Performance Modelling and the Representation of Large Scale Distributed System Functions

Obong Dennis Obot Nyong

School of Engineering
University of Durham

February 1999

A thesis submitted for the degree of Doctor of Philosophy (Ph.D.)
of the University of Durham

Obong Dennis Obot Nyong

Performance Modelling and the
Representation of Large Scale Distributed System Functions Ph.D. 1999.

Abstract

This thesis presents a resource based approach to model generation for performance characterization and correctness checking of large scale telecommunications networks. A notion called **the timed automaton** is proposed and then developed to encapsulate behaviours of networking equipment, system control policies and non-deterministic user behaviours. The states of pooled network resources and the behaviours of resource consumers are represented as continually varying geometric patterns; these patterns form part of the data operated upon by the timed automata. Such a representation technique allows for great flexibility regarding the level of abstraction that can be chosen in the modelling of telecommunications systems. None the less, the notion of **system functions** is proposed to serve as a constraining framework for specifying bounded behaviours and features of telecommunications systems.

Operational concepts are developed for the timed automata; these concepts are based on limit preserving relations. Relations over system states represent the evolution of system properties observable at various locations within the network under study. The declarative nature of such permutative state relations provides a direct framework for generating highly expressive models suitable for carrying out optimization experiments.

The usefulness of the developed procedure is demonstrated by tackling a large scale case study, in particular the problem of congestion avoidance in networks; it is shown that there can be global coupling among local behaviours within a telecommunications network. The uncovering of such a phenomenon through a function oriented simulation is a contribution to the area of network modelling. The direct and faithful way of deriving performance metrics for loss in networks from resource utilization patterns is also a new contribution to the work area.

Declaration

I hereby declare that this thesis is a record of work undertaken by myself, that it has not been the subject of any previous application for a degree, and that all sources of information have been duly acknowledged.

Acknowledgements

First and foremost I would like to express my thanks and appreciation to my supervisor Professor Philip Mars for his guidance, encouragement, enthusiasm, support and patience throughout this project.

Many people have contributed to this work in various ways. My thanks to Dr. John Ainscough for his guidance, advice and encouragement when I started identifying problems associated with distributed systems. My thanks to Professor Joseph Goguen for sending me his pre-publication paper on *sheaf semantics*; his paper has influenced greatly my thinking on declarative specification of distributed systems. My thanks to Professor Andrew Herbert, John Warne, Owen Rees, Dave Otway and Rob van der Linden all at the ANSA laboratory for guidance during my secondment to the laboratory; I developed the ideas on addressing and storage during that period.

My involvement in various research projects have also shaped my thinking on performance modelling of networking systems. My thanks to the ARMAN DTI/EPSRC supported project and the project team for their patience during long hours of discussions. I tried out a lot of ideas within the ARMAN team. My thanks go especially to Philip Aranzulla and Dr. Jonathan Pitts who have been my immediate co-workers. Philip jointly implemented the *simulation jacket*. This is the computation engine used to animate actions of the timed automaton presented in this thesis. The implementation task of such a speculative simulation environment can only be realized through team work. Jonathan joined Philip and myself in generating experimental data for several experiments used to exercise my timed automata.

My thanks go to my wife and children for total support. My thanks also to my extended family (parents, sisters, in-laws and cousins) and my friends for being patient.

My thanks to my employers for providing me with a flexible research environment at Cable and Wireless, Anite and CASE.

Contents

1 Introduction 1

1.1 The need for a reference model for system functions 1
1.2 Deriving system performance metrics from resource utilization patterns 2
1.3 Holistic modelling of telecommunications networks 2
1.4 Outline of thesis 3

2 Introduction to distributed system functions 5

2.1 Overview of system functions and large scale sub-networks 5

2.1.1 Introduction to system model generation issues 6
2.1.2 Hierarchies of network architectures 8
2.1.2.1 Network-wide and nodal resources 8
2.1.2.2 Wide area network functions 10
2.1.2.3 Nodal functions 14

2.2 A structure for system functions and computing entities 16

2.2.1 Introduction 16
2.2.2 An overview of distributed objects 17
2.2.2.1 The intelligent agent 17
2.2.2.2 A collection of intelligent agents 18
2.2.2.3 A structure for components within an intelligent agent 19
2.2.2.4 Distributed objects as co-operating agents 21
2.2.3 Locations and ports for distributed objects 22
2.2.3.1 Named locations for objects 22
2.2.3.2 Clustering objects 24

2.3 Overview of model representation for logical validation and performance characterisation 24

2.3.1 Algorithm framework for computations 24
2.3.2 Controlling the flow of information streams 26
2.3.2.1 Modelling for uncertain resource demands 26

2.3.2.2 Multiplexing and de-multiplexing of streams 28
2.3.3 Basic issues on complexity measures of representations 30
2.3.3.1 Basic issues on distributed system complexity 30

2.4 Summary 32

3 Overview and critical survey of modelling semantics for distributed algorithms 33

3.1 Introduction 33

3.1.1 Motivations, challenges and problem statement 33
3.1.2 Structure of the chapter 35

3.2 A resource allocation paradigm for specification of system functions 36

3.2.1 Introduction 36
3.2.2 The universe of resource pools and policy based resource allocation 36
3.2.3 Resource usage optimization within a single time window 37
3.2.3.1 Permutations on state variables 41
3.2.3.2 Constraints on state space 41
3.2.4 Prediction as a primitive concept within distributed algorithms 42
3.2.5 An integrated circuit of distributed algorithms 44

3.3 A critical review of modelling concepts for distributed algorithms 45

3.3.1 Introduction 45
3.3.2 Semantics and realization criteria in the assessment of distributed algorithms 46
3.3.2.1 Semantics 46
3.3.2.2 Realisation 46
3.3.2.3 A taxonomy for description of modelling techniques 47
3.3.3 A critique of important modelling concepts 53
3.3.3.1 Representation as analytic approximations 55
3.3.3.2 Representation as learning automata 56
3.3.3.3 Representation as transaction automata 57

3.4 System states as continually varying geometric quantities 58

3.4 Summary 58

4 Object interactions in the realisation of system functions 60

4.1 Introduction 60

4.2 Resource structuring for sharing 60

4.2.1 Partitioning of transmission resources 60
4.2.2 Structuring of shared resources 62
4.2.2.1 Information transmission attributes - end systems 62

4.3 Case study I: The storage functions 64

4.4 Case study II: Congestion avoidance functions 66

4.5 Summary 70

5 Product form representation of system functions . 72

5.1 Introduction 72

5.2 State co-ordinates 73

5.2.1 Computation of structured state values 75
5.2.2 Convergent sequences of timed relations 76

5.3 Classification of system state evolution 77

5.3.1 Structuring a state space into subsets of the space 77
5.3.2 Statistical values, probability values and the timed automaton 79
5.3.2.1 Deterministic and non-deterministic states 80

5.4 Admissible functions and constraints for performance characterisation 81

5.5 Product form representation of timed automata 83

5.6 Summary 84

6 Timed automaton: the graph model 86

6.1 Introduction 86

6.2 Policy criteria for resource allocation 86

6.2.1 The logical structure of system function specifications 86
6.2.2 Combining theories and generating models 87
6.2.2.1 Models from functions and relations 89
6.2.2.2 Timed automata from theories 92

6.3 Summary 101

7 Application of the timed automaton hypergraph to the specification and simulation of system functions 102

7.1 Introduction 102

7.2 The generic system function for resource management 102

7.2.1 Declarative specification of system behaviour attributes 103

7.3 The operational framework for implementation of system functions 104

7.3.1 The physical model 104
7.3.2 A solution framework for model studies of system functions 107

7.4 Adaptive resource partitioning within a distributed system 109

7.4.1 Introduction 109
7.4.2 Generation of source - destination paths 110
7.4.3 A simulation example: Adaptive resource allocation in the presence of uncertainty 115
7.4.3.1 Introduction 115
7.4.3.2 Control dynamics and transitive behaviours 115

7.5 Summary on simulation and congestion avoidance 123

8 Conclusions and areas for further work 125

8.1 Architecture, modelling components and experimentation 125

8.2 Areas for further work 126

Appendices

A Specification of storage functions 128

A.1 Introduction 128

A.2 Basic specification rules 128

A.3 Operations supporting the storage functions 128

A.4 The Interface Reference as a Type 144

A.5 Detailed specification of storage operations 128

B A graph model specification of the congestion avoidance function 166

B.1 Introduction 166

B.2 State transitions from the geometry of behaviours 166

B.2.1 Evolution bounds of system entities 166

B.2.2 State transition graph automata 167

B.2.2.1 The source module's automaton 167

B.2.2.2 A refinement of the source module's automaton 168

C A generic characterization of system function realizations 176

C.1 Introduction 176

C.2.1 Introduction to the specification of characteristic relations 176

C.2.2 Observations and attribute values 177

C.2.3 Generic state structures 180
C.2.3.1 Illustrative diagrams 180
C.2.3.2 Formalisms 183

C.2.4 Summary 190

Annex: Classical complexity measures 191

D Selected publications by author 194

References 195

Highlighted words and phrases

Word/Phrase — section described

abstract data types 3.3.2.2
abstract machine 3.3.2.2
alphabet 2.3.1
automaton 5.3.1
call pattern 3.2.3
canonical 2.3.1
capsule 4.2.2
category theory 3.3.2.3
cluster 4.2.2
common knowledge 3.3.2.3
complexity measures 2.1.1
computational object 2.2.2.4
decision engine 3.3.2.2
denotations 3.3.2.1
direct limits C.1
distributed agreement 3.3.2.3
engineering object 2.2.2.4
graph modelled timed automaton 6.1
Interface Reference 5.5
messages 2.2.2.4
modelling framework 2.1.1
NSAP address 2.2.3.1
permutations 5.3.1
plausible behaviour curves 6.3
policy based mappings 3.2.2
process 2.1
processing unit 2.1
representation 2.1.1
rewriting rules, productions 2.3.1
routeing pattern 3.2.1
schedule ports 2.2.3.1
sentences 3.3.2.1
socket 4.2.2
statements 3.3.2.1

sub-network 2.1
symbols, letters 2.3.1
system function, sub-function 2.1
theoretical concepts 2.1.1
timed automaton 6.2.2.2
traffic variables 3.2.3
word 2.3.1

Chapter one

Introduction

1.1 The need for a reference model for system functions

Large scale telecommunications networks provide services to users by pooling resources whenever it is operationally convenient because provisioning costs can be minimized through resource sharing. A simple example of resource sharing is the multiplexing of data streams from a collection of network users onto a transmission resource en-route to data bases at various destinations within the network. In executing the multiplexing function, a number of supporting computations are normally carried out at various locations within the network. The multiplexing function would in turn be part of other functions which comprise a service offered to users of a telecommunications network. Thus multiple-service networks are often provisioned with a large collection of entwined resource management functions.

There is a continuing change in the content and delivery procedure of telecommunications services. Also, existing telecommunications equipment are continually being replaced by newer equipment which incorporate newer device technologies. In keeping up with these rapid changes, various attempts at devising a reference model for defining service providing functions have failed to gain universal acceptance. Examples of such models include the International Standards Organization's basic reference model, and the International Telecommunications Union's Broadband Services (B-ISDN) reference model.

One way of developing a reference model that is never out of date is to start from networking components whose attributes are not technology or service dependent; such attributes should only evolve in a self-consistent way. Structured network locations (as a seamless primitive structure) has been identified by the author as a suitable building block for an object based distributed system architecture. This thesis presents a physical decomposition technique for the representation of networking systems; the technique is based on exploiting the inter-relationship among locations, and the derivation of information transmission attributes of channels that span pairs of locations. The technique provides a framework for generating models of system components at levels of refinement appropriate for tackling specific problems.

1.2 Deriving system performance metrics from resource utilization patterns

User demands on pooled resources are generally non-deterministic. The provisioning of resources at various locations within a network must reflect predicted behaviours of users and the feasibility of resource sharing among these users in their interactions across the network. Invariably there are times when some resources are unavailable to additional users because existing users are consuming most of the capacity. Loss or blocking is said to have occurred under such a circumstance.

Most existing modelling and simulation techniques use measures such as cell loss probability or call blocking probability to characterize system performance. A close look at how these measures should be derived reveals that it is non-trivial to generate a model that has a justification for the underlying resource saturation assumptions. In many modelling and simulation projects, one often simply makes up network behaviours that characterize loss or blocking probabilities.

A better way to characterize system behaviour performance is to generate measures of resource utilization at various locations within a network under study. This is carried out by stating explicitly the dynamic behaviours of resource bearing equipment operating at relevant locations within the network. Utilization based characterization of system performance is adopted in this thesis.

1.3 Holistic modelling of telecommunications networks

The generation of utilization based metrics results in larger and more comprehensive specifications of telecommunications functions (see appendices A and B). There is therefore a need to develop special techniques to handle the state explosion associated with such specifications. This thesis presents an *algebraic geometry* approach that can serve as a procedure for keeping track of, and classifying large state spaces generated in the specification of system functions. The new approach is therefore a superset of existing rather non-axiomatic approaches for defining loss within networks.

The importance of the proposed technique is illustrated using a case study on congestion avoidance; in the case study, a large state space is given an algebraic structure suitable for carrying out simulation experiments. Simulation experiments performed heretofore rarely address such exhaustive state inter-relationships.

1.4 Outline of thesis

There are two complementary threads in this thesis.

Chapters two, four, seven, appendix A and appendix B are mostly descriptive; very few mathematical notations are deployed. The emphasis in these chapters is on the use of holistic modelling concepts to identify and solve important telecommunications problems. Starting from the notion of named locations, a distributed object oriented representation technique is developed and shown to be sufficiently expressive in the representation of large scale system functions.

Chapters three, five, six and appendix C contain more detailed theoretical formulations in support of the notion of the timed automaton. This concept provides the operational semantics for the distributed object model put forward in chapters two and four. Algebraic and geometric approaches to the classification of large state spaces are exposed in these chapters.

Chapter two is a description of the holistic nature of system functions. The chapter illustrates how the architecture components of a modelling procedure can be built up from a basic addressing scheme. The importance of the use of geometric techniques in performance modelling is introduced in the chapter.

Chapter three is a critical survey of existing modelling techniques. A semantical framework is introduced in the chapter as a basis for evaluating formal representations of networked functions. A new architecture for carrying out modelling experiments is also introduced in the chapter.

Chapter four presents a development of a distributed object oriented modelling technique which has the capability of hiding unnecessary detail in the specification of large scale system functions. The two case studies introduced in chapter four are developed in detail in appendices A and B. The case study described in appendix A is used as a transaction based example for refining the system function developed in appendix B.

The modelling experiments described in chapter seven are based on the specification and representation of the congestion avoidance system function described in appendix B. The results from these experiments are rich in system dynamics phenomena. Local-global linkage of network behaviour is shown in the chapter.

Chapters five and six build on the architecture and semantical framework introduced in chapter three. All the technical detail that characterize the timed automaton are described in chapters five, six and appendix C. This work provides the foundation for further development of automata based operational semantics in many work areas such as pattern classification over statistical data, and teams of co-operating learning automata. Areas for further work are highlighted in chapter eight where conclusions on some of the issues addressed in this thesis are also stated.

Chapter two

Introduction to distributed system functions

2.1 Overview of system functions and large scale sub-networks

A distributed **system function** is a self-consistent set of computations carried out concurrently at two or more processing entities within a telecommunications system. In its most general form, a system function is a set of sequences of computations executed by one or more perpetual processes; it is started at an instant within an initiation time interval and continues processing for all the time. However, such perpetual system functions often generate, or accept through real-time configuration management, other functions **(sub functions)** which have finite time of existence. An example (sub-) function is the distributed call supervision function of a telecommunications network. Such a function could be enabled by a subscriber line supervisory function which perpetually scans subscriber lines, reporting activity statistics to a network management function.

A **process** is defined to be a set of sequences of computations that can be scheduled to run at a **processing unit** within a networked node. It is customary to refer to a network under study as a **sub-network** since every network can be viewed as being capable of inter-working with a larger network. Thus a sub-network is normally designed to support a large number of distributed processes which co-operate to carry out computations that animate a large number of system functions. This chapter provides an overview of the physical and logical structures that characterise distributed functions within telecommunications networks. The motivation for the presentation in this chapter is as follows:

Often many research workers attempting to solve problems in the area of telecommunications networks start off by selecting a solution technique, e.g. linear programming. A telecommunications problem is then formulated and it is shown how the selected technique solves the problem well, according to some assessment criteria. Unfortunately many of the formalised solutions so presented have met with unfavourable responses from practitioners such as network architects and network designers who specify, design and commission emerging networking systems. A common criticism of many existing formalised solutions is that idealized models adopted in solutions of many real world problems do not scale well, and often adopt unrealistic assumptions.

From a positive point of view, many successful research solutions of telecommunications

systems problems adopt real world models as the starting point. For example, contributions on encryption of transmitted packets of information in preventing un-authorized decoding of such packets. One reason for adopting the 'techniques first' approach to research is because the physical structures of telecommunications networks are still in a state of repeated revision aimed at coping with changes in equipment and service delivery technologies.

This chapter proposes that by using the basic language of set theory in an informal way, the logical and physical structures of emerging networking systems can be described in a generic manner, irrespective of the underlying implementation technology of the commissioned equipment. The structure has been generated by the author over the last five years; it brings together in a holistic way generic concepts that appear to hold as primitive. It is likely that minor changes can and will be made to this structure. The structure provides a framework for posing realistic questions on networking research problems such as: 'Is a specified research proposal dealing with a solved (or partially solved) problem, an unsolved problem, or a non-problem ? '

The novel feature of this approach is that every networking problem involving distributed algorithms can be stated in terms of system functions. Such functions can be derived using concepts built from standard notions of set theoretic functions and relations. Functions in set theory can have graphs over values. Graphs are geometric objects which can be used to isolate important system behaviours. The ramification of this simple approach to a difficult problem area is obvious.

2.1.1 Introduction to system model generation issues

The fundamental requirement for the provision of system functions is to animate declared sets of operational policies in order to control allocation of resources to users. The notions of 'resources' and 'users' are often used in a very broad sense, e.g. a resource could be a number of units of transmission link capacity; a user set could be a pair of subscribers engaged in a telephone conversation. An operational policy could be the reservation of sufficient link capacities at all times, at appropriate points within a network. Such a reservation is often necessary in case a subscriber needs to make a telephone call requesting access to a guaranteed service. An example of a guaranteed service is the completion of all calls to an ambulance station within a specified time period measured from the time the call is recognised at the calling subscriber's local exchange. Thus the operational constituents of a telecommunications network are

- users and their network access equipment
- networking nodes, their resources and local policies for allocation of nodal resources to users
- inter-node transmission resources and associated distributed (nodal) policies for allocation of inter-node resources to users.

User behaviours are generally non-deterministic. Thus policies for allocation of resources to users need to be designed such that anticipated statistical variations in user behaviours can be accommodated. Moreover, the system should be designed such that unanticipated variations in user demands on resources can be catered for either through re-provisioning of resources, or by changes in allocation policies. Thus, irrespective of the approaches adopted for adaptation to user demands, the system needs to be designed so that service disruption is minimal.

In large scale sub-networks, a large number of system functions are usually implemented to be executed concurrently within a collection of nodes. A major challenge in the specification and design of system functions is the need to ensure that concurrent computations carried out over non-deterministic input data generate results consistent with desired system behaviours. There is therefore a need for a small number of **theoretical concepts,** with sufficient expressive power, to serve as a framework for **representation** of system functions. Such concepts ought to provide the framework for specification of rules that must hold when policies are being executed in the animation of system functions. It should also be possible to verify internal consistencies among a subset of rules that are used to specify system functions. A set of theoretical concepts for representation of system functions can be referred to as a **modelling framework.**

It is important that any modelling framework adopted for the specification and design of system functions scale up gracefully when applied in large networks. Since such networks are often hierarchical in structure (with respect to provisioned system functions) modelling frameworks need to be suitable for the compositional specification of system functions. Moreover, there is often a requirement that **complexity measures** of system behaviour structures should be derivable from specifications based on the modelling framework adopted. Such a requirement ensures that there is a measure of the level of difficulty to be encountered in the validation of the implemented model of the system. Validation is often considered to be the first stage of system provisioning.

A level of confidence can be attached to performance characteristics generated by a simulation model of a set of system functions within a telecommunications network. It is important that

any simulations carried out as performance evaluation of parts of a telecommunications network should be based on a theoretically rigorous representation of parts of the system under study. It is therefore desirable to ensure that a modelling framework being developed for modelling systems can exhibit performance attributes of parts of the system under study.

2.1.2 Hierarchies of network architectures

2.1.2.1 Network-wide and nodal resources

A sub-network is normally implemented as a sub-system within a universe of sub-networks. Figure 2.1 illustrates an example of the physical topology of a corporate network implemented as a sub-system of a wide area network (WAN). The corporate network consists of three Customer Premises Networks: CPNs A, B and C inter-connected by the wide area network. In such a scenario, the CPNs A, B and C constitute a sub-network from the point of view of the WAN. In practice, the WAN could be a carrier (i.e. an authorised service provider) network within a national network; in such a case, the WAN would be a sub-network of the national network. In turn, the WAN could host a number of corporate networks as its sub-systems.

The CPNs illustrated in figure 2.1 would normally represent sites, e.g. campuses of a university. Thus CPNs A, B and C could be located at three towns within a large city. The WAN then performs a function of providing transmission resources for information transport among CPNs at the various sites.

In a different scenario, the sites could be located at different towns within a country, or within towns located in different countries. In such a case, an important issue is the cost of providing inter-site transmission resources. In assessing the importance of such costs it is useful to enumerate the key components that contribute to the cost of provisioning a telecommunications network. Such an exercise is necessary in order to highlight areas where research effort could be directed in order to generate cost saving solutions.

A convenient grouping for components of a telecommunications network is

i. nodal hardware and control software for intra-node functions, e.g. control procedures for data transport and information exchange within a node
ii. network-wide data transport and connection control software

iii. network-wide system management software

iv. hardware transmission links for inter-node transport of user data, network control and network management data

v. network-wide software platform for management of access to open distributed applications

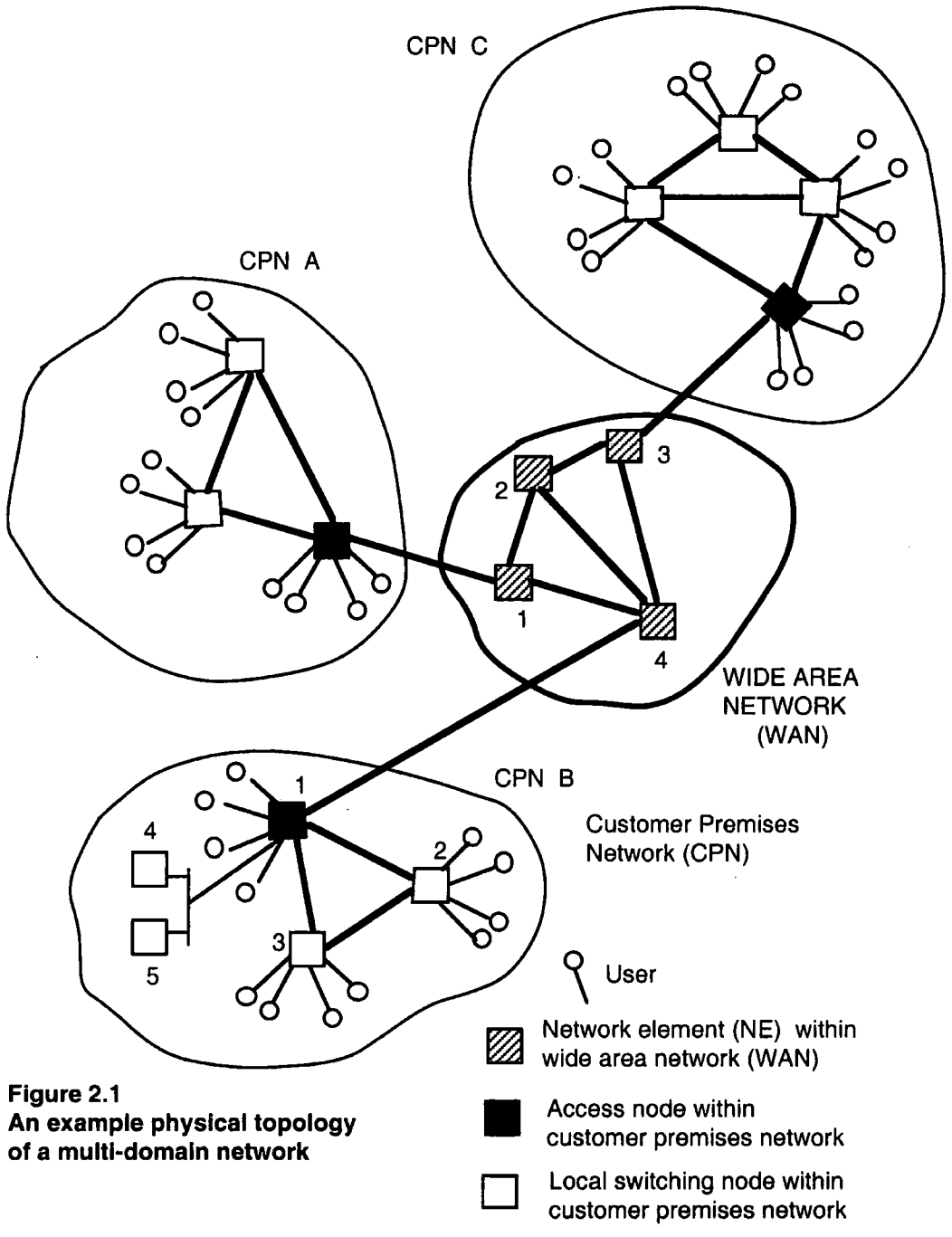


Figure 2.1
An example physical topology of a multi-domain network

These components are implemented and deployed as resources to be consumed (occupied) by users who subscribe to use the network. Thus resources that can be occupied by users are:

- processor schedules and attached memories
- time slots within transmission resources
- access ports of open distributed services

There are complex inter-relationships among resource types, location of resources within a network, sequences of operations for executing resource allocation policies, profiles of user demands for resources, and costs of provisioning resources in order to satisfy target quality of service offerings to users. These inter-relationships need to be taken into consideration when designing system functions that operate within sub-networks. In particular, emphasis needs to be focused in developing system functions that allocate resources to users in an efficient way, taking into consideration optimisation issues with respect to policies for allocation of appropriate resources.

2.1.2.2 Wide area network functions

In the scenario illustrated in figure 2.1 a CPN (e.g. CPN B) could be envisaged as being provisional using the same type of components as those used to provision the WAN. However, the occupancy cost of transmission links within a CPN would normally be several orders of magnitude lower than occupancy costs of WAN transmission resources. A number of reasons contribute to the high cost of using large scale wide-area networks.

Resources within the WAN are normally pooled for use by a number of users whose pattern of resource consumption are generally non-deterministic. Thus there is often the possibility of rendering parts of the network under-provisioned while other parts are over-provisioned. Charging for network resource usage is therefore an effective indirect mechanism for control of access to network resources.

This study does not consider charging issues directly as a policy for specification of resource allocation functions. However, such policies can be introduced if desired, as an extension of a more fundamental resource allocation policy based on monitoring the level of consumption during a preceding time window (using appropriate statistical measures). Thus any resource allocation policy can be based on comparing currently observed levels of resource consumption with previously predicted consumption levels for the time window under consideration.

The main function performed by a collection of nodes within a WAN is the (cost effective) switching, multiplexing and de-multiplexing of user data streams from source ports to distinction ports across the network. Figure 2.2 illustrates the flow of information streams

through a node in the WAN. In this example, the switching and transmission technology assumed to be implemented is the Asynchronous Transfer Mode (ATM) for information streams. However, the concepts put forward in this thesis are generally applicable to other technologies.

In the ATM technology, each user's data stream is broken up into information packets (cells) which are transported as a stream from source to destination ports. A source destination stream is abstracted as a kind of information stream called the Virtual Channel Connection (VCC). A collection of VCCs, can be multiplexed together into a larger bundle of information stream called the Virtual Path Connection (VPC). Viewed in a different way, the VPC can be envisaged as a transmission resource capable of carrying a number of VCCs between nodes. Due to practical requirements to carry out VCC switching at network nodes, there is no requirement that a VPC's payload should consist of VCC streams from only one pair of nodes.

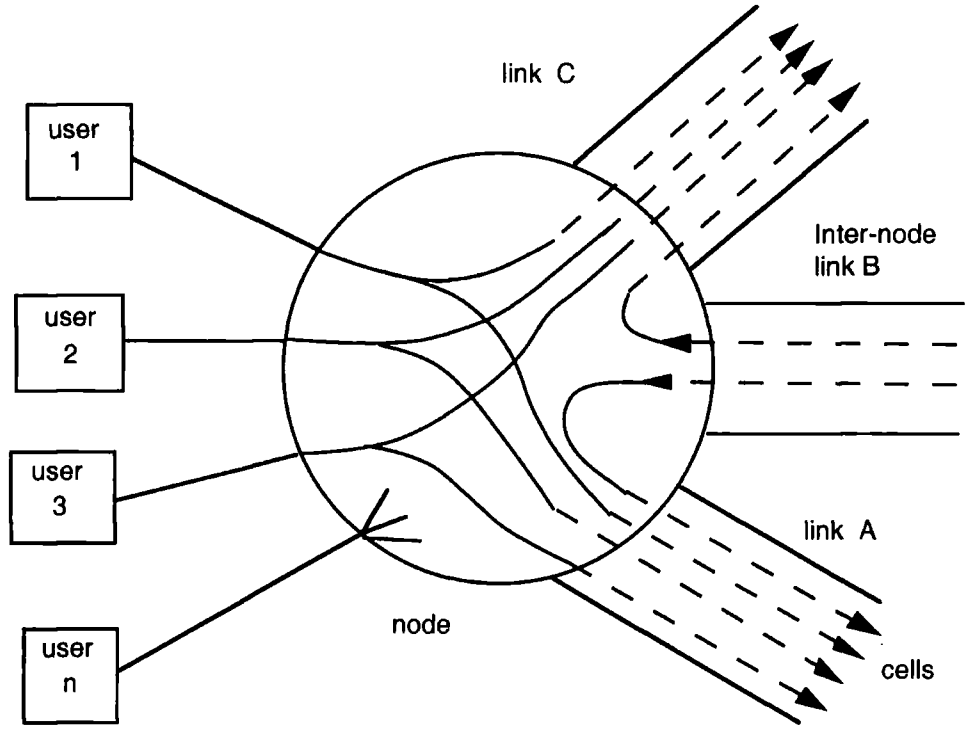


Figure 2.2 Flow of cell streams within node and links

The illustration on figure 2.2 shows three physical links A, B and C emanating from the node. Each physical link can be partitioned into a number of VPCs. The structure of VPCs that emanate from a node could have system behaviour attributes, e.g. switching and multiplexing efficiency trade off against provisioning costs. This thesis does not address such efficiency issues. However, the overall modelling framework developed in this thesis is sufficiently flexible to allow for studies to be carried out on function refinements involving characteristics of physical architecture structuring.

In general, it can be assumed that every access node into the WAN has the capability of hosting users. Thus, the structure of an architecture for system functions implemented within a node models both network specific and user behaviour functions. Such an approach to modelling system functions is attractive from a validation point of view, since the source of non-determinism in system behaviours can be accounted for directly within an implementation.

Figure 2.3 illustrates the functional architecture of an access node within a CPN. At the CPN-CPN level of abstraction, the WAN is considered to be purely a set of transmission links. An overview of system functions specified in each functional block is as follows:

i. VP and Physical processing

The hardware entities within this functional block generate a defined bit stream structure as information streams of cells, clocks and control messages. Compatible with the hardware implemented functions, the major software functions carried out within this functional block are the measurement of offered traffic into the network and congestion control on admitted VC streams. When a node acts as a transit VC switch, the congestion control function is extended to operate on a node-to-node basis in addition to effecting the user-network congestion control function.

ii. VC processing and switching

The hardware entities within this functional entity perform multiplexing, de-multiplexing, switching and bit stream integrity shaping of VC information streams. Software entities within the functional entity control the hardware functions by effecting invocations from the control and management functional entities at higher layers. The software functions include mechanisms that measure VP utilisation levels resulting from VCCs occupying VP resources, and the policing of VCC utilisation levels.

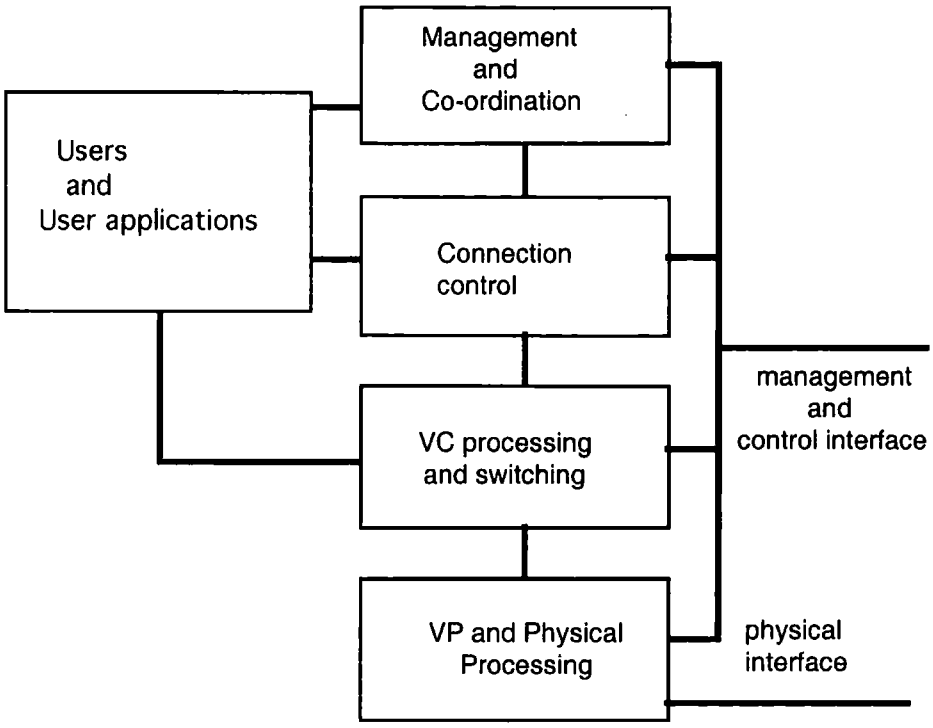


Figure 2.3
Control and management of user access to network-wide resources

iii. Connection control

Even though some of the functions located within this functional entity are normally implemented using special purpose hardware circuits, the connection control functions are often specified as operations carried out by a collection of co-operating software objects. In being invoked to make a decision on resource demand, the connection control function resolves competing options regarding choice of resources to allocate to a user. The Connection Admission Control and VCC Routeing are functions performed within this functional entity.

iv. Management and co-ordination

Functions performed within the management and co-ordination functional entity often involve selecting and enabling a desirable set of system behaviours from a large set of possible behaviours that can hold when a sequence of operations are invoked within the telecommunications network. Thus agreements by a collection of nodes within a network regarding the allocation of shared resources are often carried out within this functional entity. An example of such a function is co-ordinated fault diagnosis and containment. Another example is the execution of a distributed function that generates a set of routeing options

dynamically, with the goal of maximising the number of users concurrently utilising network resources.

It is useful to include within this functional entity the functions that support open distributed services. These functions are often designed to hide the intricacies of the underlying communications infrastructure from users and user applications. As an example, a set of management processes can be specified to operate across a telecommunications network as a system of address servers for open distributed applications. Such applications can thus offer their 'named services' open to other users in a 'location transparent' way.

v. Users and user applications

The functions provided within this functional entity vary from the specialised algorithms for encoding speech patterns to servers for printing documents at a printer. The type of application attached to the network characterises the VCC streams, the connection requests and the management functions required to provide the communications service.

2.1.2.3 Nodal functions

It is possible to specify and represent a model of the physical architecture of a node as if the node is a 'network of processing entities'. Thus it is also possible to develop a functional architecture for functions executed within a node. However, the functions executed within a node form a subset of network system functions. Thus in specifying wide area network functions, it is assumed that nodal functions are already in place.

Figure 2.4 illustrates a simplified hardware architecture of a switching node. It comprises:

- incoming and outgoing cell-stream interface modules
- switch block module
- control processor module

There are two main groups of nodal functions:

i. maintaining integrity of information streams traversing the node
ii. executing control functions in response to invocations from other computing entities distributed across the network.

Before elaborating on these functions, a refinement of the nodal architecture in figure 2.4 is useful. The physical architecture of a node can be interconnected to implement a fault-tolerant, network of processing entities. This implies that the provisioning of a node can be carried out in such a way that failure of any of the functional blocks within the node can be contained by invoking appropriate recovery functions to be processed by non-faulty blocks within the node. Also, the servers provided within a node for services such as data storage and retrieval can be tailored to suit user performance requirements.

Figure 2.5 illustrates a refinement of the generic nodal architecture. In this case, all the processing units are replicated (A and B versions). In practice, several versions of the same functional entity would be provisioned. The integrity maintenance function is provided by ensuring that both the interface modules and the control processor modules operate in replicated mode. The switch matrix would be sufficiently provisioned to allow for sparing and appropriate sharing in case of failures within switching segments. The processing units are provisioned to allow for sufficient load sharing in providing schedules for timed executions of events affecting service provision quality.

