figure 1.2 Distribution Examples

From a network-management perspective, the defining characteristics of large telecommunications networks are the scale of the distribution and the pluralistic nature of organisational authority. For example the international telephone network services many millions of subscribers and is organised into national and regional entities of varying degrees of autonomy from tight hierarchies to loose confederations. The two characteristics of scale and pluralistic authority dictate that any successful approach to management must be both fully-distributed geographically and federated organisationally. In particular a successful approach must recognise distribution strategies, such as point-to-multipoint and relay, and organisational strategies such as hierarchical, delegated, peer-to-peer, and domain (see Figure 1.2). Two other characteristics of large networks that influence the approach to management are the constant evolution of the network and the heterogeneity of both new and legacy equipment.

1.3 Standards and Consortia

The standards promulgated by ISO (International Organization for Standardization) and ITU (International Telecommunication Union) for switching, access, and transmission technologies have allowed common carriers to successfully integrate equipment from different manufacturers and provide seamless services to their customers. However, standards to permit the integration of network-management systems from different manufacturers have not been forthcoming. Whilst international standards exist, and are deployed, for the low-level management of equipment, effective standards to provide integrated management have been slow to develop.

Two initiatives to address the architectural aspects of distributed management have been published: Telecommunications Management Network (TMN) [ITU95] by the ITU and Omnipoint Integration Architecture [Kin95] by the Network Management Forum. In our opinion these initiatives have been only partially successful largely because both are actually high-level designs. They tend to be restrictive "one-size-fits-all" distribution and organisational designs that are deemed appropriate to all situations. They are just too restrictive to meet the needs of common carriers in the area of distributed management.

1.4 The Way Forward

What we believe is required to support the common carriers' need to manage large, heterogeneous networks, is a program of standardisation that develops an architecture for the design of open distributed management systems. It should be an *architecture* in the sense of a *support* for design rather than a grand design and should be *open* in the sense of subject to public consensus. It should also recognise the inherent distributed and federated nature of network management.

1.5 Layout of the Paper

The layout of the paper is as follows. In §2 we explore tools for dealing with complexity, in §3 we propose a taxonomy for software architectures, and in §4 we use the taxonomy to explore the nature of distributed systems. In §5 we propose a reference model for open distributed management and in §6 and §7 we outline the generic models and implementation framework respectively that complete the architectural design framework. In §8 we offer some conclusions and ideas for future work. In Annex A we offer a first-level confidence check by using the proposed architecture to specify a distributed telecommunications-management application.

1.6 Original Aspects of this Research

Whilst this research is based, in part, on our previous work, it goes much further. Original work reported in this paper includes the presentation of architectural-level concepts and principles in taxonomic form (§4) and two-dimensional form (§8). However, the most important part of this research concerns the Reference Model for Open Distributed Management (§5) and the generic functional models (§6). Whilst the former is based on the Reference Model for Open Distributed Processing, it is a significant refinement of the model in terms of concepts and principles and furthermore prescribes viewpoint languages, viewpoint correspondences, and requirements for programming-language bindings. The generic functional models are based on those of OSI Management but significantly extend their functionality to support fully-distributed network management. The complete specification and simulation of the small telecommunications application using the Reference Model for Open Distributed Management (Annex A) is original to this research, as is the use of a call-out mechanism in Z.

Chapter 2

Abstractions, Models, and Reality *or, Thinking about it*

In trying to design effective network-management systems to meet real-world requirements, it is necessary to set up a consistent framework with which to reason about the various aspects of the problem. An important part of such a framework is the notion of modelling as a way to describe the enterprise and implementation domains and to deal with the complexities of both. In this chapter we explore a general framework for reasoning about the problem by examining the inherent complexity of computer systems, the nature of modelling, and the relationship between enterprise-domain and implementation-domain modelling.

2.1 Dealing with Complexity

The most difficult problem facing the designer of network-management systems or, for that matter, large software systems, is that concerning their inherent complexity. As Brooks points out “The complexity of software is an essential property, not an accidental one” [Brook87]. This fact, he goes on to argue in his famous essay entitled “No Silver Bullet—Essence and Accidents of Software Engineering”, largely precludes software engineers from attaining the geometrical increases in performance that hardware engineers have achieved over the last 30 years. Also our ability as humans to cope with complexity is limited: Miller, a cognitive psychologist, reports that the maximum number of chunks of information an individual can keep track of simultaneously is seven, plus or minus two [Mil56].

The dilemma is very real: the complexity of the systems will not go away and yet mankind's ability to cope with complexity is limited. One approach to dealing with complex systems is to deal with them part at a time. As Dijkstra puts it "The technique of mastering complexity has been known since ancient times: *divide et impera* (divide and rule)" [Dij79]. There are two primary techniques for partitioning complex systems: abstraction and encapsulation. Abstraction separates out different concerns whilst encapsulation hides details of internal construction.

2.1.1 Abstraction

Abstraction—the separation of concerns—permits one to concentrate on important aspects whilst ignoring irrelevant details. In terms of software systems, Gabriel offers this definition of abstraction:

Abstraction in programming is the process of identifying common patterns that have systematic variations; an abstraction represents the common pattern and provides a means for specifying which variation to use [Gab93a].

The phrases *common patterns* and *systematic variation* imply a hierarchy: at the upper levels of the hierarchy, common patterns that represent the essential aspects of the problem at hand; further down the hierarchy, more refined patterns that represent increased detail. Class hierarchies in programming languages like Smalltalk are examples of programming abstraction.

2.1.2 Encapsulation

Encapsulation deals with complexity by the hiding of internal structure. Encapsulation in the form of subprograms was possibly the first mechanism used to try and tame the complexity of software systems. Seen originally in the 1950's as merely a labour-saving technique [Abb87], subprograms were eventually recognised as a way to abstract and

reuse program functionality and, from the 1960's on, extensive libraries of mathematical and statistical functions have been developed, for example NAG [NAG94].

Encapsulation entails the separation of a public, external interface from a private, internal implementation and forms the basis of the invoker-performer style of program design. Systems are decomposed into a number of components (performers) which are invoked through rigorously defined interfaces. The de-coupling of interface specification from implementation, permits designers to concentrate on the functionality of the components rather than their implementation. It also permits safe component re-implementation and component re-usability [Krug92].

2.2 Modelling

In order to reason about phenomena in the real (and the abstract) world, it is often convenient to construct a *model* of the system that gives rise to the phenomena. A model-theoretic approach is often taken because the systems under investigation may not be directly accessible, may not even yet exist, or may be too complex to comprehend directly. For example the world of sub-atomic particles is not directly accessible to physicists and it is unwise to carry out "what-if" investigations directly on mission-critical systems. But perhaps the most important use of models is to tame the complexity of systems. To try and understand the operation of a bank by merely observing the behaviour of customers, bank tellers, managers, computer systems, and the like, though not impossible, is daunting to say the least. Yet, by accepting abstract models of money, accounts, loans, interest, customers, and the interrelations between them, one can begin to understand banking. We offer the following definition of a model adapted from Wegner [Weg93]:

Model: a representation of behaviour or a state-of-affairs in a modelled world.

Models capture activity and existence and permit reasoning about real-world situations [ISO95b]. A model of a bank account might describes the process of debiting and crediting and also the meaning of a negative or zero balance. Similarly a model of an aircraft on landing approach might describe the configuration of the engines and control surfaces (the state of affairs) and the responses (activity) to external stimuli such as cross-winds and navigational input.

2.2.1 Modelling Concepts and Languages

There are two important aspects to the activity of modelling: the concepts that underpin the model and the language used to describe the model. The underpinning concepts of a model are those fundamental definitions and assumptions that are used as the basis for reasoning. In the case of, say, a client-server model of computer processing, the underpinning concepts would include the notions of a client, a server, a thread-of-control, and a message. The language used to capture the description must have these concepts as part of its basic lexicon, or have established mappings between these concepts and the lexicon of the language. For example in the area of modelling computer systems, the languages LOTOS [ISO88] and SDL [CCITT88] have the concepts of computer processes, thread of control, and interaction as part of their basic lexicon whereas the language Z [Spiv89] maps these concepts onto the lexicon of set theory.

To be effective, the modelling language must be able to capture both the static aspects (the state-of-affairs) and dynamic aspects (the behaviour) and precisely specify interfaces to the environment [Rum91]. Natural language is very powerful in this respect partly because of the size of the conceptual schema (the shared understanding between the users of the language) and partly because, ultimately, the underpinning concepts of the model have to be explained in natural language. Natural language can economically and rapidly describe the overall aspects of a model in an accessible manner but its lack of precision,

combined with different interpretations of the conceptual schema, can lead to an ambiguous and incomplete description. Synthetic languages with formal semantics have been proposed to alleviate this problem in the domain of computer systems [All86] and, whilst such languages do indeed describe models in a complete and unambiguous manner, they are often regarded as inaccessible to the average programmer. Semi-formal graphical languages, such as SDL [CCITT88], are often seen as a compromise between the ambiguity of natural language and the inaccessibility of formal languages.

2.2.2 Modelling and Computer Systems Design

Computer-systems design essentially consists of mapping an enterprise-domain model of a system of interest into an equivalent implementation-domain model. For example an enterprise-domain model of a bank account could be expressed in natural language and capture the processes of debiting and crediting and the semantics of a negative balance. The task of a computer-systems designer charged with automating this facet of banking is to map the enterprise-domain model of a bank account, expressed in this case in natural language, to an implementation-domain model expressed in a suitable computer language. The activity of mapping between models in the enterprise and implementation domains is the essence of computer-systems design and forms the basis of many design methodologies and, as we shall demonstrate, is the corner-stone of the Reference Model of Open Distributed Processing [ISO94a].

Some design methodologies, for example Structured Analysis [DeMa79] and Syntropy [Cook94] draw a sharp distinction between the model in the enterprise-domain and the equivalent model in a computer system. For a computerised stock-control system this differentiation appears both ontologically correct and practically useful: there is a difference between the cans of baked beans on the shelf and those represented by the system. However in many situations the computer model *is* the enterprise-domain model.

There is no meaningful difference between a personal bank account and its representation in a computer system. And the languages that are used to describe the models in the two domains are largely indistinguishable. Systems analysis can be carried out using a business-oriented language supporting the concepts of objects, classes, and messages; system's implementation can be carried out using an object-oriented programming language that supports the same concepts [Coad91].

Thus the boundary between the enterprise and implementation domains is often very fuzzy and this is the case in network management. Both management systems and the equipment to be managed are either embedded in computer systems or are controlled by computer systems. Network-management functions and their implementation in management systems can be modelled in terms of objects and interactions between objects. For example the management aspects of a telephone subscriber's line can be modelled in object-oriented terms in both the enterprise and implementation domains. In both domains, an object models the line and its attributes model the various telephony features (speed dialling, call waiting, three-way calling, etc.). In both domains, the activation of a feature is modelled as the sending of a message to the object and, in both domains, the effect will be a change in state of the object to reflect the activation of the feature. However in the implementation domain, the behaviour of the object, as represented by the program code, will perform whatever hardware-level operations are required to actually activate the feature on the subscriber's line.

Model mapping acts as a bridge between the enterprise domain—what has to be accomplished—and the implementation domain—how it is to be accomplished, but, as we have pointed out, the boundary between the enterprise and implementation domains is often indistinct.

Chapter 3

A Software Reference Architecture *or, Categorising it*

Abstraction and modelling are important aspects of a framework for reasoning about distributed network-management, but the crucial aspect is how these, and other, notions are brought together to support the design activity. In this chapter we examine the concept of a software reference architecture as a mechanism for bringing coherency to the framework.

3.1 Software Architectures

In the context of computer systems development, a software design can be regarded as a specification in a computer language of an implementation-domain model that is equivalent to some enterprise-domain model. It has been recognised for some time that the structure of such a specification is critical to the success of the development activity [Par85]; this structure has become known as *software architecture*. Software architecture is now an explicit field of study [Gar95, Mow95]. However, the term “software architecture” has two quite different meanings in the literature:

- (a) a *design* in the sense “the architecture of this system is the following component specifications”; or
- (b) a *style* in the sense “this system adopts a client/server architecture” [Gar95].

We believe that these are two legitimate aspects of software design which need support and we propose a *software reference architecture* to meet the need. A taxonomy for the reference architecture is shown in Figure 3.1.

The **architectural context** identifies the qualities of the architecture, the characteristics of the environment in which the systems are to work, and the stakeholders in the enterprise. From the architectural context, the taxonomy divides into two branches: architectural style and architectural design.

An **architectural style** provides coherence and integrity to a system by ensuring a unifying quality to the final design; it forms an important link between the architectural context and generic models.

Figure 3.1 A Taxonomy for the Software Reference Architecture

The **architectural design** supports the prescriptive definition of a system by identifying underlying concepts and paradigms. It might, for example, identify concepts of naming and binding and an object-oriented paradigm for modelling. From the architectural design, the taxonomy divides into two: an architectural reference model and an implementation framework. These three constitute our architectural design framework.

The **architectural reference model** supports design-in-the-large by providing languages for specifying models, mechanisms for combining the models into systems, and coarse

grained **generic models** that are pervasive to designs in the particular application domain. The **implementation framework**, on the other hand, supports design-in-the-small by specifying a set of fine-grained components and partially-implemented designs.

From the above taxonomy we derive the following definition for software architecture:

Software architecture: a set of models, patterns, and underlying concepts that exhibit conceptual integrity.

A software architecture enables a designer to design coherent systems. A good architecture for distributed systems facilitates the design of reliable, domain-specific distributed systems that are straightforward to build, predictable to use, and easy to maintain. What differentiates an architecture from a grab-bag of ideas is an underlying conceptual integrity and coherence that results in simplicity and straightforwardness in use of the resulting system [Brooks82].

A software architecture helps a systems designer in two ways: it provides pre-packaged solutions to recurring problems in the application domain; and it constrains the choice of a designer. One can immediately appreciate that providing pre-packaged solutions would be of great assistance to a designer, but helping a designer by constraining choice seems, at first, counter-intuitive. But a good architecture, by appropriately constraining a designer's choice, can promote interoperability between a resulting system and others built using the same architecture; can promote portability of a resulting application; and can ease future evolution.

The notion of a software architecture constraining a designer's choice obviously lays us open to the charge of promoting an aristocracy of architects over designers. We concur with Brooks's answer to the charge: yes and no. Yes, in the sense that to achieve conceptual integrity somebody must have control of the concepts, and no, in the sense

that a designer must have creativity and technical brilliance to deliver a cost-effective implementation [Brooks82].

3.2 Architectural Design Frameworks

Systems design must encompass the whole spectrum of design activity—from sub-system design, reflecting the overall structure of a system, to component design, reflecting the nuts-and-bolts of implementation. In terms of a framework to support systems design, it is however useful to divide the design spectrum into two and we shall extend Wegner's dichotomy [Weg93] of "programming-in-the-large and programming-in-the-small" to "design-in-the-large and design-in-the-small". An architectural design framework consists of: application-domain-specific concepts; an architectural reference model supporting design-in-the-large; an implementation framework supporting design-in-the-small; a set of generic models that are pervasive in a particular application domain; and a set of implementation components. Architectural design frameworks are normally application-domain specific.