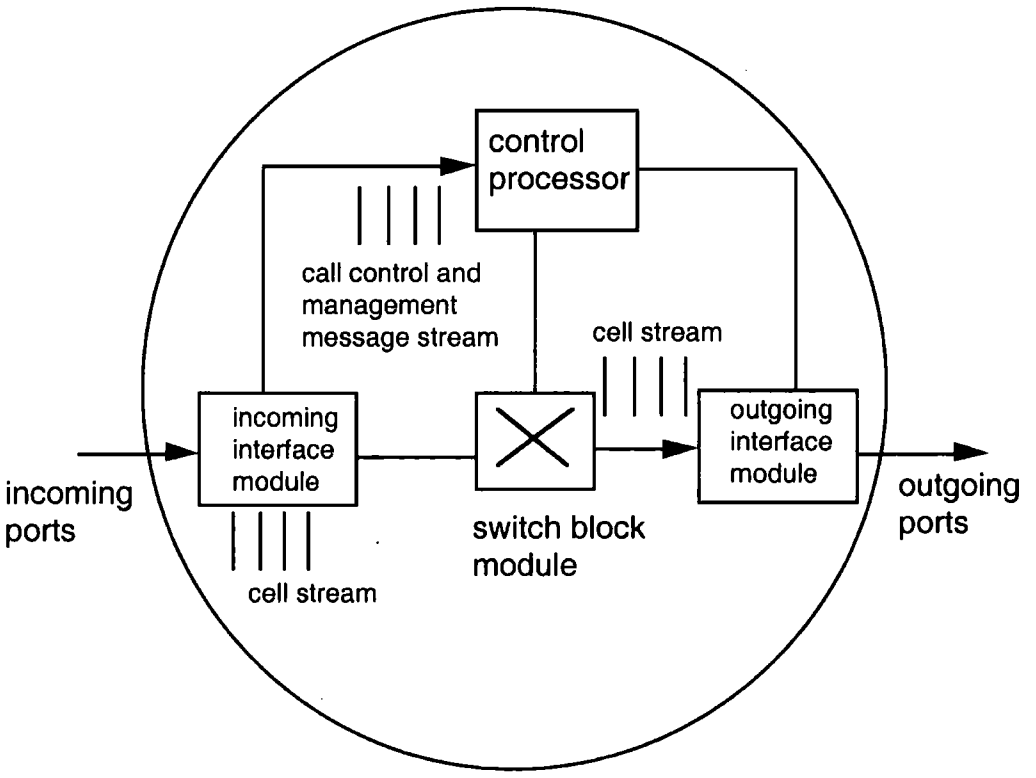


Figure 2.4 Simplified model of a switch

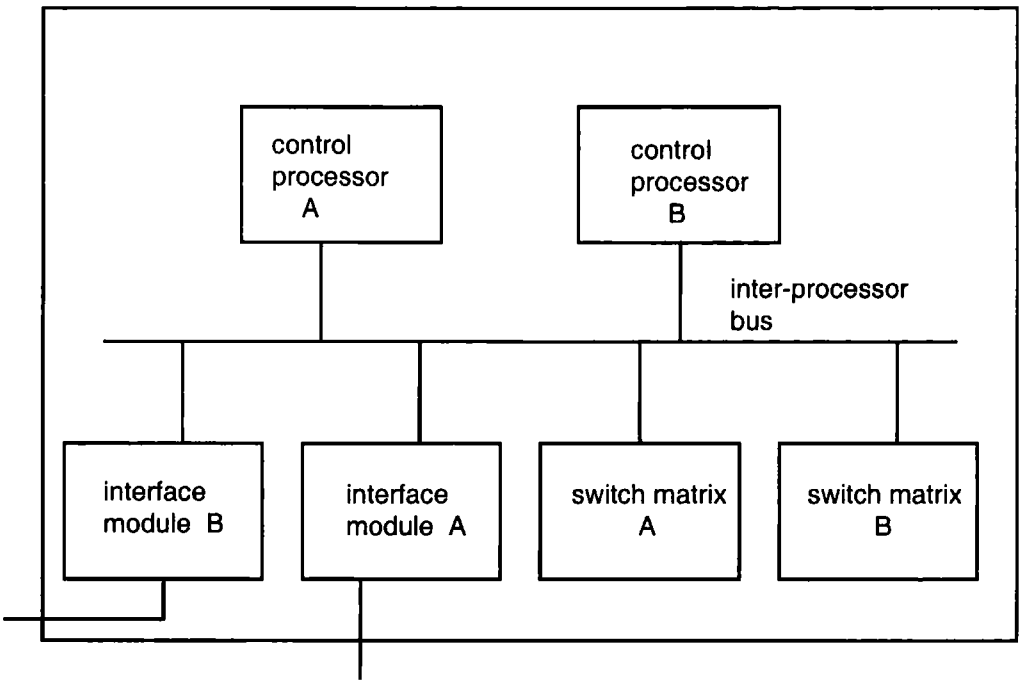


Figure 2.5
An example physical representation of processing units within a node

2.2 A structure for system functions and computing entities

2.2.1 Introduction

Specifications and implementations of system functions are usually carried out using one or more specification and programming languages. Thus specifications of a system function can be given the formal semantics of an adopted specification language. Statements about characteristic behaviours of parts of a telecommunications network can then be based on expressions about operational executions of a given system function.

There is a need to develop a structuring framework such that statements made about characteristic behaviours of a system function can be given real-world interpretations within the realm of telecommunications networks. Such a structure provides a framework for carrying out soundness checks on statements. Viewed from a different angle, it is often the case that existing specifications or programming languages are not well suited to the peculiar intricacies that need to be captured in specifying system functions. A generic informal structure needs to be developed to capture real-world characteristics of all known system functions. Such a framework can then be used as a reference model for developing formal abstract models for representation of system functions.

The specification of a collection of system functions is a formal model for these functions in the chosen formal language. Such a model encapsulates the designer's understanding of how the system will behave in executing the functions. Thus an informal real-world structure provides a framework for a formal model of system functions. Validity of statements can be made using the formal model and soundness can be checked using the informal model.

2.2.2 An overview of distributed objects

In specifying a system function, a designer imparts knowledge to a computing entity; the latter can be abstracted as an intelligent agent. Thus an intelligent agent interacts with its environment in performing computations whose results are of use to a user of the telecommunications network. Such an informal abstraction serves as an introduction to the concept of an automaton, which can be implemented as an engineering object - a computing entity encapsulating some states of a system function. Engineering objects can be distributed across locations of a telecommunications network; thus a system modelling framework can be developed based on the physical structure of such distributed objects.

2.2.2.1 The intelligent agent

An intelligent agent supports pre-programmed experiments by executing operations as a result of interpreting observed values of snapshots of its environment, and changes to relations that hold among its local representation of system states. These states are results from computations of a collection of system functions. Thus an intelligent agent executes operations within an experiment by defining another set of relations for its local data, in accordance with a pre-declared set of rules. A rule would normally have a real-world interpretation, e.g. the upper limit of resource occupancy on an inter-node link must not be exceeded.

In a recursive way, an observation by an intelligent agent could change local data that represent pre-declared sets of rules. Thus a new set of rules could be in force after some observations. Because of this, soundness and validity issues need to apply to both declared data and acceptable signatures of observations in order to ensure that executions of system functions result in desired effects. The intelligent agent therefore needs to structure its actions in such a way that when a large sequence of < observation - execution > events need to be executed to satisfy one or more entities within its environment, the agent does not generate a result that terminates the sequence purely because its rules are inconsistent. Termination should therefore only arise because a declared set of relations hold, or the environment has violated a declared contract.

2.2.2.2 A collection of intelligent agents

Figure 2.6 illustrates a single intelligent agent interacting with its environment over a fixed signature S. A signature is a set S, whose elements are called operation symbols, together with a mapping $ar : S \rightarrow N$, called the arity function, assigning each operation symbol its finite arity (or power); N is a natural number. A realization of an n-ary operation symbol in a set A is an n-ary operation on A. The notion of a data-set A of an abstract machine is described in chapter 3. A collection of intelligent agents would normally be specified to co-operate in computing system functions. Figure 2.7 illustrates four intelligent agents (T_μ, T_ω, T_α and T_β) which could be specified to co-operate over two communication environments in executing system functions.

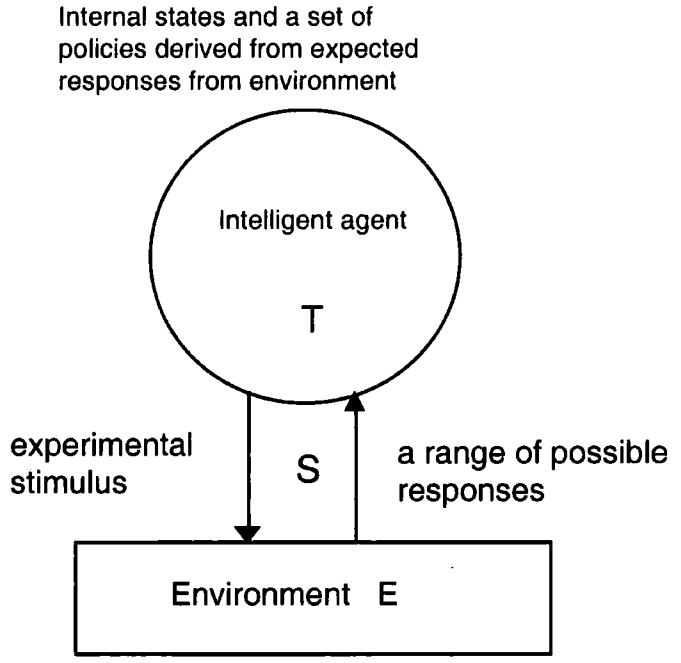


Figure 2.6 An agent making decisions in the presence of uncertain responses from its environment

On the upper left part of figure 2.7 is a refinement of the agent concept. Since each agent is a computing entity, it needs the computing resources of a scheduler, memory, timers, and such servers as required for executions of system functions. Such local computing services can be encapsulated within a concept of environment agents (T_{env}). This conceptual framework provides an abstraction for the physical environments of telecommunications systems.

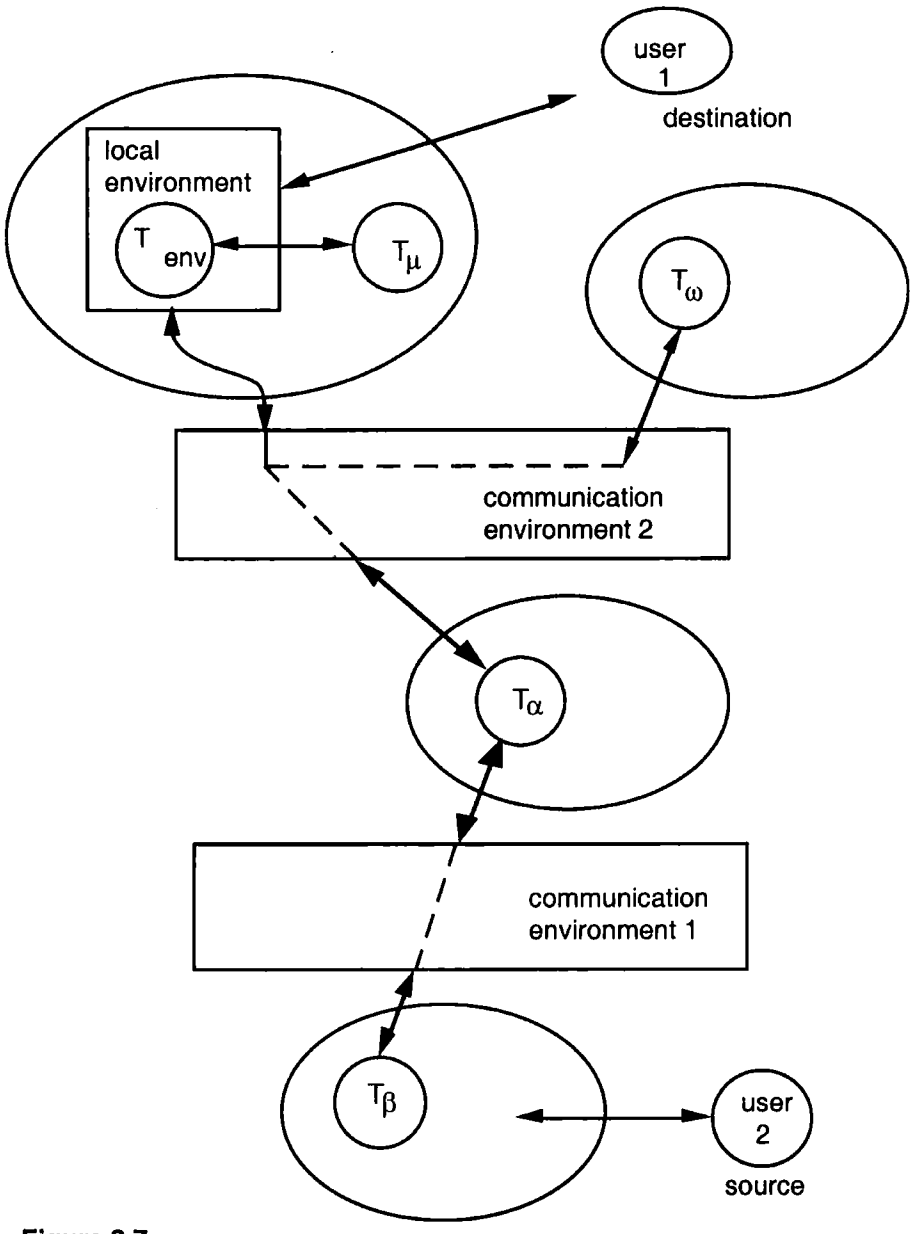


Figure 2.7
Intelligent agents providing system functions for users

2.2.2.3 A structure for components within an intelligent agent

Figure 2.8 illustrates three nodes of a network, each node hosting a single intelligent agent capable of computing system functions. In practice, each node would host a large number of 'environment' and 'functional' intelligent agents. The main components of an intelligent agent are as follows. Operations, rules and policies provide the computation script executed by an agent. It observes its environment under timed schedules to gather responses. Somehow, pre-configured data is provided within the agent (e.g. through so-called boot-strap

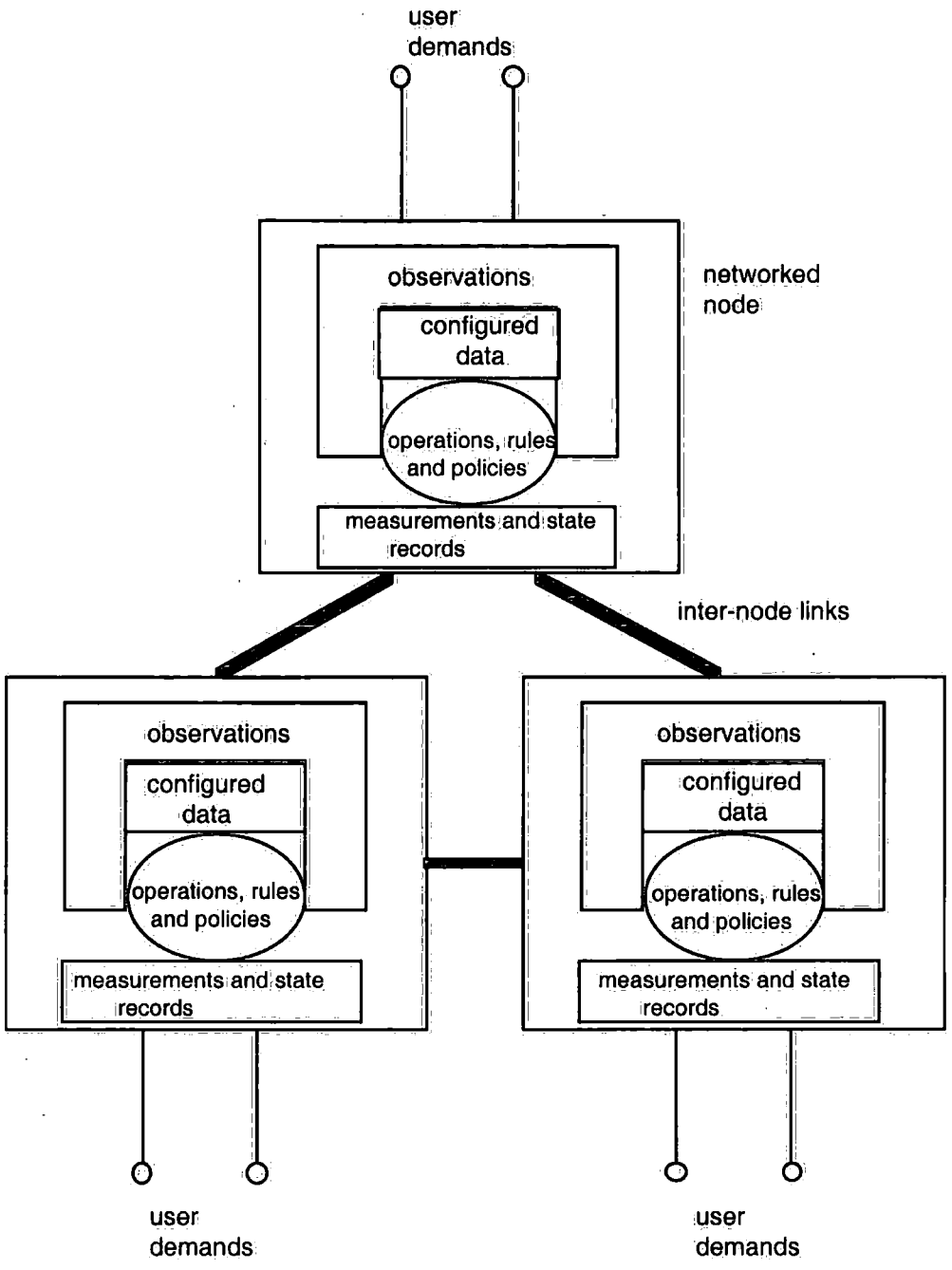


Figure 2.8 Program and data components within an intelligent agent

procedures). Once configured, further configuration data can be added or modified through computation of system functions. States of system functions are stored as measurements and state records.

2.2.2.4 Distributed objects as co-operating agents

Having provided the informal semantics of an intelligent agent, it is useful to further define the concept in order to align it with program generation and system deployment concepts. An intelligent agent would normally be specified using an appropriate programming language. When a distributed system language is available, a collection of intelligent agents can be specified to co-operate in executing one or more system functions.

A linguistic script for an intelligent agent is often described as a **computational object.** By using appropriate compilers and program transformers, a computational object can be transformed into an **engineering object** which preserves the intended behaviour of the intelligent agent. An engineering object is close in specification to the physical architecture of the system under study hence it is the level of abstraction chosen for studies in this thesis.

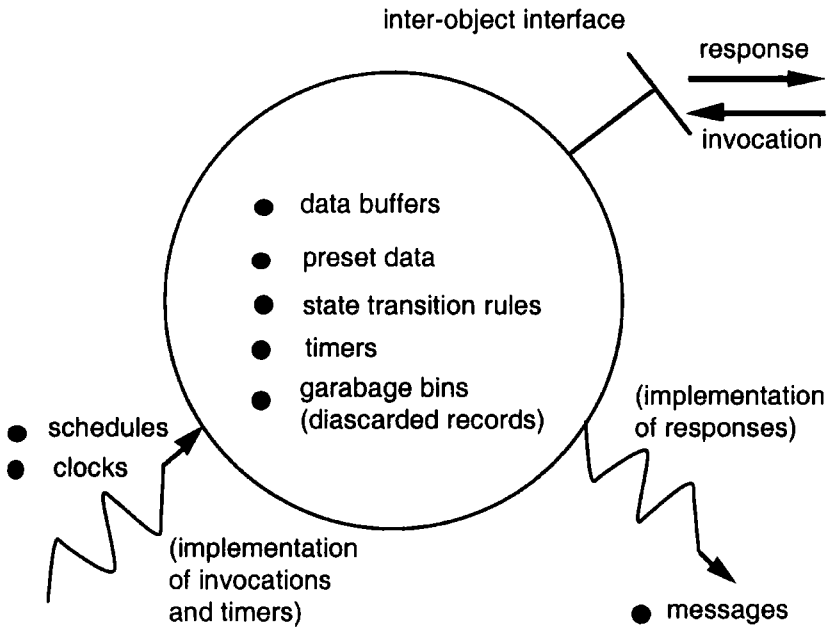


Figure 2.9 A conceptual engineering object

Figure 2.9 illustrates a conceptual engineering object. It encapsulates the components of an intelligent agent. It interacts with other engineering objects through its interface by receiving invocations and generating responses. Since it runs under the control of a scheduler, it invokes other objects indirectly by sending **messages** to its local scheduler for dispatch to the interface of a co-operating object.

Available at the local environment of an engineering object is a real-time clock. The accuracy of such a clock relative to other clocks within the system is assumed to be known. If clock

accuracy constraints do not hold, clock synchronisation system functions are specified and implemented in support of other standard system functions. An engineering object invokes another co-operating engineering object by placing an addressed message in its output port .

2.2.3 Locations and ports for distributed objects

2.2.3.1 Named locations for objects

The scheduler forms part of the local environment of an engineering object. It schedules engineering objects to execute operations and also supervises the operation of clock registers read by engineering objects. An engineering object invokes another engineering object by executing **message passing** operations to pass such invocations to its local scheduler for onward relaying.

Figure 2.10 illustrates a network which can host CPNs. At nodes 1 and 8 are illustrations of two Customer Premises Equipments (CPEs) attached to their corresponding CPNs. A name is usually reserved for the point of attachment of a node to a network. This is the **NSAP address (Network Service Access Point Address)**. Equipment attached to a network need not necessarily be switching equipment; a server database could be located at a network service access point. All locations within an attached node are named relative to the NSAP address. Thus all the names of locations within the nodes (e.g. a, b, . . . , h) are named relative to n. These locations are known as **schedule ports.**

A schedule port is a named location to which an object's message is sent in order that the object sees the message as an invocation when it is scheduled to run.

The frequency of attaching an object to a schedule port is left open according as the system function being implemented. The duration of each scheduled execution by an object is also left open.

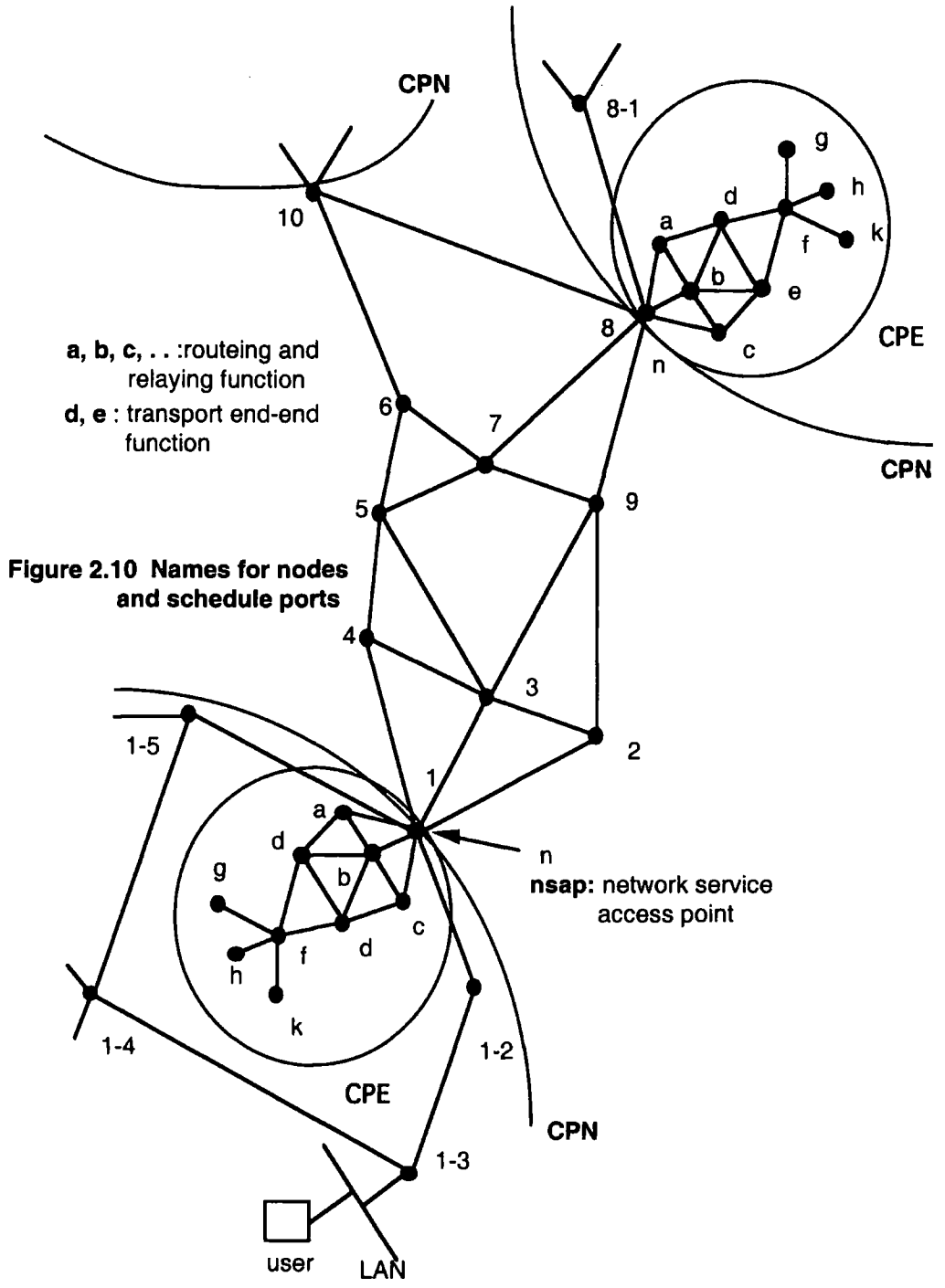


Figure 2.10 Names for nodes and schedule ports

Names of schedule ports and points of attachment are unique with respect to other such locations within a sub-network. However, relative naming and name interpretation within domains of message passing allow for non-unique sub-addresses. Engineering objects can be moved from one location to another. Hence, an address of an object's interface has two components: an object's unique identity <time of birth, location of birth>, and an object's location trail (implemented as a re-direction system function).

2.2.3.2 Clustering objects

It is useful to gather together a collection of objects which perform similar system functions as a **cluster** of objects. Such an approach allows for economy in description of a large number of system functions. Since addressing for engineering objects is relative to the NSAP address, no invocation is ever lost in such clustering.

Figure 2.11 illustrates an example of layer clustering for a three node network. It should be noticed that such clustering does not preclude full meshed inter-communications among clusters. Thus maximum inter-communication flexibility can be achieved.

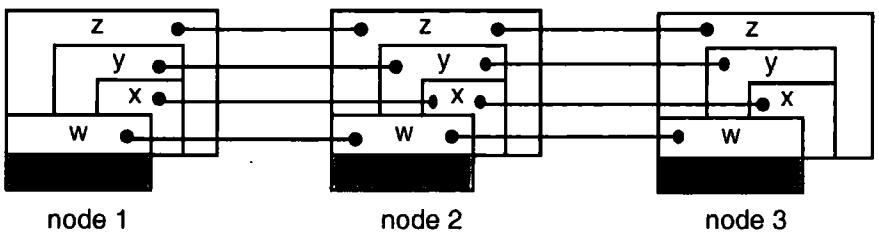


Figure 2.11 Layered functional entities

2.3 Overview of model representations for logical validation and performance characterisation

2.3.1 Algorithm framework for computations

Input-output signatures of objects can be given the model structure of automata operating over system behaviour states. Thus the behaviour of a collection of objects executing a collection of system functions can be declared as follows, using a logical model (e.g. Doets, K [1994]).

The notion of a model is useful in representations of rule based specifications. The fundamental rule based system is the notion of a language:

$$L \quad = \quad R \cup F \cup C \quad ; \qquad \text{this system being interpreted as follows:}$$

An L- model is the system

$$X \quad = \quad (A, \ldots, r, \ldots, f, \ldots, c, \ldots)_{v \in R, w \in F, y \in C}$$

Where A is a non-empty set, the universe of X.

X has the following:

i) an n-ary relation $r \subset A^n$ over A corresponding to each n-ary relation symbol $v \in R$,
ii) an n-ary function $f: A^n \rightarrow A$ for each n-ary function symbol $w \in F$,
iii) an element (constant) $c \in A$ for each constant symbol $y \in C$.

An object say ♦ is called an interpretation or the meaning of a non-logical symbol say δ in the model X. Thus it is assumed that r is the interpretation of v, f the interpretation of w, etc. Also, it is assumed that A is the universe of the model X, etc.

In the context of a model X, terms stand for certain elements of A, and rules or formulas stand for certain statements about X, when there are no free (i.e. symbols unaccounted for in the definition of *L*) variables. In the presence of variables, *L* is expanded by introducing more constants for all elements of the universe, and temporarily replacing variables by these constants. This system is clearly first-order logic which can be constrained as necessary, to achieve decidability.

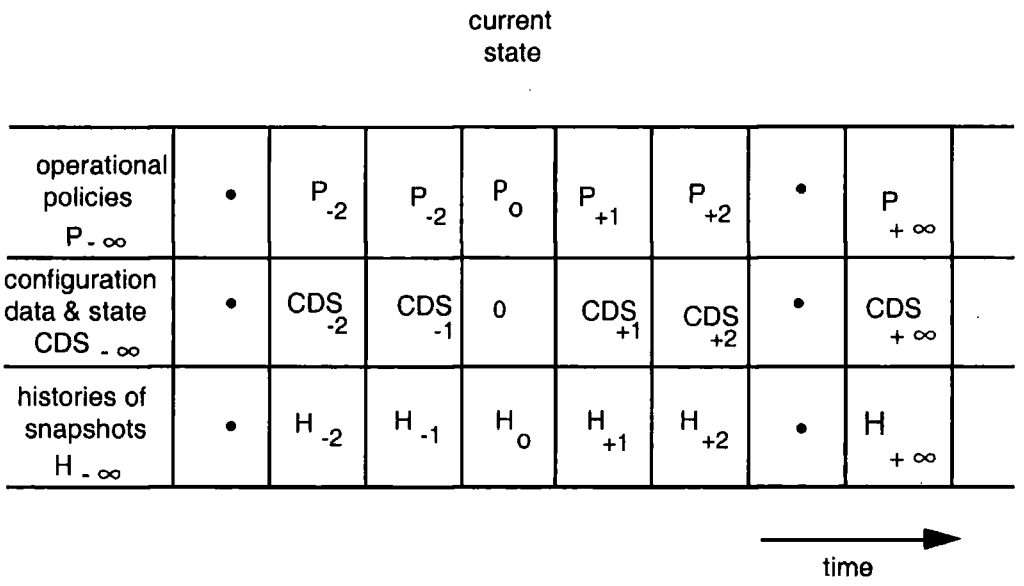


Figure 2.12 Timed snapshots of state relations, policies and events

Figure 2.12 illustrates an informal structure of a **snapshot universe** within an object executing a collection of system functions. Such a structure can be specified within the language *L*, to represent the behaviour of the object as an automaton.

Recall that for automata (see Salomaa A. [1985(a)]), an **alphabet** is a finite, non-empty set Σ. The elements of an alphabet are referred to as **symbols** or **letters**. A **word** over an alphabet is a finite string consisting of zero or more letters of Σ, in which the same letter may occur several times.

The declarative representation of the snapshot universe is basically a timed attribute of a rewriting system

$$RW = (\Sigma, P)$$

where Σ is an alphabet and P is a finite set of ordered pairs of words over Σ. The elements (u,w) of P are referred to as **rewriting rules** or **productions**, denoted by:

$$u \rightarrow w$$

When the sources of uncertainty of a telecommunications system respect agreed input-output signatures of engineering objects, the rewriting system can be animated to generate uniquely terminating productions thereby being **canonical** (Jantzen M. [1988] provides a detailed description of canonical rewriting).

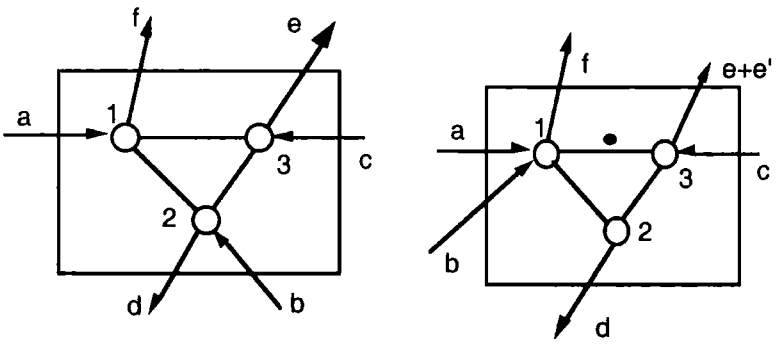
2.3.2 Controlling the flow of information streams

2.3.2.1 Modelling for uncertain resource demands

A major challenge in the specification of system functions is the need to specify, in a declarative way, algorithms and configuration data for a telecommunications network whose user demand patterns are constantly changing. This is because it is often difficult to re-provision the resources within a network in real time to satisfy a changed pattern of user demands. Thus in order to specify configuration data for use within objects, there is a need to predict a range of source-destination user demand models for connection and information stream patterns.

It is not possible to specify a comprehensive user demand model in a form that is independent of network topology, since changes in demand patterns across a sub-network can only be described in terms of network topology. Figure 2.13 (i) illustrates a simple three node network with offered traffic < a, b, c, > being carried as < d, e, f >.

Figure 2.13 (ii) illustrates a re-arranged user demand model with source stream 'b' moved from node 2 to node 1. Depending on the provisioning of link and nodal resources within the network, the source stream could be discarded, or carried (e.g. the stream fraction e'). This re-arrangement could affect consumed resource levels at all the nodes and all the links.



(i) source-destination user demands

(ii) re-arrangement of user demands

Figure 2.13 (i, ii) Traffic flows within network

Within the network nodes, a specification is provided for operations that can be executed by engineering objects to operate on input requests and configured data, and subsequently generate resource utilisation statistics. The results produced by a system function being executed can generate data which enables the invocation of appropriate resource control mechanisms e.g. congestion avoidance sub-functions. The system behaviour is often validated in the laboratory as follows.

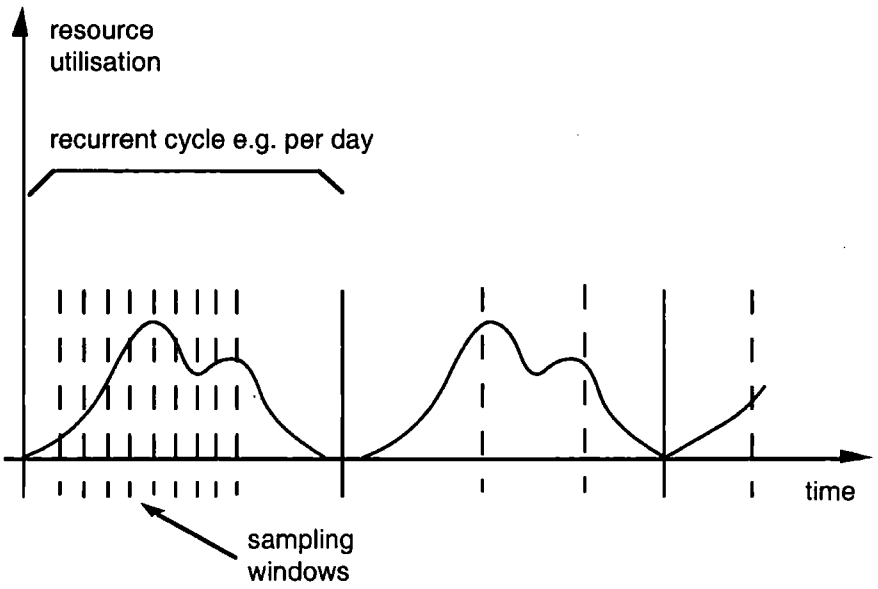


Figure 2.13
(iii) Recurrent patterns of resource utilisation

Given a network scenario, generate a set of probable utilisation vectors and other network-wide behaviour statistics, stating how all the probable behaviour types can be attributed to a set of effective user demand models and the control procedures in force. This execution scenario can be animated within a simulator. It should be noted that user-demands and allocation patterns

are often recurrent in nature (see e.g. figure 2.13 (iii)).

Figure 2.14 illustrates how the simulated or measured level of resource consumption within a link can vary over a recurrent time period. When a trend lies in regions 'b' or 'c', appropriate policies need to be enforced to reduce the level of consumption. This could be achieved where possible, by re-arranging traffic flow patterns to other parts of the network where there is low utilisation (i.e. invoking policies that effect 'a'). This is the geometric basis of what is usually referred to as dynamic routeing. The approach adopted in this study therefore subsumes such problems.

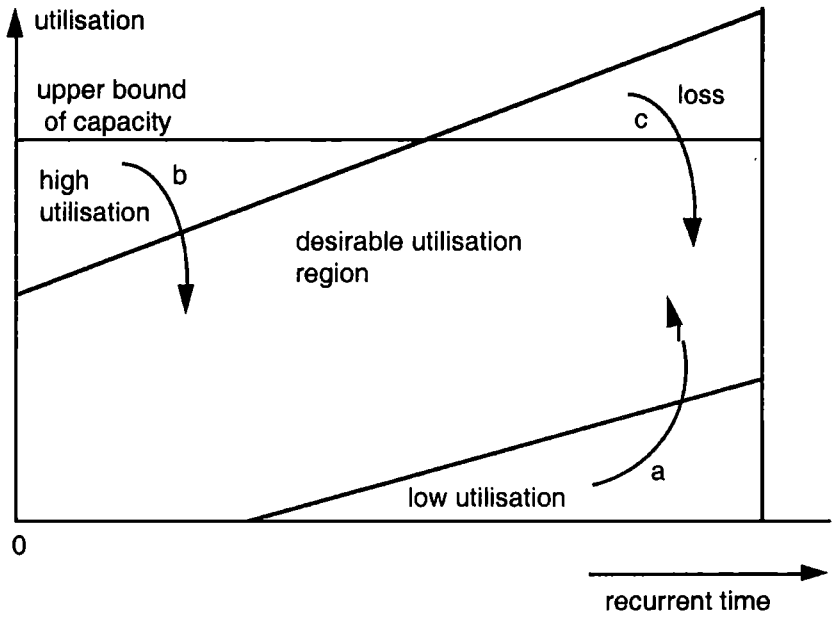


Figure 2.14 Policy selection for system behaviour constraints

2.3.2.2 Multiplexing and de-multiplexing of streams

The approach to model representation adopted in this study is the conditional re-writing operational semantics in which the specification of a system reflects beliefs held by the specifier of the system with respect to probable system behaviour. With such an approach, actual observed system behaviour provides the information required by pre-declared rules to improve the system performance. As an example, consider the following simplified queuing operations.

Figure 2.15 illustrates multiplexing and de-multiplexing functions for information streams X, Y and Z. Each stream can be considered to represent non-deterministic arrival of fixed size (in bytes) packets of information. Chains of multiplexing and de-multiplexing engines would normally be connected together to provide the transmission capability of a sub-network.

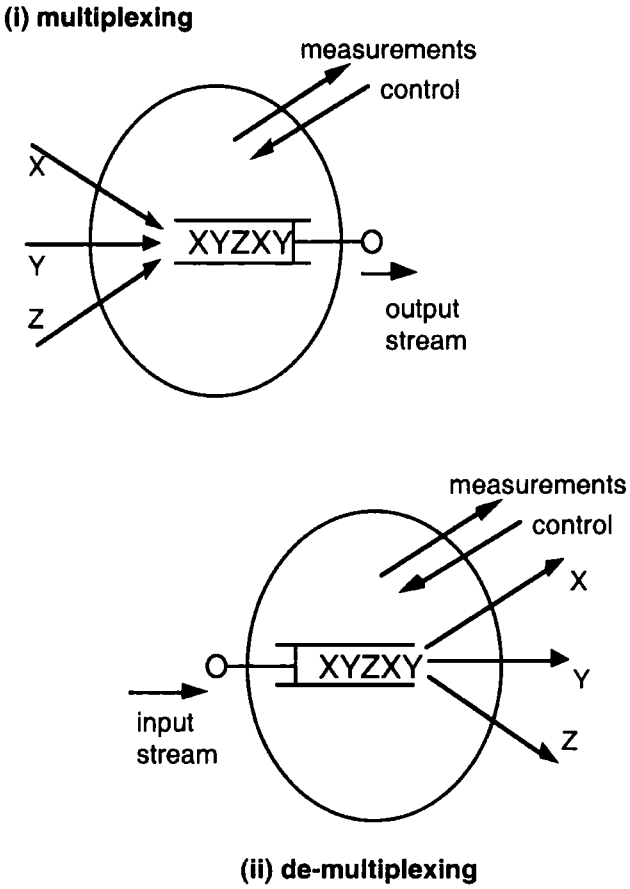


Figure 2.15
Multiplexing and de-multiplexing of information streams

When the statistical characteristics of each stream is known, the store and forward policy of the processing unit at the multiplexer and de-multiplexer can be designed to combine and separate the streams in transporting the byte streams from source to destination. Such computations at a multiplexer or de-multiplexer invariably disrupt the flow of each stream passing through the equipment. Thus in admitting a stream into a sub-network, the admission policy in operation must ensure that the stream's flow characteristics (quality) is being maintained at the originating node, and also at all subsequent nodes along the stream's path.

Since the exact pattern of an arrival composite stream may not be known, the level of disruption of output streams reflect the level of uncertainty of each stream's behaviour with

respect to the operational stream processing capability. Measurements of packet arrival patterns and output patterns provide information on the effectiveness of an invoked stream service policy. Based on analysis of the measurement information gathered during a time horizon, a policy change can be effected to operate during a future time window.

Research into queuing models and delay variations of information streams have attracted a lot of interest over the years (see e.g. Gelenbe et.al. [1987] and Saito H. [1993]). However, the resource allocation and optimisation issues that need to be addressed due to the arrival uncertainty within a given stream are often of less practical significance than those that arise due to the make and break patterns of information streams across a network.

2.3.3 Basic issues on complexity measures of representations

2.3.3.1 Basic issues on distributed system complexity

The algorithm framework for computations based on canonical rewriting provides a very powerful methodology for specification of system functions. A complete validation of a collection of system functions can only be based on practical experiments involving real equipment and real users attached to the networking system under study, exercising system functions in an intended normal way. This approach is often not acceptable due to the enormous cost of developing a system that may not behave as expected. The complexity characteristics of specifications of a set of system functions is therefore a very important measure of a networking system's behaviour at the conceptual and implementation stages of a system's design.

In the specification of a system that exhibits desirable behaviour characteristics, operational rules generate terminating sequences of operations within acceptable elapsed times, and the system consumes acceptable levels of resources. In assessing the complexity of an algorithm, the following basic questions can be posed (see e.g. Salomaa [1985(b)]).

a) given two problems P and P_1, is problem P more difficult than P_1 in the sense that every algorithm put forward for solving P is more complex than some specific reasonable algorithm for solving P_1?

b) given a problem and two algorithms A_1 and A_2, is algorithm A_1 better than algorithm A_2 in the sense that it uses fewer resources, such as time or memory?

These questions can be posed in assessing the complexity of a standard classical single processor algorithm e.g. numerical solution of a differential equation, or large scale relational algorithms such as the ones outlined in this thesis. Classical complexity theory is normally addressed at three levels of refinement:

i. axiomatic complexity
ii. machine oriented complexity
iii. low level algorithmic complexity

Axiomatic complexity addresses the application of partial recursive function theory to the description of computations which consume abstract resources. With this approach, computation problems are specified in terms of the more fundamental structure of natural number sequences and inductive definitions of system behaviours over these sequences. When a solution defined in this way exists for such a problem, the steps executed to arrive at a solution in itself defines a general complexity measure, and is thus a measure which can be applied to any problem.

At the machine oriented level of refinement, there is a need to define a generalised abstract model for computation and resources in time-space measures. Classical machine oriented complexity is developed in terms of register machines and input-output signatures of computations. In treating a networking system as an abstract machine, there is a need to define a generic model that applies to computations carried out by distributed engineering objects, the latter executing system functions. Thus there is a need to explore how the fundamental axiomatic complexity measures can be refined to networked systems which can be considered to be more expressive than register machines, in the context of system functions (see Nyong O.D.O. [1995], Abiteboul[1997]).

In the low level refinement of complexity, specific problems and their solutions are characterised. In classical complexity theory, given a problem, a question can be posed as to whether one algorithm is better than another, with respect to time and memory/space measures. In addressing the implementation of telecommunications system functions, complexity measures for resources is more complicated since there are several levels of resources ranging from processor schedule occupancy to the inter-node transmission link occupancy. Nodal equipment behaviour characteristics, the structure of networked objects, and the efficiency of operational rules all affect the values of complexity measures obtained. The first step to be carried out however, is a coherent definition of such measures. This thesis presents how system functions can be represented to satisfy resource based policies.

2.4 Summary

When posing a research problem, it is important to ensure that the specific technique being proposed to solve the problem has not distorted the problem just to satisfy the solution technique. Also, it is important to ensure that a posed problem is significant when viewed in a holistic universe where other related problems exist. Thus questions as to whether a research project addresses a solved problem, an unsolved problem, or a non-problem can be asked based on a realistic framework.

This chapter has presented a holistic description of the generic components and structures that comprise telecommunications system resources and associated system functions. These functions are large scale and inter-related so they pose challenging representation problems. The basic language used for the representation is elementary set theory, since this language does not favour any particular solution technique. The main challenge in the representation of large scale system functions is how to generate models that can be used for characterization of system behaviours and capture sufficient details such that both individual user goals and population goals of network users can be satisfied.

An important point that has emerged from the holistic description presented in this chapter is the notion of geometric regions of system behaviour. This structural aspect of a system representation's characteristic behaviour could well be the handle into ways of modelling in the large. In the next chapter, these challenges are stated explicitly in terms of a resource management algorithm framework. A critical survey of existing modelling techniques is also carried out in the next chapter. The main representation issue being investigated is then stated.

Chapter three

Overview and critical survey of modelling semantics for distributed algorithms

3.1 Introduction

This chapter provides an overview and assessment of some of the key modelling semantics for system functions. The approach adopted is to base the survey on a concrete example: resource allocation through sharing within a telecommunications network. Optimization issues are developed from first principles and primitive concepts (such as permutations on system state spaces, and predictions as declarations of system configuration data) are isolated. A reference suite of distributed algorithms is sketched as a basis for defining the scope of existing semantics.

3.1.1 Motivations, challenges and problem statement

Representation theories and models (modelling concepts) are required for carrying out realistic performance characterization of realizations of system functions. Existing models have yet to be applied satisfactorily in the modelling of practical large scale networking problems. Two impediments to the successful development of modelling concepts have been isolated in this thesis as follows.

a) System functions tend to be inter-locked regarding the data they operate on and the interconnections among objects that host the functions. Figure 3.1 illustrates an icon denoting the top level classification of resource management functions wedged between resource users and the shared resources. The resource access arbiters need to know about the current state of consumption of provisioned resources in order to invoke the necessary arbitration algorithms that select which users are allowed to consume free resources. The resource usage optimizers need to know how to partition system-wide resources efficiently in anticipation of resource requests by users. Since these functions are executed concurrently, their execution sequences form products of state sets which exhibit challenging representation characteristics.

b) System functions need to be sufficiently detailed to satisfy both individual users and populations of users. Figure 3.2 illustrates how an individual user and a population of user attributes relate. A population's behaviours (i.e. resource consumption actions) can only be obtained by the computations that resolve behaviours of individual users. The same resolution requirement on behaviours apply to the computation to establish a population's goals. Policy

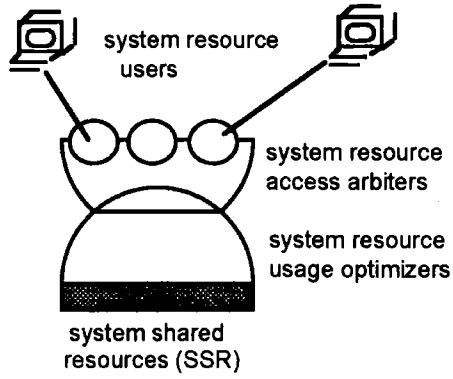


Figure 3.1 Structuring of system functions

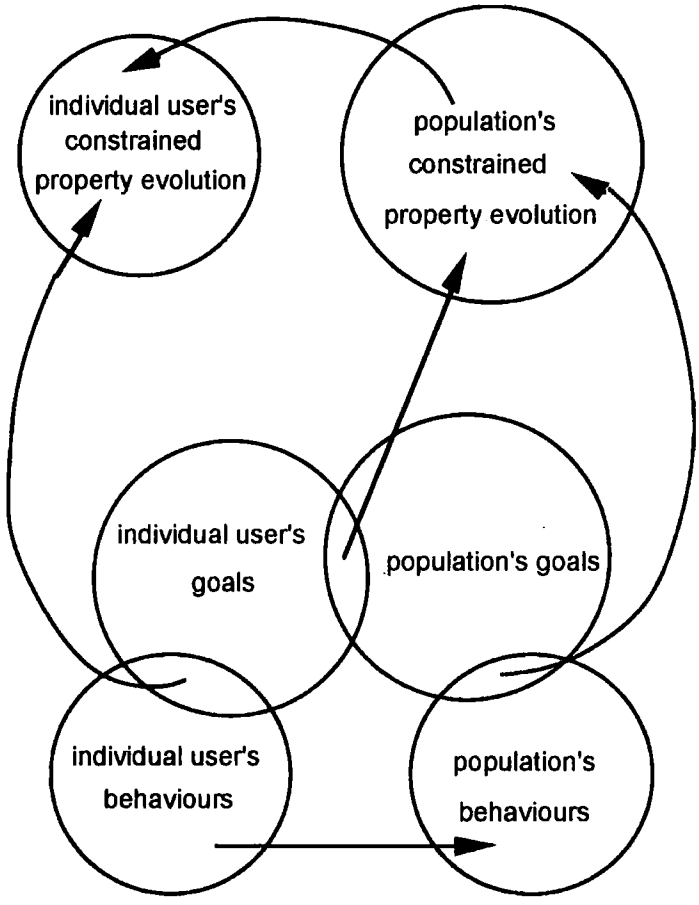


Figure 3.2 Declarative datasets and operations

constraints on a single individual must be specified to satisfy an overall system behaviour constraint. The specifications of these computations is therefore quite challenging.

Over the past twenty years, researchers into the representation formalisms for modelling

systems have been aware of the scalability problem of representing real world problems. Varaiya P. and Kumar P.K. [1986] raise this issue in the introductory chapter of their book. However, the option often adopted by researchers is to treat a proposed solution technique as a purely mathematics problem, postponing the mapping of such mathematics to real world problems. The approach adopted in this thesis is to focus on the geometric representation of system states so that the resulting patterns capture as expressively as possible both the input data and the results computed by system functions. This approach being declarative, is innovative in that specifications of computations that accept non-deterministic inputs can be re-adjusted as many times as is necessary in order to ensure that essential real world features of a problem are captured. Being a goal oriented approach, a good choice of representation theory is the notion of composition of collections of system property evolution: the calculus of relations.

The research challenge associated with devising representation theories for calculus of relations and geometric logic is well recorded in the results of the eighteen year work by Freyd P.J. and Scedrov A.[1990]. The justification for adopting this difficult approach in an attempt at representing realistic system functions is based on the success achieved so far by researchers in database theory who have represented intricate transactions using relational calculus.

3.1.2 Structure of the chapter

Section 3.2 presents the components of a system of resource allocation algorithms in order to emphasize the large scale nature of the specifications for realistic system functions. The partitioning formulation of the routeing problem is original. Section 3.3 presents a semantical unit used for classification of key constituents that ought to be present within an expressive modelling procedure. This structure is original. In this section is also presented an experimentation architecture which is used for carrying out a critical evaluation of three modelling techniques.

3.2 A resource allocation paradigm for specification of system functions

3.2.1 Introduction

This section illustrates how a large collection of inter-related distributed algorithms can be considered as a suite in order to specify the functions to be supported by a specific component within the suite. An important collection of distributed algorithms is normally provisioned within large scale telecommunications networks to carry out the allocation of network transmission resources to user demands, maintaining some adaptive **routeing pattern.** Here, routeing pattern defines the following: paths over a network topology during a specified time window, provisioned transmission resources along the paths, and offered source-destination traffic intensity (time varying information transmission rates).

Two key issues that affect the specification and design of a routeing pattern are:

- modelling for uncertain user demands
- multiplexing and de-multiplexing capabilities of networking switching nodes.

Practical networks often comprise a diverse collection of multiplexing nodes, provisioned link transmission capacities, and time-varying user demand models. It is possible to specify a set of rules that must hold in generating adaptive routeing patterns for efficient allocation of resources within a network. The specification of a collection of algorithms to satisfy such rules is a major challenge. However, an informal description of such a suite of algorithms can be presented to serve as a framework for reviewing existing modelling semantics.

3.2.2 The universe of resource pools and policy based resource allocation

The aim of this section is to introduce the notion of **policy based mappings** which bind resource pools to user demands for network-wide resources. The fundamental criterion for such mappings is the need to optimise usage of shared resources by choosing an appropriate allocation policy at any given time interval. In practice, such optimization policies are often implemented as resource partitioning and reservation procedures. These procedures can then be interpreted as resource access priorities; such priority schemes effect various levels of discrimination against various user sets.

Figure 3.3 illustrates a sub-network, a network management centre (NMC), and an example collection of routes (<A-B>, <E-F>, <J-K>, and <M-N>). In the case of the ATM networking technology, combinations of node pairs are provisioned with inter-node transmission resources packaged as virtual path connections (VPCs). For simplicity of exposition in this study, a VPC spans only two nodes. Users are allocated transmission capacities from sub-channels of VPCs as virtual channel connections (VCCs). Thus VCCs originating from a sources node are multiplexed into VPC traffic by multiplexers within source and transit nodes, en-route to destination nodes where the composite streams are de-multiplexed into individual VCCs received by user equipment.

In figure 3.3, each route comprises a collection of VPCs; a VPC does not necessarily span a route from a source node to a destination node. Thus VPCs can be concatenated within a route. Each VPC is provisioned with transmission resources for use by a collection of routes. Thus VPCs provide pooled transmission resources. Since routes share inter-node multiplexing and transmission resources, these resources can be envisaged as being dynamically allocated to routes by distributed resource allocation functions. These concepts apply to networking systems using technologies other than ATM.

3.2.3 Resource usage optimization within a single time window

In provisioning transmission resources illustrated in figure 3.3, it is assumed that the network designer can make assumptions about the structure of source-destination traffic concurrently offered by pairs of nodes. In practice, the actual values of such statistical variables are rarely known. This unknown is the main source of uncertainty in the provisioning of resources within large networks.

The importance of the need to capture statistical behaviours of non-deterministic users has prompted international standardisation effort on user demand modelling (see references ITU E.711 [1992] and ITU E.716 [1995]). The user demand modelling structure developed by the ITU is based on the following entities:

i) **call pattern** - the events of a call demand at a user-network interface. The call pattern encapsulates call **traffic variables** - the traffic characteristics of a call's events,

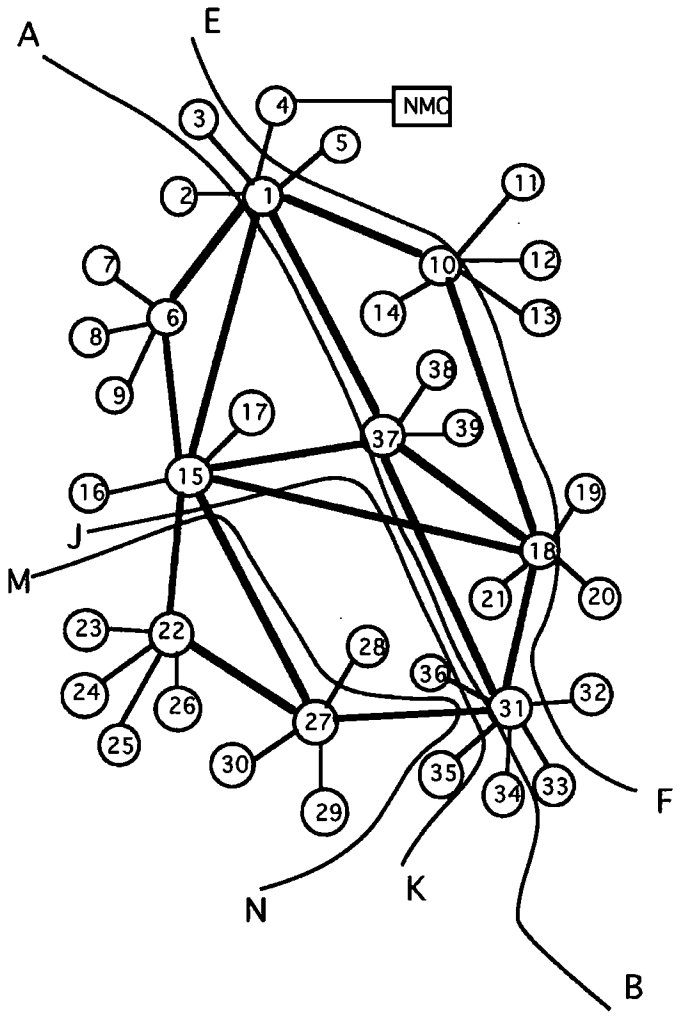


Figure 3.3 A routeing plan within a sub-network

ii) **call attempt arrival process** - call events' characteristics describing statistical behaviours of a population of users located at various nodes on a network.

Traffic variables consist of the following:

(a) transaction arrival process - information transmission event characteristics of transactions that occur within any call,
(b) transaction length - load offered by a transaction over the user-network interface,
(c) call set up and clear down signalling (time constrained invocations).

The call pattern of a call demand is defined in terms of sequences of events at the user-network interface and the times between events. The call pattern at an interface is defined by a set of traffic variables expressed in statistical terms. The statistical expressions for traffic variables contain parameters which characterise probability distributions involving the traffic variables.

The call attempt arrival process specifies the distributions for mean number of re-attempts in case of non-completion, and mean time between call attempts.

A network designer can thus construct a collection of possible sets of user demand patterns that can hold concurrently across a sub network. The network designer expects that one member of the collection of demand patterns holds during a given time window. It is however likely that the network designer's choice could be incorrect. Such an inaccurate choice can only be detected after a real network has been running for a period of time. A good measure of an appropriate duration of elapsed time is a satisfactory number of recurrent observation time windows measured in terms of pattern prediction accuracy.

Figure 3.4 illustrates how a decision engine accepts a finite collection of possible sets of user demand patterns. The decision engine is spread across the network. In executing a resource

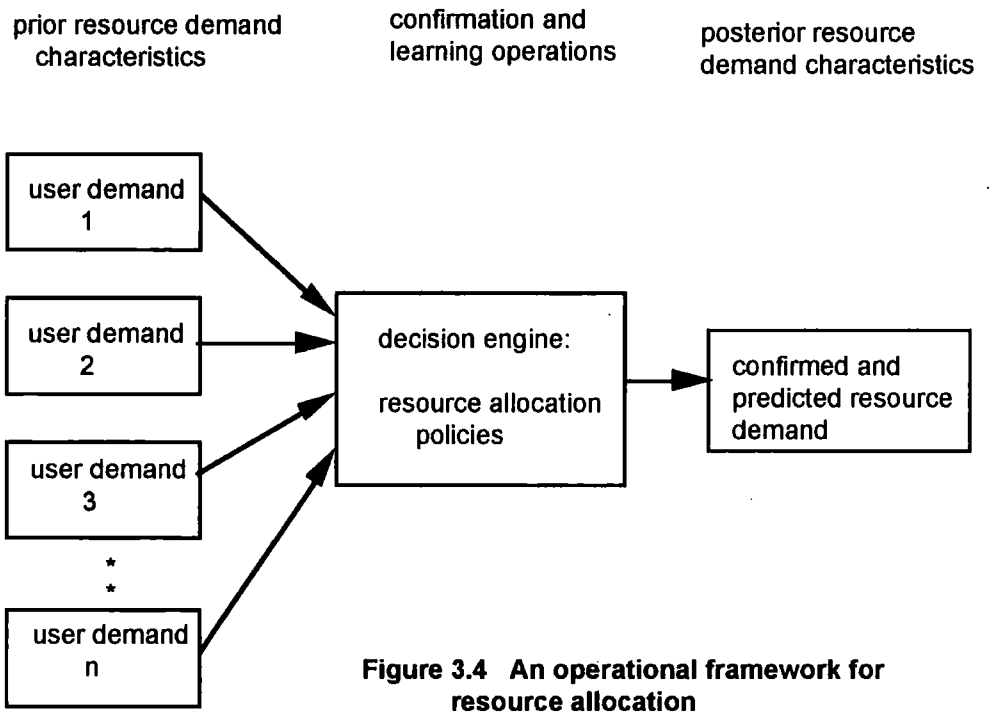


Figure 3.4 An operational framework for resource allocation

allocation policy based on a selected set of user demand patterns, the distributed decision engine can attempt to confirm that an operational choice is accurate, or predict (i.e. identify) the correct pattern when the selected set is incorrect. Structures of components within the user demand model can be specified and collected together in representing these characteristic user demand patterns. These patterns and associated information elements can then be used as input data to specify rules for optimal allocation of resources as follows.

T_{ij} is a set of all entries of offered source - destination traffic, each entry denoted by the variable λ_{ij}. R_{ij} is a set of all entries of carried traffic (utilisation of transmission resources), each entry denoted by the variable r_{ij}.

P_{ij} is a set of all possible spans of operational routes over a given network topology, each entry denoted by ϕ_{ij}.

Q_{ij} is a subset of entries in P_{ij}, entries in Q_{ij} are denoted by the variable q_{ij}. Q_{ij} is a preferred subset of P_{ij}, to carry traffic T_{ij}.

$\mu_{ij}[\lambda]_n$ is a finite set of whole and fractions of each instance of λ_{ij} ; n is a natural number denoting the number of unique members in the set.

A variable μ_{ijx} denotes a specific element in $\mu_{ij}[\lambda]_n$ that can be selected to be carried as r_{ij}.

Thus the maximum carried traffic is the summation for all ij:

$\sum r_{ij}$ less than or equal to the sum of all unique λ_{ij} in T_{ij}.

Structure values of μ_{ijx} to be less than or equal to each corresponding λ_{ij} and then select the μ_{ijx} 's that are as large as possible, at the same time meeting other system criteria such as the fair spread of carried traffic across the network. The r_{ij}'s can thus be defined using the variable μ_{ijx} to denote carried traffic across the source destination pairs.

Given constraints on provisioned resources, select members of Q_{ij} that satisfy a selected set of large values of μ_{ijx}, but have small values of q_{ij} in order that $\sum_{all_{ij}} (q_{ij} \times r_{ij})$ is smallest, to minimise occupancy costs of transmission circuit.

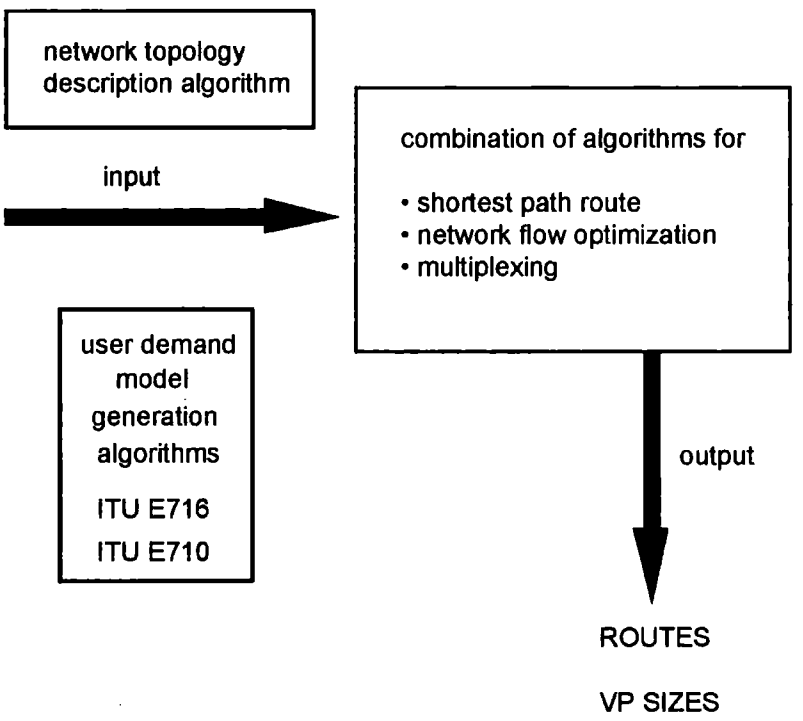


Figure 3.5 Single time window resource allocation

Figure 3.5 illustrates how these criteria can be incorporated into an algorithm that computes routes and determines resource capacities of pooled virtual path connections. These concepts are developed further in the case study on congestion avoidance described in subsequent chapters.

3.2.3.1 Permutations on state variables

In forming the set of whole-part fractions of network-wide λ_{ij}'s, logic based rules can be developed for optimal selection of carried traffic. Thus the theories of permutations and combinatorics on finite sets can be exploited in the specification of such algorithms. Clearly, the number of permutations on a set increases as the unique cardinality of the set increases. A challenge faced by a designer of distributed algorithms is the generation of constraining rules that reduce the number of admissible partitions on a system's state set.

The theory of permutations on a finite set is a well developed mathematical topic (see e.g. Blyth T.S. et al. [1986]). Logical procedures for handling permutations in equational theories are also well developed (see Burckert H.J. et al. [1990]). Since the generation of sets of state variables can always be arrived at as permutative functions, this procedure is assumed to be primitive in the specification of distributed algorithms. Further elaboration of this point will be provided throughout this thesis.

3.2.3.2 Constraints on state space

The size of a system state space being considered in a modelling experiment is directly related to the experimenter's knowledge and perception of the scenario under study. In effect, the experimenter generates user demand models to carry out experiments involving concurrent consumption of pooled resources by a fixed population of users. The state space can be significantly reduced by treating the resource demands from sub-populations of the whole user population as individual resource demands. In such cases, care is taken to ensure that network topology constraints and concurrency of resource access are not violated.

In cases where user demands are appropriately partitioned, control of allocation of pooled resources to users can then be based on opportunity costs of giving resources to sets of sub-populations relative to giving resources to other competing sets of sub-populations.

3.2.4 Prediction as a primitive concept within distributed algorithms

Each assumed (i.e. predicted) user demand model is implemented and made operational during a time window which allows sufficient statistical data to be gathered for carrying out any necessary revision to a previous prediction. Techniques for the measurement and recording of traffic in large telecommunications networks have been actively researched and standardised by working groups within the International Telecommunication Union (see ITU E.500 [1992]). Two main types of measurement are required for prediction of telecommunication traffic:

- measurement of the number of bids
- measurement of the amount of traffic carried

Bids measurements count entities denoting events, e.g. calls accepted during a certain period of time. Carried traffic measurements record averages of values of source-destination traffic over various periods of time. The unit of traffic measurement reflects in some way the occupancy characteristics of the network resource concerned.

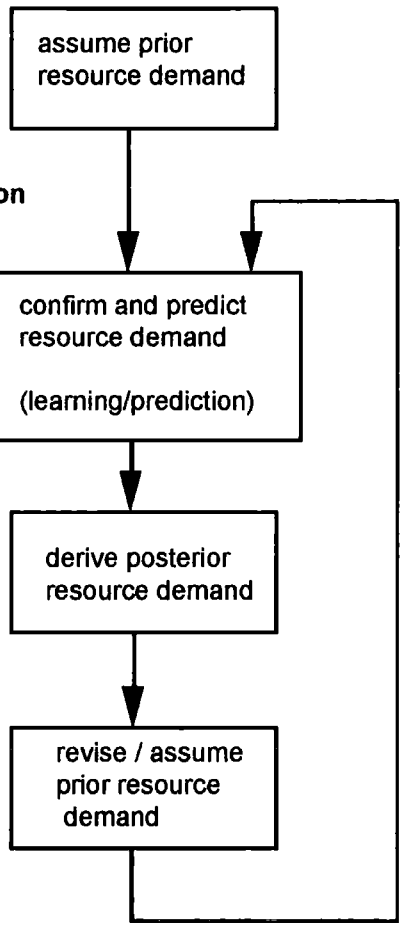


Figure 3.6 A resource allocation policy cycle

Quite often, statistical analysis procedures need to be carried out on the measured data. These involve the analysis of stability of traffic profiles and the computation of the variation factor for mean loads. Stability of traffic profiles over recurrent periods provide some guidance as to the suitability of a sampling granularity. Variation factors for mean loads provide information on changes, over recurrent periods, of parameters representing statistical distributions denoting loads. These issues are presented in detail in the ITU Recommendation E.500.

Given a prior set of user demand models, figure 3.6 illustrates how a suite of algorithms can be organised to confirm that the prior set of user demand models still holds after a given period of network operation. Concurrently, these algorithms are specified and configured to identify an alternative set of user demand models in cases when the currently identified user demand models are different from the selected prior. In some cases, the algorithms can fail to lock on to a given set of user demand models; in such situations, the closest set of user demand models is selected and an exception raised to the network management centre.

The foundations for the statistical computations required within the algorithms are well developed. However, the structuring and realization of such statistical concepts within operational distributed algorithms is challenging, and in its infancy. The theory of Bayesian estimation required for predicting sets of user demand models is well presented in Hogg R.V. and Craig A.T. [1978]. The use of Bayesian estimation for setting up sequential prediction experiments is presented in O'Hogan, A. [1994]. A detailed description of these concepts would be quite lengthy; a summary is provided as follows.

(i) Measurements of the user demand model and carried traffic parameters can be recorded over recurrent time periods (e.g. busy hours of each working day of a month) and used to generate statistical trend patterns of system behaviours. A number of non-linear trend patterns commonly adopted in classifying telecommunications systems behaviours are presented in Bhattacharyya G.K. et al. [1977].

(ii) System functions generate utilization vectors based on sequential Bayesian approach to realization of a system's behaviour.

(iii) Definition of characteristic observable behaviour types.
These behaviour types are patterns over some recurrent time window and are expressed in terms of statistical summaries (i.e. summaries for offered traffic and consumed resources). The algorithms that define the various behaviour types incorporate definitions of features that provide the basis for recognising the differences between observable statistical summary of one type from observable statistical summary of another type.

(iv) Definition of a prediction algorithm.

This algorithm takes as input a set of possible characteristic behaviour types. It can identify each of the types by recognising the properties of each type, and the differences between the types. Such a procedure provides the basis for stating how the probable behaviour types can be attributed to a set of effective user demand models and resource allocation mechanisms in force.

3.2.5 An integrated circuit of distributed algorithms

Figure 3.7 illustrates how the suite of resource allocation algorithms can be integrated together in an invocation based circuit. Even though the algorithms are arranged into columns, the objects that implement the algorithms need not respect these boundaries. The third column, a set of distributed agreement algorithms, is shown separately in order to emphasize that the main functional algorithms (centre column) must also implement distributed decision making rules which may require severe time-critical concurrency constraints.

Central to the circuit of algorithms are the predictions data and algorithms which are implemented to anticipate user demands for shared resources through configuration and learning. All the other operational algorithms either support the prediction algorithms or use the configuration and predictions data for providing services to users.

This circuit, together with the resource allocation paradigm outlined in this chapter, provide a comprehensive framework for assessing existing distributed algorithms.

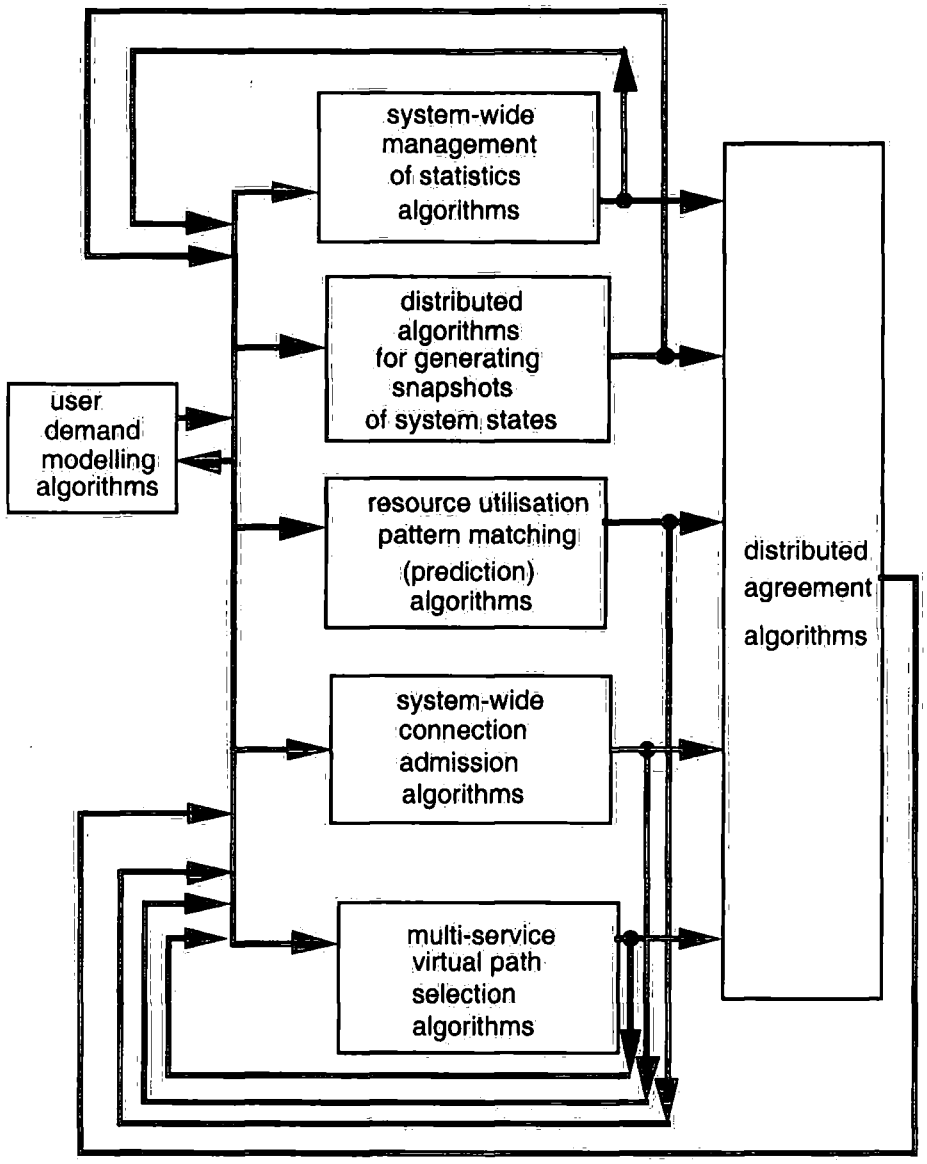


Figure 3.7 A system of network modelling algorithms

3.3 A critical review of modelling concepts for distributed algorithms

3.3.1 Introduction

In the previous section, two key issues were singled out in presenting a resource allocation paradigm for specification of system functions:

- enumeration and permutation operations on system states
- prediction within recurrent cycles of system evolution

These themes provide semantical and realization bases for a critical review of existing modelling concepts for distributed algorithms. Before stating the pros and cons of the use of existing techniques, it is important to bring together a collection of semantical concepts which, as a unit, form the target to which sufficiently expressive modelling techniques should attempt to comprise. This semantical unit is described in the next section; a critical assessment of key existing concepts is presented in section 3.3.3.

3.3.2 Semantics and realization criteria in the assessment of distributed algorithms

3.3.2.1 Semantics

Operations denoting the execution of distributed algorithms have syntactical **denotations** (symbols) used for constructing **sentences**. Sentences are used to make **statements**, simple and compound. Functions are defined to associate with each simple statement the values *true* , *false*. In some cases, neither true nor false values are assigned to simple statements in specifying representations for non-deterministic system states.

Given a set of simple statements S describing a system, a model of that system is a subset of $A \subset S$. Thus a sentence can be said to be true in a model. The interpretation or meaning of such a language (the truth or falsity of a sentence in a model) is a semantical property.

In developing modelling semantics for distributed algorithms, it is useful to ensure that both the basic mathematical semantics adopted and the system behaviour characteristics being modelled are given identical interpretations. Thus mathematical theories which are specified as axioms can play an important part in the development of appropriate semantics for modelling distributed algorithms. A standard reference for this approach is Chang C.C. and Keisler H.J. [1990].

3.3.2.2 Realization

The behaviour of a **decision engine** has been outlined in the previous section in describing a resource allocation paradigm. The concept of an **abstract machine**, well developed in the area of recursive function theory, provides a very general operational framework for the specification of distributed algorithms (see e.g. Weihrauch K. [1987]).