For example an architectural design framework for network management will likely include the concepts of managing and managed entities, security, failure, alarms, naming, and relationships. It might use the OSI Management Framework [ISO89] as a reference model for relating managing and managed entities and might use GDMO [ISO92b] as a language for specifying the managed entities. It might also include the following generic functional models:

- a model of management entities based on objects;
- a model of communications based on remote operations;
- a model of access security based on passwords; and
- a model of system-resilience based on duplicated system components.

Not every implementation of a network-management system based on the above architectural design framework will necessarily incorporate all of the models. For example, a system intended for use in a friendly situation might not implement the security model or a system intended for use in a non-mission-critical situation might not implement the model of system resilience. It is up to the systems designer to select and integrate the models that are required to meet the requirements at hand. Thus a systems design activity is, in part, the selection and integration of models to meet a particular customer's requirement.

3.2.1 Architectural Reference Models

Architectural reference models define prototypical coarse-grained components, prototypical relationships between components, and languages for expressing both. They support the designer by providing prototypes against which actual designs may be compared and precise languages for expressing designs. Examples of such reference models are the OSI Reference Model for communications [ISO84] and the Object Database Standard [Cat94].

Generic models are parameterised models that capture important aspects of domain-specific designs. A designer uses such models in a design by supplying the appropriate parameter values and instantiating the models. For example a designer might instantiate a particular class of management objects by supplying parameter values for the attribute types and supported operations to the GDMO object class template. The process is similar to that of instantiating object classes through metaclasses in Smalltalk.

Generic models differ from patterns in that models have precisely-defined semantics and are normally supported by some form of definition language. Patterns on the other hand have only generally-defined semantics and no definition language.

3.2.2 Implementation Frameworks

Whilst architectural reference models support design-in-the-large, implementation frameworks support design-in-the-small and, as such, directly support the activity of programming.

Implementation Framework: a set of software components and partially-implemented designs.

Implementation frameworks typically consist of implementation-specific components and models which allow implementors to rapidly implement designs. A well-know example of such a framework is the Smalltalk Model-View-Controller framework for user interfaces [Kras88]. It contains a model for relating the underlying information to the view (or views) of that information and how it is controlled together with a set of classes to implement the screen widgets. Perhaps not the easiest framework to use but certainly better than having to implement it one's self [Rum94].

Components included in design frameworks are either instance-based or class-based. Instance-based frameworks consist of components implemented as libraries, API suites such as OLE [Broc95], or instance-based programming systems like PARTS [Lal93]. In principle (and often in practice), the user of a component has no knowledge of the component's implementation details and knows only the interface specification. At its simplest, an application designer will specify a series of calls on the interfaces of appropriate components in order to meet the functional requirements of a given application. However, components tend to have a fine grain of functionality, for example those in the Macintosh Toolbox [App92], and developments using such fine-grained components directly, tend to be lengthy and complex. Component designers can alleviate this problem by developing higher-level components from lower-level components by means of *aggregation* or *containment* [Kil94].

Class-based frameworks consist of components specified in terms of classes which a programmer can either instantiate and use directly or add value to by developing a *refined* [ISO95b] subclass specification. If the refinement is carried out in a disciplined manner, instances of subclasses can also be *subtypes* [ISO95b] of their parent types and support polymorphic operations [Boo91]. Both subtyping and polymorphism help in promoting re-usability.

A programmer using instance-based components has access only to object instances whereas a programmer using class-based components also has access to object classes. Thus instance-based frameworks can be regarded as a “black-box” approach whereas class-based frameworks can be regarded as a “grey-box approach”. Both techniques meet the central requirement of helping deal with the complexity of software systems and they form the basis of commercial software platforms such as DSOM [Betz94] and emipWORKS [Geig94].

3.3 Architectural Style

Architectural style is a difficult concept to define precisely: it is a unifying quality of a design that gives coherence in some aesthetic sense. Poirier and Ashford [Poi95] give a useful analogy between style in software architecture and style in building architecture. They draw a parallel between a Gothic style of arch giving coherence to the vault (roof) of a church and a decentralised style of event-reporting giving coherence to an alarm-reporting system. A Gothic arch is documented as a *pattern*—essentially a description of the function of the arch and an illustration of the arch but without dimensions or an indication of building materials.

We propose the following definition of architectural style:

Architectural style: a coherent set of patterns.

A pattern differs from a model in that a model is essentially prescriptive, having precisely-defined semantics and syntax, and in that a model is restricted to a particular application domain. A pattern, on the other hand, is essentially descriptive, having only generally-defined semantics and no syntax, and can often be applied in a variety of application domains.

A whole movement has sprung up around the concept of software design-patterns --- books [Gam95, Busc96], articles[Gab93a, Gab93b], and even a recent conference [Mar95]. The spiritual leader of this movement is the idiosyncratic building architect, Christopher Alexander, whose books on architecture [Alex77, Alex79] expound a user-centred view of building architecture based on recognising and documenting “good” patterns for such things as doorways, alcoves, and so-on. The software pattern movement proposes a similar approach to software development by recognising and documenting “good” patterns for such things as brokers, command processors, and so-on.

Chapter 4

Open Distributed Systems *or, Thinking about it in different places*

In the previous chapter we discussed software architectures in general terms; in this chapter we turn to architectures specifically for open distributed systems. Many of the same issues crop up—design-in-the-small, design-in-the-large, abstraction—but two fundamental issues are of over-riding importance in the distributed case: the physical distribution of computing resources and the heterogeneity of computing systems. The physical distribution of computing resources means that different parts of a computation can potentially be performed by different parts of the system and an architecture for distributed systems must concern itself with how that is to be accomplished. An architecture must also recognise that different parts of a distributed system may fail independently and it must be able to cope with such situations. In principle distributed systems could be built of homogeneous components but, in fact, most are not. An architecture for distributed systems must cope with different instruction sets, number formats, and operating systems. Much of the effort put into architectures for distributed systems is directed to hiding the distributed and heterogeneity aspects from a systems designer. This is an attempt to have one's cake and eat it too—to get the best of centralised and distributed worlds [Shro94].

In the balance of this chapter we will expand upon the purpose of, and the stakeholders in, an architecture for distributed systems and what we believe makes a good architecture. We will draw heavily on the work of Brooks [Brook82], ANSA Architecture for Open Distributed Systems [ANSA93], ISO Reference Model for Open Distributed Processing

[ISO95a, ISO95b, ISO95c, ISO95d], and our own experience. The discussion leads naturally to the question of how to develop good architectures for distributed systems and we offer an overview of a framework, standardised by ISO, for developing such architectures.

4.1 Characteristics of Distributed Systems

To begin with, we explore the characteristics that distinguish distributed systems from their centralised counter-parts. Perhaps the most obvious characteristic of a distributed system is the **remoteness** of components that causes interactions to be carried across space. Closely linked with the characteristic of remoteness is that of **asynchrony**: activities are not driven by a global clock and atomicity cannot be assumed. Whilst atomicity cannot be assumed, **concurrency** must be: components of a distributed system can execute in parallel. Even in the best regulated of systems, failures occur and, whereas centralised systems tend to fail completely, components of distributed systems tend to fail independently of one another manifesting in **partial failure** of the system as a whole. Finally distributed systems are characterised by a **lack of global state** - that is the global state of the system cannot necessarily be precisely determined.

4.2 The Purpose of an Architecture for Open Distributed Systems

The primary purpose of an architecture for open distributed systems is to promote **interoperability** between systems components and **portability** of applications across systems components. Much has been accomplished over that last decade with carriage-protocol interworking between computer systems. Open suites of communication protocols, such as Open Systems Interconnection (OSI) of ISO and TCP/IP of the Internet Society, have made the transfer of streams of bits reliable and common-place. But *interoperability* is about more than exchanging bits, it is about exchanging information and exchanging information requires a shared understanding between sender and

receiver. Similarly, portability of applications from one programming environment to another requires more than a common set of APIs; it, too, requires a shared understanding between service user and service provider. Such an understanding is often referred to as a **shared conceptual schema** or **universe of discourse**.

In many cases of systems design, a designer could choose between a number of roughly equivalent functional models to meet a particular requirement. For example, to maintain concurrency-control in a distributed database system, a designer could choose between a locking model, a time-stamping model, or an optimistic-update model. If designers of different parts of the system choose different concurrency-control models, interoperability may be difficult or even impossible to achieve. An architecture for distributed systems meets the objectives of interoperability and portability by essentially constraining a designer's choice of model. Constraining choice enough to promote interoperability whilst not over-constraining choice and complicating designs is, of course, a delicate balancing act!

An important side effect of constraining a designer's choice is the promotion of **re-use**. To meet a recurring functional requirement, a good architecture should provide a designer with an appropriate model that can be immediately incorporated in the overall systems specification. Furthermore, an implementation framework can support the rapid development of the system by means of implementational-level components and partially-implemented designs. Thus a good architecture promotes both the re-use of models and the re-use of implementations.

4.3 The Stakeholders in an Architecture for Open Distributed Systems

The most direct stakeholder in an architecture for open distributed systems is the systems designer. A poorly-conceived, poorly-defined architecture will compromise a designer's ability to deliver a quality product; often designers will simply ignore an architecture that

inhibits rather than promotes quality. Yet a well thought-out, lean architecture can help in the design of reliable and maintainable systems [Whitt94]. The customer can also be regarded as a stakeholder at least in an indirect sense: the quality of the architecture will be reflected in the maintainability and reliability of delivered systems. But, in a more direct sense, customers who plan to subsequently extend and interwork systems will benefit from a clean architecture of well-defined interfaces and logical construction.

4.4 The Qualities of an Architecture for Open Distributed Systems

Any subjective statement on what makes a good architecture for open distributed systems is potentially contentious but most authorities would, we believe, agree with a majority of the following points.

The over-riding qualities of an architecture are that it be applicable to a **wide class of problems** within the application domain and that it should be **minimalist**. In the particular case of distributed network-management, an architecture should be applicable to voice, data, and video networks, be applicable to different transmission and switching technologies, and be applicable to both equipment and service management. A good architecture should contain only the models and concepts that are needed to meet the requirements of the problem domain in a straightforward manner. Additional models compromise interoperability and re-usability.

A good architecture should, as far as possible, be **implementation-technology neutral**. This quality is often in tension with the requirement that the subsequent specification be **complete and unambiguous**. The desire for a technology-neutral architecture militates towards more abstract models specified in more abstract languages such as Z [Spiv89] whereas the desire for completeness militates towards more implementation-like models specified in more programming-like languages such as IDL [OMG92].

A good architecture should be **durable**: it should be able to transcend changes in technology, changes in the business, and changes in the regulatory environment. In the particular case of an architecture for distributed network management it should, in addition, be **scalable** and recognise **legacy** systems.

4.5 The Reference Model for Open Distributed Processing

It is important that any architecture for open distributed systems clearly separate the essential aspects of the system from the incidental aspects of distribution. Abstraction—the suppression of irrelevant detail—is clearly a useful tool here. However, the analysis of distributed systems is complicated by the orthogonal issues of policy, functionality, distribution, and implementation technology. The traditional responses to such an increase in complexity is “bigger and better” abstractions [Gab93a]. One way to get better abstractions is by the layering of abstraction over abstraction as is done in the Open Systems Interconnection (OSI) Reference Model for open communications systems [ISO84]. In that model, each layer provides service to the layer above by providing carriage of the latter’s Protocol Data Unit (message) embedded in its own Protocol Data Unit [Ash93a]. For example the Transport Layer provides the abstraction of an end-to-end connection to the Session Layer whilst using the abstraction of a virtual circuit provided by the Network Layer. The layering of abstractions is successful in the case of the OSI Reference Model because the abstractions presented at each layer and the specification of the interfaces between the layers are well defined.

Attempts to use the layering of abstractions to model distributed systems by assigning business issues to the upper layers, informational issues to the middle layers, and implementation and distribution issues to the lower layers have, in our opinion, been less than successful. We believe that this lack of success is because the abstractions of functionality, information, and distribution are not ontologically related by hierarchy.

4.5.1 The ODP Viewpoints

The approach taken to analysing the various ontological aspects of distributed systems proposed by ISO in its Reference Model for Open Distributed Processing (RM-ODP) [ISO95a, ISO95b, ISO95c, ISO95d] is one of abstracting systems from five orthogonal viewpoints, see Figure 4.1. The salient aspects of a distributed system can be specified and reasoned about from the five viewpoints and, by combining and transforming specifications from these viewpoints, a system can be fully specified. The five viewpoints reflect the five important areas of concern, namely:

- enterprise:** what the system is about, its goals, how it achieves its goals and how it fits into its environment;
- information:** the information the system needs for processing and storage to meet the enterprise viewpoint goals;

Figure 4.1 ODP-Reference Model Viewpoints

- computational:** the interaction of the various components of the system viewed as sources and sinks of information;

- engineering:** the handling of the details of distribution including such things as name resolution and binding, and the provision of various transparency services for such things as failure-recovery, persistency, and relocation;
- technology:** the detailed specification of appropriate hardware and software components.

It is important to note that the RM-ODP does not specify an architectural reference model, say for client-server systems or, in our case, network management—it merely specifies the framework for developing such a reference model. At first glance this seems a bit of a cop-out. But it is, in fact, a measured response to the realities of designing distributed systems. Not all classes of distributed systems will need the same set of concepts or models: simple client-server systems may not need dynamic binding, whereas sophisticated distributed processing systems may not only need dynamic binding but may also need access to trading and repository services. An architectural reference model for such classes of systems should reflect the requirements of such systems—and nothing more. ODP provides a set of concepts, functions, and transparencies that are generally useful for all classes of distributed systems; a particular reference model may select from, add to, or refine the set.

4.5.2 Viewpoint Languages

Each viewpoint of a system is expressed in a particular viewpoint language: a set of concepts and rules for expressing the area of concern. Whilst the ODP standards do not prescribe languages for the viewpoints, they do specify the basic requirements for such languages. It is assumed that architects for particular classes of systems will choose appropriate languages for the viewpoints based on such concerns as formality, accessibility, and tool support and show how the viewpoint-language requirements are

met by the chosen language. Part of the RM-ODP [ODP95d] is devoted to **architectural semantics**—the mapping of ODP viewpoint concepts to the standardised formal description languages LOTOS, SDL, and Z.

4.5.3 Viewpoint Correspondence

Whilst each viewpoint language expresses only the concepts relevant to its viewpoint, it is important to remember that each viewpoint expresses different aspects of the *same* system and it is self-evident that specifications of the different viewpoints of the same system must be consistent. ODP proposes that, for a given system, consistency between viewpoint specification can be demonstrated by defining a set of correspondence rules that steer the specifications of a system in one viewpoint language to a corresponding specification in another viewpoint language. The transformation provides the basis for an analysis of behavioural compatibility between the two specifications.

In fact the RM-ODP only requires that a statement of correspondence between the computational and informational viewpoint specifications and between the computational and engineering viewpoint specifications be made. The RM-ODP requires that the information and computational viewpoints be expressed in terms of objects—but not necessarily the same objects. For example, an object in the information viewpoint could, for practical reasons, correspond to a number of objects in the computational viewpoint but any intermediate transitional states in the computational objects are abstracted as an atomic transition in the information object. In simple cases, appeals to intuition, if not formality, can be made to show equivalence. More complex cases may prove to be less tractable. One obvious and straightforward approach to showing equivalence is to make the computational and information objects congruent and to use a single language to express the informational schemata and the computational interfaces. We shall propose this approach for open distributed management.

The RM-ODP prescribes a one-to-one correspondence between a computational interface and an engineering interface supported by a **basic engineering object**. The basic engineering object is itself supported by stub, binder, and protocol objects which form the channel that facilitates location-independent interactions between basic engineering objects. Although the ODP Reference Model goes into great detail about the concepts and structuring rules for the engineering viewpoint (reflecting the realities of operating systems, processes, linking, and so on), in practice this is of little concern to the systems designer. For a given distribution technology, say CORBA [OMG92] or DCE [Ros92], tools such as compilers and stub generators and libraries of routines for marshalling and un-marshalling data, converting data formats, and locating interfaces take care of the engineering concerns.

4.5.4 Transparencies

One of the most important aspects of the RM-ODP is the notion of transparencies. Transparencies allow computational viewpoint specifications to be made without concern for such things as physical distribution and partial failure. The co-ordination functions that support the transparencies are implemented in the engineering viewpoint.

4.6 Some Observations on Open Distributed Processing

The five viewpoints were first proposed in the late eighties by the Advanced Networked Systems Architecture (ANSA) project, originally part of a United Kingdom government initiative to support research into advanced, distributed systems and later supported by the European Community. The approach was subsequently submitted to ISO during the development of the Reference Model for Open Distributed Processing which has gained wide support in the standards community, including accredited standards bodies such as the International Telecommunication Union (ITU) and consortia such as the Object Management Group (OMG) and TINA-C.

It can be argued that the ODP standards merely codify the abstraction mechanisms systems designers have always used and, in many ways, this is true. But what ODP has also done is thrown down a challenge to architects to develop formal languages to specify these abstractions and formal techniques to show correspondence between the languages. Many of these challenges can be met with tools that are available today: object technology, standardised formal description languages, distribution platforms, execution platforms, and on. The biggest challenge is probably to be able to develop a formal language to specify the enterprise viewpoint and to show the correspondence between such a language and those of the other viewpoints.

Chapter 5

An Architectural Framework for Open Distributed Management *or, On being more formal*

In the previous chapter we discussed architectures for distributed systems in general terms. In this chapter, and the two following chapters, we turn our attention to an architectural design framework for open distributed management. In this chapter we will develop a Reference Model for Open Distributed Management (RM-ODM) by examining the fundamental concepts underpinning open distributed management and proposing languages for the five viewpoints. In the two subsequent chapters we will outline our approach to developing generic functional models and an implementation framework.

5.1 Open Distributed Management Concepts

The primary sources for these concepts are Open Distributed Processing [ISO95b] and Ashford [Ash93b]; OPD offers more general definitions whilst Ashford offers definitions more specific to network management.

5.1.1 Object-oriented Concepts

The following are definitions of the underpinning object-oriented concepts for open distributed management.

Management object: a model of an entity of interest to management; an object is characterised by its behaviour and its state.

The management object is the basic unit of modelling: it possesses behaviour, which reflects the active nature of the entity it models, and state, which reflects the history of the behaviour.

State: a stable, internal condition of a management object.

The state of an object captures the previous behaviour of an object and influences future behaviour: it may be changed by interactions with other objects or by internal events. State reflects the informational aspects of entities.

Behaviour: a change in state of a management object.

Encapsulation: a property of a management object such that its state can only be accessed by means of prescribed signals across the object boundary or by means of prescribed relationships between objects.

Encapsulation is the means by which the external view of an object is separated from its internal construction. By ensuring that access to the object is limited to prescribed mechanisms, a designer, whilst required to honour the interface specification, is free to implement (and re-implement) an object without affecting the behaviour of clients.

Operation: a behaviour of a management object; it is initiated by the reception of an invocation.

Operations reflect the information processing aspects of entities.

Event: an autonomous behaviour of a management object; it may be signalled by the emission of a notification.

The concept of an autonomous event for an object is extremely important in network management and reflects the notion of autonomous changes in state, such as alarm conditions, in network entities. It also implies that the architecture must support multiple threads of control.

Invocation: a signal, possibly parameterised, requesting an operation be performed by a management object.

The operation invocation is the primary mechanism by which the information and information-processing aspects of an object are made available to a client.

Notification: A signal, possibly parameterised, indicating that an event has occurred.

The notification is the basic mechanism by which an object signals the occurrence of an event.

Reply: a signal, possibly parameterised, indicating a response to an invocation or notification; replies are often classified as normal or exceptional.

Interface: a point on the boundary of a management object at which a prescribed protocol exists.

Open interface: an interface that supports a published protocol and at which conformance may be tested.

Protocol: a set of syntactic and semantic rules for exchanging information.

Exception: a condition that precludes normal execution.

5.1.2 Information Modelling Concepts

Informational modelling is fundamental to any distributed processing architecture—so much so that ODP assigns one of the five viewpoints to it. ODP requires that the information model be expressed in a language that captures the static, dynamic, and invariant aspects of the information model. The following definitions are refinements of those in RM-ODP.

Static Schema: a specification of the state of one or more management information objects at some point in time.

Dynamic Schema: a specification of the allowable behaviour of one or more management information objects.

Invariant schema: a set of invariants over one or more management information objects that must remain true for some period of time; the predicates constrain the possible states and behaviour of the objects.

Relationship: an invariant schema between two or more management information objects.

Composition: a combination of two or more management information objects yielding a new information object at a new level of abstraction.

The characteristics of the new object are determined by those of the original objects and the manner in which the objects are combined.

5.1.3 Architectural Design Concepts

The following are definitions of the architectural concepts relevant to open distributed management.

Name: a token used to refer to a management object.

A name is not necessarily an unambiguous reference to an object. Different objects may be assigned the same name and the same object may be assigned different names; names may be re-assigned during the lifetime of the object. Names are essentially for human consumption.

Address. The name of a communicating entity; the format of the name is oriented towards machine processing.

Naming and addressing often become confused because of humans' tendency to overload names with elements of addressing, routing, and object description [Hauz86]. Shoch [Shoch78] defines three distinct concepts with respect to communications systems:

the **name** of a resource is *what* we seek, an **address** is *where* it is, and a **route** is *how to get there*.

In general we are interested in the objects bound to communicating entities rather than the communicating entities themselves; addresses are thus said to *refer* to objects. The binding of an object to an address is not necessarily permanent and, in fact, may be changed to support object mobility [Levy90]. Addresses are special types of names and exhibit many of the characteristics of object names: they can have structural, temporal, and cardinal aspects. However addresses are generally distinguished by obeying strict syntactical rules, are often numeric in format but, above all, are recognised by the communicating entities [Haus86].

Name Resolution: the mapping of a name to its associated address.

Contract: an agreement governing part of the collective behaviour of a set of management objects; a contract is specified in terms of obligations, permissions, and prohibitions.

Obligations require a behaviour; *permissions* permit (but do not require) a behaviour; and *prohibitions* forbid a behaviour.

Policy: A set of rules related to a particular purpose; rules can be expressed in terms of obligations, permissions, or prohibitions.

Binding: a contractual context over object interfaces; it results from a given establishing process and enables object interaction.

The process of establishing a binding is probably the most distinguishing feature of distributed systems. In all but the most simple of systems, it is done at runtime and can often involve negotiation within the parameters of a given contract. Intimately connected with the process of binding are the concepts of name resolution and trading. The ability

to bind (and re-bind) at runtime enables such services as trading and object relocation, and transparencies such as migration and failure.

Trading: the exchange of information, via a third party, by objects regarding contracts between them.

5.1.4 Specification Concepts

Architectural designs, to be of value, must be prescriptive and ODP offers a framework for the specification of viewpoints. The precise way in which the following concepts are expressed in open distributed management will be dependent on the particular language chosen for the particular viewpoint.

Type: a predicate over the characteristics of a management object, managed relationship, interface, operation, or notification.

Subtype: an additional predicate over a type, termed the supertype, such that the resultant type, termed the subtype, satisfies the supertype.

Object class: a template for the generation and management of objects.

Template: the specification of the common features of a collection of management objects, relationships, interfaces, operations, or notifications such that an instance may be created.

Inheritance: a mechanism for deriving the specification of a descendant from the specification of an ancestor such that, in addition to its own characteristics, a descendant acquires all (strict inheritance), or part, of the characteristics of its ancestor. A descendant may have a single immediate ancestor (single inheritance) or multiple immediate ancestors (multiple inheritance). The relationship resulting from single inheritance may be represented by a tree; that resulting from multiple inheritance, by a directed, acyclic graph.

Invariant: a logical predicate that must remain true during some scope.

Pre-condition: a logical predicate that must be true immediately before an operation or immediately before an event.

Post-condition: a logical predicate that must be true immediately after an operation or immediately after an event.

5.2 A Reference Model for Open Distributed Management

ODP does not prescribe languages with which the various viewpoints in a particular architecture should be described—it only specifies requirements for the languages. It is assumed that the architect will pick the most appropriate language for the particular problem domain.

In choosing languages for the Reference Model for Open Distributed Management (RM-ODM) viewpoints, we have had to strike a balance between a number of factors: formality, accessibility, availability of tools, and acceptability to the telecommunications industry. In our choices we have recognised the need to retain the notion of an implementation-technology neutral architecture—at least in the enterprise, informational, and computational viewpoints.

5.2.1 Enterprise Viewpoint for Open Distributed Management

5.2.1.1 Description

The enterprise viewpoint specification for the RM-ODM is concerned with the purpose, scope, and objectives of the network-management system in terms of the following items:

- (1) the roles to be played by the enterprise objects;
- (2) the activities to be undertaken by the system; and

- (3) policy statements about the system.

Both roles and policy statements are defined in terms of permissions, obligations, and prohibitions with respect to the enterprise objects fulfilling the roles.

In general, enterprise objects will be coarse-grained such as “manager”, “agent”, or “customer”, and may play more than one role in a network-management system—a supplier of one service may be a consumer of another service. The policies that apply to enterprise objects will be similarly coarse-grained: obligations such as “forward all critical alarms” or permissions such as “use backup facilities in case of failure”.

5.2.1.2 Language

We are not aware of a formal specification language devoted to enterprise modelling although a proposal has been accepted by ISO to begin an item of work in this area [X395]. In meantime, we propose to use natural language stylised to meet the enterprise viewpoint specification requirements. Roles and policy statements are stylised in terms of permissions, obligations, and prohibitions and activities are specified in a narrative style describing sequences of behaviours.

5.2.2 Information Viewpoint for Open Distributed Management

5.2.2.1 Description

The information viewpoint for the RM-ODM is concerned with the information requirements of a distributed management system: what information is required; how the various items of information relate to one another; and how the information is permitted to change. The specification is couched in terms of management information objects, which reflect the real-world entities of interest, and three schemata, which capture the semantics of the information. The invariant schema specifies predicates over the

information objects that must remain true for some period of time; the static schema specifies the state of the information objects at some point in time (often after initialisation); and the dynamic schema specifies how the information objects may change. These schemata are similar to those found in traditional data modelling activities.

Management information objects are based on the management object model described in [ISO92c]. Information objects are atomic but more complex objects can be represented by composites of management information objects. However, as Odell points out [Ode94], there are a number of interpretations of the semantics of composition and we shall therefore not attempt a precise definition at the architectural level. We shall permit designers to define the semantics of composition by means of the invariant schema.

5.2.2.2 Language

The selection of an appropriate language for information-viewpoint specifications is complicated because of the large choice available. We can however divide candidate languages into three broad categories: completely formal languages, primarily from academia, such as Z [Spiv89]; semi-formal languages from the telecommunications-standards community, such as GDMO [ISO92b]; and semi-formal languages from the general object-oriented design community such as OMT [Rum91].

Completely formal languages would entirely fulfil the requirements for an information viewpoint language in RM-ODM but we recognise that, without adequate training, it is difficult for developers to reach a reading proficiency in any of the formal languages. The lack of immediate accessibility and the lack of industrial-strength tools would, we believe, compromise the acceptability of RM-ODM in the telecommunications industry. On the other hand, the use of any of the semi-formal languages from general object-oriented design community, whilst more accessible and better supported by industrial-

strength tools, is likely to be frowned upon by the standards organisations because of the strong commercial overtones. We therefore believe the most prudent choice is to take the most widely accepted information specification language in the telecommunications industry, GDMO/GRM [ISO92b, ISO95e], and extend it to meet the requirements of the information viewpoint language.

GDMO/GRM offers formal (formal in the sense of unambiguous) mechanisms for the specification of the syntax of management objects (attributes, operation signatures, etc.) and managed relationships (cardinality, relationship operations, etc.) but offers only natural language for the specification of the behavioural aspects. However, ISO is actively studying the use of Z as an adjunct to GDMO/GRM to formally specify behaviour and we therefore propose that Z (liberally commented) be used to specify the behavioural aspects of management objects and managed relationships.

5.2.3 Computation Viewpoint for Open Distributed Management

5.2.3.1 Description

The computational viewpoint for RM-ODM is directly concerned with packaging for distribution. Whilst it does not address interaction mechanisms directly (this is left to the engineering viewpoint) it does decompose the system into objects that interact with one another at well-defined interfaces. The specification of the computational viewpoint is expressed in terms of management computational objects, interfaces, and bindings between interfaces.

Management computational objects are based on the management object model described in §6.1 and are specified in terms of:

- computational interfaces; and
- management computational object behaviour.

Computational interfaces are specified in terms of an interface signature that comprises of:

- the set of operation invocations available at the interface;
- the set of event notifications available at the interface;
- interface behaviour; and
- a statement of the causality (i.e. managed or managing) of the interface.

Operation invocations and event notifications are specified in terms of signatures that comprise:

- the name of the invocation or notification;
- the number, names, and types of the parameters;
- a set (possible empty) of responses containing the name of the response and the number, names, and types of the parameters.

5.2.3.2 Language

Whilst the computational viewpoint is modelled in terms of objects, the major aspects of the specification are in terms of interface signatures and interface behaviour. Interface signatures indicate the permissible interactions (operation invocations, event notifications, and responses) and behaviour indicates the particular interactions in a given situation. Behaviour can specify sequences of interactions at one or more interfaces of one or more management computational objects. The style of behaviour specification is in terms of sequences of interactions between interfaces.

GDMO provides suitable tools for specifying computational interface signatures and, given our choice of GDMO for specifying management information objects, we propose to use GDMO for specifying interface signatures. This choice will not only provide a

consistent notation for both viewpoints but will help in demonstrating consistency between viewpoint specifications.

As we have pointed out, GDMO only provides formal tools for specifying the syntactical aspects of management objects and that additional techniques have to be adopted to formally specify behaviour. In the information viewpoint we proposed Z and we believe that Z would be an appropriate choice for specifying computational interface behaviour.

5.2.4 Engineering Viewpoint for Open Distributed Management

5.2.4.1 Description

The engineering viewpoint focuses on the way object interaction is achieved. While the computational viewpoint is concerned with when and why objects interact, the engineering viewpoint is concerned with how they interact. ODP requires a direct correspondence between objects in the computational viewpoint and those in the engineering viewpoint: there is a one-to-one correspondence between computational objects and basic engineering objects. Within the engineering viewpoint, basic engineering objects communicate with one another by channels which provide data conversion, communications protocol functions, and, if necessary, protocol conversion.