The notion of **abstract data type** is a very powerful concept for writing specifications of system functions in logical semantics. Such data type specifications abstract away from specific concrete implementations, but provide sufficient information for various realizations of the functions being computed. The following mappings illustrate how abstract machines and abstract data types can be used as the foundation for the specification of system functions.

D is the data set of an abstract data type A, $f_A: D \rightarrow D$ is the function computed by A,

$s \in N$, $s \geq 1$ is the natural number indexing the possible final outcome set of the computation,

$t_A: D \rightarrow \{1, \ldots, s\}$ are the (conditional) tests performed by A.

M = (A, X, Y, IC, OC) is an abstract machine such that

A is an abstract data type over D,

X and Y are input and output sets respectively,

IC : $X \rightarrow D$ is input signature encoding,

OC : $D \rightarrow Y$ is output signature encoding.

$f_M: X \rightarrow Y$ is the function computed by M,

$t_M: X \rightarrow \{1, \ldots, s\}$ are the (conditional) tests performed by M, defined as

$f_M := OC \bullet f_A \bullet IC;$

$t_M := t_A \bullet IC;$

where • is a binary operation.

A larger structure for setting up abstract data types is the algebraic specification. A detailed exposition of the standard concepts for algebraic specification of computation functions can be found in Wechler W. [1992]. However, these concepts need to be investigated in depth in the representation of distributed system functions.

3.3.2.3 A taxonomy for description of modelling techniques

Figure 3.8 brings together the concepts outlined in this chapter as a taxonomy for classifying modelling techniques for system functions. The top level realization semantics presented in the previous sub-section is sufficient for describing each constituent of the semantical unit illustrated in figure 3.8.

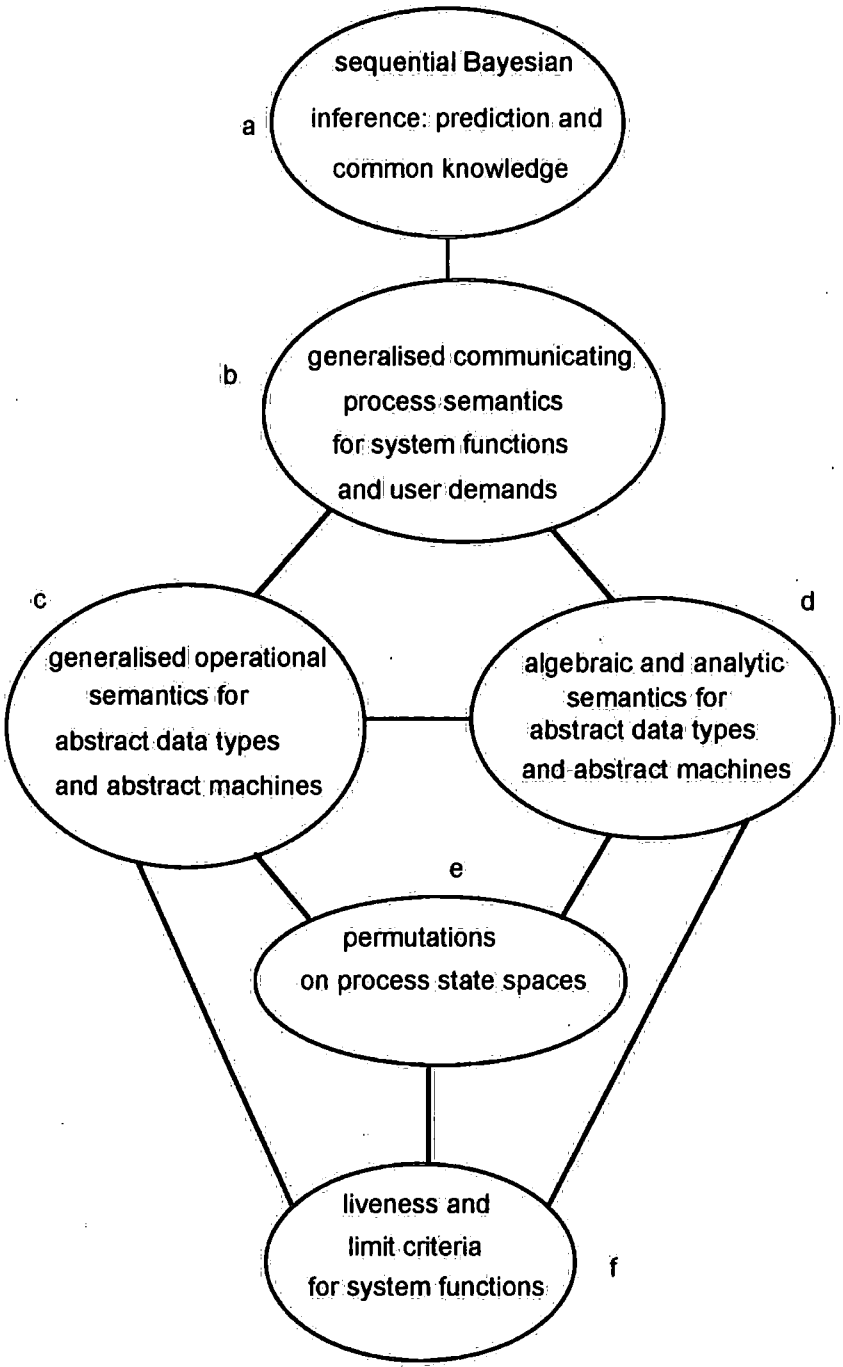


Figure 3.8 A taxonomy for operational semantics of system functions

a) Prediction and common knowledge:

The specification of system functions is inherently declarative, and can only be such in order to justify the selection of any resource allocation policy. This means that the data sets operated

upon by the abstract machine must be based on predicted behaviours of user demands for system services. Resources are shared across a network, and abstract machines are distributed across the network to serve as resource access arbiters and resource usage optimizers. There is therefore a need for distributed abstract machines to have some form of common knowledge about shared resources.

b) Generalized communicating process semantics:
Distributed agreement is only one attribute of the consequences of communication among distributed abstract machines. There are other semantical issues that need to be brought together to characterize distribution. Two main issues are the fair scheduling of the executions of abstract machines to run in a concurrent processing environment and the ordering of messages as they arrive at an abstract machine.

c) Generalized operational semantics:
In expanding the basic outline of abstract data types and abstract machines, the notion of theories provide a finite set of rules for setting up specifications of specific problems. Models for such theories provide the basis for realizations of problem specifications. A generalized operational semantics is required for the realization of abstract machines as agents that can be placed at locations of a network.

d) Algebraic and analytic semantics:
Algebraic specifications as refinements of the abstract machine provide detailed representations of snapshots of system states. Analytic criteria ensure the existence of system property relations among sequences of snapshots. These criteria provide the important basis for ensuring that the algebraic specifications satisfy requirements of non-deterministic user demands. Thus algebras and analysis go hand in hand in the definition of details for use of abstract machines to specify system functions.

e) Permutations on process state spaces:
Permutations over appropriate state spaces characterize all meaningful solutions from which optimal solutions can be selected. This means that the functions computed by the abstract machine must be sufficiently flexible to generate permutations for optimality definitions.

f) Limit and liveness criteria for system functions:
Liveness criteria ensure that computations executed by abstract machines do not go into a loop in a meaningless way. Limit criteria ensure that termination is inherent in the actions of the abstract machine. The definition of limit criteria for algebraic representations is a challenging task since it involves the definition of terminations for composed relations. Note that such

relations represent the evolution of system properties.

The following paragraphs cite references which develop components of the overall semantical unit in various contexts of research investigations, some not necessarily related to telecommunications.

The notion of system function described here is a generalisation of the classical system functions of engineering and physics (see e.g. Lancaster G. [1992]). The notion refers to the synthesis of a network to obtain a desired response to a specified excitation. In this study, the excitation is non-deterministic.

Distributed control of resource allocation in the presence of uncertain user demands can be modelled in a comprehensive way by using the concept of **common knowledge** among distributed objects (see Werlang S.R. da C. [1989]). This paradigm is based on distributed objects exchanging messages about the state of some network phenomenon, with a truth functional probabilistic attribute. The co-operating objects achieve common knowledge of a phenomenon when the belief probability of all the objects take value 1.

The operational framework for the common knowledge model is the probability space given by $(\Omega,\Im,P)$, derived from an exhaustive experiment such that:

ζ is an exhaustive experiment,
Ω is a sample space of all possible outcomes of ζ ,
$\Im$ is an event space (subset of the power set of Ω) ,
P is the probability measure on (Ω , $\Im$).

To characterize the evolution of system properties, the probability space is decomposed for sequential observations of events and the attribution of conditional probabilities to events (see e.g. Grimmett G. and Welsh D.[1986]). The resulting structure is the fundamental framework for the representation of system functions. The complexity of such a structure has been a major conceptual barrier to the modelling of large scale system functions.

In systems theory, metric spaces provide axioms for representing sequences of points that can be characterized with limit and continuity properties. Continuity is important in the modelling of system states when schedules of objects are specified to ensure that important events are not missed due to infrequent observations. Metric spaces can represent points as mappings without the burden of elaborate algebraic characterization of the points, but at the expense of expressiveness with respect to the representation of system property evolution. Topological

spaces generalize metric spaces and at the same time allow the exploitation of algebraic relations in the classification of system state spaces. Semigroups provide basic algebraic structures over and above set theoretic constructions; its structure is also compatible with sequences which are primitive in analytic expressions.

Mane R. [1987] presents an overview of the structuring concepts for the sample space as metric and topological spaces; this structuring is useful in models for system state space. Sequences of partitions of the event space are also characterised in that reference. Knight F.B. [1991] presents characterizations of the probability space as a prediction space, encapsulating probable event occurrences as topological spaces. Rosenblatt M. [1974] presents Markov processes as semigroups; such a structure can be enhanced to generate algebraic and analytic semantics for distributed statistical processes. These references show the interplay among measure theory, probability space, metric space, topological space and semigroups in the development of operational semantics for modelling system functions.

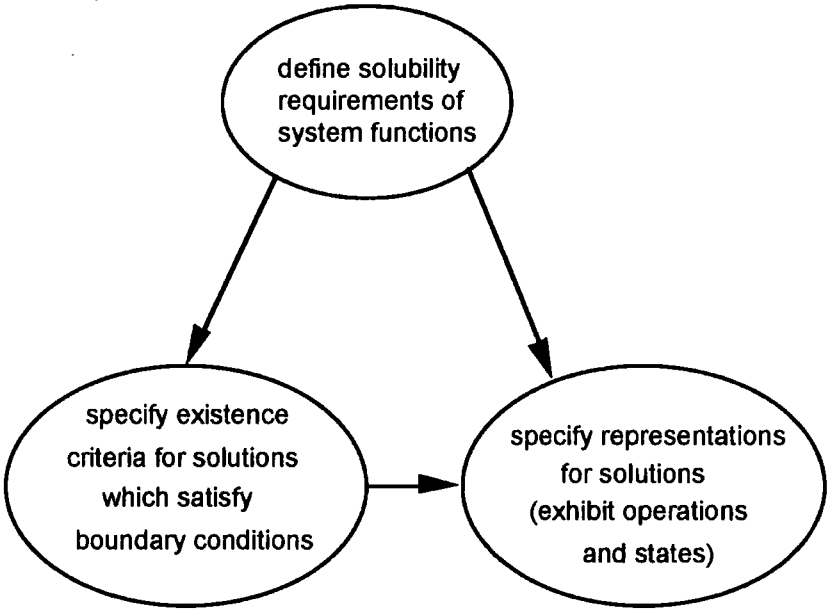


Figure 3.9 Refinements of system function specifications

Higgins P. M. [1992] shows how products of semigroups can be represented as hypergraphs. Degano P. and Montanari U. [1984(b)] use hypergraphs as the basic framework for representing events and schedules within a distributed system. The same authors use the notion of convergence in metric spaces as the analytic semantics for the specification of event and schedule sequences ([1984(a)]). Such semantics can be characterised in a very generic way by using a structuring technique which models mappings among algebraic structures as primitive (i.e. the use of **category theory**, see de Bakker J.W. and Rutten J. [1991]). The standard

reference for category theory is MacLane S. [1971].

Figure 3.9 illustrates how the definition of system functions in terms of requirements can be refined to generate detailed specifications. The approach of generating formulae to denote the existence of solutions is quite common in the literature. However, the representation of solutions is not a trivial task; in practice, representations often illustrate the inadequacies of existence formulae. Representation issues and examples are illustrated in this thesis.

The existence and representation requirements for solubility of system functions is a very challenging problem in distributed systems. A good starting point is to borrow techniques from the solution of (non-linear) polynomial expressions. An excellent description of the technique for formulating solutions of polynomial expressions is in Cohn P.M. [1981], describing how rings (algebraic laws) can be embedded in a prescribed field (algebraic laws). The analogy of this for system functions is that fields represent the most expressive rules of states of a system function while rings represent some acceptable constraining rules. The classical solution to this embedding problem was presented by Higman G. [1952]; it forms the basis of most existing iterative approaches to finding roots of algebraic expressions.

Solubility criteria assume that user demands are predicted correctly. Assume that interactions among communicating objects is well engineered. Located within each object are the function mappings computed by the abstract machines. These mappings are the algebraic operations of an algebraic specification. The algebraic operations can be constrained in terms of the values of the data they operate upon, to obey some familiar algebraic laws e.g. groups, rings and fields. Such laws serve as theories for generating model expressions as problem specifications.

In the detailed representation of problems, values taken by operations satisfy additive and multiplicative laws. Solutions are basically relations which ensure that a high level structure still satisfies a refined low level structure. This is often achieved by generating a sufficient set of low level values (system states) and defining relations that ensure that appropriate subsets of such low level values satisfy high level, low level and real world operations.

Possible results of sequences of computations computed by the abstract machine functions can be structured in such a way that the values belong to contracting mappings, as in metric space theory. Bertsekas D.P. [1983] formulates the optimal routeing problem using this concept, and generates representations as differential operators (see also Bertsekas D.P. [1982], Tsitsiklis, J.N. et al.[1986]). This approach forms a problem oriented framework for enriching representations to cater for statistical analysis of gathered resource utilisation data.

For system functions represented as non-linear mappings within high dimensional spaces, Narendra, K.S. et al. [1990] deploy the existence semantics of the Stone-Weierstrass theorem, and then generate required differential operators for dynamical systems. When applied to routeing in networks, Narendra et al. [1989] specialize the operational semantics as learning automata; they define a range of convergence criteria for use with these automata, including the fundamental concept of convergence in distribution. Common knowledge is a sub-concept of the Narendra convergence rules: convergence with probability one.

Perhaps the most powerful concept for developing both algebraic and analytical semantics of dynamical systems is the notion of sheaf theory. This theory is the foundation for several expressions in analysis (see e.g. Kashiwara M. et al.[1986]). Goguen J. [1990] develops limit processes for distributed systems in a very powerful way that encapsulates system behaviour attributes (see also Ehrich H.D. et al. [1990], where snapshots, behaviours and distributed agreements are described using sheaf semantics).

Meseguer, J. [1992] adopts the categorical logic for representing a range of inter-related algebraic laws for modelling concurrent systems. Embedded within these structures is the Petri Net (see Pezze M. et al. [1995]). Another approach to developing semantics of operations for distributed systems is to start from semigroup theory and specify the operational sequences as automata (sequential machines). This approach is attractive since relational concepts from database theory can be incorporated in a simple way. Further axioms can then be added to these theories to ensure fair scheduling and analytical fixed point semantics.

These models provide a large number of possible options for representing operations and behaviours of system functions. They all have in common the need to generate models using a large number of algebraic and analytic concepts in a unified way.

3.3.3 A critique of important modelling concepts

Recall that the main requirements on a modelling concept is that it should be sufficiently expressive to represent the behaviours of both an individual user and a population of users as illustrated in figures 3.1 and 3.2. The challenge posed by this requirement is so great that existing contributions are in the infant stage.

Figure 3.10 illustrates an architecture for conducting learning and prediction experiments. The non-deterministic behaviour generator (GFP) and its user demand entities generate resource

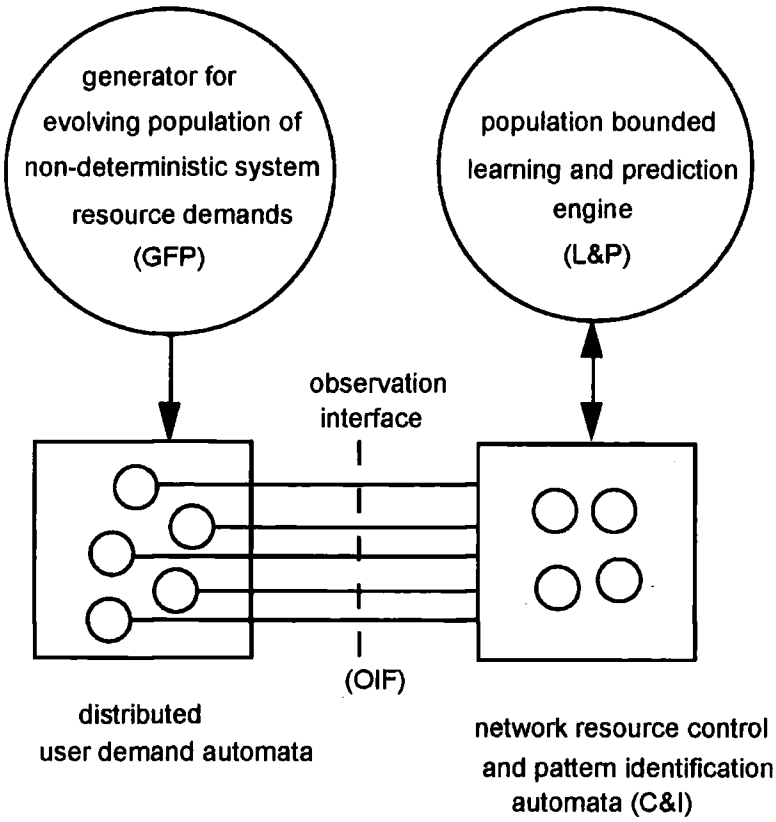


Figure 3.10 An architecture for learning and prediction experiments

demand patterns not known to the network control entities (C&I) nor the learning engine (L&P). The C&I and the L&P can only make inferences based on observations made at the observation interface (OIF). The C&I are distributed across the network whereas the L&P would be placed at strategic nodes of a network, or at a network management centre. The L&P is updated over time as necessary by the C&I entities; the latter are instructed to implement pre-declared policies using the procedure described in section 3.2.4. The L&P and C&I can be implemented as automata specified to carry out computations which identify characteristic system properties that hold at the observation time period. Such system properties evolve over time.

In selecting an operational characteristic behaviour pattern, the identification automaton is carrying out statistical analysis of observation data. A similar exercise is carried out by the L &P automata. Such an exercise involves recording observation data as random vectors and, in many cases, assigning probabilities to competing characteristic patterns. The procedure for assigning probabilities to competing characteristic patterns is a very challenging problem, and is a topic of research in its own right (see e.g. Mortimer H.[1988]). Therrien C.W.[1989] provides combined algebraic and analytic procedures for models that can be used for tackling

simple pattern recognition problems in image processing. The only acceptable procedure for assigning probabilities to characteristic patterns built from random vectors of observations data appears to be the use of *large sample inference* (as in e.g. Gelman A. et al.). This procedure is a formalisation of the measurement techniques described in section 3.2.4.

The abstract machine formulation can be used to represent both the deterministic and the non-deterministic automata of figure 3.10. As defined in Salomaa A.[1985(a)], non-determinism is characterized by the following. Given partitions of the sets X and Y of the abstract machine, sequences can be constructed as mappings among sets formed by the partitions. Elements of the sequences induce relations among pairs of input and output encodings of the abstract machine. Non-determinism means that given a sequence of input encodings, the associated output encodings are not always the same each time the abstract machine is enabled to run.

It is important to note that the distributed user demand automata are inherently non-deterministic. However, the C&I automata are not inherently non-deterministic even though they operate upon random vectors. Three important modelling techniques are evaluated in the following sub-sections.

3.3.3.1 Representations as analytic approximations

The queueing modelling approach to the characterization of system behaviours under specified loading conditions has been researched thoroughly over the past twenty years. A good description of queueing disciplines is presented in Gelenbe E. et al.[1987]. Taking a physical system, the queueing approach involves generating a model of the system components as a system of analytic equations which capture both the resource demands on the system (including probabilistic assumptions in the form of Markov chains or Markov processes), and the control actions (queue service procedures) of the system. The system equations can then be simplified so that approximate characteristic solutions (i.e. behaviours) are derived.

The cited reference is chosen here because one of the example problems tackled in that text is the modelling of a packet switched network. Several other papers and books tend to tackle smaller problems which do not pose the level of challenge addressed in the present thesis. The main benefit of the queueing technique is that it attempts a holistic formulation of representations of components of a system. The use of analytic semantics is also a benefit since such semantics provide goal oriented definitions of the problem being tackled.

There are a number of drawbacks in the use of the queueing modelling technique. Firstly, the

queueing formulation of analytic equations (expressions) tends to lack the expressiveness of algebraic representations which explicitly state desirable results of computations. Since equations are often represented as linear or non-linear polynomial or exponential expressions, the scope of the representations are often dictated by the tendency of these expressions to look clumsy when several variables are being addressed. Real world system functions are best represented as functions of several variables.

A second major drawback in the use of queueing models is that assumptions of non-deterministic demands on resources are often woven directly into the system equations being solved. Recently improvements have been suggested to make system equations more flexible in order that measurements from real networks can be incorporated into operational models of queueing systems (see e.g. Saito H.[1993]). Such improvements weaken queueing theory in a way that makes it more in line with the concepts proposed in this thesis.

3.3.3.2 Representations as learning automata

The basic denotation of the learning automaton model can be generated from the abstract machine model described in section 3.3.2.2 earlier, non-determinism being defined as in Salomaa A.[1985(a)] through output encodings. An excellent introduction to the learning automaton is the presentation of Mars P. et al.[1996]. In the reference is also described the application of learning automata for characterization of resource access arbitration (access control) and routeing in telecommunications networks. A comprehensive treatment of the analytic semantics for learning automata is presented in Narendra K.S and Thathachar M.A.L.[1989]. The challenges of representing system behaviours using a system of learning automata is exposed thoroughly in Thathachar M.A.L and Phansalkar V.V.[1995].

A key strength of the learning automaton concept is that it addresses in a very concise way the following aspects of figure 3.10.

i) Non-deterministic user demands which are incorporated into the model of the status of shared resources,
ii) Assignment of probabilities to the choice options seen by the automata as resource access arbiters. This procedure forces the system designer to develop an appropriate pattern identification scheme; such a scheme must incorporate convergence criteria since this is the core semantical basis of the approach.

The main drawback of the learning automaton modelling concept is well articulated in Thathachar M.A.L. and Phansalkar V.V.[1995]. This is the inability of the learning automaton

to represent effectively collections of characteristic pattern or context vectors. These investigators suggest that either the structure of the automaton is generalized to capture context vectors, or attempts are made to couple teams of automata to form connectionist systems for pattern analysis and optimization.

Perhaps the main impediment to the use of learning automaton in modelling large scale systems is the bundling up of several formalisms within the notion of a non-deterministic action. A close examination reveals that the assignment of probabilities to actions can be achieved through statistical analysis of observation vectors, the construction and execution of discriminant functions that characterize system state patterns, and the preference ranking of competing characteristic patterns. All these sub-concepts are described in detail in Therrein C.W.[1989]. This critique is in line with the development of discriminant functions in the paper of Thathachar M.A.L. and Phansalkar V.V.[1995].

The use of learning automata for modelling large scale system functions is a possibility in the future after some basic representation issues have been clarified.

3.3.3.3 Representations as transaction automata

N.Lynch in Fekete A. et al.[1993]) has developed a technique of connecting together collections of abstract machines to form a system referred to as the I/O automata. In that model, output signatures of automata are matched to input signatures of co-operating automata thereby creating potentially unlimited sized conglomerates of abstract machines. Two structuring criteria are required to make the automata work:

i) Assignment of time bounds to event arrivals which trigger input driven computations (see Merrit M. et al.[1991]),
ii) Set theoretic interpretations of theories that specify a problem's constraints.

The power of this concept is demonstrated by using it for the specification of large problems in database theory (e.g. Weihl W.E.[1988]), and communications protocols (e.g. Welch J. et al. [1988]). In the latter reference, the specification of a realistic resource partitioning algorithm (the spanning tree algorithm described in Gallager R.G. et al.[1983]) is validated.

The I/O automata has been used to generate permutative characterizations of a large range of realistic networking protocols (see e.g. Weil W.E.[1988]). The representation of system functions in a way similar to the representation of transactions in database theory is primitive to

the modelling approach. There is therefore a good prospect that the modelling procedure could be enriched with the kind of semantical concepts proposed in figure 3.8. The main drawback of the modelling procedure is that it lacks analytic and geometric semantics. The introduction of the latter features into I/O automata is a non-trivial task since product form representations of state sets need to be formulated and represented as geometric patterns.

3.4 System states as continually varying geometric quantities

The following problem statement is a challenging task which could yield useful results in a research project developing scalable modelling concepts.

Can system functions be realized as timed automata whose computations are synthesised as permutations and limit preserving relations on geometries of system states ? Can representations of system states be defined as continually varying geometries that capture meaningful realizations of system functions ?

It may be possible to generate techniques involving algebraic classification of geometric structures; such techniques would provide axioms and constraint satisfaction rules for developing control and identification algorithms. It may also be possible to define limit preserving structures for representation of data sets classified using such algebraic axioms.

The greatest challenge to developing usable semantics for modelling large practical system functions is the issue of scalability of representations for system state structures. Since permutations on system state spaces are basically relational concepts, it is imperative that any proposed concept should incorporate relational complexity measures. This problem is yet to be addressed; relational complexity concepts are only just being defined (see e.g. Abiteboul et al. [1997]).

3.5 Summary

An introduction to resource based policies has been presented in carrying out an outline description of routeing rules for resource sharing among users whose demand models are non-deterministic. Instead of working with a sparse level of modelling data such as blocking probabilities, a detailed approach based on resource utilization levels is adopted. The

comprehensive resource based approach encapsulates a lot of important detail which cannot be sacrificed if results of modelling experiments are to be useful in the characterization of behaviours of practical networks.

A survey of existing work shows that appropriate semantics for comprehensive representation for system functions is yet to be developed and tried out on realistic network scenarios. In carrying out such semantical model evaluation experiments, there is an interplay between logical models and object oriented models. The latter models are closer to the physical structures of networks being modelled. The next chapter presents a development of an object model and the use of such a model in two modelling case studies.

Chapter four

Object interactions in the realization of system functions

4.1 Introduction

This chapter presents an outline of two case studies which are used to illustrate the modelling concepts proposed in this thesis. A presentation of the architectural structure of resource pools precedes the description of the two case studies. The first case study illustrates how user demands on resources can be restructured in order to reduce network resource consumption. The second case study addresses resource allocation issues directly by describing how user demands on resources are accepted by a resource access arbiter. This case study is used to introduce the notion of co-ordinates for specification of system state geometry.

The main contribution of this chapter is the way system state variables are built up from names of locations which, as a collection, is self contained within a sub-network. Geometries of system states subsume locations where measurements and control actions occur.

4.2 Resource structuring for sharing

4.2.1 Partitioning of transmission resources

Figure 4.1 illustrates an example fragment of the information streams that are enabled within a network for short periods of time. The nodes of the network are labelled using capital letters C, E, ... , etc. The information streams are labelled using lower case letters. The streams here are unidirectional purely for simplicity; the concepts being developed for individual streams also apply to bi-directional streams. Streams on the diagram denote traffic flow over routes that are selected during a specific time window. Thus paths <a-a'> and <f-f'> denote two distinct routes provisioned for use by users at nodes E, G and D. Also, path <c-c'> denotes a route for users at nodes B and D to communicate over. In the example outlined here, the routeing scheme has been invoked to enable at nodes E, B and D to share the transmission resource pool at link N-D.

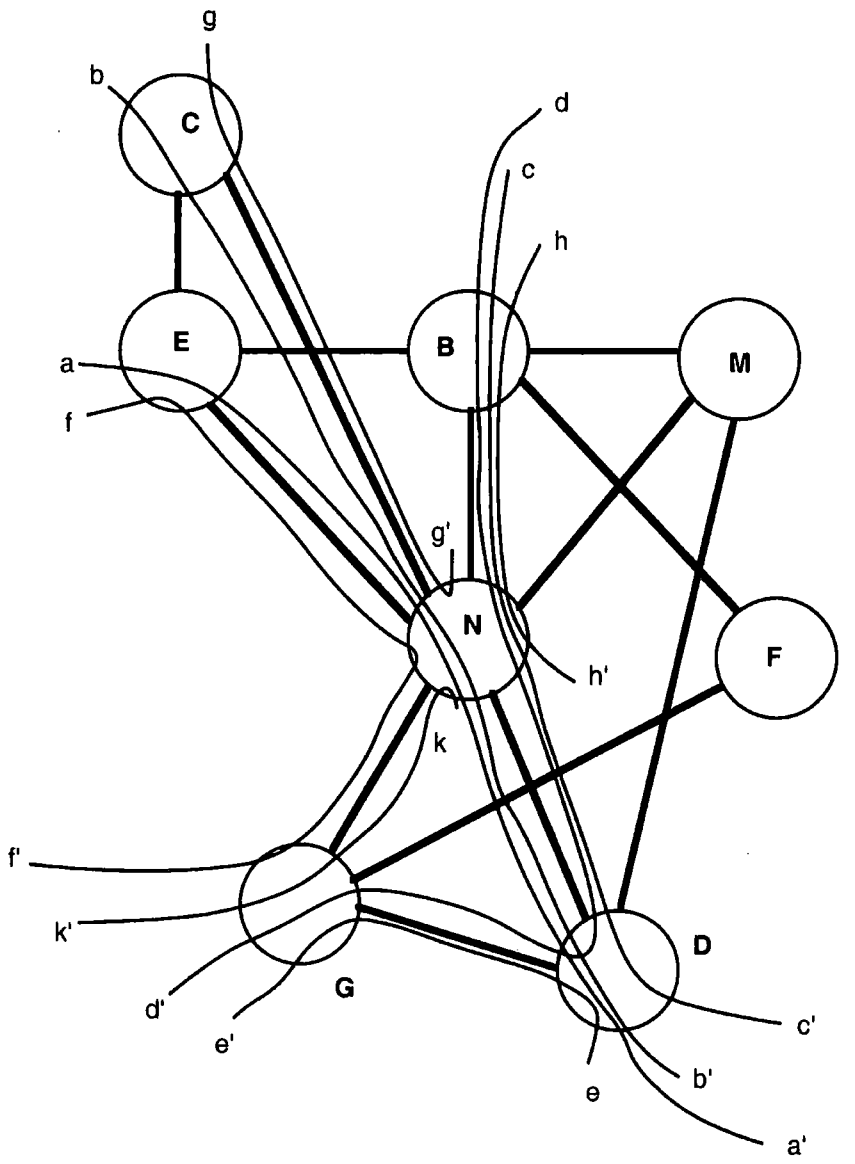


Figure 4.1 Information streams through a network

The concept of resource sharing outlined here provides more detail on the resource allocation concepts described in the previous chapter. Thus the development of resource allocation algorithms can be synthesised by considering behaviours and goals of user collections. The resource access arbiters implemented within the nodes of the network continually observe each user in order to detect when a user bids to consume pooled resources. The system functions hosted by the resource access arbiters continually constrain the way resources are allocated in such a way that various system properties (e.g. information packet loss or delay) evolve according to user, user populations, and overall system performance goals.

4.2.2 Structuring of shared resources

An introduction to the structuring of distributed objects was presented in chapter two in providing a holistic picture of system functions within distributed systems. Traditionally, the modelling of distributed systems is normally split into two work areas: communications technology and time constrained distributed transaction management (see e.g. Soparkar N. R. et al. [1996]). In the communications technology case, system functions are often specified as elaborate communication protocols. In transaction management, system functions use basic macros such as locking and commit protocols. It is essential that the two work areas be integrated into one structure since simplification can be achieved by ensuring that the operational capabilities of these two areas inter work correctly.

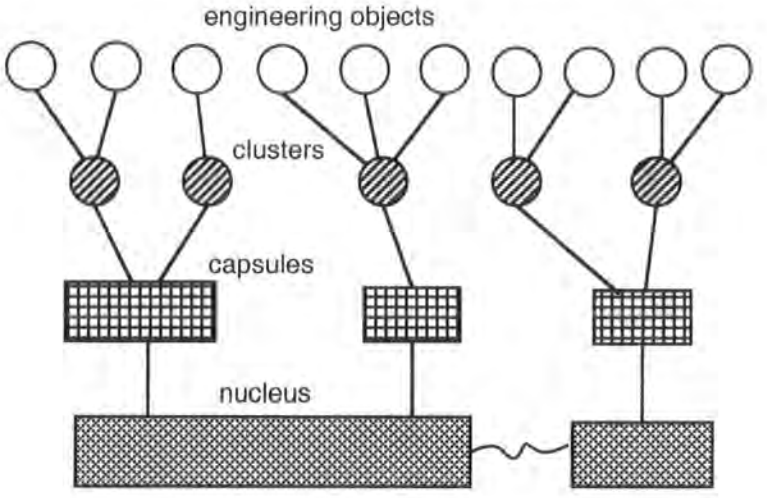


Figure 4.2 The engineering entities

Figure 4.2 illustrates the engineering entities required to host system functions. The engineering object is the basic abstraction for hosting a system function's computation at a location. The engineering object can be plugged in to a **socket,** to receive invocations for performing computations. An engineering object consumes processing and memory resources encapsulated as the **capsule**. Capsules host **clusters** of engineering objects, in implementing the notion of economies of scale. Transmission resources are encapsulated into the nucleus. Thus the traditional notion of storage is encapsulated within objects and capsules, and can be made location transparent.

4.2.2.1 Information transmission attributes - end systems

Similar to transmission links within networks, channels are provisioned within end-systems to carry information streams to and from sockets where objects are located. Figure 4.3

illustrates four networking nodes $m_1, \ldots, m_4$ and locations within end-systems at nodes m_1 and m_2. Assume that a service is located at *d* on the left hand side end-system. A client located at *g* on the right hand side end-system can invoke the service by sending a message via arcs gx, xo, . . . ,m_2m_1, m_1h, . . , kd. However, a client at location *n* could invoke *d* by sending a message through nx, xw, ... ,m_2m_1, m_1h, . . , kd. Thus both invocations are multiplexed at location m_2. It is of no significance to the server where multiplexing is carried out provided that specified channel attributes are not violated. Let the locations *d, e, w, g, n* and *j* denote sockets and *k, r, x* and β denote capsule addresses.

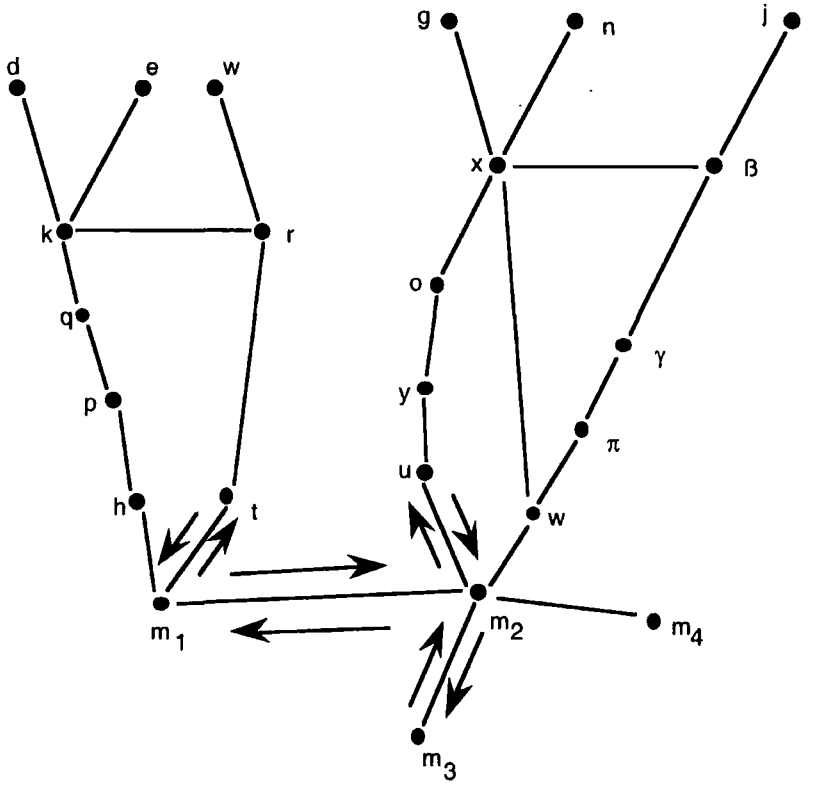


Figure 4.3 The structure of channels across end-systems

The client at *g* in intending to invoke the service at *d* does not need to know that *d* is remote. It invokes the service anyway, providing its binder with a set of attributes. The invocation can be interpreted as a remote procedure call; the attributes stipulate information transmission attributes for that computation.

The locations *k* and *x* can be interpreted as capsule locations with structured sub-locations to which are attached support objects which carry out presentation syntax conversions. Locations for interfaces (d, e,.., g, . . etc.) can also be interpreted as structured locations to which are attached remote procedure call support objects.

Thus in order to satisfy stipulated information transmission attributes of an invocation,

messages are routed through support objects located at vertices of the end-system graph. Support objects often need to invoke other support objects which are not directly adjacent to each other in the provision of services for their clients. Such invocations are relayed by intermediate support objects on the network graph.

As an example, consider a client located at g wishing to consume a duplicated service located at *d* and *n*. In order to ensure synchronization of service provision, support objects are provided at locations *o* and *q*. Thus an invocation by a client at *g* is routed to a replication support interface at *o*, which carries out a number of interactions with its peer at *q* in order to synchronize service provision by the replicated set of applications.