5.2.4.2 Language

The specification of basic engineering objects is the traditional craft of programming—that is, the specification of algorithms to execute on some hardware platform. We do not propose to prescribe a language for the engineering viewpoint but rather to permit developers to choose the most appropriate language for their environment. We expect object-oriented languages such as C++, Smalltalk, and Eiffel will be the most popular choices but we do not rule out the use of procedural languages such as C and Ada. We

propose that whatever language is chosen, a mapping between the linguistic constructs of the language and those of the computational viewpoint specification language be stated. The requirements on these mappings are defined in §5.3.

The specification of the functions within a channel are largely the responsibility of communications protocol designers and the choice of an appropriate language for the specification of these aspects of the engineering viewpoint of RM-ODM is not necessary. In fact, these aspects are nearly always automatically generated from computational viewpoint specifications by means of software tools.

5.2.5 Technology Viewpoint for Open Distributed Management

The technology viewpoint is concerned with the selection of particular hardware and software components to implement an open distributed management specification. Whilst the selection must be consistent with the other viewpoints (and, in particular, the engineering viewpoint) it will largely be governed by commercial and practical considerations and is thus beyond the scope of RM-ODM.

5.3 Correspondence between the viewpoints

In principle, the complete specification of an open distributed management system will consist of five separate viewpoint specifications each couched in a different language, or language sub-set, but each describing the same underlying system. In order to demonstrate that the complete specification is coherent, it is not only necessary to demonstrate that each viewpoint-specification is coherent, but to demonstrate that the correspondence between the viewpoint-specifications is coherent. As we pointed out in Chapter 4, this does not mean that we have to demonstrate consistency between each viewpoint-specification and those of the other viewpoints, but rather key correspondences between viewpoints.

In reality, the essential specification of any open distributed management system will consist of three viewpoint specifications: the enterprise, the information, and the computation. Important specification issues will revolve around the enterprise and information viewpoints—what has to be achieved and the information processing necessary to achieve it. The correspondence between these viewpoints is difficult to define in a precise manner but, since we propose a consistent information modelling paradigm in each viewpoint—objects and relationships—it will be possible to express the coarse-grained objects of the enterprise viewpoint as aggregates of the finer-grained objects of the information viewpoint.

Figure 5.1 Correspondence between the RM-ODM Viewpoints

The RM-ODP permits, but does not require, a one-to-one correspondence between objects in the information viewpoint and those in the computational viewpoint. Although in some cases of re-use, granularity of distribution, and packaging, it may be desirable not to do so, in most cases an object in the information viewpoint can map directly onto an object in the computational viewpoint which, in turn, maps directly onto a basic engineering object. We thus propose that, in general, there be a one-to-one correspondence between objects in the information viewpoint, those in the computational viewpoint, and basic engineering objects, see Figure 5.1.

The programming-language binding requirements (q.v. §5.2.4.2) we propose are:

- (1) that the mapping from ASN.1 data types to those of the programming language be defined;
- (2) that the mapping from the GDMO signatures to the procedure call/return and exception mechanisms of the programming language be defined; and
- (3) that the mapping from the declarative specification of the behaviour in Z to the linguistic constructs of the programming language be defined.

5.4 Transparencies

We propose that the RM-ODM support three transparencies: access, location, and transaction. RM-ODP models the functions that support these transparencies in the engineering viewpoint and thus completely invisible to the computational viewpoint. Given the legacy of OSI Systems management, location and transaction functions cannot be made completely invisible to the computational viewpoint, although access transparency is fully supported by the Common Management Information Protocol (CMIP) [ISO91a].

Chapter 6

Generic Functional Models for Open Distributed Management *or, On helping the designer*

An important component in an architectural design framework for open distributed management is the set of generic functional models that is relevant to the domain of network management. Many of the generic models that will be required to support open distributed management are extensions to those models already used in OSI Management. An overall illustration of OSI Management is shown in Figure 6.1.

Figure 6.1 OSI Management Point-to-point Model

In this chapter we present the key models that assist in the design of open distributed management systems.

6.1 The Management Object Model

We propose that the fundamental unit of modelling in the five open distributed management viewpoints be the *management object*, see Figure 6.2. Depending on the particular viewpoint, different aspects of the management object model (e.g. informational, computational, etc.) will assume different levels of importance but nonetheless all viewpoints will be modelled in terms of management objects.

Figure 6.2 The Management Object Model

The management object model is based on the OSI Management managed-object model [ISO92c] which supports a single interface-type—the managed interface. In order to adequately model fully-distributed management systems, the managed-object model must be refined by the introduction of a second interface type, the *managing interface*. A management object can support at most two interfaces, each of different causality:

- a managed interface that accepts operation invocations and emits event notifications; and
- a managing interface that emits operation invocations and accepts event notifications.

A management object that only exhibits a managed interface is a managed object in the OSI Management sense; a management object that only exhibits a managing interface is a managing object and it is that which we propose for modelling managing-system functionality.

GDMO, the template language we are proposing to be used to specify management objects, does not, in its current form, explicitly support the managing interface. Whilst we had considered extending the template notation to permit the specification of invoked operations and received event notifications, we concluded that the existing behaviour template was the most appropriate mechanism for capturing such specifications.

6.2 The Naming Model

The naming model used by OSI Management is based upon that defined in the international standard for the Directory [ISO91b]—an ambitious standard to promote global naming, name-resolution, and name-authorisation services. In OSI Management, the name of a managed object is a single, mandatory, immutable attribute of the object that is unambiguous within some naming context.

Figure 6.3 RM-ODM Global Naming Model

A naming context is normally realised by another managed object which has name-authorisation responsibilities assigned to it. Naming contexts can be nested within one another to form a single-rooted tree of managed objects, often referred to as the *naming tree*. The name of a managed object can thus be made unambiguous within some containing naming context by a concatenation of the names of all the containing management objects to, and including, that of the managed object in question; this name is termed the *relative distinguished-name*. The Directory standard defines mechanisms that have been implemented by international and national naming authorities to allow a relative distinguished name to be qualified to make it globally unambiguous; such a qualified name is termed a *distinguished name*.

We propose that the OSI Management naming model be used to name management objects in RM-ODM. A single distinguished name is used to refer to both the managed and the managing interfaces; the context of the use of the name implies the particular interface. OSI Management standards permit the use of relative distinguished names within a single managed system relying on the client's being aware of the naming context. However, in a fully-distributed environment this is likely to cause name clashes. We thus propose that only distinguished names be used in RM-ODM, see Figure 6.3.

Actual naming contexts are specified by supplying parameters to name-binding templates defined in GDMO. The name-binding templates inherently support distinguished names and thus need not be amendment for RM-ODM.

6.3 The Binding Model

The binding model defined in OSI Management is similar to the compound binding model described in the RM-ODP except that the former is couched in non-object terms.

Figure 6.4 RM-ODM Point-to-point Binding Model

The OSI Management model consists of a managing-system interface that is bound to a managed-system interface and an agent that provides access to a number of named managed-object interfaces in the managed system, see Figure 6.1.

The model is semi-dynamic: dynamic with respect to the binding between the managing system and the managed objects but static with respect to the binding between the managing system and the managed system.

6.3.1 The Point-to-point Binding Model

The point-to-point binding model we propose for RM-ODM is based on the semi-dynamic OSI Management model and on the management object model. A managing object models the functionality of the managing system and an *agent management-object* models the functionality of the agent. See Figure 6.4.

The managing-object interface is statically bound the single agent management-object managed interface by means of Association Control Service Element (ACSE) [ISO87]. The managing object will use the A-ASSOCIATE service of ACSE with appropriate parameters such as the presentation address of the agent management-object interface, the presentation context, and quality of service requirements. ACSE will use services of the presentation layer which will, in turn, use services of the session layer to establish a connection-oriented association between the managing object and the agent management-object.

The managing-object interface is dynamically bound to appropriate managed object interfaces by means of the Common Management Information Service Element (CMISE) [ISO91a]. The managing object may use any of the CMISE services (M-ACTION, M-GET, M-SET, M-CREATE, M-DELETE, or M-CANCEL-GET) with appropriate parameters including the name of the target managed object. CMISE will use presentation-layer services to transfer the associated CMIP PDU in a connectionless fashion over the connection-oriented association previously established by ACSE.

6.3.2 The Point-to-multipoint Model

Figure 6.5 RM-ODM Point-to-multipoint Binding Model

In order to support the need for a single managing system to manage a number of managed systems, we propose the point-to-multipoint model shown in Figure 6.5. Here a single managing-object interface can be statically bound to a number of agent-object managed interfaces by means of ACSE; again the interfaces of the respective managing object and managed objects are dynamically bound by means of CMISE. To permit point-to-multipoint binding, multiple copies of ACSE and CMISE are made available in the engineering viewpoint on the managing system.

6.3.3 The Relay Binding Model

We propose a third binding model to support mediated and delegated styles of management. In this model, a managing object in a managing system has indirect access to the managed objects of a managed system through a relay management system, see Figure 6.6.

The agent management-object of the relay system acts as an OSI application-layer relay and the management objects of the relay system provide the delegation and mediation functions. The managing-object interface in the managing system is bound statically to the agent-object managed interface in the relay system and the agent-object managing

interface is, in turn, bound statically to the agent-object managed interface in the managed system, in both cases using ACSE.

Figure 6.6 RM-ODM Relay Binding Model

The managing-object interface is bound dynamically to the management-object managed interfaces in the relay system and the managing interfaces of the management objects in the relay system are dynamically bound to the managed-object interfaces in the managed system, in both cases by means of CMISE. The behaviour clauses of the management objects in the relay system define their dynamic binding to the managed objects in the managed system.

6.4 The Invocation Model

The invocation model of OSI Management is defined by the Common Management Information Service Element (CMISE) [ISO91a] which supports both attribute-based operation invocations (set-value, get-value, etc.) and class-specific object-based operation invocations. CMISE also permits a single operation to be invoked on a number of managed objects within a single system and the specification of predicates to control the invocation of operations on individual objects. These facilities, called respectively scoping and filtering, are notionally mediated by the agent. Operation invocation within OSI Management is accomplished by supplying parameters to a single copy CMISE located in the engineering viewpoint.

For the purposes of open distributed management, we propose to adopt the OSI Management invocation model but interpret it within the context of RM-ODP. In general terms, then, operations are invoked on the managed-interfaces of management objects.

We recognise that, at the raw computational (CMISE) level, a programmer has to be, to some extent, aware of the distribution of the management objects and thus is not being provided with true location transparency. However, we believe that true location transparency can be provided by developing higher-level APIs based on CMISE and the Directory that include appropriate name-resolution features¹.

6.5 The Managed Relationship Model

The OSI General Relationship Model (GRM) [ISO95e] extends the OSI Management managed-object model to support the concept of relationships, termed managed relationships, between management objects. The following is a description of the GRM is adapted from Ashford [Ash94].

The GRM supports relationships of any degree (i.e. any number of roles) and roles of any cardinality (i.e. any number of participants in a given role). The GRM also supports a number of canonical operations for managing relationships. Managed-relationships are first-class concepts ("first-class" in the sense that the concepts are not reducible to simpler concepts) within OSI Management but no special constructs have been added to the managed-object model to represent and manage relationships. Managed relationships are represented by existing concepts in the model and managed by side effects of object operations. The approach was influenced by the early work of Rumbaugh in the area of associations in object-oriented systems [Rum87].

The GRM makes a clear distinction between the ontology of a relationship and the representation of a relationship within the OSI managed-object model. This distinction has two ramifications: firstly, a particular relationship type (e.g. back-up) might be represented

¹Migration transparency to support mobile objects will require a more sophisticated approach.

by different methods in different designs (e.g. pointers in one and names in another) and, secondly, the integrity of the representation of the relationship is automatically maintained (e.g. no dangling pointers). This distinction also reflects the distinction made by ODP between the informational viewpoint (the semantics of information) and the computational viewpoint (the realisation of the informational viewpoint).

The GRM provides a template language in a similar style to GDMO for the specification of managed relationships. Two templates are provided: the Relationship Class Template and the Relationship Mapping Template. The class template specifies the informational aspects of the relationship (cardinality, degree, relationship operations, etc.) and the mapping template specifies the representation of the informational aspects in the computational aspects of the managed-object model (e.g. pointers, management operations, etc.).

We propose to adopt the GRM for open distributed management but interpret it within the context of RM-ODM. In general terms, then, managed relationships can exist between management objects. The GRM model can be used without change except with respect to the maintenance of the integrity of the invariant and the integrity of the representation. In the single-managing/single-managed system distribution model of OSI

Figure 6.7 RM-ODM Relationship Model

Management, all managed objects are assumed to reside in a single system and thus the actual mechanism for integrity maintenance is not subject to standardisation. However in the distributed case this is not so: objects may reside in different systems and the maintenance of relationship integrity becomes a matter for standardisation.

We propose that the integrity of relationships that span systems be maintained by *relationship-maintenance management objects*, one in each system for each relationship instance, see Figure 6.7. A relationship-maintenance object conceptually monitors the behaviour of related objects in its system and invokes appropriate operations on peer relationship-management objects to maintain the integrity of the relationship.

6.6 The Transaction Model

The transaction model we propose is based on the application context developed by ISO for OSI Management [ISO95f] which is itself based on the OSI Transaction Processing Application Service Element (TP-ASE) standard [ISO92f]. The application context provides two types of service:

- (a) the grouping of a number of CMISE requests to be carried out atomically but without the need for rollback and recovery features; and
- (b) the grouping of a number of CMISE request to be carried out atomically and with the provision of rollback and recovery features (to maintain the so-called ACID properties [Ber81]).

The application context is modelled in the computational viewpoint by use of TP-ASE service primitives such as TP-BEGIN, TP-HANDSHAKE, and TP-COMMIT to bracket the CMISE primitives and to indicate the supporting semantics of the services.

We propose to adopt the model but to interpret it within the context of the proposed management and binding models. Thus a TP transaction manager is mapped to a managing object and a TP transaction subordinate to an agent management-object. The agent management-object may, in turn, play the role of TP transaction manager for subordinate transactions.

Transactions are accomplished by supplying parameters, within the rules of the application context, to copies of CMISE and TP-ASE in the engineering viewpoint.

Chapter 7

An Implementation Framework for Open Distributed Management, *or On helping the programmer*

The implementation framework directly supports the specification of the computational viewpoint by defining components that can be directly used in programming an open distributed management system. In this chapter we outline the components defined in OSI Management standards that can be used without change and additional components that are needed to support the models we propose for the RM-ODM.

7.1 OSI Management Components

A number of components defined in OSI Management standards support models that are directly relevant to the RM-ODM and the components can be used without change. They are documented using GDMO with semantics defined in natural language in the international standard entitled "Definition of Management Information" [ISO92d]. These includes the **log** object, the **logRecord** object (and subclasses **eventLogRecord**, **alarmRecord**, and **attributeChangeRecord** objects), and **discriminator** object classes.

7.2 RM-ODM Components

We have proposed a number of generic models based on those defined in OSI Management that include new, or extended, component definitions. These include the **agentManagementObject** and the **relationshipMaintenanceManagementObject**.

7.3 Commercial Frameworks

Obviously endorsements of particular framework products are not appropriate in an international standard but we believe that, at least for the purposes of this discussion, we would be remiss not list some of the more important products, namely:

- cmipWorks [Geig94] from IBM;
- Agent Toolbox [DSET95] from DSET;
- GDMO Editor [Mjo93] from Mjolner;
- Archon Toolset [Arc94] from Retix; and
- IM Tools [Eri95] from Erisoft.

Chapter 8

Summary and Conclusions *or, So far—so good*

Developing open distributed network-management systems for evolving telecommunications networks is both difficult and urgent. Difficult, because of the size and complexity of the problem and urgent, because of the continual pressure on common carriers to reduce costs and provide better service. In this chapter we summarise the important aspects of our research, draw some conclusions, and propose some further areas for investigation.

8.1 Summary

Perhaps the greatest difficulty in reasoning about modern network-management systems is because of their immense complexity. In order to address the issue of complexity we look, in Chapter 2, at tools for reasoning about the design of software systems in general. By taking Brook's standpoint that complexity is an essential and not an accidental aspect of software systems, we explore abstraction and encapsulation as mechanisms for taming complexity. We then explore the characteristics of modelling as a way of reasoning about software systems and their associated phenomena.

In Chapter 3 we postulate that abstraction and modelling are not, in themselves, enough to meet the goal of reasoning about complex software systems and that other, architectural-level, issues such as reference models, patterns, design granularity, and languages need to be addressed. We drawn these concepts together into an original taxonomy, or software reference architecture, which includes two key classifications: architectural design and architectural style.

Next, in Chapter 4, we present an original taxonomy of architectural issues—the software reference architecture—as a basis for discussing the context—characteristics, qualities, purpose—of architectures for distributed software systems. We go on to introduce the Reference Model for Open Distributed Processing (RM-ODP) with its five viewpoints as a basis for developing architectural reference models for classes of distributed systems.

Our discussion to this point has avoided issues specific to distributed management but, in Chapter 5, we begin the development of an architectural design framework for distributed network-management by first proposing a Reference Model for Open Distributed Management. Whilst the basis for much of the reference model is RM-ODP and our previous work, the interpretation of the concepts within the context of network management and the rationale for the selection of viewpoint languages is original to this research.

In Chapter 6 we continue the development of RM-ODM with the addition of various generic models. In order to meet the goal of an evolutionary path to distributed management, we extend the models defined in OSI Management to develop a set of generic models to support the design of open distributed management systems. We identify three generic models important to distributed management: the management object model; the relay binding model; and the distributed relationship model. These extensions are original to this research and significantly augment the functionality of OSI Management to support fully-distributed network management.

The approach we propose is evolutionary in nature and builds upon international standards that have been in place for a number of years. In particular we do not propose changes in base communications protocols nor incompatible changes to binding and invocation models. By this means we propose to preserve the investment in systems that have already been deployed.

In Chapter 7 we round out the architectural design framework with an outline of components that will be necessary to support RM-ODM.

In Annex A we provide a first-level of confidence in our architecture by using it to develop an original specification of a simple, distributed telecommunications-management application and, in an appendix, we show a simulation of the specification in Smalltalk.

8.2 Conclusions

The main conclusion we have drawn from our research is that there is no quick fix to developing open distributed management systems for common carriers. The complexity of the systems—a complexity born out of the very nature of network management and compounded by the inherently distributed nature of the application domain, pluralistic authority structures, and the size of networks—precludes a simple solution. Existing approaches, such as TMN, are unsatisfactory because each tends to be a high-level design that fails to address many of the realities of network management. This forces developers to adopt ad hoc, closed solutions to meet actual requirements. What is needed is a tractable approach that enables developers to design open, interoperable systems.

We conclude that the structure of designs for distributed network-management systems is the *sine qua non* of a tractable approach and that an architecture to guide and assist systems designers in the structuring of straightforward and logical designs is required. The need for an architecture is, in fact, not restricted to the design of distributed network-management systems but is common to the development of all distributed systems. Industry and academia have developed a number of independent architectural-level tools—reference models, patterns, specification languages—but these need to be brought together in a single framework to assist the software designer.

Designing an architecture is hard: there is debate on the scope and content of architectures; there is a tension between design and style; and there are charges of elitism. Our research has shown that, in the area of network management, RM-ODP is an important tool for reasoning about and developing an appropriate architecture. We were able to develop a satisfying interpretation of an existing architecture—OSI Management—within the context of RM-ODP. But, more to the point, we were able to extend that architecture, again within the context of RM-ODP, to one that supports fully-distributed network management. We believe that RM-ODP will soon be to systems architecture what the OSI Reference Model is to systems interconnection. Not that all architectures will necessarily be conformant to RM-ODP but that it will be possible to compare architectures against a single yard-stick and thus to reason about issues such as interworking, completeness, and formality.

In view of the huge capital investments in network infrastructure, it is not possible for common carriers to follow the rapid-obsolescence model of the computer industry and it is crucial that moves from point-to-point to fully-distributed systems be evolutionary. On the basis of the direction that we have mapped out for ISO Management, we conclude that similar evolutionary directions can also be mapped out for management based on CORBA and TINA-C DPE. Whilst the specification languages in both cases may be different, the concepts of the extended object and binding models are transferable.

8.3 Future Work

We have presented the broad sweep of an architecture for open distributed management, but our work indicates a number of items for further investigation. The items fall into three main categories:

- (1) improving the completeness and formality of RM-ODM viewpoint languages;
- (2) specifying additional generic models; and

- (3) better presenting the relationships between the elements of the software reference architecture.

We will deal with each of these categories in turn.

8.3.1 Improved Formality

Perhaps the most urgent improvement to RM-ODM is the development of a formal language for the enterprise viewpoint. We currently propose that the policies of the enterprise viewpoint be specified in stylised natural language but we believe that a more formal approach, in keeping with the formality of languages proposed in the other viewpoints, is required. Deontic logic [Lee88], rule-based languages such as Prolog, and scripting languages such as Telescript [Whi94] seem to offer fruitful areas for investigation.

Demonstrating consistency between viewpoint specifications by means of behavioural compatibility will probably prove quite difficult and an appeal to the formal methods community might be in order. Proving conformance between incrementally more concrete specifications by means of proof-obligations is standard in textbooks on formal methods [Word92, Jones86]. This form of refinement may be analogous to showing behavioural compatibility between viewpoint specifications. The obvious challenges to adapting the proof-obligations approach are the semantic gaps between, and the different languages used in, the viewpoints. We also believe that more formal notions of aggregation and composition of objects would enrich RM-ODM.

8.3.2 Additional Generic Models

The relay-binding model enables the specification of two additional models of management: delegation and mediation. In a delegation model of management, responsibility for carrying-out high-level functions is delegated to one or more

subordinate managing systems which interpret the high-level functions within the context of their span of control. Mediation involves a subordinate managing system providing, to its immediate managing system, a higher-level abstraction of objects in its span of control. These are important models of management and additional research is need to characterise the models and develop any additional notations that may be required.

8.3.3 Refinements to the Software Reference Architecture

Figure 8.1 Visualisation of a Software Reference Architecture

Whilst the taxonomic structure of the software reference architecture described in Chapter 3 has the advantage of simplicity, it suffers from undue compartmentalisation. Issues of style, abstraction, and granularity appear isolated from one another and yet in practice they are all inter-related. We propose research on how the relationships between the various elements might be better presented. As a starting point we propose a visualisation in two dimensions: one of abstraction and an orthogonal one of granularity, see Figure 8.1.

8.4 Final Observations

Despite the urgency, achieving the goal of interoperable components and portable applications for distributed management is likely to take some years. We believe that the standardisation of an architectural framework similar to that which we propose is an important first step and will galvanise the various stakeholders in the industry. We recognise that such an architecture, even if quickly standardised, cannot be implemented overnight and that strategies for interworking with legacy equipment and long-term co-existence with the evolving signalling network will be crucial.

Annex A

A Small Specification or, *Trying it out*

The purpose of this annex is to show how a simple, distributed telecommunications-management application might be specified using the proposed Open Distributed Management architecture. In the main body of the annex, the enterprise, the combined information/computational, and the engineering viewpoints are defined. In the enterprise (§A.2) viewpoint we used stylised natural language to define the entities of interest and the policies to which they are subject. In the combined information/computational viewpoint (§A.3), we use GDMO to specify the syntactical aspects of management objects, GRM to specify the syntactical aspects of the schemata, and Z to specify the behavioural aspects of both. In the engineering viewpoint (§A.4) we use Smalltalk to define the basic engineering objects and the Smalltalk runtime system to provide the channel support. In the appendix, we present a simulation of the application.

A.1 Overview of Telecommunications Application

Figure A.1—Application Scenario:

The scenario is illustrated in Figure A.1. A managing-system workstation permits an operator to manage (make, clear, and query) telecommunications circuits by means of a GUI. Circuits are abstractions that provided a telecommunications connection-service between two endpoints with a time-varying quality of service (QoS). Circuits are realised as a number of trails each of which provides a section of the connection service, also with a time-varying quality of service. For load-sharing reasons, the circuits are maintained by a network controller that performs the equipment-level operations on trails and the trail pool. Communications between the managing system and the controller is not secure, but communications between the controllers and the trails and trail pool is.

A.2 Enterprise Viewpoint

A.2.1 Managing System

The managing system provides the management interface to the network for the human operator.

A.2.1.1 Obligations

- (1) Maintain a graphic display of the topology of the network, assigned circuits, and serviceability of circuits.
- (2) Attempt to honour operator requests for the making, clearing, and querying of circuits.
- (3) Ensure feedback to the operator in respect of responses to all operator requests.
- (4) Provide appropriate security credentials when requesting service of controllers.

A.2.1.2 Prohibitions

- (1) May not communicate directly with trails or the trail pool.

A.2.2 Controller

Provides equipment-level control and carries out operations delegated by managing system.

A.2.2.1 Obligations

- (1) Attempt to honour authorised requests to make, clear, and query circuits.
- (2) Request and return trails to trail pool.

A.2.3 Trail Pool

Provides the inventory facility for trails.

A.2.3.1 Obligations

- (1) Maintain an inventory of trails.
- (2) Attempt to honour requests to assign and return trails.

A.3 Information and Computational Viewpoint

A.3.1 Overview

A schematic overview, using the OMT notation [Rum91], is shown in Figure A.2. The managing-system object in the enterprise viewpoint is represented by two management objects in the information viewpoint: the GUI and Manager. Circuits are represented by managed relationships and controller, trails, and the trail pool in the enterprise viewpoint are represented respectively by management objects in the information viewpoint. The GUI is not defined.

Figure A.2—Information Viewpoint Overview;

A 3.2 Z Mapping

The mapping of GDMO constructs to those of Z and the mapping of ASN.1 data-types to those of Z reflect on-going work of the ISO working-group ISO IEC/JTC1/SC21/WG7 [ISO 95g]. Briefly, Z schema are used to map GDMO classes and actions respectively and a syntactical convention is used to map data-types.

The following are global declarations of data-types:

```

QoS == R [0 ... 1.0]
Success ::= {true | false}
RequestedQoS == QoS
Password == seq Char
Char == {a, b, ... y, z}
IsServiceable ::= {true | false}

```

The following are global declarations of predicates:

```

averageQoS : QoS
maxTrails : N
allocation : N
-----
averageQoS = 0.5
maxTrails = 5000
allocation = 3

```

A.3.3 Manager Class Definition

osiManager MANAGEMENT OBJECT CLASS

DERIVED FROM "CCITT REC X.721 | ISO/IEC 10165-2":top
CHARACTERISED BY

connectionManagementPackage PACKAGE
BEHAVIOUR connectionManagementPackageBehaviour;

ACTIONS

makeConnectionWith;
clearConnection;
mgrIsServiceable;;;

REGISTERED AS {thesisApplication n };

osiManager class inherits from **top** and is characterised by a single package, **connectionManagementPackage**. Its basic behaviour is defined by **connectionManagementPackageBehaviour** and it supports three operations (actions), namely: **makeConnectionWith**, **clearConnection**, and **mgrIsServiceable**.

connectionManagementPackageBehaviour BEHAVIOUR

DEFINED AS "

OSIManager

activeController : OSIController
ctrlPassword : Password

ctrlPassword ≠ < >

CreateOSIManager

OSIManager'
ctrl? : OSIController
pwd? : Password

activeController' = ctrl?
ctrlPassword' = pwd?

REGISTERED AS {thesisApplication n };

The behaviour of the connection management package defines two variables, *activeController* and *ctrlPassword*, the identifier of the manager's controller and the managers authorisation password, respectively. The predicate specifies that the password must be non-null. The creation of an *OSIManager*, *CreateOSIManager*, entails the assignment of values to the two variables.

makeConnectionWith ACTION

BEHAVIOUR makeConnectionWithBehaviour;
MODE CONFIRMED;
WITH INFORMATION SYNTAX makeConnectionWithRequest;
WITH REPLY SYNTAX makeConnectionWithReply;

REGISTERED AS {thesisApplication n };

makeConnectionWithBehaviour BEHAVIOUR

DEFINED AS "

MakeConnectionWith $\equiv$ *MakeConnectionWithOkay* $\vee$ *MakeConnectionWithFail*

MakeConnectionWithOkay

$\exists$ *OSIManager*

q? : *QoS*

c! : *OSICircuit*

success! : *Success*

BuildCircuitOkay

BuildCircuitOkay [ctrlPassword/p?, q?/q?, c!/c!, success!/success!]

MakeConnectionWithFail

$\exists$ *OSIManager*

q? : *QoS*

c! : *OSICircuit*

success! : *Success*

BuildCircuitFail

BuildCircuitFail [ctrlPassword/p?, q?/q?, c!/c!, success!/success!]

*

REGISTERED AS (thesisApplication n);

MakeConnectionWith expects a single parameter, *q?* (the desired QoS), and returns the results, *c!* (a circuit) and *success!* (the outcome of the operation). The normal behaviour, *makeConnectionWithOkay*, entails the "call-out" of *BuildCircuitOkay* with input parameters of *ctrlPassword* and *q?* and return parameters of *c!* (the circuit assigned) and *success!* set to *true*. Exception behaviour, *MakeConnectionWithFail*, entails the "call-out" of *BuildCircuitFail* with *success!* set to *false*. The complete behaviour of *MakeConnectionWith* is the disjunction of *MakeConnectionWithOkay* and *MakeConnectionWithFail*.

clearConnection **ACTION**

BEHAVIOUR clearConnectionBehaviour;

MODE **CONFIRMED;**

WITH INFORMATION SYNTAX clearConnectionRequest;

WITH REPLY SYNTAX clearConnectionReply;

REGISTERED AS (thesisApplication n);

clearConnectionBehaviour **BEHAVIOUR**

DEFINED AS *

ClearConnection $\equiv$ *ClearConnectionOkay* $\vee$ *ClearConnectionFail*

ClearConnectionOkay

$\exists$ *OSIManager*

c? : *OSICircuit*

success! : *Success*

TearDownCircuitOkay

TearDownCircuitOkay [ctrlPassword / p?, c? / c?, success! / success!]