The following statements define information transmission attributes of multiplexed channels that join together pairs of communicating users. For a non-replicated server at a socket, the location of attachment of an end-system on a network (the m-location on figure 4.3) is the multiplexing location. Within the end-system, various paths can be constructed between the m-location and a specified socket. These paths are characterized by the operations and observations executed by the objects attached at named locations within the end systems. These objects serve as controllers of access to transmission resources used in peer to peer communications among users. The computations carried out by support objects within end systems are aimed at achieving both user and system-wide goals. It should be noted that support objects are themselves users of transmission resources and so do consume shared resource pools in exchanging network-wide messages among themselves.

During any instant in time when a support object is invoked, the results of the operations executed by the support object can be encapsulated within the names of interfaces to other objects that exist within the distributed system. This is because in a declarative system, it is necessary to give names to collections of entities whose values denote observable system attributes. Support objects can carry out both monitoring and control of multiplexing events involving end to end channels between transmitters and receivers. The state of these channels can be referred to as information transmission attributes. Values denoting these states can be named in the context of location addresses of objects that encapsulate the co-ordination of computations in which the system states are derived. Such a location address is called the Interface Reference (see Nyong O.D.O.[1993] for a detailed specification).

4.3 Case Study I : The storage functions

Storage functions are provided by systems programmers for effecting efficient resource consumption by users. Thus the collection of functions provides an indirect way of influencing user demand models for consumption of shared resources. This case study

shows how an individual's behaviour can be transparently modified to suit a population's behaviour, and therefore satisfy both the individual user's and the population's goals.

A detailed specification of the storage functions is presented in appendix A. Though formal in its derivation, the specification script is written in a notation-free style. This approach is taken because in the specification of large scale system functions, it is desirable to isolate a few operational and semantical concepts that can serve as the basis for representing and validating a large number of operational sequences. The formal justification is developed in chapters 5, 6, and appendix C. In appendix A, the specification of the storage functions is presented as collection of self-consistent operational sequences.

The communication infrastructure within a network has been described in the previous section where abstractions for the physical structures of networking components have been described in the definition of resources as clusters, capsules, nucleus and channels. The specification of storage functions is carried out using this framework.

The key points are as follows:

- Objects provide services at interfaces. Interfaces provide operations. All access to objects is by operation invocation. Object state consists of both internal sub-objects and references to other external objects. Internal state can be represented as data values. Objects consume processing (i.e. processor and memory) and communication resources in carrying out operations. Passive representations of objects can be defined and derived at run time; they consist of snapshots of the objects' states. Passive representations can be recombined with processing and communications resources to create the original object. Interactions between objects require processing and memory resources. When either resource is not immediately available, such interactions are delayed. The communication infrastructure can pose a further delay when objects are remote. This latency can be overcome by moving objects closer together.

- An object can be **active**, **inactive** or **passive**. Active and inactive objects are provided with processing resources whereas passive representations are not. An inactive object carries out no useful computation during the period of inactivity. Passive representations of objects consume minimal processing-resources whereas inactive and active objects consume significantly more. There are, therefore, advantages to *passivating* inactive objects: objects can be moved from location to location to optmimize system-wide resource allocation and use.

- Storage functions are specified as operations executed by resource management objects,

designed and installed by system programmers. These objects are required to provide management operations on their representations. A management operation can be initiated internally within an object, or externally by a peer object. Application programs encapsulated by the storage functions can be ported from one execution environment to another, independent of the underlying technology.

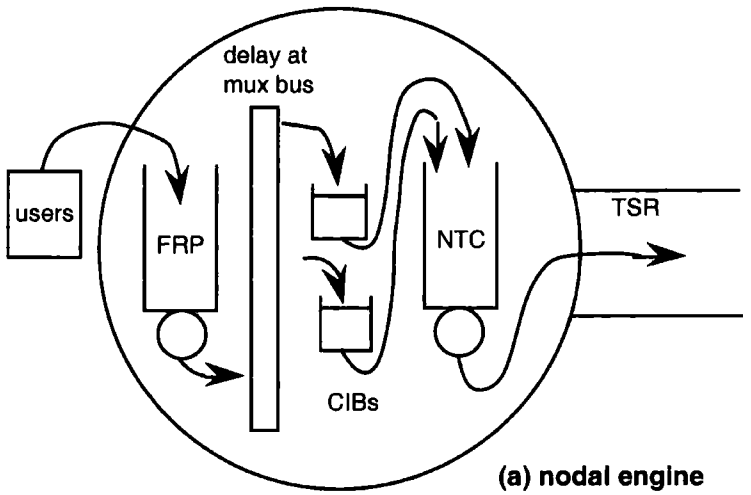
• The management functions defined by the storage system functions are:
i) the **snapshot** function, used to capture an object's representation to be placed in stable storage, or sent to another object,
ii) the **passivate** function, to detach an object's representation from processing resources and move it into a passive environment, retaining sufficient information to enable later re-activation,
iii) the **activate** function to restore a passive object to an active state,
iii) the **migrate** function to move an active object from one location to another.

4.4 Case Study II: Congestion avoidance functions

The congestion avoidance requirement in telecommunications networks brings together, in one set of related system functions, issues concerning the requirement to satisfy both individual user goals and user population goals. The case study introduced in this section complements the storage functions introduced in the previous section since it addresses directly the arbitration of access to shared resources. User streams exist as on-off system property states. When a stream state is on, packets of data are then sent in on-off bursts. Packets may be grouped into frames. This approach to modelling user demand models is sufficiently flexible to be used in generating traffic patterns that are characterised by the traffic variables specified in the previous chapter.

Figure 4.4 illustrates the internal structure of switching and multiplexing engines. Figure (a) illustrates the node engine:

User demand streams are received at the input buffer of the Frame and Packet Port (FRP) where it is scheduled and relayed by the multiplexing bus, to be transmitted across the network. The Transit Shared Resource (TSR) is loaded by the Network Trunk Controller (NTC) which receives its input at a set of Conceptual Input Buffers (CIBs). Thus local and transit streams are multiplexed by the NTC. The NTC can service its CIBs in a fair or priority scheme, as necessary.



FRP: FRame and Packet Port
CIB: Conceptual Input Buffer
NTC: Network Trunk Controller
TSR: Transit Shared Resource

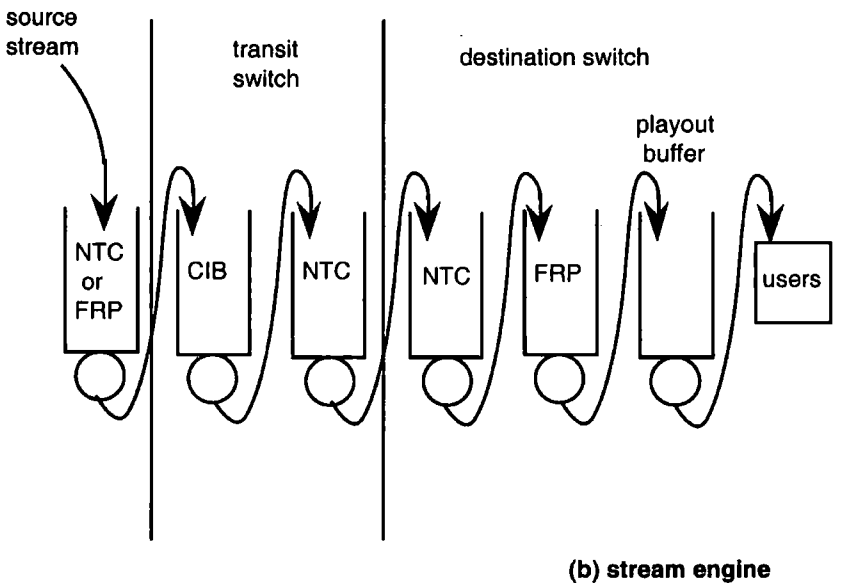


Figure 4.4 The internal structure of switching and transmission engines

Figure (b) illustrates how the object components of the nodal engine form an end-to-end chain. At the destination exchange, the play out buffer provides a function that models the receiving user's behaviour.

Figure 4.5 illustrates how the distributed control of allocation of shared resources is carried out by inter-connected resource management objects.

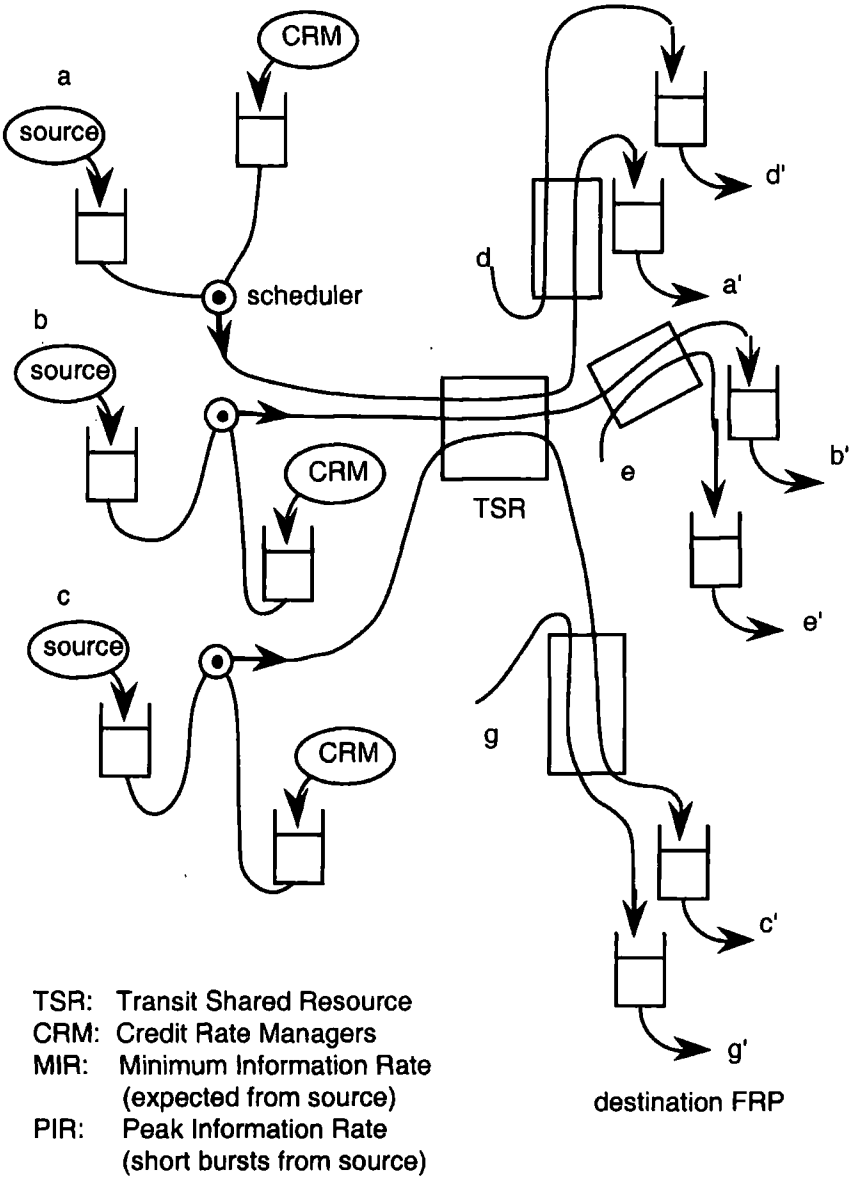


Figure 4.5 The inter-connection of distributed arbiters

The Transit Shared Resource (TSR) object encapsulates the functions performed by the NTC with respect to the following system functions. It maintains local statistics of resource consumption trends so that the information can be collected and used for prediction of resource consumption by route sets managers. In the short term, it maintains a snapshot of local resource consumption status.

The destination Frame and Packet Port (FRP) maintains and manages statistics of information transmission attributes along the path of the stream it supervises. This information is thus available as a basis for prediction of imminent congestion along a path, or the need to allocate unused resources to other routes.

The Credit Rate Managers (CRM) located at source nodes execute distributed agreement algorithms in deciding the rate at which to admit user demands. A credit award rate manager is used to implement admissible resource allocation constraints for each stream. When resource allocation is carried out as timed discrete event operations, credit awards are maintained as credit levels which can then be monitored in ensuring that a user's goals and a population's goals are satisfied by the underlying software and hardware resource allocators.

The scheduler performs the local resource admission function, guided by the credit level it sees, as well as the request level at its input buffer.

All the resource management objects maintain various state variables over time, in carrying out control of allocation of shared resources. Figure 4.6 illustrates an important set of co-ordinates for this case study.

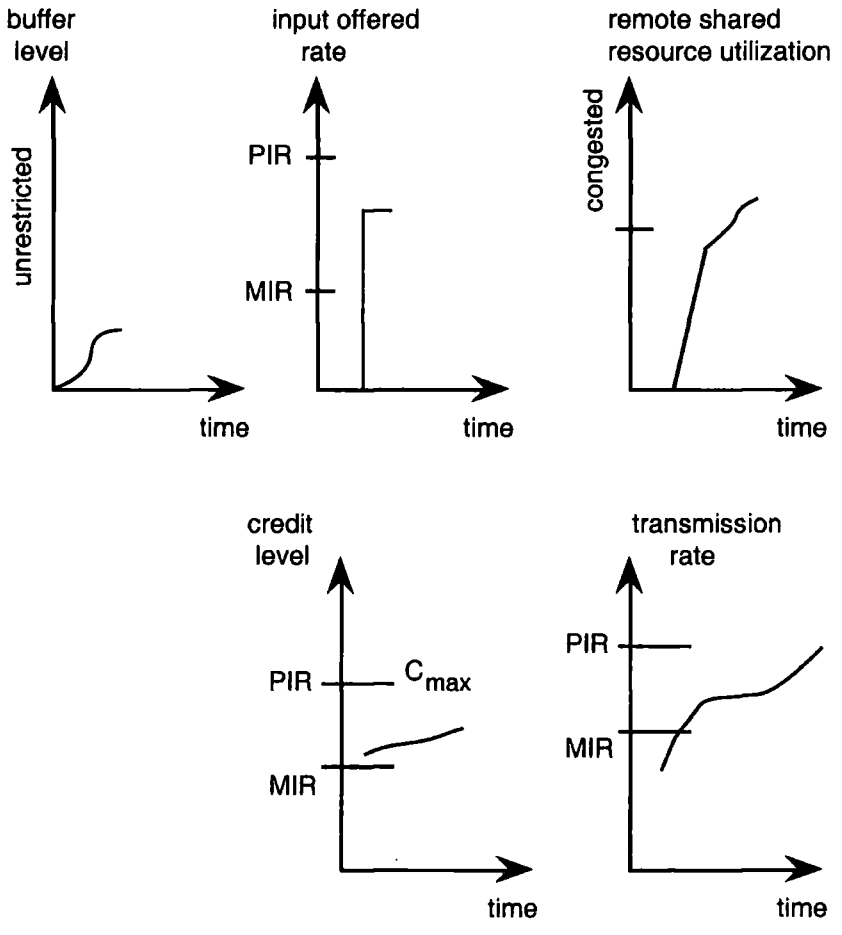


Figure 4.6 Coordinates for system state geometry

i) The input buffer level provides the vehicle for snapshots of offered user demand model. For experimentation purposes, it can be generated to take unrestricted values.

ii) The input offered rate is predicted to vary between zero, and the Peak Information Rate (PIR). The Minimum Information Rate (MIR) indicates the traffic rate that the stream can offer continuously, and expect to have guaranteed network resources. However, the stream can offer less traffic than MIR over a time period, and have some credit saved for future use in periods when its offered traffic can exceed MIR, up to PIR. The carried traffic is controlled using the credit management mechanism. Clearly the 0-MIR-PIR range is obtained by both declaring some values a priori and predicting new values a posteriori.

iii) The remote shared resource level indicates the current utilisation status along the path of the stream. This variable is observed by the Transit Shared Resource object.

iv) The credit level indicates the discretized stable snapshots of short term transmission entitlements of a stream. The full entitlement cannot be exercised if congestion exists. Its value reflects both the effect of credit award rate on consumed resources, and the effectiveness of predicted user demand patterns across the network.

v) The transmission rate from the input buffer into the network indicates the effectiveness of implemented resource allocation policy.

These co-ordinates can be combined in various ways in specifying admissible behaviour patterns of the system. This task is carried out in appendix B, and in chapter 7.

4.5 Summary

This chapter has provided an informal introduction to two groups of system functions used for detailed investigations in this research: the storage functions and the congestion avoidance functions. A detailed illustration of object communication infrastructure is presented showing how information packets sent and received by objects are carried within streams that span sub-networks. Abstractions for processing engines within a sub-network are also presented. These engines support networked objects. This architecture provides a framework for describing all the necessary data required for declarative specification of system functions.

The structuring framework for objects, clusters, capsules and nucleus was developed by a team of architects at the Advanced Networked Systems Architecture (ANSA) research

laboratory in Cambridge, UK, in 1991. The author was one of the networking architects at that laboratory. The structure of channels and information processing attributes has been developed solely by the author.

In a layered network architecture, the storage functions would normally be placed as an application support layer on top of the congestion avoidance functions. Thus the two groups of system functions provide a good span of typical communications systems functions. Higher layer functions (e.g. the storage functions) would normally exploit permutative and logical concepts, absorbing the lower layer functions in their information transmission attributes. Lower layer functions (e.g. congestion avoidance functions) would in addition exploit the benefits of constructing detailed geometric structures of system state. Co-ordinates for geometric patterns are introduced using the congestion avoidance system functions.

In the next chapter, the storage collection of functions is used as a practical example for developing product form representation concepts. Geometric patterns are given further intuitive interpretations.

Chapter five

Product form representation of system functions

5.1 Introduction

When objects interact in the execution of system functions, they operate on data denoting system state variables. A structure is developed in this chapter for representing system evolution data. A theoretical foundation is put forward for structuring the internal states of interacting objects as they execute system functions. The theoretical foundation must have both of the following characteristics:

- problem specific representation model in order that it can satisfy system functions that are specified to carry out resource usage control and system management.

- problem independent generic structure with deductive features that allow for validation of every implementation's correctness, and operational semantics that allow for evaluation of characteristic system behaviour performance.

A geometric structure has been developed for structuring system states in such a way that the methodology appeals to intuition, but also has a sound formal basis. The notion of state co-ordinates provides the necessary theory as a generator for functions of two variables, with values in a time induced plane. Discrete planes are specified to continuously merge with each other thereby generating paths of system values in a system of R^3 spaces.

There is a need to classify the enormous size of state spaces that are often generated by specifications of large system functions. Classification provides a framework for making statements about system properties and also forms the basis for developing automata theoretic concepts. Automata are useful for organising large specifications and form a basis for linguistic oriented specifications suitable for mechanization and automated correctness checking.

Before embarking on a detailed exposition and example specification of a realistic case study, a toy specification is described. Expressions illustrating the behaviour of a simple pendulum is used as an example dynamical system. The toy example is then used as a starting point for presenting issues involved in generating and combining system functions.

A detailed specification of a case study is presented in appendix A. The specification and verification task for system functions illustrated in appendix A was carried out over a year, full

time, by the author while being seconded to the Advanced Networked System Architecture (ANSA) laboratory in Cambridge, UK. The specification was implemented in parts by an implementation team at the ANSA laboratory. The size of the case study provides an illustration of the complexity of system functions encountered in the design of practical networks.

The specification makes a contribution in the area of distributed algorithms in the following way. The variational approach to specification of operations and rules governing the behaviours of a dynamical system has been successfully developed by applied mathematicians and physicists over the past fifty years. The approach is extremely powerful since a dynamical system can be specified, built and tested to satisfy known physical laws. By analogy, a large scale distributed system function has been specified to satisfy a large collection of rules which are self-consistent in their prescription of expected system behaviours. The generation of such a collection of operations and constraints is a novel application of a well established powerful concept. The encapsulation of information transmission attributes in the specification of computations carried out at a high layer object within a telecommunications system is original. A contribution is therefore made in devising this way of refining distributed algorithms.

In contribution, this chapter illustrates the intuitive use of theories to guide the specification of a large system function. It is an experimental attempt at combining the notion of state geometries with emerging concepts of algebraic specifications of abstract data types. Performance related issues are not addressed in this chapter. The next chapter focuses on state structuring in modelling for performance characterisation.

5.2 State co-ordinates

A thorough definition of the notion of entities and attributes is postponed to the next chapter where curves are defined formally. In this chapter, an intuitive use of this concept is adopted to avoid cluttering up this introductory description of state evolution geometry. In the previous chapter, some important co-ordinates were selected as basic variables to keep track of system property evolution in specifying distributed functions that control user access to shared resources. As an example, consider the hypothetical variation of credit award rate against user traffic demand, in controlling traffic admitted into a network's route. The credit award rate can be adjusted to admit traffic in a way that satisfies system goals, as follows.

Referring to figure 5.1, the variation of credit award rate and user demand can be observed at specified times of a clock tick. On the figure, t_π indicates planes carrying loci of states, e.g. states <a,b,c,g,k> which have been declared as part of a large collection of possible system

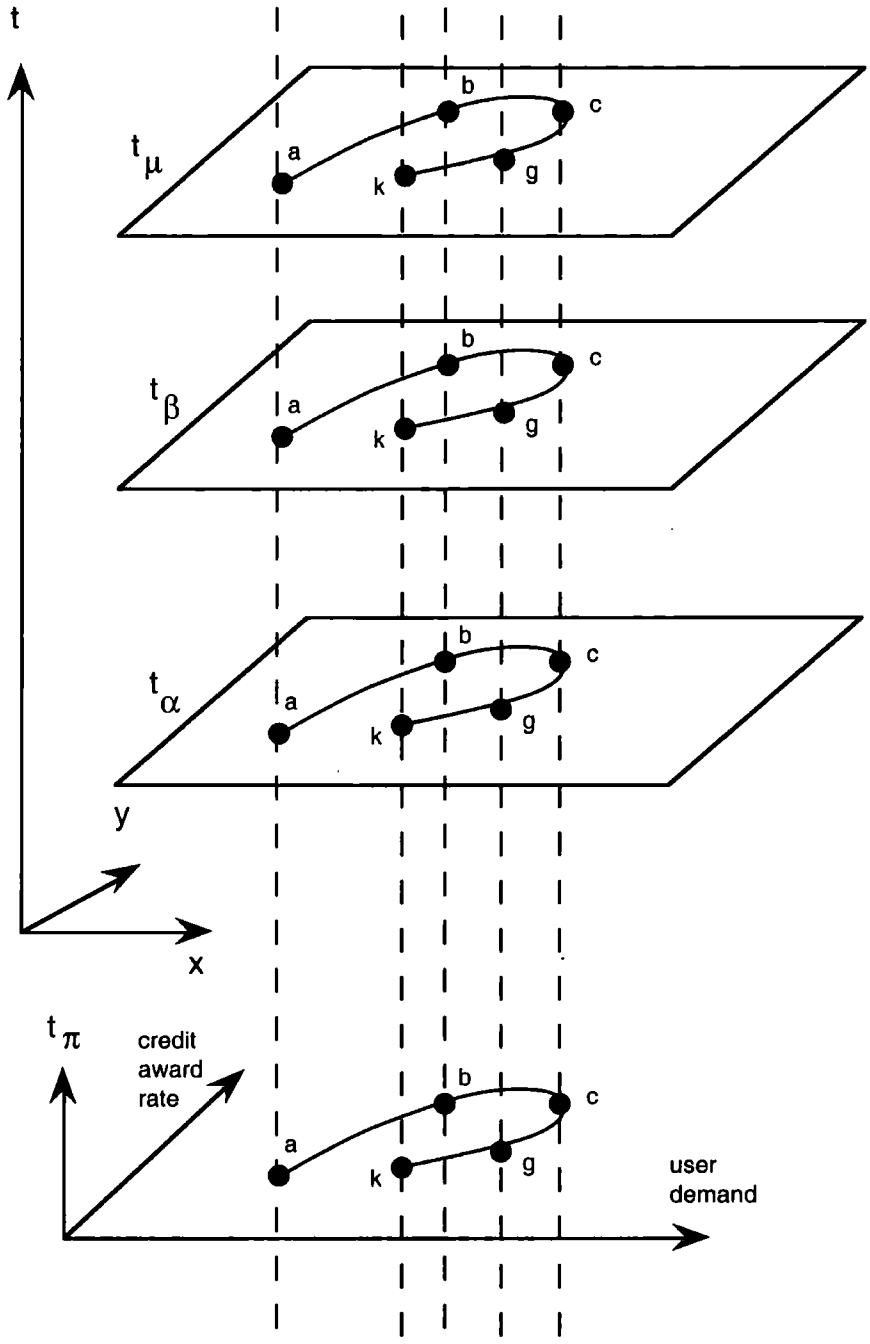


Figure 5.1 Conceptual geometry of system state values

behaviours. Each of the points being an approximation to an observable value, is given attributes which can also be declared to be observable, and relate to other system states. At time t_α, the system's state attribute could be <a>, moving towards <b>. Between time t_α and t_β, the user demand increases such that the state attribute <b> is reached. At this point, the credit award rate is held constant as per user-network pre-defined contract. When the demand drops, for example, to enable the state attributes <g> and <k>, the credit award rate is decreased accordingly. The <k> state could occur say at time t_μ.

5.2.1 Computation of structured state values

The state transitions described in the previous paragraph need not be discrete. As an example, point <b> could be expected to be obtained from a convergent sequence after a number of observations. Figure 5.2 provides a conceptual illustration of how a system state such as <b> is obtained. Assume that a set of points S is obtained from a sufficiently large number of snapshots by a system function. Figure 5.2(a) illustrates how a sequence of values V1,...,V10 relate to each other, after being computed by a system function using data declared within the system, and summaries using points in S. As will be seen in the case study described in appendix B, such a **goal oriented** configuration of system data is highly useful and desirable.

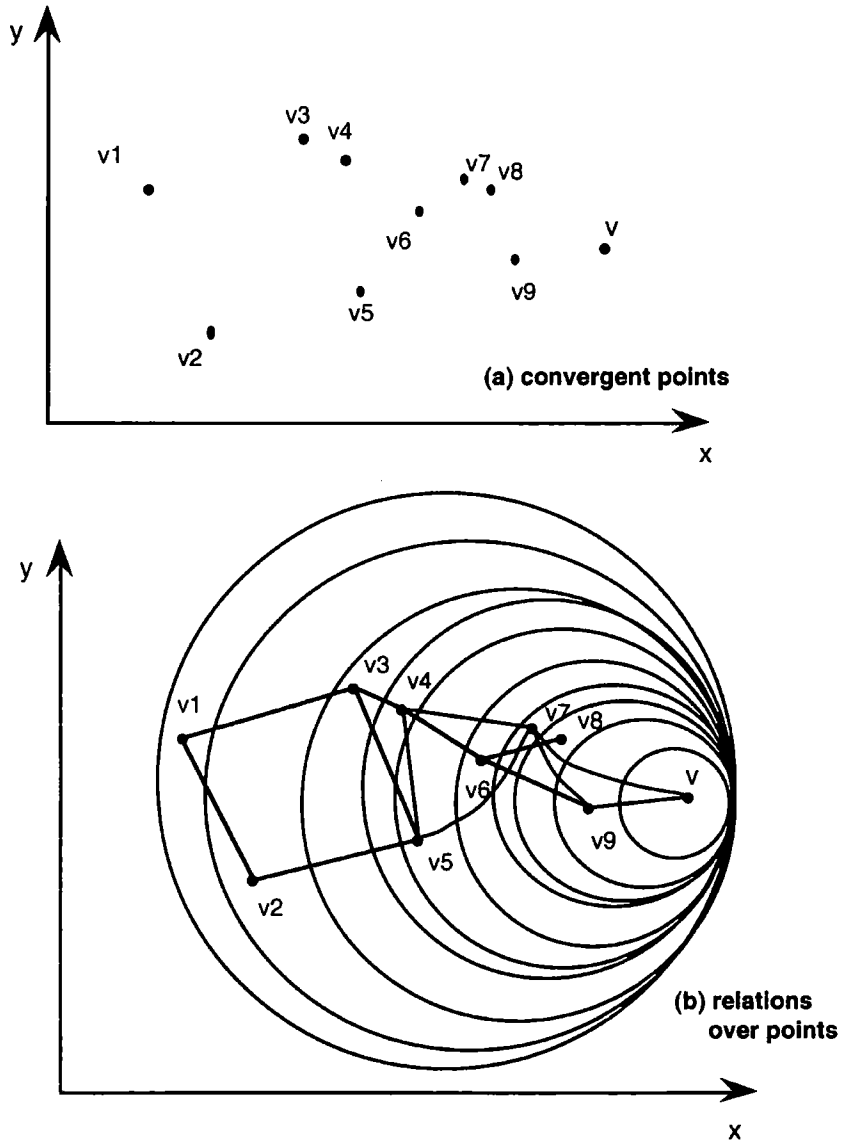


Figure 5.2 (a,b) Conceptual definition of states' convergent relations

Figure 5.2(b) shows how the value points V1, ...,V10 can be allowed to relate to each other as a system property's state evolves from V1 to V10. The goal state is V10. An example sequence is:

<(V1-V2), (V2-V5), (V5-V7), (V7-V9), (V9-V10)>

An alternative sequence resulting in a different goal state is the following:

<(V1-V3), (V3-V4), (V4-V6), (V6-V8)>

Figure 5.2(c) illustrates how a sequence can be attributed relations over time where points V1, V2, V4 are enabled at times t_1, t_2, t_3 respectively.
These ideas are made concrete in the implementation of the case study in appendix B.

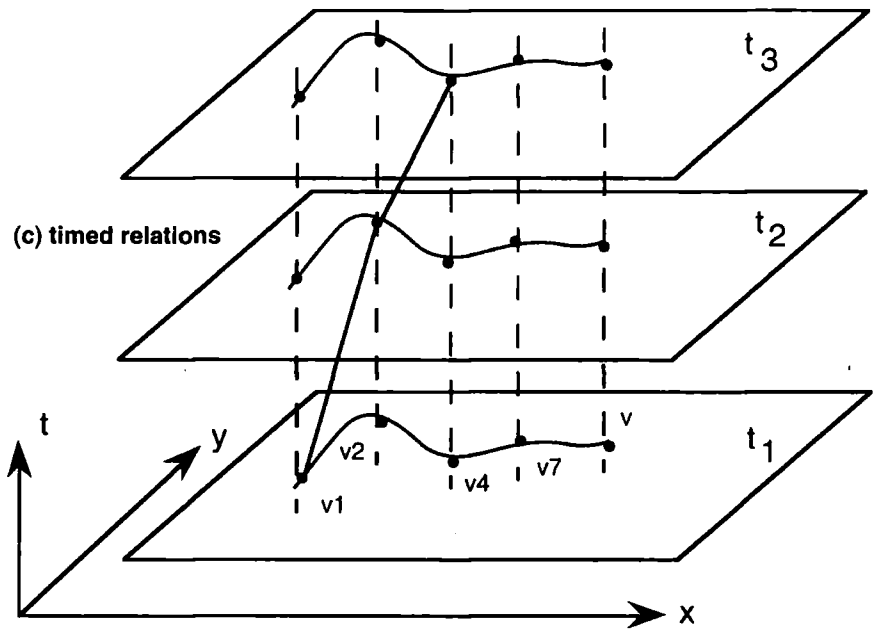


Figure 5.2 (c) Conceptual definition of states' convergent relations

5.2.2 Convergent sequences of timed relations

The distance relation and the time attribute relation among points can be exploited in specifying system functions by use of the theory of metric spaces.

A metric space is a set E, together with a rule which associates with each pair p, q $\in$ E a real number d(p,q) such that

(i) $d(p,q) \geq 0$ for all $p, q \in E$

(ii) $d(p,q) = 0$ if and only if $p=q$

(iii) $d(p,q) = d(q,p)$ for all $p,q \in E$

(iv) $d(p,r) \leq d(p,q) + d(q,r)$ for all $p,q,r, \in E$ (triangle inequality)

Thus a metric space is an ordered pair (E,d), where E is a set and d a function

$d: E \times E \rightarrow R$

satisfying properties (i) - (iv) above.

Let $p_1, p_2, p_3, \ldots$ be a sequence of points in the metric space E. A point $p \in E$ is called a limit of the sequence if, given any real number $\varepsilon > 0$, there is a positive integer N such that $d(p,p_n)<\varepsilon$ whenever $n>N$. If the sequence has a limit it is said to be convergent; if the limit is p, the sequence is said to converge to p.

The notion of metric spaces does not possess sufficient relational structure for implementation as limit processes of system functions. It is however a good starting point since a weaker concept, the notion of an **inductive limit**, can be defined over states, to resemble the limit structure of metric spaces. The approximation technique of the inductive limit is described in appendix C.

Sets of sequences of timed relations illustrated in figure 5.2 can thus be constructed, fused together to reflect possible system behaviours and given appropriate metrics to serve as models for system functions.

5.3 Classification of system state evolution

5.3.1 Structuring a state space into subsets of the space

The metric space on its own does not provide a sufficiently strong theory for problem independent classification of system states generated in the realization of system functions. What is required is a collection of algebraic laws that provide a timed structure for sets of values denoting configured data, observed and computed states. The destination of this exposition is the notion of **permutations** on sytem states introduced in chapter three. A good starting point is to define a mapping

f: A $\rightarrow$ B from a domain A to a co -domain B to be a subset of $A \times B$ such that

$(\forall a \in A)\ (\exists! b \in B)\ [(a,b) \in f]$

Note that the exclamation emphasises the definite existence of b; A and B represent some components of system state space. This approach provides a way of defining functions as relations over state spaces, thereby opening up the possibility of representing system properties as relations.

Given a state space E, a binary operation (law of composition) can be defined as:

$$\bullet : E \times E \rightarrow E,$$

whereby elements x, y of E can be combined to form another element $x \bullet y$ of E. Such a system is called a groupoid.

The equality relation can then be applied in generating richer structures over the set E.

When the operation $\bullet$ is associative on E, the law is said to form a semigroup, if

$$(\forall\ x,y,z \in E)\quad x\bullet(y\bullet z) = (x\bullet y)\bullet z\ .$$

If the groupoid E has laws such that

i) $(\forall x \in E)\quad e\bullet x = x$

ii) $(\forall x \in E)\quad x\bullet e = x$

iii) $(\forall x \in E)\quad e\bullet x = x = x\bullet e\ ,$

it is said that the groupoid i) has a left identity, ii) has a right identity, iii) has a two sided identity (or simply an identity).

The composition, associative and identity laws can be specialized in many ways to characterize system state spaces. Some important examples of such specialization are as follows. Denote an arbitrary binary operation on E as Map(E,E). By starting with composition of mappings as a law of composition on Map (E,E), and denoting such composition operator by $\circ$, the system $(Map(E,E),\circ)$ is a semigroup. Addition and multiplication of real numbers are associative laws of composition so (R, +) and (R, .) are semigroups. The power set $\wp(E)$, together with the union $\cup$ forms a semigroup $(\wp(E),\cup)$ with identity $\emptyset$ the null element. Also $(\wp(E),\cap)$ forms a semigroup with identity E.

Let $(S, \bullet)$ be a semigroup with an identity e, and let G be the set of invertible elements of S. Then $(G,\bullet)$ is a group. In the semigroup $(Map(E,E),\circ)$ the invertible elements are the bijections. Thus the set Bij(E,E) of all bijections $f: E \rightarrow E$ is therefore a group under composition of mappings. Recall that in a finite set E consisting of n elements, a bijection $f: E \rightarrow E$ is called a

permutation on E. The group thus formed is the symmetric group on n symbols S_n. Since there are n! permutations on a set of n elements, S_n has n! elements.

5.3.2 Statistical values, probability values and the timed automaton

The notion of relations over convergent points coupled with the laws of semigroups provide powerful model generation semantics for specification of all kinds of system functions encountered in telecommunications networks. The traditional approach to generating system behaviours is to write down equations involving some form of sequences of operations with values specialized by use of power series. These sequences are often denoted using differentiated and integral operations (see e.g. Cohn P.M [1991], Aliprantis C.D. et al. [1990]).

A nice way to build up measures and integrals is to combine the specialized semigroup operations <+, ·> and group operations into the semiring structure from which measures and integrals are defined. Statistics relating to specific experiments are represented as relations over convergent points. Classification of sample spaces to generate state event spaces as probability measures is carried out by partitioning the sample spaces and mapping subsets to bounded real values (see e.g. Mane R. [1987]).

The notion of an abstract machine outlined in chapter three can be recovered in terms of semigroups and automata.

An alphabet B is a finite nonvoid set. The elements of B are called letters. The number of occurrences of letters in a word p is the length of p. For the empty word, i.e. for word of length 0, denote it as e. The set of all words over B is denoted as B*. Multiplication can be introduced in B* as follows:

Concatenation of words, i.e. for arbitrary two words $p=b_1 \ldots b_m$ and $q=b_{m+1} \ldots b_n$

$(b_i \in B, i = 1, \ldots, n)$,

$pq = b_1 \ldots b_m\, b_{m+1} \ldots b_n$.

Moreover, for arbitrary $p \in B^*$ and $n \in N$ (natural numbers), the power p^n is defined by $p^0 = e$ and

$p^n = pp^{n-1}$ if $n>o$.

Under the above multiplication B* forms a monoid, i.e. a semigroup with identity element. An explicit representation of this multiplication is presented in chapter appendix C. This intuitive description of multiplication is applied in appendix A in preparation for the detailed exposition in appendix C.

A system T = (B, A, δ) is an **automaton**, where

i) B is an alphabet of input signals called the input alphabet,
ii) A is the nonvoid set of states,
iii) δ is a mapping of A × B into A called the transition function (or next state function).

The automaton T is finite if A is finite. A system M = (B, A, Y,δ, μ) is a sequential machine (or a Mealy machine), where

i) B is an alphabet of input signals, the input alphabet,
ii) A is a nonvoid set of states,
iii) Y is an alphabet of output signals, the output alphabet,
iv) δ is the transition function,
v) μ is the output function.

The sequential machine is finite if A is finite. T is the underlying automaton of the sequential machine M.