ClearConnectionFail

<i>c?</i> : <i>OSICircuit</i> <i>success!</i> : <i>Success</i> <i>TearDownCircuitFail</i> <i>TearDownCircuitFail</i> [<i>ctrlPassword</i> / <i>p?</i> , <i>c?</i> / <i>c?</i> , <i>success!</i> / <i>success!</i>]

•

REGISTERED AS {thesisApplication n};

ClearConnection expects a single parameter, *c?*(the circuit to clear down), and returns the result *success!* (the outcome of the operation). The normal behaviour, *ClearConnectionOkay*, entails the “call-out” of *TearDownCircuitOkay* with input parameters of *ctrlPassword* and *c?* and a return parameter of *success!* set to *true*. Exception behaviour, *ClearConnectionFail*, entails the “call-out” of *TearDownCircuitFail* with *success!* set to *false*. The complete behaviour of *ClearConnection* is the disjunction of *ClearConnectionOkay* and *ClearConnectionFail*.

mgrIsServiceable **ACTION**

BEHAVIOUR *mgrIsServiceableBehaviour*;
MODE **CONFIRMED**;
WITH INFORMATION SYNTAX *mgrIsServiceableRequest*;
WITH REPLY SYNTAX *mgrIsServiceableReply*;

REGISTERED AS {thesisApplication n};**mgrIsServiceableBehaviour** **BEHAVIOUR****DEFINED AS “***MgrIsServiceable* $\equiv$ *MgrIsServiceableOkay* $\vee$ *MgrIsServiceableFail***MgrIsServiceableOkay**

$\exists$ <i>OSIManager</i> <i>c?</i> : <i>OSICircuit</i> <i>success!</i> : <i>Success</i> <i>svc!</i> : <i>IsServiceable</i> <i>IsServiceableOkay</i> <i>IsServiceableOkay</i> [<i>ctrlPassword</i> / <i>p?</i> , <i>c?</i> / <i>c?</i> , <i>success!</i> / <i>success!</i> , <i>svc!</i> / <i>s?</i>]
--

MgrIsServiceableFail

<i>c?</i> : <i>OSICircuit</i> <i>success!</i> : <i>Success</i> <i>IsServiceableFail</i> <i>IsServiceableFail</i> [<i>ctrlPassword</i> / <i>p?</i> , <i>c?</i> / <i>c?</i> , <i>success!</i> / <i>success!</i> , <i>svc!</i> / <i>s?</i>]

•

REGISTERED AS {thesisApplication n};

MgrIsServiceable expects a single parameter, *c?*(the circuit to query), and returns the result, *success!* (the outcome of the operation) and *svc?* (the serviceability of the circuit). The normal behaviour, *MgrIsServiceableOkay*, entails the “call-out” of *IsServiceableOkay* with input parameters of *ctrlPassword* and *c?* and return parameter of *success!* set to *true* and *svc!* set to *true*. Exception behaviour, *MgrIsServiceableFail*, entails the “call-out” of *IsServiceableFail* with *success!* set to *false*. The complete behaviour of *MgrIsServiceable* is the disjunction of *MgrIsServiceableOkay* and *MgrIsServiceableFail*.

A.3.4 Controller Class Definition

osiController MANAGEMENT OBJECT CLASS

DERIVED FROM "CCITT REC X.721 | ISO/IEC 10165-2":top
CHARACTERISED BY

controllerPackage PACKAGE
BEHAVIOUR controllerPackageBehaviour;

ACTIONS

buildCircuitWith;
tearDownCircuit;
isServiceable;;;

REGISTERED AS (thesisApplication n);

osiController class inherits from **top** and is characterised by a single package, **controllerPackage**. Its basic behaviour is defined by **controllerPackageBehaviour** and it supports three operations (actions), namely: **buildCircuitWith**, **tearDownCircuit**, and **isServiceable**.

controllerPackageBehaviour BEHAVIOUR

DEFINED AS "

OSIController

allocatedCircuits : OSICircuit → RequestedQoS

currentPassword : Password

currentPassword ≠ < >

CreateOSIController

OSIManager'

ctrl? : OSIController

pwd? : Password

allocatedCircuits' ≠ ∅

currentPassword' = pwd?

"

REGISTERED AS (thesisApplication n);

The behaviour of the controller management package defines two variables, *allocatedCircuits* and *currentPassword*, a partial function relating circuit and requested QoS, and the password of the controller, respectively. The predicate specifies that the password must be non-null. The correct provisioning of an **OSIController** entails the assignment of a values to *currentPassword* and the partial function.

buildCircuitWith ACTION

BEHAVIOUR buildCircuitWithBehaviour;

MODE CONFIRMED;

WITH INFORMATION SYNTAX buildCircuitWithRequest;

WITH REPLY SYNTAX buildCircuitWithReply;

REGISTERED AS (thesisApplication n);

buildCircuitWithBehaviour BEHAVIOUR

DEFINED AS "

BuildCircuitWith $\equiv$ *BuildCircuitOkay* $\wedge$ *BuildCircuitFail*

BuildCircuitOkay

Δ *OSIController*

$p? : \text{Password}$

$q? : \text{QoS}$

$c! : \text{OSICircuit}$

AllocateTrailsOkay

CreateOSICircuitOkay

$\text{success!} : \text{Success}$

$t : \mathbb{P} \text{ OSITrails}$

$p? = \text{currentPassword}$

AllocateTrailsOkay $[t / t!]$

CreateOSICircuitOkay $[t!t?, c!/c!]$

$\text{success!} = \text{true}$

$\text{allocatedCircuits}' = \text{allocatedCircuits} \cup \{c! \mapsto q?\}$

$\text{currentPassword}' = \text{currentPassword}$

BuildCircuitFail

Ξ *OSIController*

$p? : \text{Password}$

$q? : \text{QoS}$

$c! : \text{OSICircuit}$

$t : \mathbb{P} \text{ OSITrails}$

$\text{success!} : \text{Success}$

AllocateTrailsFail

CreateOSICircuitsFail

$p? \neq \text{currentPassword}$

AllocateTrailsFail $[t / t!] \vee$ *CreateOSICircuitsFail* $[t? / t]$

$\text{success!} = \text{false}$

BuildCircuitWith expects two parameters, $q?$ (the requested QoS) and $p?$ an authorisation token, and returns two parameters, success! (the success of the operation) and $c!$ the assigned, initialised circuit. The normal behaviour, *BuildCircuitOkay*, entails the “call-out” of *AllocateTrailsOkay* with input parameter of $p?$ and return parameters of t , the trails assigned, and success! set to *true* followed by a call-out to *CreateOSICircuitOkay* to create the circuit from the trails, t . Exception behaviour, *BuildCircuitFail*, entails “call-outs” of *AllocateTrailsFail* and *CreateOSICircuitFail*. The complete behaviour of *BuildCircuitWith* is the disjunction of *BuildCircuitOkay* and *BuildCircuitFail*.

•

REGISTERED AS {thesisApplication n};

tearDownCircuit ACTION

BEHAVIOUR tearDownCircuitBehaviour;

MODE CONFIRMED;

WITH INFORMATION SYNTAX tearDownCircuitRequest;

WITH REPLY SYNTAX tearDownCircuitReply;

REGISTERED AS {thesisApplication n};

tearDownCircuitBehaviour **BEHAVIOUR**

DEFINED AS "

TearDownCircuit = *TearDownCircuitOkay* $\vee$ *TearDownCircuitFail*

TearDownCircuitOkay

Δ *OSIController*

p? : Password

c? : *OSICircuit*

ReturnTrails

success! : *Success*

p? = *currentPassword*

c? $\in$ **dom** *allocatedCircuits*

allocatedCircuits' = *allocatedCircuits* $\ominus$ {*c?*}

ReturnTrails [*c?.CircuitTrails/t?*]

success! = *true*

TearDownCircuitFail

Ξ *OSIController*

p? : Password

c? : *OSICircuit*

success! : *Success*

p? $\neq$ *currentPassword* $\vee$ *c?* $\notin$ **dom** *allocatedCircuits*

success! = *false*

"

REGISTERED AS {thesisApplication n};

TearDownCircuit expects two parameters, *p?* (the authorisation token) and *c?* the circuit to clear down. The normal behaviour requires that the password be correct and that the circuit is one in the *allocatedCircuits* table; the entry is removed and the trails are returned to the trail pool.

isServiceable **ACTION**

BEHAVIOUR *isServiceableBehaviour*;

MODE **CONFIRMED**;

WITH INFORMATION SYNTAX *isServiceableRequest*;

WITH REPLY SYNTAX *isServiceableReply*;

REGISTERED AS {thesisApplication n};

isServiceableBehaviour **BEHAVIOUR**

DEFINED AS "

$IsServiceable \equiv IsServiceableOkay \wedge IsServiceableFail$

IsServiceableOkay

$\exists OSIController$
 $q : QoS$
 $p? : Password$
 $c? : OSICircuit$
 $s! : IsServiceable$
 $success! : Success$

$p? = currentPassword$
 $c? \in \mathbf{dom} \text{ allocatedCircuits}$
 $CircuitQoS \ [q \neq q!]$
 $s! = q \geq allocatedCircuits \ c?$
 $success! = true$

IsServiceableFail

$\exists OSIController$
 $p? : Password$
 $c? : OSICircuit$
 $success! : Success$

$p? \neq currentPassword$
 $c? \notin \mathbf{dom} \text{ allocatedCircuits}$
 $success! = false$

REGISTERED AS (thesisApplication n);

IsServiceable expects two parameters, $p?$ (the authorisation token) and $c?$ the circuit to query. The normal behaviour requires that the password be correct, that the circuit is one in the *allocatedCircuits* table, that the actual QoS of the circuit is greater than the requested QoS stored in the *allocatedCircuits* table and that the output parameter, $s!$, be set to *true*.

A.3.5 Circuit Definition

The generic aspects of the circuit relationship (roles, cardinality, operations, etc.) are specified in §A.3.3.5.1; a realisation of the relationship in terms of a relationship object is specified in §A.3.3.5.2; and the relationship object is defined in §A.3.3.5.3.

A.3.5.1 Relationship Class Definition

circuitRelationship RELATIONSHIP CLASS

SUPPORTS

ESTABLISH establish,
TERMINATE terminate,
USER DEFINED circuitQoS;

ROLE trailRole

COMPATIBLE WITH osiTrail
PERMITTED-ROLE-CARDINALITY OneToTen
REQUIRED-ROLE-CARDINALITY OneToTen;

REGISTERED AS {thesisApplication n };

circuitRelationship supports three operations: **establish** (to establish the relationship), **terminate** (to terminate the relationship), and **circuitQoS** (a user-defined operation to return the current QoS of the circuit). One role, **trailRole**, is defined that takes its behaviour definition from the management object class, **osiTrail**; an implementation is required to support from one to ten objects in the role.

A.3.5.2 Relationship Mapping Definition

circuitMapping RELATIONSHIP MAPPING

RELATIONSHIP CLASS circuitRelationship;
RELATIONSHIP OBJECT osiCircuit;

ROLE trailRole

RELATED CLASSES osiTrail
REPRESENTED BY

RELATIONSHIP-OBJECT-USING-POINTER trails;

OPERATION MAPPING

ESTABLISH establish
MAPS-TO-OPERATION CREATE OF RELATIONSHIP OBJECT
TERMINATE terminate
MAPS-TO-OPERATION DELETE OF RELATIONSHIP OBJECT
USER DEFINED circuitQoS
MAPS-TO-OPERATION ACTION circuitQoS OF RELATIONSHIP OBJECT;

REGISTERED AS {thesisApplication n };

The relationship mapping, **circuitMapping**, specifies that the relationship, **circuitRelationship** be implemented by means of a *relationship object* that acts as a surrogate for the relationship. Objects bound in the role are defined by a multi-valued pointer, **trails**; generic operations **establish** and **terminate** are mapped, respectively, to the create and delete of the relationship object and the user-defined operation, **circuitQoS** is mapped to an operation of the same name on the relationship object.

A.3.5.3 Circuit Class Definition

osiCircuit MANAGEMENT OBJECT CLASS**DERIVED FROM "CCITT REC X.725 | ISO/IEC 10165-7":genericRelationshipObject;****CHARACTERISED BY** **osiCircuitPackage PACKAGE****BEHAVIOUR** **osiCircuitPackageBehaviour;****ACTIONS****circuitQoS****ATTRIBUTES****trails GET::;****REGISTERED AS {thesisApplication n };**

osiCircuit class inherits from **genericRelationshipObject** and is characterised by a single package, **osiCircuitPackage**. Its basic behaviour is defined by **osiCircuitPackageBehaviour** and it supports a single operation: **circuitQoS**, and a single, readable, attribute: **trails**.

osiCircuitPackageBehaviour BEHAVIOUR**DEFINED AS "***OSICircuit**circuitTrails : P OSITrails**# circuitTrails > 0 ∧ # circuitTrails ≤ 10**CreateOSICircuit* $\sqsubseteq$ *CreateOSICircuit* $\vee$ *CreateOSICircuitFail**CreateOSICircuitOkay**OSICircuit'**t? : P Trails**c! : OSICircuit**success! : Success**# t? > 0 ∧ # t? ≤ 10**circuitTrails' = t?**c! = OSICircuit'**success! = true**CreateOSICircuitFail**OSICircuit'**t? : P Trails**success! : Success**# t? ≤ 0 ∨ t? > 10**success! = false***"****REGISTERED AS {thesisApplication n };**

The normal operation of *CreateOSICircuit*, *CreateOSICircuitOkay*, ensures that the cardinality constraints of the relationship class are maintained.

circuitQoS ACTION

BEHAVIOUR circuitQoSBehaviour;
MODE CONFIRMED;
WITH REPLY SYNTAX circuitQoSReply;

REGISTERED AS {thesisApplication n };

circuitQoSBehaviour BEHAVIOUR

DEFINED AS "

CircuitQoS

$\exists$ *OSICircuit*

$q! : QoS$

$q! = (\sum \text{circuitTrails.currentQoS}) / \# \text{circuitTrails}$

"

REGISTERED AS {thesisApplication n };

trails ATTRIBUTE

DERIVED FROM "CCITT REC X.725 | ISO/IEC 10165-7":participantPointer;

REGISTERED AS {thesisApplication n };

A.3.6 Trail Pool Definition

osiTrailPool MANAGEMENT OBJECT CLASS

DERIVED FROM "CCITT REC X.721 | ISO/IEC 10165-2":top
CHARACTERISED BY

osiTrailPoolPackage PACKAGE
BEHAVIOUR osiTrailPoolBehavior;

ACTIONS

allocateTrails;
 returnTrails;;;

REGISTERED AS {thesisApplication n };

osiTrailPoolBehaviour BEHAVIOUR

DEFINED AS "

OSITrailPool

$\text{provisionedTrails} : \mathbb{P} \text{ OSITrails}$

OSITrailPoolInit

OSITrailPool'

$\# \text{provisionedTrails} = \text{maxTrails}$

"

REGISTERED AS {thesisApplication n };

osiTrailPool class inherits from **top** and is characterised by a single package, **osiTrailPoolPackage**. Its basic behaviour is defined by **osiTrailPoolBehaviour** and it supports two operation: **allocateTrails** (to allocate trails) and **returnTrails** (to return them to the pool).

allocateTrails **ACTION**

BEHAVIOUR **allocateTrailsBehaviour**
MODE **CONFIRMED**
WITH INFORMATION SYNTAX **allocateTrailsRequest**
WITH REPLY SYNTAX **allocateTrailsReply**

REGISTERED AS {thesisApplication n };

allocateTrailsBehaviour **BEHAVIOUR**

DEFINED AS "

AllocateTrails $\equiv$ *AllocateTrailsOkay* $\vee$ *AllocateTrailsFail*

AllocateTrailsOkay

Δ *OSITrailPool*
 $q? : QoS$
 $t! : P\ OSITrails$
 $success! : Success$

$\# provisionedTrails \geq allocation$
 $t! \subseteq provisionedTrails$
 $\# t! = allocation$
 $provisionedTrails' = provisionedTrails \setminus t!$
 $success! = true$

AllocateTrailsFail

Ξ *OSITrailPool*
 $success! : Success$

$\# provisionedTrails < allocation$
 $success! = false$

REGISTERED AS {thesisApplication n };

returnTrails **ACTION**

BEHAVIOUR **returnTrailsBehaviour;**
MODE **CONFIRMED;**
WITH INFORMATION SYNTAX **returnTrailsRequest;**
WITH REPLY SYNTAX **returnTrailsReply;**

REGISTERED AS {thesisApplication n };

returnTrailsBehaviour **BEHAVIOUR**

DEFINED AS "

ReturnTrails

Δ *OSITrailPool*
 $t? : P\ OSITrails$
 $success! : Success$

$provisionedTrails' = provisionedTrails \oplus t?$
 $success! = true$

"
REGISTERED AS {thesisApplication n };

A.3.7 Trail Definition

osiTrail MANAGEMENT OBJECT CLASS

DERIVED FROM "CCITT REC X.721 | ISO/IEC 10165-2":top
CHARACTERISED BY

osiTrailPackage PACKAGE
BEHAVIOUR osiTrailBehaviour

ATTRIBUTES
currentQoS GET;;

REGISTERED AS {thesisApplication n };

osiTrailBehaviour BEHAVIOUR

DEFINED AS "

OSITrail

currentQoS : QoS

currentQoS will vary with time

"
REGISTERED AS {thesisApplication n }

osiTrail class inherits from **top** and is characterised by a single package, **osiTrailPackage**. Its basic behaviour is defined by **osiTrailBehaviour** and it supports a single, readable, attribute, **currentQoS**, the value of which will vary with time.

A.3.8 ASN.1 Definitions

allocateTrailsReply ::= SEQUENCE{
success Success
trails SET OF Trails OPTIONAL}

allocateTrailsRequest ::= QoS

buildCircuitWithReply ::= SEQUENCE{
success Success,
circuitID CircuitID OPTIONAL}

buildCircuitWithRequest ::= SEQUENCE{
withQoS QoS,
authorisation Password }

CircuitID ::= DistinguishedName

clearConnectionReply ::= Success

clearConnectionRequest ::= CircuitID

```

circuitQoSReply ::=          SEQUENCE{
    success      Success,
    qos         QoS}

isServiceableReply ::=      SEQUENCE{
    success      Success,
    serviceable  BOOLEAN OPTIONAL}

isServiceableRequest ::=    SEQUENCE{
    circuitID    CircuitID
    authorisation Password}

makeConnectionWithReply ::=  SEQUENCE{
    success      Success,
    circuitID    CircuitID OPTIONAL}

makeConnectionWithRequest ::= QoS

mgrIsServicableReply ::=    SEQUENCE{
    success      Success,
    serviceable  BOOLEAN OPTIONAL}

mgrIsServicableRequest ::=  CircuitID

Password ::=  PrintableString

QoS ::=  REAL

returnTrailsReply ::=  Success

returnTrailsRequest ::=  SET OF Trails

Success ::=  BOOLEAN

tearDownCircuitReply ::=  Success

tearDownCircuitRequest ::=  SEQUENCE{
    circuitID    CircuitID
    authorisation Password}

```

A.4 Engineering Viewpoint

For this example we have chosen SmalltalkV as the specification language for the basic engineering objects. In the following sections we develop a programming-language binding; detail the correspondence between the information/computational viewpoints and the engineering viewpoint; and document the SmalltalkV objects.

A.4.1 SmalltalkV GDMO Language Binding

In §5.3 we required that a programming-language binding be specified for the language selected to specify the basic engineering objects—such a binding is required specify three mappings: data types; signatures; and behaviour.

The SmalltalkV language binding used in this example is based upon the Smalltalk OMG IDL binding [OMG92, Eas93]. Briefly: multiple-parameter replies are returned as Smalltalk arrays; error replies are handled using the standard SmalltalkV exception mechanism, readable attributes are accessed by appropriate methods, and ASN.1 data types are mapped to the equivalent Smalltalk classes. The mapping of Z declarative constructs onto the programming linguistics of SmalltalkV is not formally specified and we rely on the craft of programming to provide an informal mapping.

A.4.2 Information/Computational Object to Basic Engineering Object Mapping

In §5.3 we proposed that there be a one-to-one correspondence between the objects in the information/computational viewpoint and basic engineering objects. In table A.1 we document this correspondence.

Informational/Computational Viewpoint		Engineering Viewpoint	
GDMO Label	Section	Smalltalk Name	Section
osiManager	A.3.3	OSIManager	A.4.3.6
osiController	A.3.4	OSIController	A.4.3.5
osiCircuit	A.3.5.3	OSICircuit	A.4.3.3
osiTrailPool	A.3.6	TrailPool	A.4.3.7
osiTrail	A.3.7	OSTrail	A.4.3.4

Table A.1 Correspondence between the Information/Computational and Engineering Viewpoints

A.4.3 Smalltalk Class Definitions

The following is a listing of the Smalltalk classes developed from the information/computational viewpoint specification.

A.4.3.1 OSI ManagedObject

```
Object
  subclass: #OSIManagedObject
  instanceVariableNames: 'nameBinding distinguishedName'
  classVariableNames: ''
  poolDictionaries: ''
```

A.4.3.2 OSIRelationshipObject

```
OSIManagedObject
  subclass: #OSIRelationshipObject
  instanceVariableNames: 'relationshipClass'
  classVariableNames: ''
  poolDictionaries: ''
```

A.4.3.3 OSICircuit

```
OSIRelationshipObject
  subclass: #OSICircuit
  instanceVariableNames: 'trails'
  classVariableNames: ''
  poolDictionaries: ''
```

Class methods

```
establish:aSetOfTrails
  "answer a new circuit consisting of the trails in aSetOfTrails"

  (aSetOfTrails size >= 1 and:aSetOfTrails size<= 10) ifTrue.
    [^ self new initialise:aSetOfTrails] ifFalse:
      [Error signal. ^self]
```

Instance methods

```
circuitQoS
  "answer average QoS of trails making up the receiver"

  | averageQoS |
  averageQoS:= 0.
  trails do: [:each | averageQoS := averageQoS + each currentQoS].
  ^averageQoS := averageQoS/trails size
```

```
initialise:aSetOfTrails
    "Private"
    "initialises receiver circuit with the set of trails in
aSetofTrails"

    trails:=aSetOfTrails.
    ^self

terminate
    "Answer the set of trails that made up the circuit"

    ^trails
```

A.4.3.4 OSITrail

```
OSIManagedObject
    subclass: #OSITrail
    instanceVariableNames: ''
    classVariableNames: ''
    poolDictionaries: ''

Instance methods

currentQoS
    "answer QoS of trail"

    ^(SmallInteger random:11)asFloat/10
```

A.4.3.5 OSIController

```
OSIManagedObject
    subclass: #OSIController
    instanceVariableNames: 'allocatedCircuits currentPassword'
    classVariableNames: 'NetworkTrailPool'
    poolDictionaries: ''

Class methods

classInitialise:aTrailPool
    "answer the receiver with the class variable, NetworkTrailPool
initialised"

    NetworkTrailPool:=aTrailPool.
    ^self

createWithPassword:aPassword
    "answer an instance of the receiver initialised with aPassword"

    ^self new privateInitialise:aPassword
```

Instance methods

```
buildCircuitWith:qosValue authorisation:password
    "answer a circuit from startPoint to Endpoint"

    | trails newCircuit |

    self ifAuthorised:password.          "check password"

    trails:= NetworkTrailPool allocateTrailsWith:qosValue.
    newCircuit:=OSICircuit establish:trails.

    allocatedCircuits at:newCircuit put:qosValue.    "save Id and
requested QoS"
    ^newCircuit

ifAuthorised:password
    "answer true if password check passes else raises error"

    password=currentPassword ifFalse:
        [Error signal:'authorisation failure'.
         ^self].
    ^True          "password okay"

isServicable:aCircuit authorisation:password
    "answer aBoolean indicating servicability
    ie actualQoS of aCircuit is >= requestedQoS"

    self ifAuthorised:password.

    ^aCircuit circuitQoS >=
        (allocatedCircuits at:aCircuit ifAbsent:
         [Error signal:'bad CircuitID'.
          ^self])

privateInitialise:aPassword
    "answer the receiver initialised"

    allocatedCircuits:= Dictionary new.
    currentPassword:=aPassword.
    ^self

tearDownCircuit:aCircuit authorisation:password
    "answer the receiver; trails returned to TrailPool"

    self ifAuthorised:password.

    allocatedCircuits removeKey:aCircuit ifAbsent:
        [Error signal:'Bad CircuitID'.
         ^self].

    NetworkTrailPool returnTrails:(aCircuit terminate).
    ^self
```


A.4.3.6 OSIManager

OSIManagedObject

subclass: #OSIManager

instanceVariableNames: 'activeController ctrlPassword'

classVariableNames: ''

poolDictionaries: ''

Class methods

provision:controllerID withPassword:pwdValue

"create an instance of the of the receiver with
controllerLoad and password initialised"

^super new initialise:controllerID withPassword:pwdValue

Instance methods

clearConnection:aCircuitID

"answer the receiver with circuit:aCircuit cleared
or error raised"

activeController tearDownCircuit:aCircuitID
authorisation:ctrlPassword.

^self

initialise:controllerID withPassword:pwdValue

"Private"

"answer receiver initialised"

activeController:=controllerID.

ctrlPassword := pwdValue.

^self

isServicable:aCircuitID

"Boolean indicating if circuit is servicable
ie. meets requested QoS"

^activeController isServicable:
aCircuitID authorisation:ctrlPassword

makeConnectionWith:qosValue

"answer CircuitID of connection built with nominal
QoS of qosValue"

^activeController buildCircuitWith:qosValue
authorisation:ctrlPassword

A.4.3.7 TrailPool

OSIManagedObject

subclass: #TrailPool

```

instanceVariableNames: ''
classVariableNames: 'ProvisionedTrails'
poolDictionaries: ''
Class methods

```

```

Provision:numberOfTrails
    "answer the receiver provisioned with the number of trails"

    ProvisionedTrails:=numberOfTrails.
    ^self

```

Instance methods

```

allocateTrailsWith:qosValue
    "answer a set of trails with nominal QoS equal
    to qosValue"

    | allocation trails count |

    allocation:=3.
    count:=0.
    (ProvisionedTrails < allocation) ifTrue:
        [Error signal. ^self].          "not enough trails"
    trails:= Set new.
    [count<allocation] whileTrue:[ trails add: (OSTrail new).
        count:=count+1].
    ProvisionedTrails:=ProvisionedTrails-allocation.
    ^trails

```

```

returnTrails:trails
    "answer receiver with trails returned to warehouse"

    ProvisionedTrails := ProvisionedTrails + trails size.
    ^self

```

```

"answer a set of trails linking startPoint to endPoint"

    | allocation trails count |
    allocation:=3.
    count:=0.
    (ProvisionedTrails < allocation) ifTrue:
        [Error signal. ^self].          "not enough trails"
    trails:= Set new.
    [count<allocation] whileTrue:[ trails add: (OSTrail new).
        count:=count+1].
    ProvisionedTrails:=ProvisionedTrails-allocation.
    ^trails

```

```

returnTrails:trails
    "returns trails to warehouse"

    ProvisionedTrails := ProvisionedTrails + trails size.
    ^self

```

PM-1 3½"x4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT

PRECISIONSM RESOLUTION TARGETS

A.5 Conclusions

The specification of the telecommunications-management application using the proposed viewpoint languages was successful in the result but tedious and error-prone in the execution. The overall configuration and operation of the system was difficult to grasp from initial text-only specifications but the subsequent addition of informal diagrams greatly improved accessibility. Syntactical and type checking by hand proved to be error-prone partly because of the inherent nature of the three languages—GDMO/GRM, ASN.1, and Z—and partly because of the imperfect coupling of the languages into a single viewpoint language. Obviously much of this tedium could have been alleviated by the use of type-checking and formatting tools currently available for each of the languages, but a tool (or tool-chain) that, in addition, recognises the coupling of the languages would have been more useful.

Owing to the object-orientation of GDMO/GRM, specifications written in GDMO/GRM mapped naturally into Smalltalk classes and methods; the mapping of Z specifications was less natural due to its procedural orientation. In particular we had difficulty in specifying inherent object-oriented aspects of the application in Z, aspects such as name scoping, parameterised schema inclusion, and inheritance. Many of these problems have been reported in the literature [Jud93, Wil93] and on-going work into object-oriented dialects of Z, such as ZEST [Cus92], is producing encouraging results [ISO95g].

In summary, we believe that the proposed architecture has stood up well to a limited trial but that a more extensive one is now required. However, a pre-requisite to such a trial is the development of appropriate software tools for syntax and type checking.

Appendix to Annex A

SmalltalkV Simulation

The simulation of the distributed application is constructed on a single Smalltalk image. In the interests of simplicity, Smalltalk naming and binding were used rather than those proposed in §6.

AA.1 Simulation Script

The following is a script to exercise the various aspects of the application simulation.

```
"Test of Distributed Telecommunications Application"
```

```
|myController myCircuit circuits count myManager|
```

```
"Unit test of Controller"
```

```
Transcript clear.
```

```
Transcript show: 'Test of Application '; print: (Date today); cr.
```

```
Transcript show: 'Unit Test of Controller'; cr.
```

```
TrailPool Provision: 5000. "Provision TrailPool with 5000 trails"
```

```
OSIController classInitialise: (TrailPool new). "Set TrailPool reference"
```

```
myController:=
```

```
    OSIController createWithPassword 'Secret'. "Provision controller"
```

```
myCircuit:=
```

```
    myController buildCircuitWith: 0.5 authorisation: 'Secret'. "build circuit"
```

```
Transcript cr.
```

```
Transcript show: 'Circuit '; print: myCircuit id; show: ' is '.
```

```
(myController isServicable: myCircuit authorisation: 'Secret')
```

```
    ifFalse: [Transcript show: 'not ']. "check servicable"
```

```
Transcript show: 'servicable'.
```

```
myController tearDownCircuit: myCircuit authorisation: 'Secret'. "clear circuit"
```

```
[myController tearDownCircuit:myCircuit authorisation:'Secret'] "clear
same circuit"
```

```
on:Error
```

```
description].
```

```
[myController buildCircuitWith:0.5 authorisation:'NotSecret'.] "bad
password"
```

```
on:Error
```

```
description].
```

```
"Test of Manager"
```

```
Transcript cr;cr; show:'Test of Manager';cr.
```

```
myManager:= OSIManager provision:myController withPassword:'Secret'.
```

```
myCircuit:= myManager makeConnectionWith:0.5.
```

```
Transcript cr.
```

```
Transcript show:'Circuit ';print:myCircuit id;show:' is '.
```

```
(myManager isServicable:myCircuit)
```

```
ifFalse:[Transcript show:'not ']. "check servicable"
```

```
Transcript show:'servicable'.
```

```
myManager clearConnection:myCircuit. "clear circuit"
```

```
[myManager clearConnection:myCircuit] "clear same circuit"
```

```
on:Error
```

```
description].
```

```
"create 10 circuits"
```

```
circuits:= Set new.
```

```
count:=0.
```

```
[count < 10] whileTrue:[circuits add:
```

```
(myManager makeConnectionWith:0.5).
```

```
count:=count+1].
```

```
"check servicable"
```

```
circuits do:[eachCircuit|
```

```
Transcript cr.
```

```
Transcript show:'Circuit ';print:eachCircuit id;show:' is '.
```

```
(myManager isServicable:eachCircuit)
```

```
ifFalse:[Transcript show:'not '].
```

```
Transcript show:'servicable'.
```

```
Transcript cr;cr; show:'Testing finished'
```

AA.2 Transcript of Simulation

The following is a transcript of the output of the application simulation exercised by the script detailed in §AA.1.