5.3.2.1 Deterministic and non-deterministic states

It is important to note how non-deterministic states can be interpreted within the automaton framework defined in the previous sub-section. An automaton can be used to simulate a user, demanding to consume resources over time. In this case, the internal states of the automaton is non-deterministic and the output of the automaton as seen by invoked system functions is also non-deterministic.

When an automaton of a sequential machine represents system functions, the objective of the representation is to satisfy system behaviour goals despite imprecise knowledge of inputs from user demands. In such a case, the approach to modelling system functions is to generate a sufficiently large state space for the control automata such that configuration data and imprecise user demands for resources can be modelled. The actions of the automata operate on

constrained data sets in such a way that system goals are satisfied. Such constrained operations can be carried out in the following way.

Given two state sets C and D, relations can be defined over these sets. Also total functions can be defined over these sets, or partial functions can be defined over subsets such that

$$f : C \rightarrow D, \text{ domain } [f] \subseteq C$$
$$\text{or image } [f] \subseteq D .$$

The semigroup operations and metric space structure for sets still hold under such cases. In terms of representations for system behaviours, state values are generated as follows, the examples being taken from the theory of groups.

Using the following notation for values of functions,

$(c)f = d$ in the function $f : C \rightarrow D$,

A group is a structure $(G,\alpha,\beta,1)$ of sets in which α is a function $G\times G \rightarrow G$, β is a function $G \rightarrow G$ and $1 \in G$. The associativity law is:

$((c,d)\alpha,g)\alpha = (c, (d,g)\alpha)\alpha$ for all $c,d,g \in G$

The identity law is:

$(1,c)\alpha = c = (c,1)\alpha$, for all $c \in G$.

and the inverse law is:

$(c,c\beta)\alpha = 1 = (c\beta,c)\alpha$, for all $c \in G$.

These concepts are implemented in a more verbose way in the case studies (appendices A and B).

If an automaton executes an action at every clock tick of its time, the state transition functions result in some *group* or *semigroup laws* holding over system state data. Some non-relevant values may also be generated, so need to be recognized and ignored; this is the classification task. The action of automata are specified to be biased in such a way that meaningful sets of states are observed to hold at specified time windows. This concept is the notion of the **timed automaton** developed in this thesis.

5.4 Admissible functions and constraints for performance characterisation

It is instructive to compare the notion of timed automaton described in the previous section to classical variational problems, the sources of several performance characterisation research problems. As an example, the classical pendulum can be described as a variational problem

with side conditions (see Brechtken-Manderscheid U.[1991]).

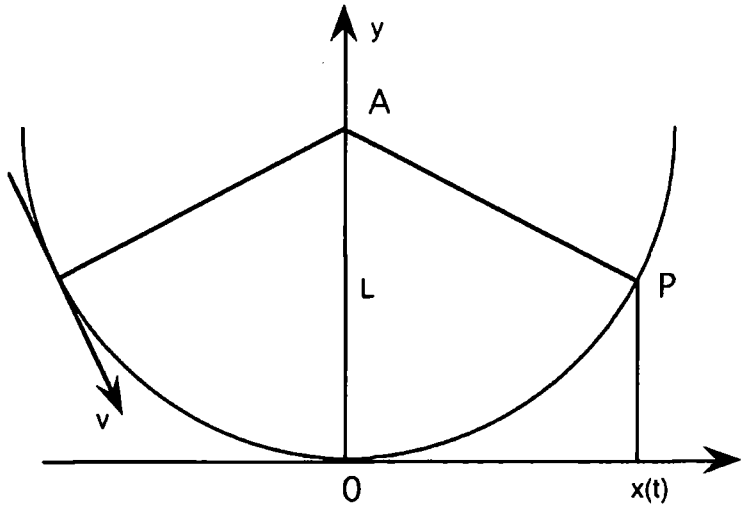


Figure 5.3 The mathematical pendulum

The mass of the pendulum is attached to a taut string of length L and fixed at point A = (0, L). Assume that at time t_0 the bob is displaced to a certain initial position (x_0, y_0), and that it has an initial velocity $[v_1, v_2]$ as a vector. An expression is required for the path (x(t), y(t)) of the bob.

Using Hamiltonian principle for conservation of energy, an expression for stationary energy quantity (resource being allocated and re-allocated) is as:

$$J(x,y) = \int_0^t [T(t)-U(t)]\ dt$$

where T(t) denotes the kinetic energy and U(t) the potential energy of the pendulum at time t. Admissible functions are the piecewise smooth vector functions (x,y) with time as the independent variable. The initial conditions

$$x\ (o) = x_O\ , \qquad y(o) = y_O$$

$$x\ '\ (o) = v_1, \qquad y\ '\ (o) = v_2\ .$$

Constraints of the behaviour expressions are:
for all t, points (x(t), y(t)) lie on a circle with centre at A, and radius L given by

$$x^2(t) + (y\ (t) - L)^2 = L^2$$

This is the standard equation of a circle with centre (0,L) and locus traced by P(x,y).
Thus the integral operator provides the generator for system state evolution, subject to the constraint relation stated in terms of equations. Generated in this thesis is a state evolution structure that is much more general than the integral operator, and allows for easier composition of system functions. In the mathematical programming work area, the state generator is normally represented as soluble differential expressions within Banach spaces (see e.g. Craven, B. D.[1978]). Note that Banach spaces can be built up as vector spaces and metric spaces; vector spaces can be built up from groups, embedding semigroups.

Optimisation and performance characterisation tasks involve selection of paths as sequences within admissible functions in such a way that various performance constraints hold. Given the complexity and large size of system functions encountered in practice, it is a major challenge to specify plausible sequences of admissible functions. Such an exercise should state the optimization criteria that should hold in the execution of the system functions.

5.5 Product form representation of timed automata

The physical example of the simple pendulum described in the previous section illustrates how a system evolution's behaviour is presented using the notion of generators and relations. This concept provides a far reaching procedure for specification of system functions. It is, however, not practical to write down a single expression for the governing relations holding together a collection of system functions. It is therefore more convenient to compose system functions in such a way that the realized operational sequences maintain the desired effect (e.g. in the allocation of shared resources to users).

In chapter three, while reviewing modelling concepts for distributed algorithms, the notions of abstract data types and abstract machines were introduced. These are standard concepts in modern theory of computation. In such a context, a semigroup is an algebra with an associative binary operation. In composing system functions, one is in effect multiplying abstract machines. In algebraic terms, one is fusing together a number of semigroups.

The semigroup amalgam $[S_i;U]_{i\in I}$ is an indexed family $\{S_i;i\in I\}$ of semigroups, the pairwise intersection of which is a common core semigroup U. Since the core of an amalgam is the unifying object in its structure, there are embeddability implications due to the character of the core. Recall that the final *outcome set* as results of computations carried out by an abstract data type can be indexed by a natural number. For each computation fragment of a system function, results obtained can be typed, e.g. denoted as μx, μ being the type indicator for a result item of type x within a result set. Thus a specific result can be chosen from a result set,

the type of the result being x, and some property holding on the chosen result. The property is denoted as p(x), and the choice is denoted as

$\nu\{\mu x{:}p(x)\}$, where ν is the choice operator,

saying that for some type x (e.g. real numbers), there are choice elements such that property (p) holds.

Thus properties such as ordering and commutativity can be expressed to hold on values generated in computations based on semigroup algebras. This approach allows for a type based specification of what would otherwise be an intractable problem.

A self contained specification of the storage functions example is presented in appendix A. The framework for the specification is presented in Nyong, D. [1993], where it is shown how the operations executed by system functions are named in terms of interfaces of objects that host the operations. Thus the notion of **Interface Reference** is described as a typed entity reflecting the quality of service implications in the invocation of operations that are executed at remote locations.

In the case study, four system functions are specified:

ACTIVATE, PASSIVATE, MOVE and MIGRATE .

These system functions are then amalgamated in order to ensure that concurrency issues such as commutativity and abortion of computation sequences (also referred to as activities in appendix A) are effected in an efficient manner.

The size and complexity of this case study serves to emphasise the need for an expressive modelling semantics for distributed algorithms. By basing such a specification on a rigorous set of operational semantics framework, correctness of the specification follows closely the adherence to well-defined algebraic and analytic concepts.

5.6 Summary

An introductory development has been presented of the interplay between declarative operational sequences and the geometry of system states as induced by the operations. The notion of the timed automaton has been defined and shown to be compatible with operational

sequences that can represent products of system functions. These concepts though standard in the mathematics literature, have been applied in a specific way in the area of telecommunications system functions. The specification of the storage function in appendix A though large in scope, can thus be read using the presented small set of concepts.

In the next chapter, the notion of geometric paths is developed in more detail and linked to the logical framework aimed at satisfying individual user and population goals.

Chapter six

Timed automaton: the graph model

6.1 Introduction

The specification of a system function for a telecommunications system needs to be sufficiently flexible in order to capture faithfully the non-deterministic nature of event arrivals at the user-network interface. Given a specification expressing the behaviours of a collection of system functions, there is a need to base an assessment of the correctness and practical realizability of such a specification on some logical framework. The timed automaton graph model presented in this chapter aims to provide a logical oriented operational semantics for the specification of system functions.

Section 6.2.1 is a presentation of the interplay between the operational nature of system functions that satisfy a population of users, and the logical basis for generating control rules in satisfying system goals. Section 6.2.2 develops an operational semantics from basic logical concepts, culminating in a four point definition of the **graph modelled timed automaton**. A formal definition is presented for conceptual sketches of geometric curves; these curves are used to illustrate the valuation framework for states of a system as the system executes system functions. This geometric approach is put to practice in the next chapter which illustrates through a case study, the use of these concepts in the realization of a large scale system function modelled for performance characterization of system behaviours.

This chapter makes an original contribution in developing a basic structuring framework for declared data as user and population objects over which logical propositions can be extended.

6.2 Policy criteria for resource allocation

6.2.1 The logical structure of system function specifications

The behaviours of distributed systems executing system functions can be described in theoretical terms as the animation of logical arguments. With such an approach, logical arguments are applied to systems whose operational framework are based on predictions of expected events. Thus there is a need to develop a structure on which arguments about system behaviours can be based. Two components for the analysis of logical arguments are the premisses and the conclusions. Premisses

are the propositions which provide grounds for conclusions, and also additional propositions to be affirmed. A valid and sound argument about a system behaviour is deduced when its premisses and conclusion are so related such that the premisses are true and the conclusion are also true. The soundness component encompasses both logical validity and establishing the truth or falsity of premisses. The latter is normally the task of a physical science enquiry but in the specification of system functions, premisses must be stated explicitly for both deterministic and non-deterministic phenomena.

Figure 6.1 illustrates how the goal oriented policy framework introduced in chapter 3 is being enhanced to serve as a logical basis for the development of operational rules in animating system behaviours. The term operational logic is used loosely in that diagram to mean the sequences of operations executed by the system in real time, as the objects that encapsulate system functions interact with each other, and their environments.

Several inter-related deductions are normally specified for a set of system functions in formulating soundness, and optimality criteria. The declarations of policy criteria illustrated in figure 6.1 can be interpreted simply as sequences of relations over observable and declared data values. These sequences are named as behaviours, goals and constrained property evolution. The arrows indicate how some properties are exported to other properties (from tail to head of arrow), and how properties overlap with each other.

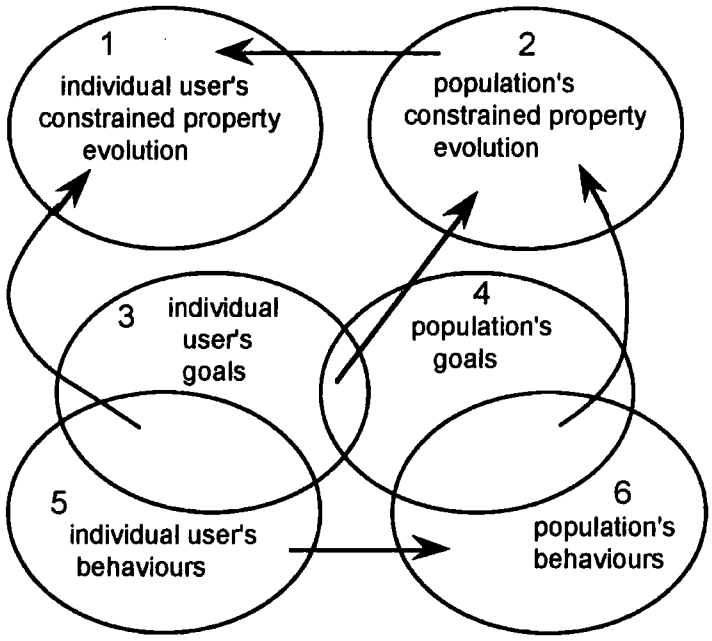
An important point to recall in the declarative approach is that predictions are based on previously observed characteristic phenomena and are therefore carried out over recurrent time windows. On figure 6.1, advanced policy enforcement rules are effected at operation sequence groups 4 and 2 respectively. It should also be noted that operational sequences group 1 use more immediate gathered data for decision making whereas operational sequences group 2 use stable statistical data for decision making.

A detailed description of the logical development of this declarative scheme is presented in the rest of this chapter. A practical case study on the use of this scheme is presented in the next chapter.

6.2.2 Combining theories and generating models

A collection of theoretical concepts have been proposed in chapters 3 and 5 to serve as the basis for model generation in the specification of system functions. In particular, the concepts that serve as foundations in specifications are:

- limit processes in metric spaces,
- semigroup algebras as classifying concepts for system state spaces.



1 As derived by operational logic within network, using derived information denoting population's resource demands, individual user's current actions, configured resource allocation rules, and resulting consumption patterns	2 As derived by operational logic within network, using derived information denoting population's resource demands, configured resource allocation rules, and resulting consumption patterns
3 As declared by users in bids for network resources	4 As derived by operational logic within network, from observed user goal declarations
5 As observed by operational logic within network from user bids for network resources	6 As observed by operational logic within network from user bids for network resources

Figure 6.1 Declarations of policy criteria for resource allocation

A standard reference describing the basic concepts of model theory is Chang C. C. and H. J. Keisler [1990]. A brief overview of the definition of models and theories is presented in this section. One approach to exposing the modelling concepts developed in this thesis is to adopt a linguistic framework as follows: specify a list of symbols and then give precise rules by which sentences can be built up from the symbols. The linguistic approach is not adopted in this work mainly because techniques for modelling large scale system functions are still in their infancy. Thus it is more useful to explore how any developed technique scales up by applying the concepts on realistic case studies. A set theoretic framework is adopted instead of the linguistic approach since basic mathematical theories and their models can then be applied in describing the behaviours of objects executing system functions.

6.2.2.1 Models from functions and relations

Formally, a starting point for generation of models is the classical sentential logic. Syntactically, symbols are built into words which can then be used to construct sentences. Propositions are statements which are built up using sentences. However, it should be noted that a sentence can be used to make different statements depending on the context in which the sentence is uttered. Propositions are traditionally true or false. However, where Bayesian inference is used to derive possible states of system functions executed at remote nodes, a proposition may be neither exactly true nor exactly false, just possibly true or possible false.

Starting from set theory, simple and compound statements can be made. A simple statement contains no other statement apart from itself. Compound statements contain other statements as a component part. Given a set of simple statements ζ , a function F associates with each simple statement s the values *true* , *false* (or indeterminate, in the case of states within remote objects).

A model is a subset A of ζ; $s \in A$ indicates that the statement s is true, and $s \notin A$ indicates that the simple statement is false. A may be partitioned to cater for the indeterminate state. Note that the set of all models for ζ has power $2^{|\zeta|}$, from the basic notion of power sets. A sentence φ is true in a model A by writing

$$A \models \varphi$$

Other utterances are: φ is true in A, or φ holds in A. An important semantical notion is that a sentence φ is called valid if φ holds in all models for ζ. Thus for valid sentences, one writes

$$\models \varphi$$

In formal linguistic terms, syntactic denotations play an important part in describing system evolution. In this work, the notion of timed executions of traces within graphs provide the syntax. Thus operations executed by objects generate sentence symbols (as illustrated in the storage model case study). Thus let φ be a sentence and $s_0, \ldots, s_n$ be all the sentence symbols occurring in φ. In this case, φ is said to be a tautology if φ has the value true for every assignment $a_0, \ldots a_n$ of values to the sentence symbols. Thus

$$\vdash \varphi .$$

At this point it is convenient to introduce the notion of graphs and hypergraphs as the syntactic framework for denoting sequences of values of sentence symbols. The goal for such an exercise is to exploit the completeness theorem of sentential calculus,

$$\vdash \varphi \quad \text{iff} \quad \models \varphi$$

Stating that a sentence is a tautology if and only if it is valid. Thus the detachment inference can be used for rewriting as follows:

$$\text{From } \psi \text{ and } \psi \Rightarrow \varphi, \text{ infer } \varphi \quad .$$

Since some sentence symbols cannot always be assigned absolute *true* or *false* values in modelling a distributed system, an agent's assignment of values is often sufficient for making progress in ensuring a system's state sequences converge to fixed point sets.

In generating timed automata, a starting point is set Y representing a set of system states. The power set of Y can be generated as usual by considering all subsets of Y. When the state set is large, an easier way of handling the state permutations is to consider the so-called r-subsets i.e. subsets of size r, denoted as $Y^{(r)}$. Thus with such a structure, an r- hypergraph can be defined as a pair (Y,ε) where $\varepsilon \subset Y^{(r)}$. An element ε is an edge of the hypergraph. A point x is said to be joined to a point y if $\{x,y\}$ is an edge of the graph. Given the powerset $\wp(Y)$, and edges $\varepsilon \subset Y^{(r)}$ for all $r = o, \ldots, n$ where n is generated by the size of $\wp(Y)$, the semigroup operations of $(\wp(Y),\cup)$ can be used to join hyper-edges together in generating a graph structure over the set Y.

The elements of a set Q with the hypergraph structure and the semigroup structure can be organised as a POSET = (Q, <) in representing how system properties evolve over time. An element x of a

poset is said to cover an element y if x>y and no element z satisfies x>z>y. Thus the covering graph of POSET has vertex set Q and x is joined to y if x covers y or y covers x. A representation of this relational structure that is easier to visualize is presented in appendix C where mapping diagrams from category theory is used.

Figure 6.2 illustrates the transformation procedure carried out in this study to provide structures for a large system state set generated for a collection of system functions. The metric space structure is used to ensure that the sample clock ticks for a system's operational decision making is sufficiently fine that no valuable behaviour observations are missed. The semigroup structure classifies the large state space into graph-based relations which denote specific system properties that hold at various instants in time.

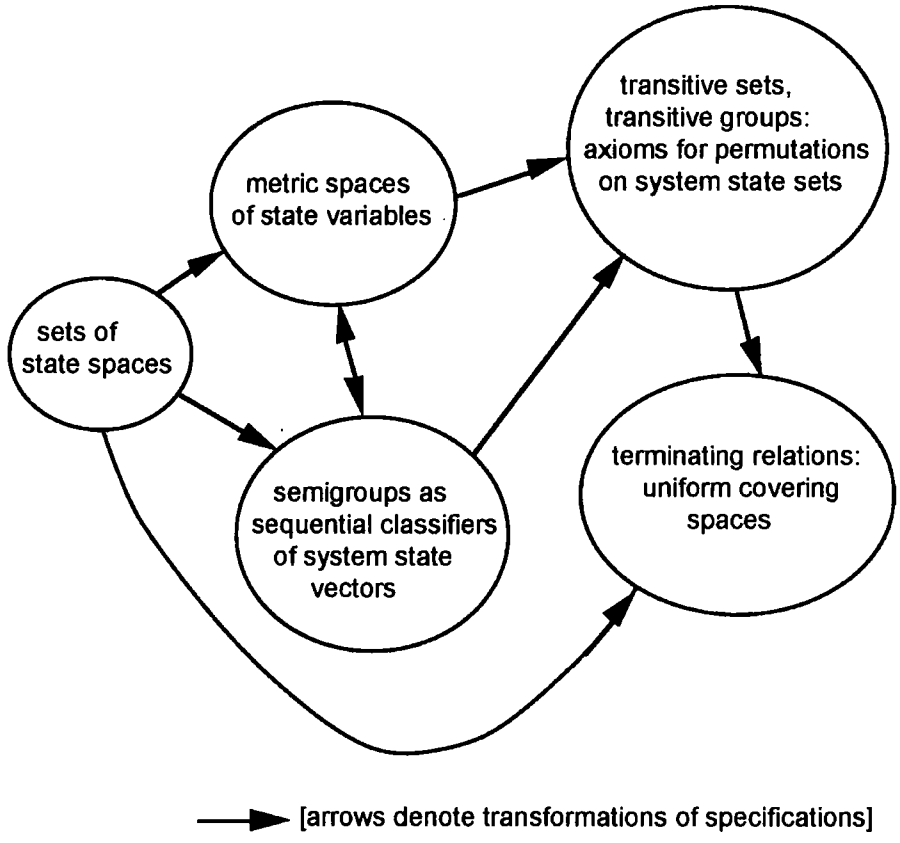


Figure 6.2 Generators for system state spaces

Mathematical structures are built up from axiomatic theories which serve as a generic constraining basis for models. As illustrated in figure 6.1, within the framework of metric spaces and semigroup

algebras, theories of transitive sets and transitive groups can be deployed to provide constraint in the specification of system behaviour properties. The following section is a description of how timed automata can be generated from theories.

6.2.2.2 Timed automata from theories

Given a finite or infinite set Σ of ζ (i.e. simple statements) a sentence φ can be deduced from Σ, written

$$\Sigma \vdash \varphi$$

if and only if there is a finite sequence $\psi_0, \psi_1, \ldots, \psi_n$ of sentences such that $\varphi = \psi_n$ and each sentence ψ_m a member of the sequence is either valid, belongs to Σ or is inferred from earlier sentences of the sequence by detachment. The sequence above is thus termed the deduction of φ from Σ. By definition Σ is inconsistent if and only if

$$\Sigma \vdash \varphi$$

for all sentences φ otherwise Σ is consistent. A is a model of Σ

$$A \models \Sigma$$

if and only if every sentence $\varphi \in \Sigma$ is true in A; Σ is said to be satisfiable if and only if it has at least one model. φ is a consequence of Σ if and only if every model of Σ is a model of φ. Using these definitions, the notion of a theory can be defined.

A set Γ of sentences is called a theory; a theory is said to be closed if and only if every consequence of Γ belongs to Γ. A set μ of sentences is a set of axioms for a theory Γ if and only if Γ and μ have the same consequences. A theory is finitely axiomatisable if and only if it has a finite set of axioms. Thus μ is a set of axioms for a theory if and only if μ has exactly the same models as Γ.

Figure 6.3 illustrates how the hypergraph - based operational framework is derived as a model structure from a collection of theories, and then transformed into a standard automata theoretic operational framework. This approach which forms a powerful specification and implementation procedure is described in this sub-section.

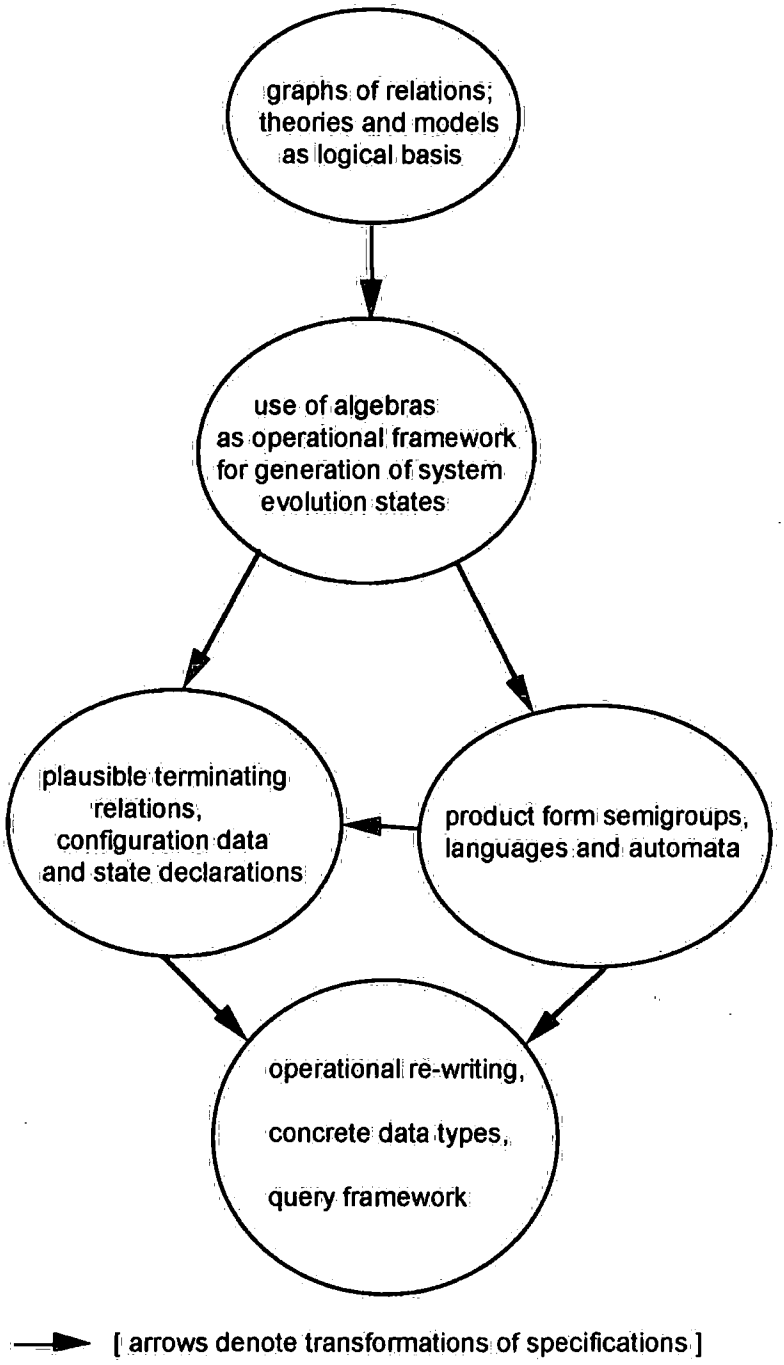


Figure 6.3 Definition of timed automata from algebraic framework

The graph automaton: point 1 ~ operations on state sets.

Recall from chapter two (see section on algorithm framework for computations) that the notion of an L- model is a system comprising the n-ary relation, the n-ary function and the constant c. Each of these components is a structure built from a non-empty set, the universe of the L-model.

n-ary relations are built up from binary relations through pairing of ordered n-tuples. Also, starting from relations, n-ary functions can be built up from the constituent relations. These n-ary functions are effectively algebra's

$$\alpha = (A, \Sigma^{\alpha})$$

where Σ^{α} is a set of operations on A, and

$\sigma : A^n \rightarrow A$ is an n-ary operation on A. Where A is a non-void set, $n \geq 0$ is an integer, and every $\sigma \in \Sigma^{\alpha}$.

There is a need to ensure that operations on state sets can be traced back to behaviours of real world entities. It is therefore important that data values operated upon in the realization of system functions are constructed in a consistent way. The entity relationship paradigm provides a good starting point for building a formal model for the generation of system evolution states (see e.g. Vossen G. [1991]).

An entity, e.g. a transmission channel, possesses a fixed number of properties. A property of an entity such as a transmission channel could be that it transports information bit streams in an asynchronous end-to-end clocking; another property could characterise the burstiness acceptable in the transport of information streams over the channel. Each property is allowed to take a finite set of unique atomic values. Figure 6.4 illustrates this concept.

When a collection of entities share a property, such a property is referred to as an attribute of the entities. As is defined for any system property, attributes have a domain of unique atomic values. It is also useful to classify attributes of an entity set as being time-invariant or time-varying. Thus time-varying attributes can take different values at various instants in time. The notion of a system evolution can therefore be described in terms of relations that hold due to the following.

a) observed values of properties as generated by reactive agents within the environments of a distributed system,

b) derived values of properties as computed by the operational logic within a distributed system, after carrying out a sequence of observations of its environment,

c) a number of decisions effected by the operational logic within a distributed system, after carrying out steps a) and b).

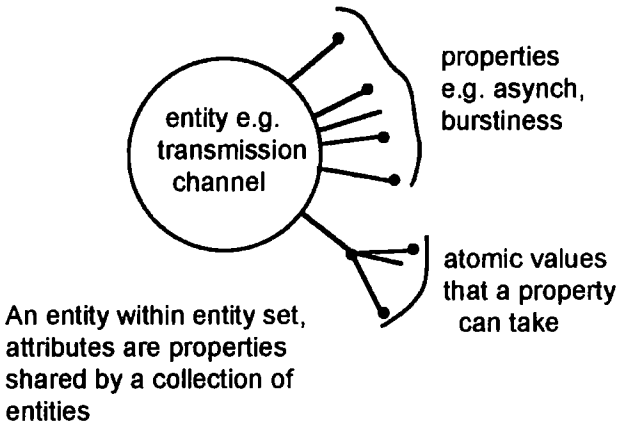
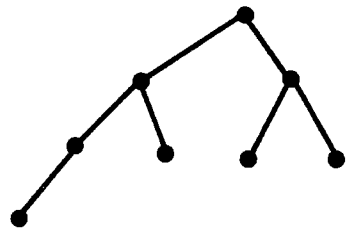


Figure 6.4 Definition of system attributes

Figure 6.5 illustrates how system property trees and hypergraphs are constructed using the Polish notation for standard propositional logic (see e.g. Korfhage R. R. [1966]). These concepts form the basis for specifications through transformations as shown in figure 6.3. Languages and tree automata constitute a topic of great depth and importance; it is outside the scope of this thesis. The timed automaton is built around tree automata, exploiting any inherent geometric characteristics of system attribute value evolution.

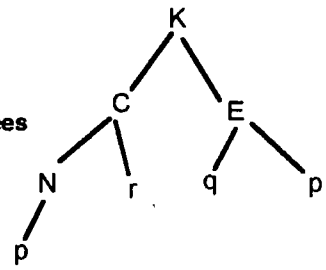
The graph automaton: point 2 ~ curves for attribute values.

Figure 6.6(i) illustrates how values of system evolution state variables are plotted in a standard way. The h- axis represents a range for the attribute values of some entity set, as the values vary over a recurrent time span. In practice, these values are obtained through observations or derivations as described in point 1. The recurrent time span is selected to represent a period when sufficient information is captured about a distributed system, as the system provides a service to its environment (i.e. users). An example recurrent pattern could be the period between 9.00 a.m. and



(i) A tree of relational propositions

(ii) Polish notation for relational proposition trees



KCNprEqp

Polish notation	Standard notation
Np	~p
Cpq	$p \rightarrow q$
Apq	$p \vee q$
Kpq	$p \wedge q$
Epq	$p \equiv q$

Figure 6.5 Construction of system property trees

3.00 p.m. of every day, irrespective of the day of the week, or the week of the year. During a recurrent time span, attributes are computed at appropriately selected snapshot intervals. Criteria for determining various snapshot intervals within a recurrent time span can only be determined by trial and error, when considering a specific system function and the behaviour characteristics of an in-situ distributed system.

Assume that a sequence of clock ticks of appropriate interval has been enabled for carrying out observations, and executing derivations. Attributes common to two entity sets can be represented by a curve in the two-dimensional Euclidean space. Such a curve represents an *admissible sequence of attribute evolution values.* Figure 6.6(ii)-(iv) illustrates such a curve obtained as follows. Take the variable t in R, the real numbers, as the time variable. Consider only functions defined on a closed interval [a,b] and let P(t) denote the image of t. [a,b] can be considered as a time window within which countably many snapshots are taken, and P(t) the value of a system state variable in R^2 , a plane, at time t.

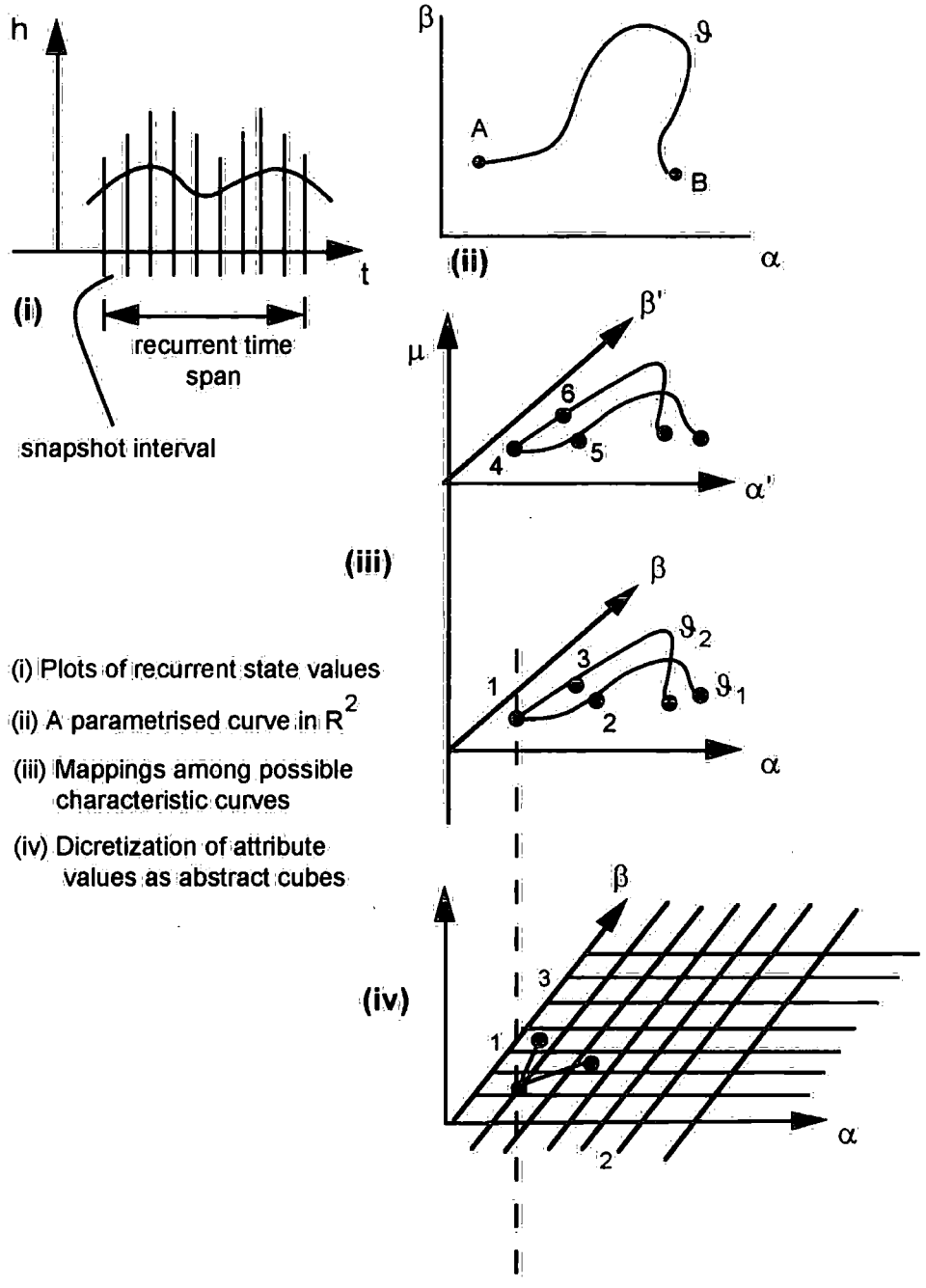


Figure 6.6 Representation of system property curves as points sets

Considering P(t) as having co-ordinates ($\alpha(t)$, $\beta(t)$). P is said to be continuous if $\alpha(t)$ and $\beta(t)$ are continuous respectively. Thus the mapping denoting a parametrised curve is

$$P: [a, b] \rightarrow (\alpha(t), \beta(t))$$

Assume that the first and second derivatives of P exist, and are continuous, and that the partial derivatives with respect to α, β and t are non-zero (i.e. $P' \neq 0$, $P'(t) = (\alpha'(t), \beta'(t))$.

Referring again to figure 6.6, a directed curve in R^2 is a triple $<\vartheta, A, B>$ consisting of a set of points ϑ in R^2 and points A, B in ϑ called the *initial* and *final* points of ϑ. Parametrisation of the curves as the mapping

$$P : (a, b) \rightarrow R^2$$

satisfies the following conditions:

i) P ' and P " both exist and are continuous
ii) $P'(t) \neq 0$ for all $t \in [a, b]$
iii) $P([a,b]) = \vartheta$.
iv) P(a) = A and P(b) =B
v) if A = B then, as t increases, P(t) moves around ϑ in an anti-clockwise direction.

The tuple of points on the curve P(t) are often mapped onto an attribute value space (e.g. h(t) of figure 6.6(i)) in order to give the curve a specific interpretation with respect to the system function being implemented. Assume that the co-ordinate pair <h,t> is reserved for displaying specific attribute values. The curve for a pair of entity sets can be used to construct a three dimensional curve ϖ, the 3-tuple of attribute values being $<\alpha, \beta, \mu>$. μ, the third co-ordinate provides values for the specific attributes h(t). This process can be continued recursively thereby providing a declarative schema for the specification of system property evolution. A detailed description of analytic expressions useful for generating basic three-dimensional curves can be found in Dineen S. [1995], and O'Neill, B. [1966]. Linear and non-linear relations as models for attribute values are described fully under 'Model Checking and Multiple Regression' in Bhattacharyya, G. K. and Johnson, R.A. [1997].

The graph automaton: point 3 ~ stability of attribute membership within curve pattern.

The specification of a system property evolution as the realization of a system function would normally be sufficiently flexible in that the specified attribute evolution sequences represent all the admissible behaviours when the system interacts with its environment in executing the function. Thus a system function's attribute evolution sequences can be represented as a collection of

admissible n-dimensional curves. Each admissible curve possesses sufficient geometric discriminant characteristics which can be used by a suitable algorithm to identify the curve within its membership collection. Since points that denote such attribute evolution curves are generated using statistical input data from the environment of the distributed system under study, there is a need to develop an appropriate mathematical structure for use in evaluating the stability characteristics of a set of points with respect to a specific curve when such points can belong to more than one curve within a membership collection.

Figure 6.6(iii) is an example system of two attribute evolution curves in ϑ_1 and ϑ_2 in E^2, which can be valuated in E^3 as ϖ_1 and ϖ_2. These curves have practical interpretation as follows. At time t_i, the value of a state variable as an attribute value is $<\alpha_i, \beta_i, \mu_i>$. In the generic case, collect together all the points $<\alpha, \beta, \mu>$ into sets:

Λ the α values, $\prod$ the β values and Φ the μ values.

Thus given the set $< \Lambda, \prod, \Phi>$, the discriminating algorithm identifies characteristic ϖ curves as ϖ_1, ϖ_2 etc., ensuring that each point set for a characteristic curve is stable with respect to membership of the points as they belong to a curve. The Hurewicz and Wallman stability specification is a mathematical structure for identifying such characteristic curves (as in Hurewicz W. and Wallman H.[1969] : 'Mappings in Spheres and Applications').

The Hurewicz - Wallman structure embraces both analytic formulations for convergence of mappings into a cube, and algebraic formulations for generating point sets as the expressive Hilbert Spaces. However, these structures do not provide ways of discriminating among characteristics of curves. Statistical analysis techniques are required for expressing the characteristics of curve patterns within a membership collection. Such statistical techniques belong to a vast work area comprising Non-linear Multilevel Models (see Goldstein H. [1995]: Model Checking and Multiple Regression and also see Bhattacharyya G. K. and Johnson R. A. [1977]). Such statistical analysis techniques are incorporated in the timed automaton as computations executed by auxiliary routines while a system function is tracing through an attribute curve.

Referring to figure 6.6(iii), point 1 on curves ϑ_1 and ϑ_2 denote the $<\alpha_i, \beta_i, \mu_i>$ value of the attribute of a number of entity sets. As time progresses to t_{i+1}, point 1 transforms to point 6 or point 5, depending on whether the invoked statistical inference has placed the attribute sequence on curve $\vartheta_2$or ϑ_1, respectively. Point 4 maintains a history trace of previous values in the sequence, when restricting a curve to an E^2 plane. Figure 6.6(iv) illustrates in E^2 how a choice of stable

transition is made from point 1 (projected on a current plane, from a plane in the previous clock tick), to point 2 or point 3 on the same or a different curve, as evaluated in the execution of a system computation. The implementation of these concepts is illustrated using the congestion avoidance case study in the next chapter.

The graph automaton: point 4 ~ system evolution states as canonical mappings.

The characteristic ϖ curves presented in figure 6.6 are values of relational propositions which are generated as specifications that obey logical connectives (recall the tree of relational propositions presented in figure 6.5). In algebraic terms, the characteristic curves are specified as relational structures which can be composed or refined (decomposed) over several steps. The calculus of relations pioneered by C.C. Chang, B. Jonsson, and A. Tarski during 1961-1967, and developed further in McKenzie R.N. et al. [1987] provides a comprehensive framework for the generation of relational specifications.

Given a non empty set X representing points denoting admissible attribute values for a collection of system functions, a semigroup can be defined on the set X as follows:

$$P\,\tau_X \text{ consists of all functions } \nu\text{: dom } \nu \rightarrow \text{ ran } \nu \ .$$

where dom ν and ran ν are subsets of X, and $P\,\tau_X$ is the partial transformation semigroup on X. The set of binary relations on X (subsets of $X \times X$) forms a semigroup B_X under the relational product as follows:

$$\tau \circ \lambda = \{(a,c) \in X \times X \mid \exists\ b \in X \text{ such that } (a,b) \in \tau,\ (b,c) \in \lambda \ \}.$$

If these relations are functions, this product reduces to function composition and thus $P\,\tau_X$ is a sub-semigroup of B_X.

Thus the theory of partial transformation semigroup provides the logical rules for the generation of system evolution attributes as characteristic relations and admissible n-dimensional curves.

Referring back to figure 6.3, the notion of a timed automaton graph model is a transformation of basic set theoretic structures, algebraic structures and analytic convergent sequences. These transformations are generated from concepts which are developed from first principles as concrete geometric objects. This specification of canonical mappings (terminating relations) as a graph

automaton provides a simple intuitive basis for operational re-writing and query based deduction formally described in, say, Jantzen M. [1988].

6.3 Summary

This chapter has provided the formal details required for constructing **plausible behaviour curves** as part of the declarative configuration and observation data located within interacting objects. It also describes the logical basis for specifying algorithms which operate as discriminant functions over such curves. The algorithms are important components of the specifications and realizations of system functions. The power of these formal constructs are better appreciated through the specification and simulation of a realistic case study such as the example described in the next chapter.

Some of the technical constructs such as the notion of convergence of sequences and the composition of relations are difficult to implement incrementally. Representation techniques for transaction based implementations are developed in appendix C where incremental implementation issues are defined.

Chapter seven

Application of the timed automaton hypergraph to the specification and simulation of system functions

7.1 Introduction

The graph model for specification of system functions presented in the previous chapter ties together both the problem related soundness issue with a logical and algebraic operational framework. Plausible system evolution states are given generic geometric and analytic structures. The soundness issue is developed further in this chapter through the specification and simulation of a large scale system function, the ISO transport class four with congestion avoidance enhancements.

Section 2 presents a statement of the practical problem being tackled. Section 3 presents a collection of components which are brought together to solve system behaviour characterization problems in simulating a collection of system functions. The solution framework is comprehensive ranging from the generation of models for operations of networking equipment components to the encapsulation of processing engines as interacting objects. Section 4 presents a detailed simulation of the congestion avoidance system function. Detailed specifications of this case study are presented in appendix B.

7.2 The generic system function for resource management

The congestion avoidance system function is a classical distributed algorithm required for controlling access to shared resource pools. An introduction to a subset of this system function was presented in chapter 4. There are two components of the system function. These are the functions executed at the end-systems, and the functions executed at the resource pools. Both components satisfy the main system goal: the allocation of resources to user populations in a specified way, in accordance with predicted user demand patterns.

Even though the congestion avoidance function has been described and implemented in several networks, it is quite a large protocol suite which poses interesting challenges in the configuration of its parameters. This challenge arises because the protocol needs to be configured to satisfy the goals of users attached at various locations within the network. The protocol also controls the flow of input packet streams in accordance with feedback indications

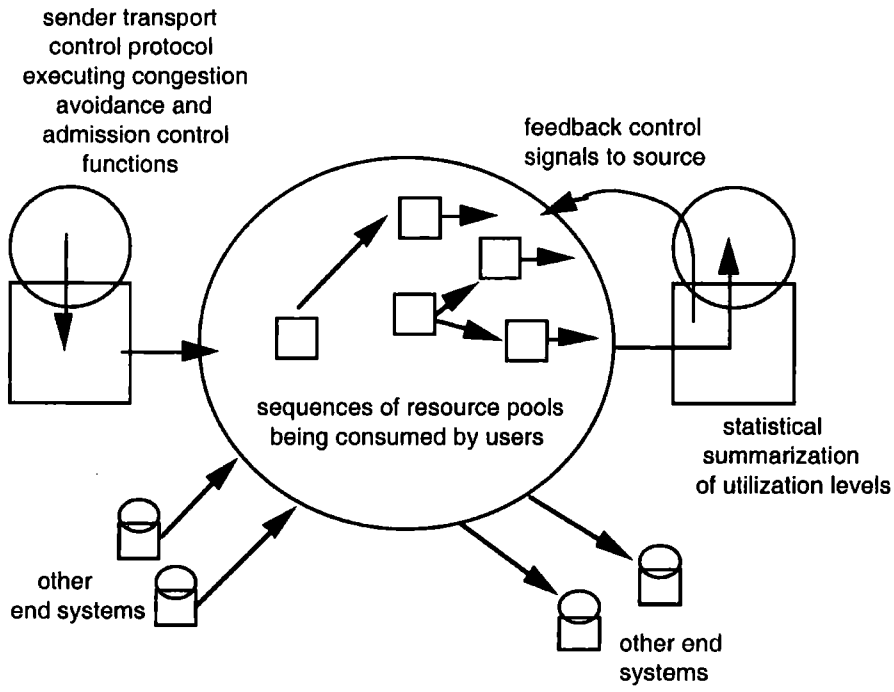


Figure 7.1 Unidirectional flow of information streams

of congestion status at shared resource pools. Two standard references available in the open literature provide detailed descriptions of the congestion avoidance function. The references are: ISO 8073[1986], and Ramakrishnan, K. K. and Jain R. [1988].

Figure 7.1 illustrates a unidirectional flow of information streams as such streams consume sequences of pooled resources. The ISO transport layer protocol Transport Class 4 (TP4) describes the sender and receiver transport control protocol for information streams and the credit based congestion avoidance procedures. The feedback scheme provides mechanisms for detecting utilisation threshold points that indicate imminent congestion at a resource pool. Prediction algorithms based on geometric concepts described in the previous chapter are provided to generate statistical summaries of utilisation status.

7.2.1 Declarative specification of system behaviour attributes

Figure 7.2 illustrates an inheritance structure for the declarative specification of a generic distributed system protocol. The user-network traffic contract is a good starting point for the specification of distributed control systems since it captures both the user and population goals, and the system provider's service capabilities. Expected transaction characteristics of user demands for system resources are generated by enrichment of the user-network traffic contract specification (see ITU User Demand Modelling: ITU E.711[1992]).

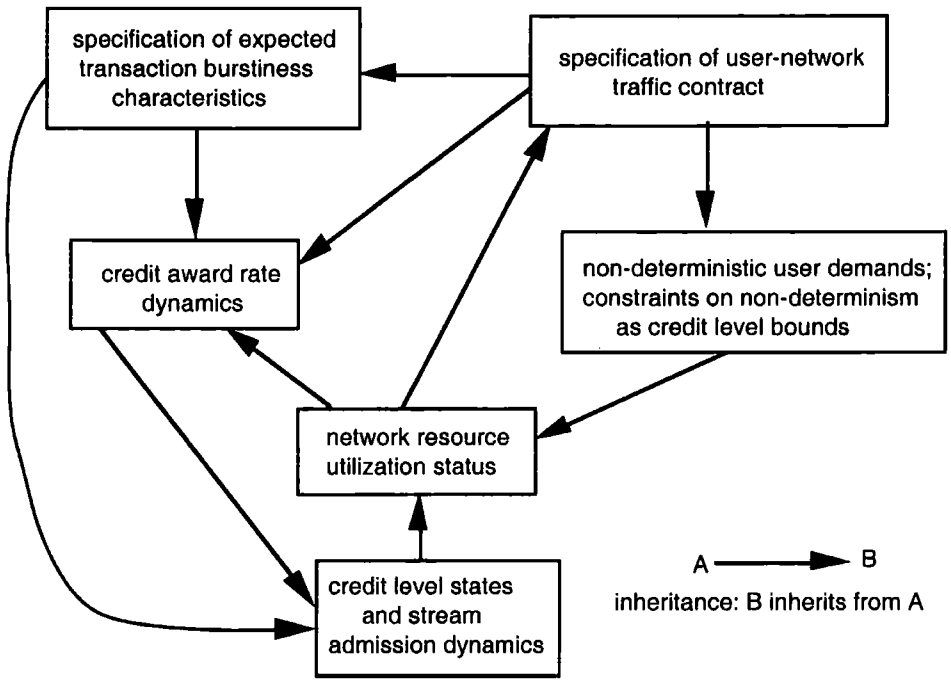


Figure 7.2 Declared and inherited system evolution properties as state attributes

During normal burstiness of resource consumption, the credit award rate dynamics (as specified within the traffic contract) effects admission control of transactions. However, it is often the case that user demands and the resulting transactions cannot be fully specified in a deterministic way (i.e. values of statistical variables cannot be completely specified, given a specific time window). It is therefore useful to allow a certain level of non-deterministic user demand patterns to be admitted into the network, in the hope that peaks from one set of users correspond to troughs from another set of users, the sets related by concurrent access to resource pools.

By monitoring credit level states, network resources' consumption status, and effecting bounding constraints such as admissible credit level state transitions, the credit award rate dynamics provides a control mechanism for enforcing resource based policies within networks.

7.3 The operational framework for implementation of system functions

7.3.1 The physical model

Figure 7.3(i) illustrates the generic mechanism for modelling the transport of information packets through a network of nodes. Packet streams enter an originating node, say at 'a' in the figure, into a conceptual input buffer 'b'. The behaviour of a packet stream as it travels from the

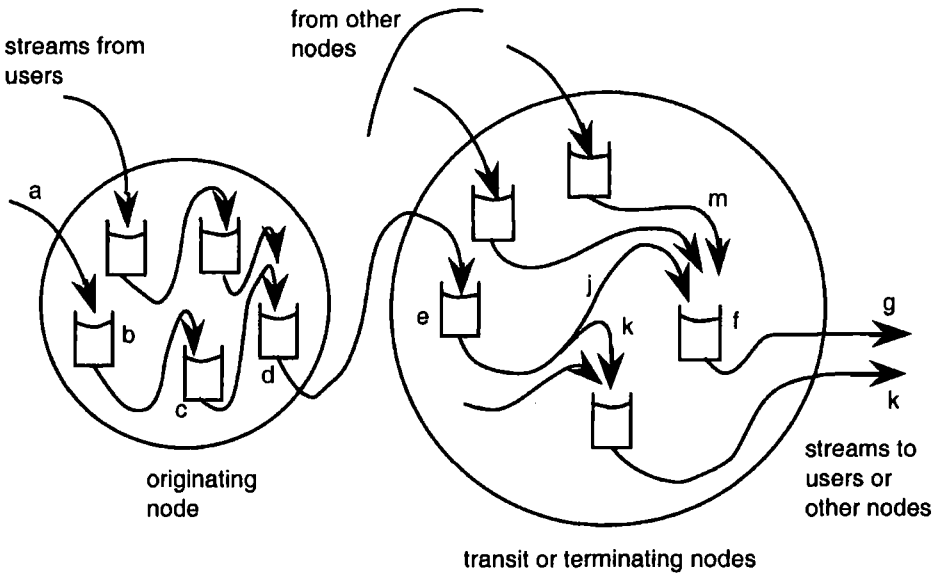


Figure 7.3(i) Flow of streams within nodes of network

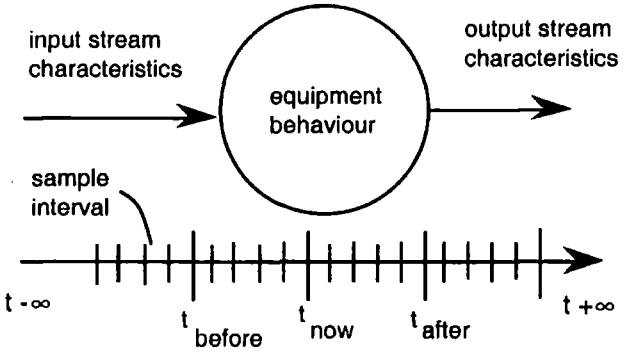


Figure 7.3(ii) The equipment behaviour as an automaton

entry point into a network to the exit point from the network is modelled as a system of automata sequential machines. Figure 7.3(ii) illustrates the encapsulation of transmission control as an equipment behaviour automaton which transforms an input stream into an output stream, satisfying some specified stream behaviour and network behaviour constraints. In this case, the automaton encapsulates the rules for emptying the input buffer into the internal nodal transmission mechanism which in turn is another automaton at 'c' (figure 7.3(i)). The automaton at 'd' multiplexes various streams similar to the <a-b-c> stream into the shared <d-e> transmission resource.

At transit nodes both de-multiplexing and multiplexing are performed as necessary. <e-j> and <e-k> are de-multiplexed streams whereas <j-f> and <m-f> are two of the three streams multiplexed into the <f-g> streams.

The equipment behaviour automaton is designed to operate at discrete clock ticks; the interval of the tick is chosen to be sufficiently fine such that quantization errors in the representation of plausible geometric behaviour curves of the system function being simulated are maintained at an acceptable level. Figure 7.3(ii) also illustrates the time train specifying the times at which an automaton must be scheduled to execute its operations. Each operation executed at say time t_{now} builds on information gathered by the system at time t_{before}, so that goals obtained at time t_{after} are in line with predicted system behaviour.

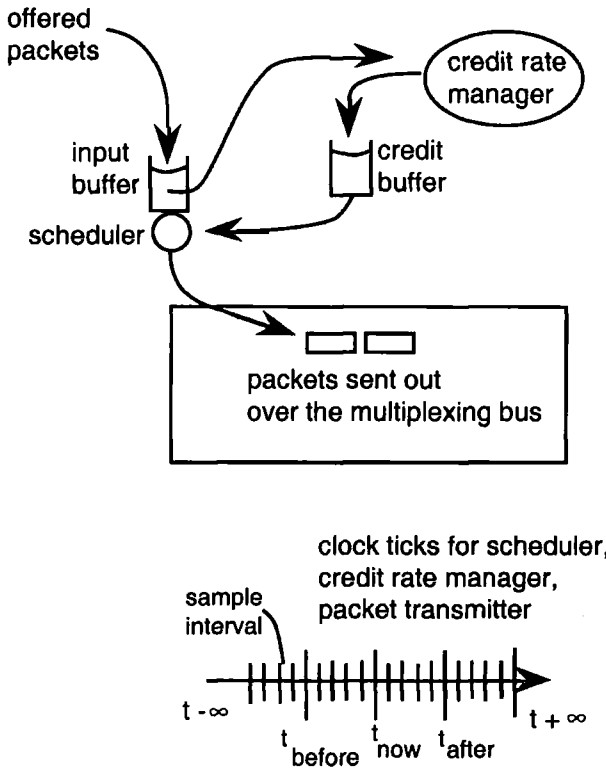


Figure 7.4 Schedules for transmission of packets of information

Figure 7.4 illustrates the behaviour of the automaton at location 'b' of figure 7.3(i). The credit rate manager is specified to award credits to the stream at a rate which satisfies both the individual stream's contracted goals, and the system's (population) goals. The credit rate manager operates a clock tick for placing credits into the credit level buffer. The scheduler is scheduled to execute, and empty credits from the credit buffer at a clock tick rate and operational characteristics that satisfy the modelled equipment behaviour constraints. Thus the

scheduler never violates the behaviour of the equipment being modelled. Every component of the system is therefore accounted for in the model.

7.3.2 A solution framework for model studies of system functions

Figure 7.5 illustrates how a collection of operational models can be used to generate executable specifications of system functions. The goal of this section is to bring together the exposition presented in the previous chapters as framework for performance modelling behaviours of distributed systems. On the left hand side of the figure are three components which are normally encapsulated within a programming system for simulation and verification of specifications of system functions. On the right hand side are concepts which are used to exhibit specifications of distributed system functions.

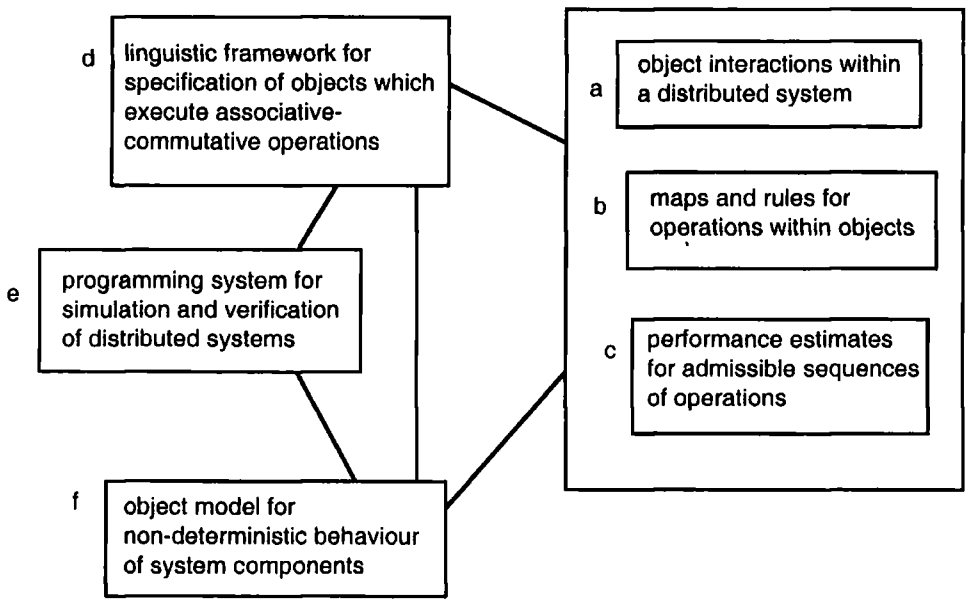


Figure 7.5 A solution framework for modelling distributed system functions

Item (a) denotes concepts which are obtained by further development of the notion of the simulation automaton presented in the previous sub-section. The simulation automaton is embedded within an object; the latter provides encoding for a recognisable input-output signature of the enclosed automaton. Since the notion of non-deterministic behaviour within a distributed system results from characteristic (though random) behaviours of some entities within the system, the following description holds for a non-deterministic simulation automaton. Given an internal start state and a sequence of input invocations as its signature

component, the sequence of output responses is not a unique sequence within the signature when the object hosting the automaton is instantiated at various times. Thus an object hosting an automaton can generate an output after receiving an input by applying intrinsic judgement which cannot be represented explicitly because the system designer who specifies the object has incomplete information about the behaviour of the object at every invocation.

The two types of objects within a distributed system are the deterministic and non-deterministic objects. The deterministic objects provide control logic for the allocation of system-wide resources reflecting the attempt by a system designer to specify operational behaviours of these objects in order that system-wide goals are satisfied despite the behaviours of non-deterministic objects within the system.

Item (b) denotes concepts, i.e. mathematical structures for representing operations over time, within distributed objects. For non-deterministic objects, such maps and rules reflect the designer's predictions of the behaviours of the objects; these objects would normally model users of a network, and networking equipment that exhibit non-deterministic behaviours. The system goals reflected in the maps within the non-deterministic objects range from desirable behaviours to various levels of undesirable behaviours in the eyes of a system designer. An undesirable behaviour could be that a non-deterministic object requests for the allocation of resources to it over time, in an erratic way in which no apparent recurrent pattern is discernible.

Maps and rules within deterministic objects aim to represent admissible sequences which reflect the system designer's policies in satisfying system goals. Various combinations of operational sequences can be admissible (i.e. acceptable given a range of performance constraints). Item (c) is a set of specifications which simulate possible behaviours of a collection of objects that execute specific system functions. These specifications can be animated in order that the strengths and weaknesses of various resource allocation policies can be evaluated in terms of performance constraints. The specifications and simulations described in the next section illustrates such a solution procedure.

It is useful to factor out specification and simulation tasks common to the evaluation of most networking scenarios of interacting objects and system functions under study. A simulation environment is often provided with features that encompass such generic functions. Such a simulation environment would normally provide a linguistic framework for expressing declarative operations, data domains and conditional statements (item (d) on figure 7.5). Also, such a linguistic framework needs to be sufficiently flexible in order that it can serve as a specification language for a wide variety of system functions.

A mathematically rigorous simulation system has been implemented by a software research company called the Math Works Inc., in Natick, Massachusetts, USA. The system is developed using MATLAB™ [1995] which combines a simple procedural programming language with a collection of standard matrix algebra and analytic functions. The linguistic framework adopted by the Math Works Simulation Suite (MWSS) is called STATEFLOW™ , an implementation of the **state charts** formalism of Harel D.[1987]. Harel's state charts notation uses the set theoretic Venn Diagrams to structure predicates that hold within a reactive system as the system evolves over discrete time windows. State charts is powerful in its representation of system evolution states since hierarchical structures of predicates can be composed to any desired level of height. Also, any number of state hierarchies can be given a parallel structure in accordance with some physical structure of the system being simulated. Items (d) and (e) on figure 7.5 can thus be realised using STATEFLOW™ program objects within MWSS.

The very expressive formalism of state charts needs to be suitably constrained so that system functions can be checked for soundness. The object model presented in chapter four is a basis for specification of system functions so that such specifications can be interpreted in terms of the physical system being modelled. Item (f) on figure 7.5 shows how the object model developed in chapter four fits into the proposed solution framework. A detailed development of such an object model has been carried out by the author, and implemented by the author in collaboration with his co-workers (Aranzulla P., and Pitts J.) at Cable and Wireless Communications, Watford, UK. The implemented system is called the SIMULATION JACKET: an outer shell of system functions for the analysis and validation of telecommunications networks. A publication of the SIMULATION JACKET cannot be carried out until appropriate patent and intellectual property rights have been concluded. An object model developed by the author for the specification presented in appendix A can be found in Nyong O.D.O.[1993].

7.4 Adaptive resource partitioning within a distributed system

7.4.1 Introduction

This section is a presentation of a policy framework for allocating inter-node resources to source-destination paths over a network topology. Each pair of nodes is allocated a single source-destination primary path for carrying predicted traffic during a defined time window. Thus a set of source-destination paths is generated to span appropriate link segments of a network topology in such a way that the network is provisioned in an efficient way. The notion of efficient allocation of resources is a structured representation of characteristic sequences of attribute values generated for entities within the network. Such entities include inter-node

transmission resource pools; an example attribute of a resource pool is the occupancy level of the pool.

Since user demands on pooled network resources are inherently non-deterministic, it is acceptable to expect prediction errors with respect to the pattern of offered source-destination traffic during any given future time window. However, it is only meaningful to invoke a collection of source-destination paths as operational within a live network during a period of time when the predicted traffic is close to the traffic being offered by users attached to the network. During the period when a route pattern is operational, flow control procedures are also operational to serve as constraints on non-deterministic user behaviours.

As described in chapter 3, the computation of a map of source-destination paths is an optimization problem which has been studied in various forms (see e.g. sections on shortest paths and network flows in Tarjan R. E. [1983]). Given a predicted user demand traffic pattern across source-destination node pairs, a set of paths can be calculated to satisfy a set of constraints. Since such a path set can be computed off-line without a severe time constraint (e.g. sub-minute constraint), this computation is not necessarily a challenging exercise. Moreover, the computation is not a distributed algorithm and so such algorithms do not pose the 'product form' challenge in the generation of valid attribute values. The next sub-section is a description of a path generation procedure followed by a specification of the challenging congestion avoidance algorithm implemented to enforce traffic admission constraints.

7.4.2 Generation of source-destination paths

This subsection describes a simple suite of algorithms for setting up a single set of source destination paths across an irregularly connected network. Figure 7.6 illustrates a network showing three primary paths <A-B>, <C-D>, and <J-K>. The algorithm described in this subsection is aimed at generating such primary paths for all pairs of nodes in the network. A complementary set of policies and procedure would be provided to generate secondary paths such as <E-F>, and <M-N>.

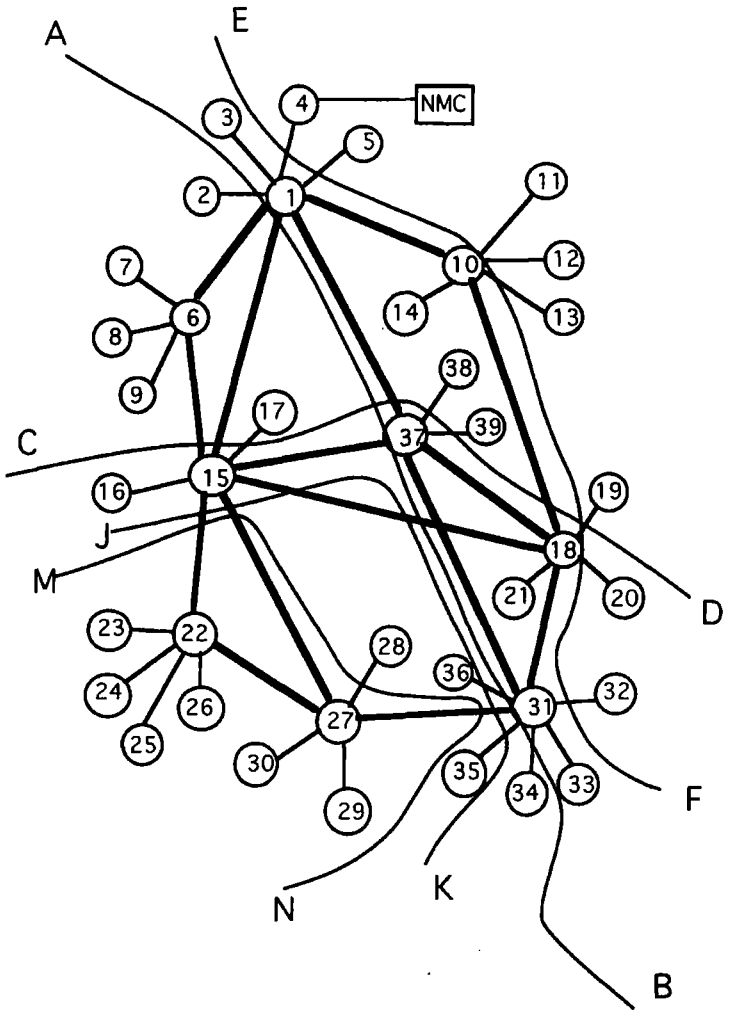


Figure 7.6 A routeing plan within a sub-network

The following definitions describe the operational mappings for the procedure illustrated in figure 7.7.

Definition 1:

Given any sub network, pairs of nodes are denoted as <i, j>, the user offered traffic intensity at a given time window can be denoted as $\lambda_{ij}(t)$.

Definition 2:

Given a sub-network with pairs of nodes denoted as <i, j>, paths can be selected to carry $\lambda_{ij}(t)$ traffic during a given time window. Based on various policy criteria (e.g. optimization through minimization of the quantity of pooled resources provisioned and maximization of carried traffic), paths can be re-structured across the network; thus paths continuously vary over a time horizon. Denote time varying paths as $\mathbf{P}_{ij}(t)$.

Definition 3:

As in 1 and 2 above, observed consumption of resources can be denoted as time varying vectors $\mathbf{R}_{ij}(t)$.

Definition 4:

In provisioning paths in a network to satisfy the distributed nature of users and the need to pool resources, a mapping is defined over a fixed set of paths as follows:

$$f_a: \lambda_{ij}(t) \rightarrow \mathbf{P}_{ij}$$

Definition 5:

In operational networks, it is necessary to classify traffic types or user types and so allocate network resources to satisfy specific user or traffic type constraints. The following mapping defines such utilisation effect.

$$f_b: < \lambda_{ij}(t), \mathbf{P}_{ij} > \rightarrow \mathbf{R}_{ij}$$

Definition 6:

With learning and prediction in place, improvements in resource allocation algorithms can be made. Moreover, adaptation to non-deterministic user demands can be accounted for. The following mapping defines policy revisions.

$$f_c: <\lambda_{ij}(t), \mathbf{P}_{ij}, \mathbf{R}_{ij} > \rightarrow < \mathbf{P}_{ij}, \mathbf{R}_{ij} >$$

Referring to figure 7.7, the data items 1, 3 and 2 provide basic user demand, network topology and link segment cost constraint declarations by the network designer. Items 4 and 5 provide biasing information in the generation of all possible source destination paths. Algorithm A simply enumerates all plausible source destination paths that do not violate the weak constraints of 4 and 5, and fair resource allocation using data item 1.

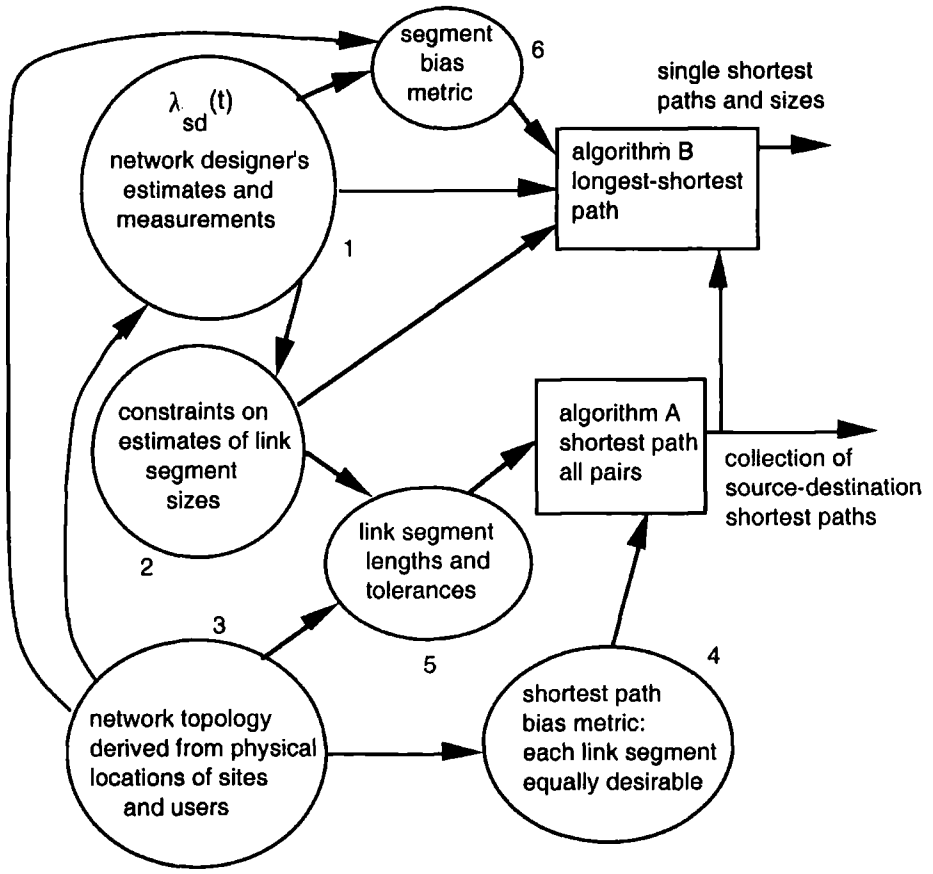


Figure 7.7 Holistic resource partitioning algorithm

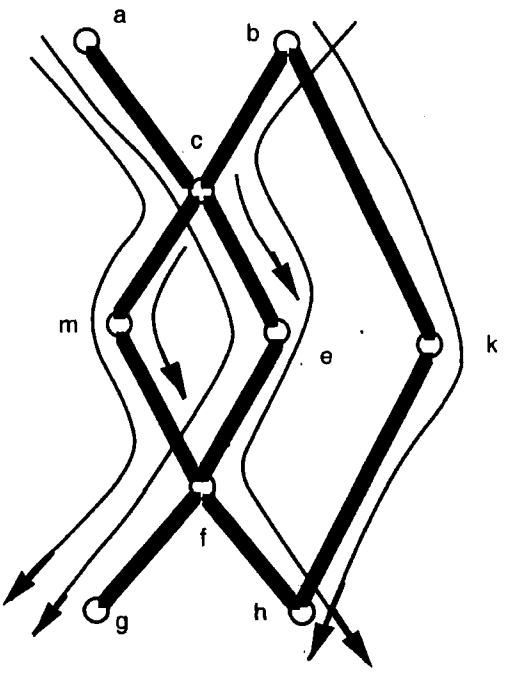


Figure 7.8 Segment metric biasing

The collection of source-destination shortest paths generated by algorithm A are ordered in ascending order of a quantity denoting a product of the path length metric and the predicted carried traffic. One policy for generating a singe set of shortest paths is to invoke the notion of economies of scale as the link segment bias (see data item 6 on figure 7.7). Consider as an example a fragment network topology [a, b, c, ..., f, g, h] as shown in figure 7.8. The pair of paths <b-k-h> and <b-c-e-f-h> are equally acceptable for b-h traffic. Likewise, the pair of paths <a-c-e-f-g> and <a-c-m-f-g> for a-g traffic. Assume that the <b-h> path is longer than the <a-g> pair in terms of source-destination path length over the whole network topology. Assume that the <b-c-e-f-h> path can carry an amalgamation of higher traffic, as per the predicted demand, than the <b-k-h> path. Then by notion of economies of scale, path <b-c-e-f-h> is selected instead of path <b-k-h>. Moreover path <a-c-e-f-g> is selected automatically instead of path <a-m-g>. This procedure is continued until all the single shortest source-destination paths are generated. This is an example policy criterion for use in algorithm B of figure 7.7. Such a policy procedure is rational with respect to any primitive provisioning criteria, and incorporates the uncertain user demand behaviours through the predications of offered source-destination traffic patterns.

The conclusion to be drawn from the example illustrated in this section is that an optimization problem can always be treated as a 'permutation with constraints' problem. Constraints on admissible system behaviours are usually specified as system resource access or consumption policies.

7.4.3 A simulation example: adaptive resource allocation in the presence of uncertainty

7.4.3.1 Introduction

It is useful to provide an overview of the experimental set up of the simulation before a detailed presentation of the simulation task is described. Referring back to the network on figure 7.6, the paths have been set up for the admission of traffic into the network. As an example, consider the streams denoted as <A-B> and <J-K> which consume resource pool 37-31 as well as other resource pools in their paths. The existence of these paths and the allowable consumption pattern of each stream are pre-planned in accordance with predicted user demand patterns of these streams, and other streams that could concurrently consume pooled resources.

Figure 7.9(a) illustrates (using a free hand sketch) predicted user demand models of streams <A-B> and <J-K> during a normal recurrent cycle. Such a pattern would normally bear some relationship to an agreed user-network traffic contract. Figures (b) and (c) serve to emphasise that such summaries would normally be obtained over several recurrent cycles.

Figure 7.10(a) illustrates how admitted traffic from various streams are interpreted as system-wide attributes of entities within a network. Admitted traffic from each stream is constrained to enforce pre-declared traffic contracts. The attribute space in this case is defined to denote the time varying resource consumption level at a shared resource pool. An example of such variation is shown in figure 7.10(b). There is a strong coupling between statistical summarization of resource demands, statistical summarization of resource consumption, and constrained admission of traffic onto resource pools within a network. An example specification of such a control procedure is presented in appendix B.

7.4.3.2 Control dynamics and transitive behaviours

An example of the variations of a buffer level at an entry point into a shared transmission resource is illustrated in appendix B. Thus in a given time window, it is possible to define a bounded range of values referred to as an **attribute value space**. Sequences of values of an attribute are used as an important parameter in the logical rules for constraining the behaviour of an individual resource consumer and thus the behaviour of any population of resource consumers. More directly, such sequences generate the rules required for selecting appropriate actions executed by equipment behaviour automata described in section 7.3.1.