Test of Application 8/4/96
Unit Test of Controller

Circuit 12937 is servicable
Bad CircuitID
authorisation failure

Test of Manager

Circuit 32300 is not servicable
Bad CircuitID
Circuit 44489 is not servicable
Circuit 44999 is not servicable
Circuit 43367 is not servicable
Circuit 43758 is servicable
Circuit 44149 is not servicable
Circuit 44659 is not servicable
Circuit 43537 is servicable
Circuit 44319 is servicable
Circuit 44829 is servicable
Circuit 43979 is not servicable

Testing finished

References

- [Abb87] Abbot, R. J. Knowledge Abstraction. *Comm ACM*, (30)8, August 1987.
- [Aid94] Aidarous, S. and Plevyak, T., (Eds). *Telecommunications Network Management into the 21 Century*. IEEE Press, 1994.
- [Alex77] Alexander, C., Ishikawa, S., Silverstein, M. *A Pattern Language*. OUP, 1977.
- [Alex79] Alexander, C. *A Timeless Way of Building*. OUP, 1979.
- [All86] Allison, L. *A Practical Introduction to Denotational Semantics*. CUP, 1986.
- [ANSA93] ANSA Architecture for Open Distributed Systems. *Architecture and Design Frameworks*. Architecture Projects Management, Document TR.38.00, 1993.
- [App92] Apple Computer Inc. *Inside Macintosh*. Addison-Wesley, Reading, Mass, 1992.
- [Arc94] *Archon Toolset*. Retix, 2401 Colorado Ave, Santa Monica, CA 90404.
- [Ash93a] Ashford, C. The OSI Managed-object Model. In *Proceedings of European Conference on Object-Oriented Programming* (Kaiserslautern, Germany, 1993). LNCS 707, Springer-Verlag, Berlin, 1993.
- [Ash93b] Ashford, C. (Ed.) *Comparison of the OMG and ISO/CCITT Object Models*, Network Management Forum, Bernardsville, NJ, 1993.
- [Ash94] Ashford, C. Specifying Relationships between Resources in Telecommunications Systems. In *Proceedings of Workshop on Specification of Behavioral Semantics in Object-Oriented Systems, OOPSLA 94*. Institute for Information Management, Robert Morris College, Coraopolis and Pittsburgh, Pennsylvania, USA, 1994.

-
- [Ber81] Bernstein, P. and Goodman, N. Concurrency Control in Distributed Database Systems. *ACM Comp. Surv.* 13(2), 1981.
- [Betz94] Betz, M. Interoperable Objects. *Dr. Dobbs Journal*, (19)11, October 1994.
- [Boo91] Booch, G. *Object Oriented Design with Applications*. Benjamin/Cummings, 1991.
- [Broc95] Brockschmidt, K. *Inside OLE 2*. Microsoft, 1995.
- [Brooks82] Brooks, F. *The Mythical Man-month Essays in Software Engineering*. Addison-Wesley, 1982.
- [Brooks87] Brooks, F. No Silver Bullet. *IEEE Computer*, (20)4, April 1987.
- [Buhr96] Buhr, R. and Casselman. *Use Case Maps for Object-oriented Systems*. Prentice Hall, 1996.
- [Busc96] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., and Stal, M. *Pattern-oriented Software Architecture*. John Wiley (to be published).
- [Cat94] Cattell, R., (Ed). *The Object Database Standard*. Morgan Kauffmann, 1994.
- [CCITT88] CCITT Recommendation Z.100: *Specification and Description Language SDL*. AP IX-35, 1988.
- [Coad91] Coad, P. OOA & OOD: a continuum of representation. *JOOP* (3)6, February 1991.
- [Cook94] Cook, S. and Daniels, D. Software isn't the real world. *JOOP* (7)2, May 1994.
- [Cox90] Cox, B. There Is a Silver Bullet. *Byte*, October, 1990.
- [Cus92] Cusack, E. and Fafsanjani, G. H. B. ZEST. In *Object Orientation in Z*, Eds Stepney, S. and Barden, R. Springer-Verlag, 1992.
- [DeMa79] DeMarco, T. *Structured Analysis and System Specification*. Prentice Hall, 1979.
- [Di79] Dijkstra, E. Programming Considered as a Human Activity. In *Classics in Software Engineering*, Yourdon Press, New York, 1979.
- [DSET95] *Agent Toolkit*. DSET, 1011 Route 22 West, Bridgewater, NJ 08807.

-
- [Eijk89] Eijk, van P.H.J., Vissers, C. A., and Diaz, M. Eds. *The Formal Description Language LOTOS*. North Holland, 1989.
- [Eri95] *IM Tools*. Erisoft AB, Box2003, S-930 30 Ursviken, Sweden.
- [Gab93a] Gabriel, R. P. Abstraction Descant, part 1. *JOOP* (6)1, March/April 1993.
- [Gab93b] Gabriel, R. P. Abstraction Descant, part 2. *JOOP* (6)2, May 1993.
- [Gam95] Gamma, E., Helm, R., Johnson, R., and Vlissitudes, J. *Design Patterns*. Addison-Wesley, 1995.
- [Gar95] Garlan, D. Research Directions in Software Architecture. *ACM Comput. Surv.* (27)2, June 1995.
- [Geig94] The IBM cmipWORKS. IBM Technical Report, IBM Research Triangle Park, May 1994.
- [Hausz86] Hauzer, B. M. A Model for Naming, Addressing, and Routing. *ACM Trans. Off. Inf. Sys.* 4(4), Oct 1986.
- [ISO84] International Organization for Standardization and International Electrotechnical Commission. *Information Technology — Open Systems Interconnection — Basic Reference Model*. International Standard ISO/IEC 7498 : 1984.
- [ISO87] International Organization for Standardization and International Electrotechnical Commission. *Information Technology — Open Systems Interconnection — Service Definition for the Association Control Service Element*. International Standard ISO/IEC 8649 : 1987.
- [ISO89] International Organization for Standardization and International Electrotechnical Commission. *Information technology — Open Systems Interconnection — Basic Reference Model — Part 4: Management Framework*. International Standard ISO/IEC 7498-4 : 1989.
- [ISO91a] International Organization for Standardization and International Electrotechnical Commission. *Information technology — Open Systems Interconnection — Common Management Information Service Definition*. International Standard ISO/IEC 9595 : 1991.

-
- [ISO91b] International Organization for Standardization and International Electrotechnical Commission. *Information technology — Open Systems Interconnection — Management Information Services — The Directory*. ISO/IEC 9594. Dec. 1991.
- [ISO92a] International Organization for Standardization and International Electrotechnical Commission. *Information Technology — Open Systems Interconnection — LOTOS: A Formal Description Technique Based on the Temporal Ordering of Observational Behavior*. International Standard ISO/IEC 8807 : 1988.
- [ISO92b] International Organization for Standardization and International Electrotechnical Commission. *Information Technology — Open Systems Interconnection — Management Information Services — Structure of Management Information — Part 4: Guidelines for the Definition of Managed Objects*. International Standard ISO/IEC 10165-4 : 1992.
- [ISO92c] International Organization for Standardization and International Electrotechnical Commission. *Information Technology — Open Systems Interconnection — Management Information Services — Structure of Management Information — Part 1: Management Information Model*. International Standard ISO/IEC 10165-1 : 1992.
- [ISO92d] International Organization for Standardization and International Electrotechnical Commission. *Information Technology — Open Systems Interconnection — Management Information Services — Structure of Management Information — Part 2: Definition of Management Information*. International Standard ISO/IEC 10165-4 : 1992.
- [ISO92c] International Organization for Standardization and International Electrotechnical Commission. *Information Technology — Open Systems Interconnection — Distributed Transaction Processing*. ISO/IEC 10026 : 1992.
- [ISO95a] International Organization for Standardization and International Electrotechnical Commission. *Information Technology — Open Distributed Processing— Reference Model— Part 1: Overview*. International Standard ISO/IEC 10746-1 : 1995.

-
- [ISO95b] International Organization for Standardization and International Electrotechnical Commission. *Information Technology — Open Distributed Processing— Reference Model— Part 2: Foundations*. International Standard ISO/IEC 10746-2 : 1995.
- [ISO95c] International Organization for Standardization and International Electrotechnical Commission. *Information Technology — Open Distributed Processing— Reference Model— Part 3: Reference Model*. International Standard ISO/IEC 10746-3 : 1995.
- [ISO95d] International Organization for Standardization and International Electrotechnical Commission. *Information Technology — Open Distributed Processing— Reference Model— Part 4 : Architectural Semantics*. International Standard ISO/IEC 10746-4 : 1995.
- [ISO95e] International Organization for Standardization and International Electrotechnical Commission. *Information Technology — Open Systems Interconnection — Management Information Services — Structure of Management Information — Part 7: General Relationship Model*. International Standard ISO/IEC 10165-1 : 1995.
- [ISO95f] International Organization for Standardization and International Electrotechnical Commission. *Information Technology — Open Systems Interconnection — Application Context for Systems Management with Transaction Processing*. International Standard ISO/IEC 11587 : 1995.
- [ISO95g] International Organization for Standardization and International Electrotechnical Commission. UK Comments on the use of Z for the definition of the behaviour of managed objects. Document reference: ISO/IEC JTC1/SC21/WG4 N2161, June 1995.
- [ITU95] International Telecommunication Union Recommendation M.3000. *Overview of TMN Recommendations*. ITU 1995.
- [Jud93] Judge, A. J. and Wezeman, C. D. Formal specification of managed objects—a case study. *BT Technology Journal*, (11)3, July 1993.

-
- [Kil94] Kilov, H. and Ross, J. *Information Modeling: an Object-oriented Approach*. Prentice-Hall, 1994.
- [Kin95] Kincel, N. (Ed.) *OmniPoint Integration Architecture*. Network Management Forum, Morristown, NJ, 1995.
- [Kras88] Krasner, K. and Pope, S. A Cookbook for Using the Model-View-Controller User Interface in Smalltalk-80. *JOOP* (1)3, August/September 1988.
- [Krug92] Krueger, C. W., Software Reuse, *ACM Comput. Surv.* 24(2), June 1992.
- [Lal93] LaLond, W. and Pugh, J. Instance-based Programming with PARTS. *JOOP*, (6)1, March/April 1993.
- [Lee88] Lee, R. M. Bureaucracies as Deontic Systems, *ACM Trans. Off. Inf. Sys.*, 6(2), April 1988.
- [Levy90] Levy, E and Silberschatz, A. Distributed File Systems: Concepts and Examples. *ACM Comput. Surv.* 22(4), Dec 1990.
- [Mar95] Martin, R. Patterns: PLoP, PLoP, fizz, fizz. *JOOP*, 7(8) , Feb 1995.
- [Mil56] Miller, G. The Magical Number Seven, Plus or Minus Seven: Some Limits on Our Capacity to Process Information. *The Psychological Review*, (63)2, March 1956.
- [Mjo93] *GDMO Hyper-structure Editor*, Mjolner Informatics, Science Park Aarhus, Gustav Wiedsvej 10, Aarhus, Denmark.
- [Mow95] Essentials of Object-oriented Architectures, *Object Magazine* 5(5), September 1995.
- [NAG94] *The NAG Fortran Library*, The Numerical Algorithms Group, Oxford, 1994.
- [Ode94] Odell, J. J. Six Different Kinds of Composition. *JOOP* 5, 8 (January 1994).
- [OMG92] The Object Management Group. *The Common Object Request Broker: Architecture and Specification*. OMG Document Number 91.12.1, Rev. 1.1, 1992.
- [Par85] Parnas, D, Clements, P., and Weiss, D. The modular construction of complex systems, *IEEE Trans Softw Eng*, SE-11, March 1985.

-
- [Poir95] Poirier, S. and Ashford, C. Semantics: the Key to Interoperability. In *Proceeding of the OOPSLA 95 Workshop on Business Objects*. (To be published).
- [Ros92] Rosenberry, W., Kenney, D., and Fisher, G. *Understanding DCE*. O'Reilly and Associates, 1992.
- [Rum87] Rumbaugh, J. Relations as Semantic Constructs in an Object-Oriented Language. In *Proceedings of the Object-Oriented Programming Systems, Languages, and Applications*. ACM Press, New York, 1987.
- [Rum91] Rumbaugh, J., Blaha, M., Premelani, W., Eddy, F., and Lorenson, W. *Object-oriented Modelling and Design*. Prentice-Hall, 1991.
- [Rum94] Modelling Models and Viewing Views. *JOOP*, (7)2, May 1994.
- [Shoch78] Schoch, J. Inter-network Naming, Addressing, and Routing. In *Proceeding of IEEE Computer Conference, Compcon (Washington, D.C., Fall)*. IEEE New York, 1978.
- [Shro94] Schroder, M. State-of-the-Art Distributed System: Computing with BOB. In *Distributed Systems*, Ed. Sape Mullender. Addison-Wesley and ACM Press, 1994.
- [Spiv89] J.M. Spivey, *The Z Notation: A Reference Manual*. Prentice-Hall International, 1989.
- [Weg93] Wegner, P. *Towards Component-based Software Technology*. Technical Report No CS-93-11, Brown University, RI.
- [Whi94] White, J. *Telescript Technology: the Foundation for the Electronic Marketplace*. General Magic Inc., California 94040, USA, 1994.
- [Wil93] Wilson, J. C. Formal methods in object oriented analysis. *BT Technology Journal*, (11)3, July 1993.
- [Witt94] Witt, B., Baker, F., and Merritt, E. *Software Architecture and Design*. Van Nostrand Reinhold, New York, 1994.

-
- [X395] *RM-OPD Enterprise Viewpoint New Work Proposal*. Object Information Committee X3H7 of American National Standards Institute. X3H7-95-22 1995.

Abbreviations

ACSE	Association Control Service Element
ANSA	Advanced Network System Architecture
API	Application Programming Interface
ATM	Asynchronous Transfer Mode
CMISE	Common Management Information Service Element
COM	Common Object Model
CORBA	Common Object Request Broker Architecture
DCE	Distributed Computing Environment
DPE	Distributed Processing Environment
DSOM	Distributed System Object Model
IDL	Interface Definition Language
ISO	International Organization for Standardization
ITU	International Telecommunication Union
LAN	Local Area Network
ODP	Open Distributed Processing

OLE	Object Linking and Embedding
OMG	Object Management Group
OMT	Object Modelling Technique
OSI	Open Systems Interconnection
RM-ODM	Reference Model for Open Distributed Management
RM-ODP	Reference Model for Open Distributed Processing
SONET	Synchronous Optical Network
TCP/IP	Transmission Control Protocol/Internet Protocol
TINA-C	Telecommunications Intelligent Network Architecture-Consortium
TMN	Telecommunications Management Network
TP	Transaction Processing
TP-ASE	Transaction Processing-Application Service Element

END

2 1 - 1 1 - 9 6

FIN