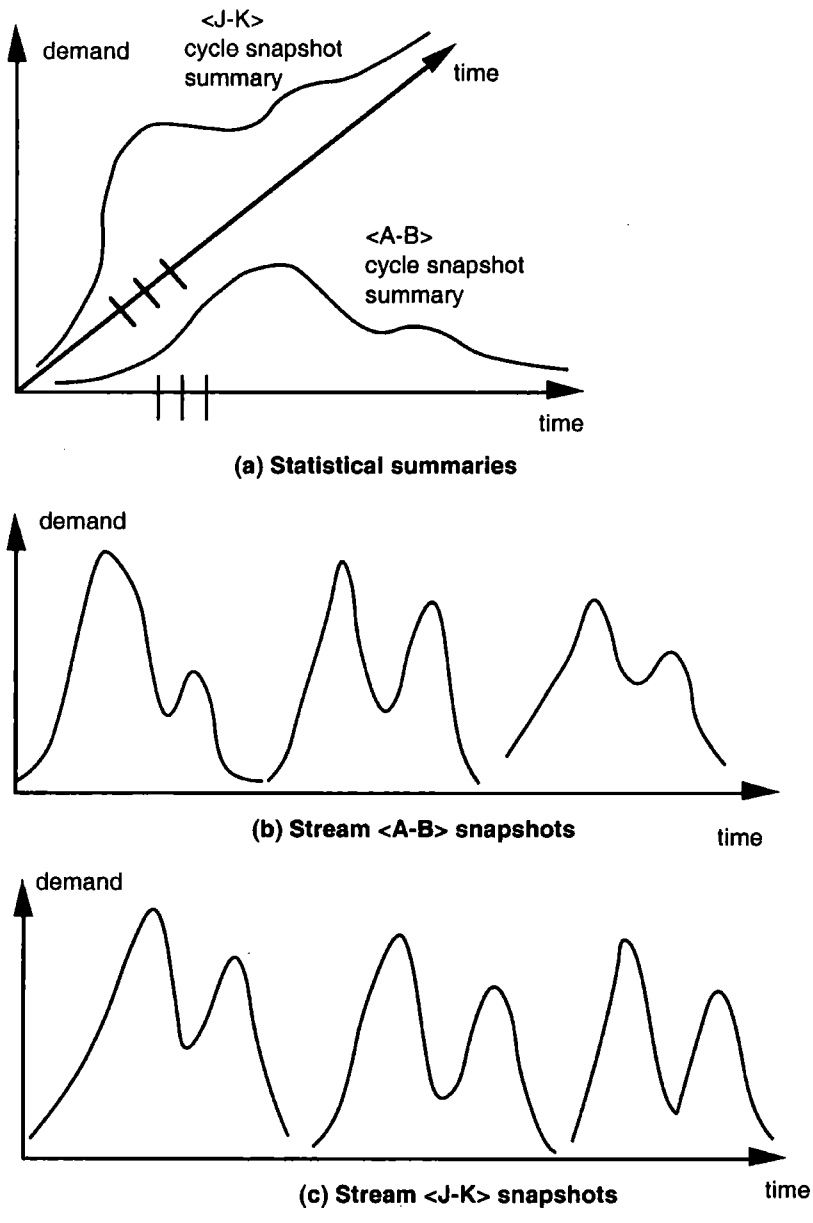
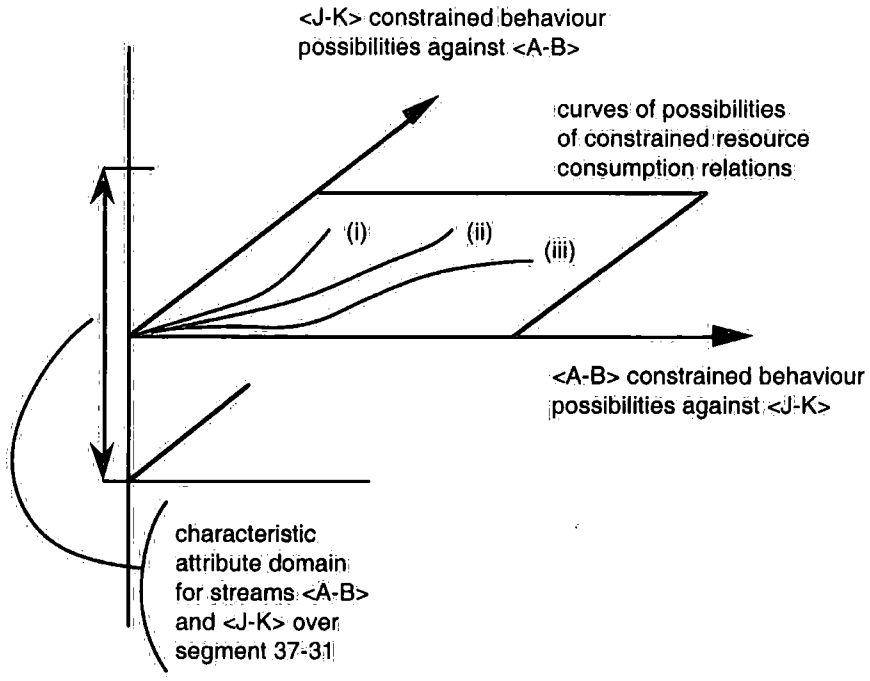
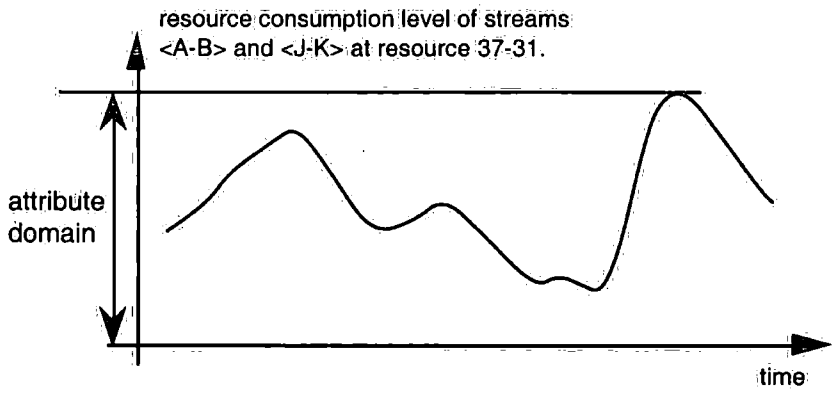


Figure 7.9 Summarizing user demand models

It is an extremely challenging task to generate consistent sets of attribute value sequences for various user demand scenarios within a distributed system in such a way that system behaviour goals are met. This difficulty is compounded by the fact that user demands on pooled resources are inherently non-deterministic. One way of controlling this difficulty is to work within the solution framework described in this chapter. The approach being proposed is to generate experimental system behaviours and use the results of the experiments to provide information on how to select acceptable attribute sequences that match system realization goals. Two experimental scenarios are used to illustrate this point.



(a) Constrained behaviour during a recurrent cycle



(b) Consumption pattern at a resource pool, during a time window

Figure 7.10 Constrained system behaviour patterns

Scenario 1.

Referring to figure 7.6, this scenario is concerned with resource consumption at shared resource pools <37-31> and <15-37> due to demands on streams <A-B>, <C-D> and <J-K>. Feedback control signals are sent from the distributed control centres to the sources of data streams requesting that the respective sources' window sizes should be decreased down to, or increased above their contracted minimum (the Contracted Information Rate ~ CIR). The decision to place these streams onto the shared resources is based on predictions of user demands of these and other streams that consume pooled resources across the network.

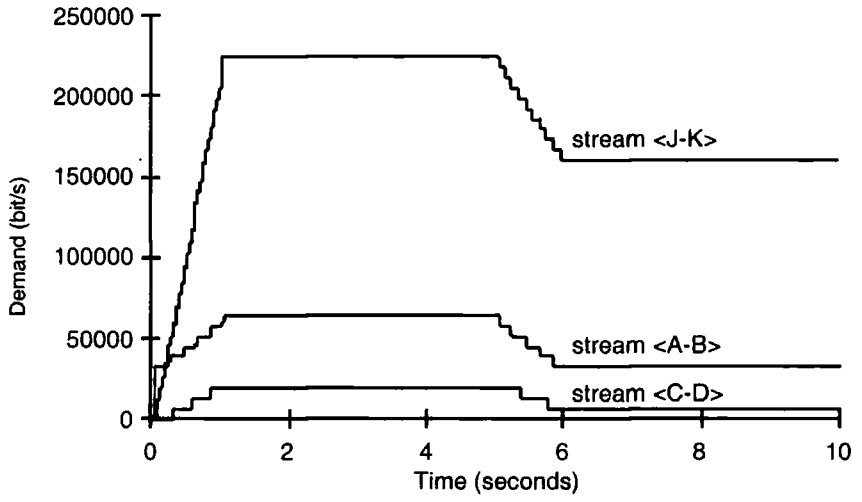
In this scenario, the CIR is configured such that streams <A-B> and <J-K> share equally the resource <37-31>. Refer to the simulation graphs (a) to (f). It is possible in practical networks that the predicted traffic differs from the actually experienced traffic such that the pattern shown on plate (a) holds: user demands on stream <A-B> remain below its CIR whereas those on stream <J-K> strays above its CIR during most of the cycle of recurrence. The effect of the flow control procedure specified in transport class 4 is studied by focusing on the system behaviour as traffic on stream <J-K> is being constrained. Note that this stream consumes resource pools <15-37> and <37-31>. Saturation thresholds at buffers serving these resources are set at level of 30 packets each, an experimental value.

Referring to plates (b) and (e), the label 'before' denotes the first part of this scenario when the credit award rate has not been altered. The label 'after' denotes plots for the second part of this scenario when the credit award rate has been altered.

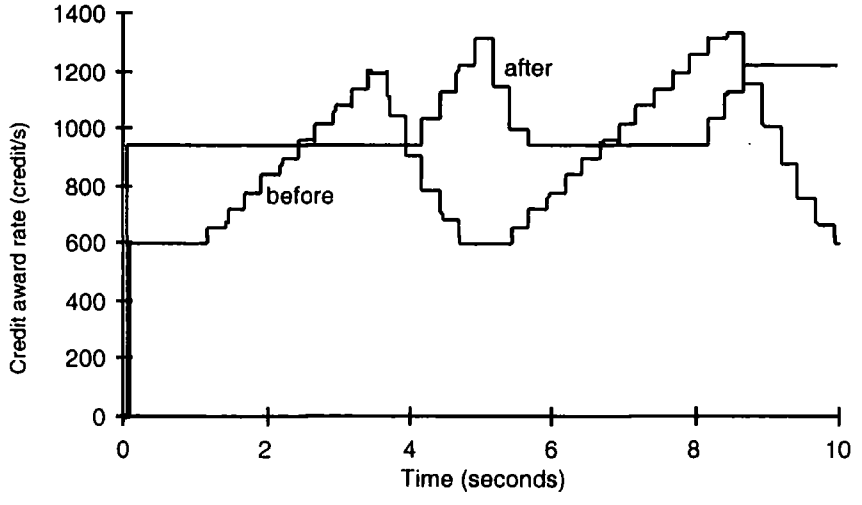
Consider the credit award rate 'before' on plate (b). End systems K and B construct statistical summaries of utilization status of resource pools along the paths of their streams, as specified in appendix B. As shown on the plot on plate (b), binary feed back control signals derived from statistical summaries are sent to the end system J which increases or decreases its admission window size (i.e. credit award rate) as stipulated in the signals.

From t=1.0 to t=3.5 seconds, the statistical summaries at K indicate that the credit award rate for admission into the shared pools can be increased above CIR. At t=3.5 seconds and henceforth until 4.5 seconds, control signals from K reduce the credit award rate back to CIR. At t=5.0 seconds, the oscillatory phenomenon commences, in line with the offered traffic pattern. Stream <A-B> is not affected by the congestion at the shared resource pool even though end system B is aware of the problem. This is because stream <A-B> is offering traffic into the network at a rate well within its user-network traffic contract.

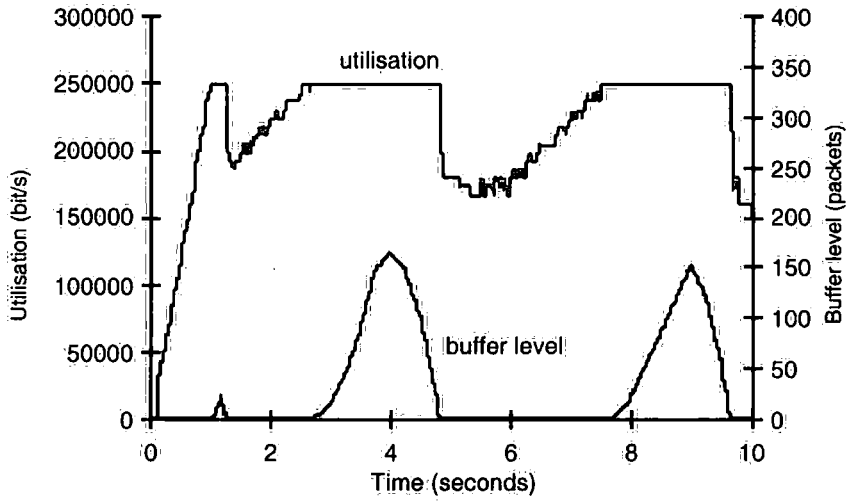
Plate (c) illustrates the system behaviour at the shared resource pool <37-31> in terms of its utilization and buffer levels. The transmission resource capacity is exhausted initially soon after bursts from the streams are detected. However, smoothening takes happens due to the re-distribution of packet which have built up at various location within the network. The first major congestion at the resource pool <37-31> occurs at about 3.0 seconds. At the time when congestion commences, summarization at the resource pool is initiated.



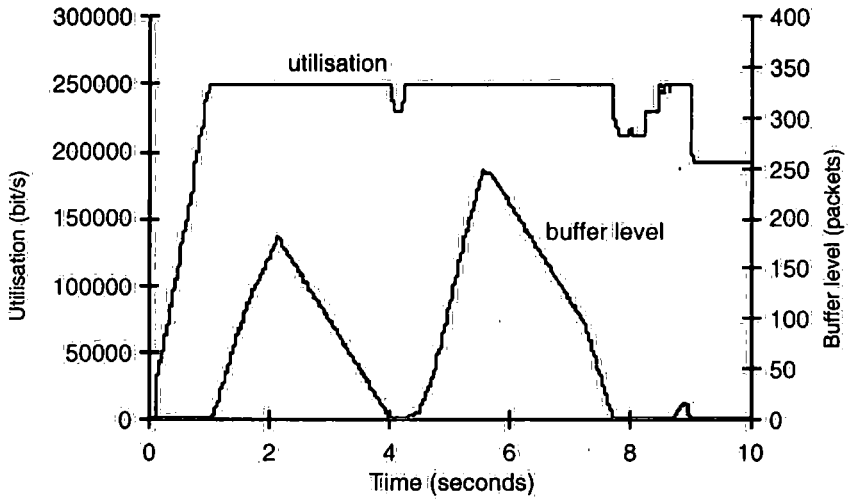
(a) User demand with stream <J-K> deviating from expected demand



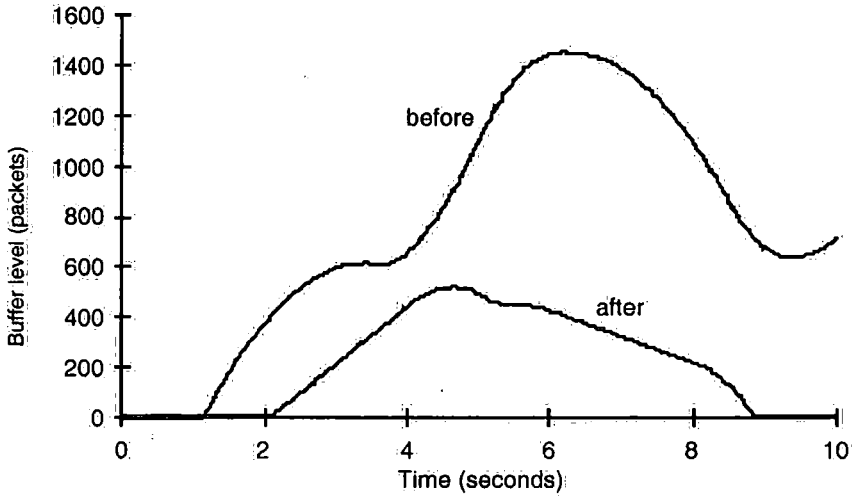
(b) Credit award rate for stream <J-K> before and after reconfiguration



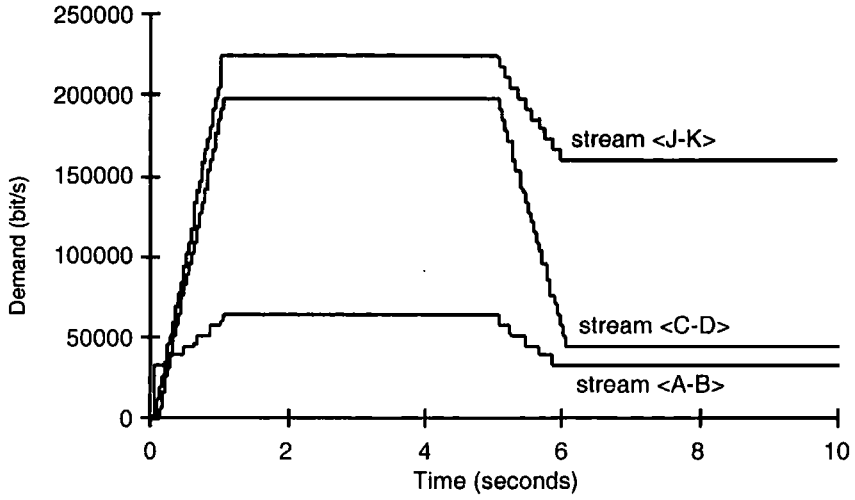
(c) Resource utilization and buffer level at <37-31> before reconfiguration



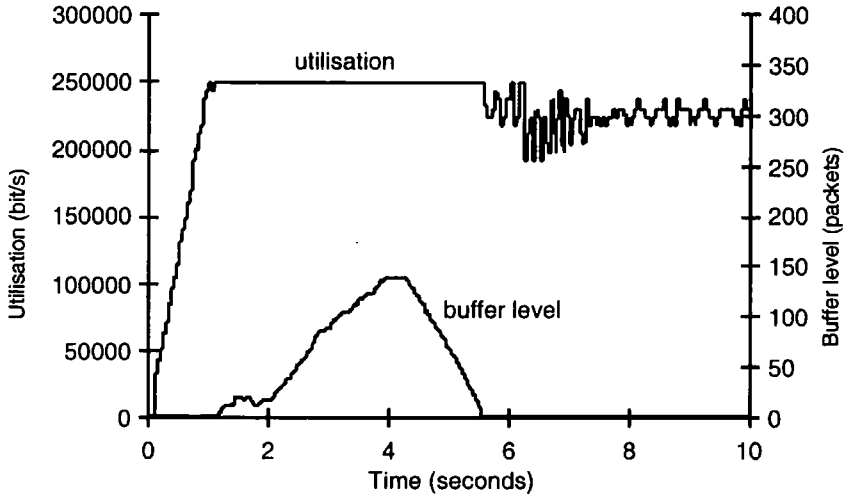
(d) Resource utilization and buffer level at <37-31> after reconfiguration



(e) Buffer level at end stystem J before and after reconfiguration



(f) User demands with stream <C-D> deviating from expected demand



(g) Resource utilization and buffer level at <37-31> after reconfiguration and with user demand given in (f)

Most high speed packet switching equipment currently in use provide mechanisms for summarization as follows. As each packet traverses shared resource pools (buffers) in its source-destination journey, if there is congestion at a shared resource pool, an information element is inserted in the packet as a way of signalling to the destination end system indicating the possibility of congestion at a location along the path. Since the build up of congestion along a path may not result in every packet traversing the path having the congestion marker set, the end systems need to invoke summarization procedures that detect true congestion, as opposed to intermittent buffer congestion. However, such persistency checks cause delays in the invocation of flow control procedures when congestion actually do occur.

In the simulation results shown on plates (b) and (c), the end system K invokes flow control procedures on source stream J a number of clock ticks after the transmission resource has been exhausted. When the flow control mechanism at J takes effect, packet entry into the network is reduced and the build up of packets at the shared buffer is cleared. Utilization of the shared transmission resource drops temporarily to about 65%. At about t=5.0 seconds, the end system K detects that congestion has been cleared, and that there is slack in the shared transmission resource. End system J is again invited to increase its packet transmission into the network.

Plate (e), the 'before' curve shows that the transmission resource provision for stream <J-K> is severely inadequate since there is a high build up of packets at the end system's entry point into the network. The presence of oscillations in the pattern of resource consumption at a resource pool can be interpreted to mean that a fraction of that pool's resource is wasted, probably due to badly chosen configuration and control parameters within the system. In order to revise the configuration and control parameters when maintaining live networks, several measurements of offered traffic and network behaviour patterns need to be carried out over a number of recurrent patterns. The procedure for changing from one set of configuration and control parameters to another set can be built into the evolution policy of the operational software. The model generation framework proposed in this thesis does hold for the specification of policy revision as a system function.

For this scenario, a parameter reconfiguration algorithm could recommend a restructuring of the routes across the network. However for illustrative purposes in this experiment, this scenario maintains the current set of routes in order to illustrate the effect of reconfiguration of the admission control policy. The CIRs for streams <J-K> and <A-B> are configured such that the resource <37-31> is shared in proportion to the demands by the two streams (i.e. 78:22 ratio). The CIR for stream <C-D> is left unchanged.

Plate (b) shows the behaviour of the credit award rate for stream <J-K> 'after' reconfiguration of the credit rate in line with the new traffic contract. Flow control operations are also invoked in this configuration. Plate (d) shows the impact of reconfiguration on the shared resource <37-31>; the transmission resource is fully utilised during most of the period.

Plate (e) further illustrates the effect of the reconfiguration on stream <J-K>. In the 'before' configuration, more packets are buffered in the end system J.

It is tempting to conclude that this reconfiguration policy is sound, and has resulted in efficient utilization of the shared resource pool <37-31> since oscillations have been reduced. However, such a conclusion is ill-judged since the build up of packets within the high speed network (up to 200 packets at a particular time window) far exceeds the saturation threshold of 30. High

speed networks are generally designed in such a way that information streams suffer minimum transit delays. This scenario emphasises the need for carrying out several *what if ?* experiments in order to determine an appropriate set of configuration and control parameters. In this case, such experiments involve analysing the trade-off between pooled resource utilization efficiency and service provision quality.

Scenario 2.

Suppose that the user demand of stream <C-D> on plate (f) deviates from the expected demand shown on plate (f). Recall from the network diagram that the stream <C-D> and <J-K> share resource <15-37> and that stream <C-D> does not require resource <37-31>. When stream <J-K> loads up resource <37-31> while sharing it with stream <A-B>, the distributed control algorithms try to satisfy the demands of the two streams on the shared resource. The increased traffic input of stream <C-D> forces stream <J-K> to reduce its share of consuming resource <15-37>, and hence the reduction of stream <J-K>'s consumption of all resource pools along its path. The impact of this reduction is felt at locations not traversed by stream <C-D>. Comparing plates (d) and (g), it can be seen that altering, through control policy constraints, the resource consumption at one point in a network can result, through transitive effect, in a change of behaviour patterns at another part of the network. This transitive phenomenon emphasises the need for holistic modelling and experimentation in the provisioning and configuration of networks.

The results presented in this section can also be applied to scenarios where a number of sub-networks agree to co-operate, and share local (i.e. per network) resources. The plates shown in this chapter were published as part of the paper that exposed some inter-networking policies (see Nyong O.D.O.et al.[1998].

7.5 Summary on simulation of congestion avoidance

This chapter has illustrated through specification and simulation the diverse components required to solve a resource allocation problem in a telecommunications network. Emphasis has been placed on all the steps required for problem solution, ranging from the specification of user - network traffic contracts through to the simulation of an industry standard system function.

The simulation case study presented in this chapter has illustrated the very important transitivity phenomenon in the performance characteristics of resource consumption at interconnected resource pools. Whereas the control of resource allocation at a resource pool is

a local matter, the policy for placing user demands on shared resource pools is a global matter. This phenomenon can be interpreted to mean that most studies on admission control or predictions of an individual stream's traffic pattern is always a sub-problem of a global system function. It is often the study of such global system functions than can provide insight into the level of optimization that are practically meaningful while formulating rules for local resource allocation.

In the specification of the case study described in appendix B, the mathematical and logical framework underlying the simulation has been hidden away in order to avoid obscuring the practical issues concerning the reasons for solving such problems. However, without the formal concepts developed in the previous two chapters, it would be very difficult to keep track of the large numbers of system evolution states and rules normally encountered in the modelling and implementation of such system functions. A reconciliation of this chapter's informal presentation with the formal framework presented in the previous two chapters is carried out in appendix C.

Chapter eight

Conclusions and areas for further work

8.1 Architecture, modelling components and experimentation

The problem of finding a reference model for representing large scale telecommunications system functions has been tackled by developing abstract concepts for both physical network topologies and measurable characteristic behaviours of networked entities. By treating any network as a structured mesh of inter-connected service provision points, the physical structure of any network topology is interpreted as part of a large set of declarative data universes. Since operations on these data universes are compositional, the building blocks for constructing hierarchies of specifications representing system behaviours are divorced from the purely physical structures of network topologies. Important issues addressed in the specification and representation of admissible network behaviours can thus be treated as purely algebraic, analytic and model theoretic concepts.

User demand models and equipment operational characteristics are attributes of reactive entities within a networking system. Their behaviours, even when non-deterministic, can be specified in a declarative way. The notion of a timed automaton formalizes the operations executed by these reactive agents. The control automata (described in appendix C as the *limit preserving transaction controller*) complements user demand and the equipment behaviour automata; the three types of modelling components are sufficient for specification of operations representing any system function.

The three types of automata have been used in the specification, representation and simulation of realistic system functions. By adopting this approach, model generation and characterization for any type of network topology is turned into a task involving step by step composition of system functions and simulation experimentation. It can therefore be concluded that the *system function* based modelling approach developed in this thesis provides a compositional simulation approach backed up by formal semantics. This approach to simulation has hitherto been largely unexplored.

The timed automaton encapsulates both the object oriented paradigm of software engineering and the faithful representation of switching and control equipment within a network. This approach is useful because utilization levels of pooled network-wide resources can be specified or derived directly during a simulation. Thus notions such as packet loss probability and call blocking probability can be given justifiable meaning

based on statistical data gathered at resource pools.

The embedding of geometric patterns within system evolution states provides a powerful technique for keeping track of large state spaces generated in the modelling of realistic system functions. This approach scales well and overcomes the conceptual barrier of how to model large topology irregular networks. Convergence preserving relations developed for transactions (in appendix C) complement the geometric patterns as part of the limit criteria proposed in the semantical framework (chapter three).

The results from simulation experiments have shown the importance of holistic model study in performance evaluation of network-wide system functions. The experiments reported in chapter seven have shown that changes to resource consumption at one location within a network can significantly affect the quantity of resource available for consumption at another part of the network due to the complex coupling of resource consumption streams across a network. Thus the entwined structure of system functions is isomorphic to the coupling of resource consumption streams across a network.

8.2 Areas for further work

The discriminant function used for the detection of congestion in the simulation example (case study II) relies on the persistency of measured behaviours at resource pools. Such an approach is a coarse control procedure; it can result in an inefficient resource allocation to users because the threshold for detecting congestion at a resource needs to be quite low, in order to minimise loss of transmitted packets of data. More efficient control actions can be achieved by deploying powerful discriminant functions that can identify a current characteristic trend among other plausible geometric structures. This is a major area for further work; an initial attempt by a team of co-workers was reported in Cosmas J. et al. [1997].

The notion of convergence to optimal resource allocation policies, given a non-deterministic resource demand scenario, is a major subject of research in the area of stochastic optimization. The algorithm framework reported in appendix C is convergence preserving. Further work could focus on the complexity of achieving convergence by building on the structure proposed in appendix C, along the line initiated in Nyong O.D.O[1995].

It is a challenging task to construct problem specific rules for implementing the notion of non-deterministic actions to be executed by learning automata in the optimal control of network-wide resource consumption. Results computed in each simulation experiment of

case study II could be treated as part of the collection of information on plausible system behaviours, given a user demand scenario. Thus such experiments can serve as the building blocks for constructing actions executed by teams of learning automata in their enforcement of resource allocation policies. The notion of *product form representation of system functions* developed in chapter five could be built into the structure of the learning automaton. This is an interesting and important area of further work.

Appendix A
Specification of the storage functions

1 Introduction

The motivations and architecture for the storage functions are presented in chapter 4. The specification presented in this appendix is aimed at providing an insight into the typical size and complexity of large scale system functions. The specification is provided at a detail that allows for correct implementation using an appropriate high level programming or deduction language. The high level of complexity is unavoidable thus the specification can be laborious to read. Since the theoretical framework used for the specification is fairly crisp (see chapters three, four and five) the reader may not need to read the specification in a strict sequential order in assessing the completeness of the operations.

2 Basic specification rules

An introduction to the algebraic and limit semantics applied in the specification of this collection of functions is presented in chapter five.

Structural relations over observation and declared data are defined as follows. A *conditional* statement is made if a choice sub-operation within a computation clarifies the execution of the invoked operation. A *leads-to* statement is made to emphasize the sequence structure of relations over data. This is in the spirit of the logic of Chandy K.M. and Misra J.[1988]. The limit or fixed point statement is made to emphasize the situation when a computation has recognized a goal state.

3 Operations supporting the storage functions

3.1 Introduction

This section describes operations that support storage functions. Section 3.2 is an outline description of the entities that support storage functions. Section 3.3 and 3.4 describe example operational sequences for activities that can result from the **passivate, activate, move** and **migrate** operations.

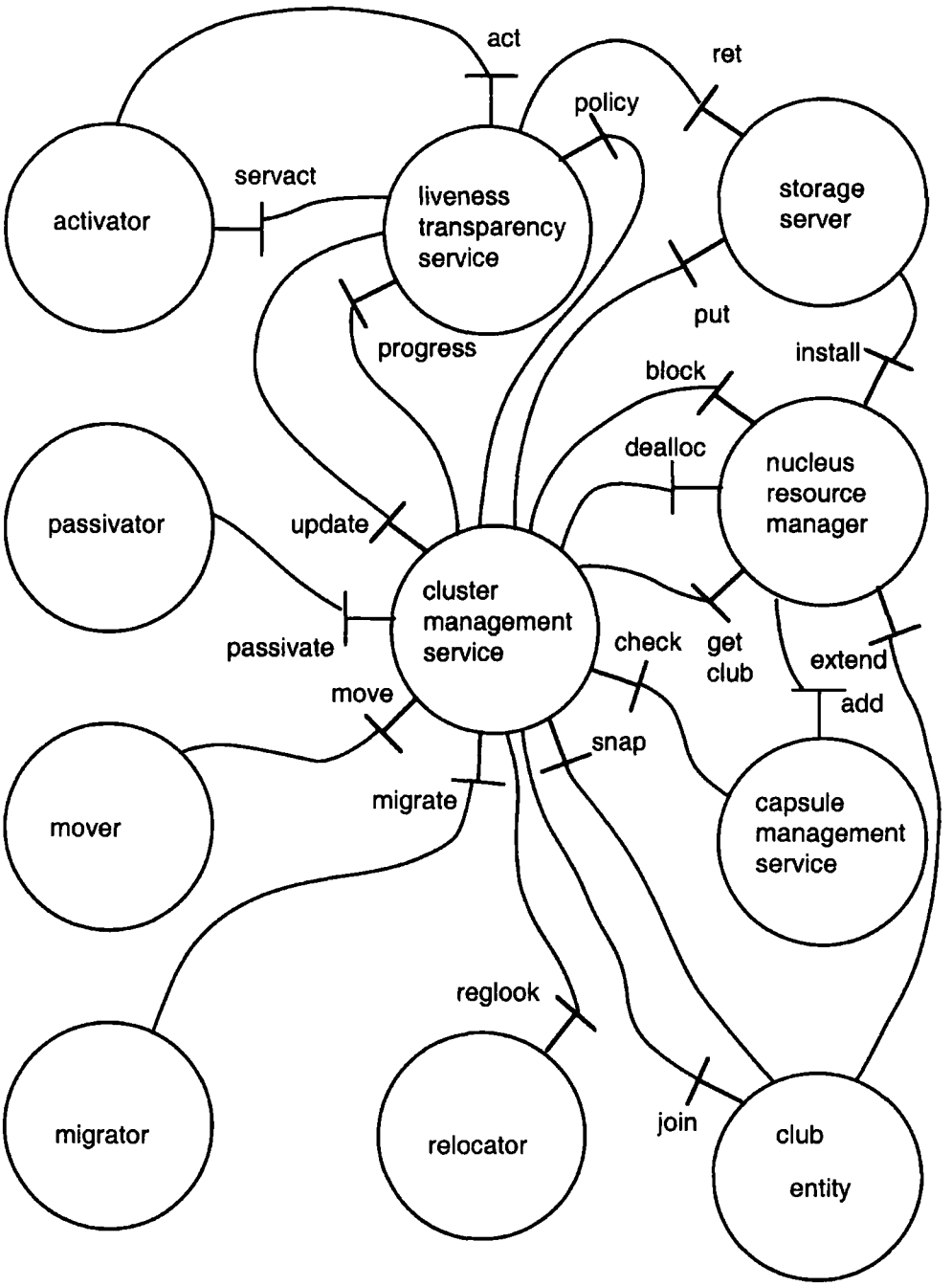


Figure 1 Interfaces supporting storage functions

3.2 The entities supporting storage functions

Figure 1 illustrates a system of entities supporting storage functions. The configuration of the system with respect to the number and distribution of interfaces in the figure is to serve as a guideline only; other configurations could be devised without altering the effect of the behaviour of the system.

The services provided by the activator, passivator, mover and migrator are not described in

this document; their functions are better understood in an overall system-wide context. Also, it is not necessary to state the locations of the following supporting services relative to the clusters involved in storage functions: activator, passivator, mover, migrator, relocator, storage server, liveness transparency object. The migration transparency object is not described as its function is similar to that of the liveness transparency object in the support of storage functions. Thus for exposition convenience, the liveness transparency service is invoked in support of migration functions.

The club entity, the capsule management service, the nucleus resource manager and the cluster management service are located at the same node as the objects being moved in storage operations. The club entity allows an interface to be replicated (form a club) for purposes of fault tolerance, and similar requirements. In some cases, it is useful to have more than one instance of these services, e.g. originating and terminating cluster management services in the migrate activity. The services supported by each operation are outlined in this section. Their use in support of storage functions are described in sections 3.3 and 3.4. A detailed specification is presented in section 5.

3.2.1 Liveness transparency service

3.2.1.1 *interface:* *act*

operation: activate

returns: activation_in_progress, passivation_in_progress

service_activated

The liveness transparency service holds references of passivated services for which it provides liveness. This operation is invoked in activating such services. The passivation_in_progress response indicates that an activate activity clashes with a passivate activity.

3.2.1.2 *interface:* *policy*

operation: passivation_initiated

returns: BOOLEAN

The liveness transparency service holds policy information on services for which it provided liveness. The cluster management service of the cluster being passivated invokes this operation on its liveness transparency service in ensuring that none of its policies is being violated.

3.2.1.3 interface: progress

operation: migrate_initiated

returns: continue_migrate, abort_migrate

This operation solicits the liveness transparency service to authorize continuation of a migrate activity.

3.2.2 Cluster management service

3.2.2.1 interface: upd

operation: update_relocator

returns: relocator_updated

This operation instructs the cluster management service to perform an update on the relocators of all the services it hosts, registering with these relocators the new interface references of its services.

3.2.2.2 interface: pasv

operation: passivate

returns: BOOLEAN (e.g. passive_ok)

This operation instructs the cluster management service to execute the activity which makes itself passive.

3.2.2.3 interface: move

operation: move

returns: BOOLEAN (e.g. moved_ok)

This operation instructs the cluster management service to execute the activity which results in moving its self-representation to another capsule where it is instantiated.

3.2.2.4 interface: migrate

operation: migrate

returns: BOOLEAN (e.g. migrated_ok)

This operation instructs the cluster management service to execute the activity which results in all the services within the cluster migrating to another capsule.

3.2.2.5 interface: snap

operation: install_bindings

returns: installed_ok, send_bindings, joining_at

This operation, sent by an originating cluster management service in the migrate activity, instructs the destination cluster management service to accept pre-declared self-representation information elements of the migrating cluster. The parameters of the response specify further anticipated information elements.

operation: handed_over

response: BOOLEAN

This operation informs an originating cluster management service that migration is complete. It is invoked by the originating club service.

3.2.3 The relocator

3.2.3.1 interface: reglook

operation: lookup

returns: interface_reference

The relocator holds information on the current location of a service interface. This operation supplies such information.

operation: register

returns: BOOLEAN

This operation is used to update the relocator with current location of a service.

3.2.4 The storage server

3.2.4.1 interface: ret

operation: retrieve_and_place

returns: cluster_placed_at_location

The storage server holds a cluster's passive representation in secondary storage. This operation requires the storage server to retrieve a cluster's passive representation from secondary storage and place it at a specified location.

3.2.4.2 interface: put

operation: store

returns: passive cluster name

This operation is used for storing a cluster's passive representation inside a storage server. The passive cluster is named in the context of the storage server.

3.2.5 Nucleus resource manager

3.2.5.1 interface: inst

operation: instantiate

returns: BOOLEAN, (e.g. cluster_instantiated)

The nucleus resource manager is given access to a passive cluster's self-representation in the activate activity. The operation instructs the nucleus resource manager to construct an active cluster from the passive cluster, and to schedule the services on the activated cluster for execution.

3.2.5.2 interface: block

operation: block_sockets

returns: BOOLEAN

The nucleus resource manager schedules services on an object to run, according to a specified threading policy. The block_sockets operation removes a designated set of interfaces from the nucleus's schedules. The nucleus then invokes the end-to-end invocation entity (binder) to reject further invocations on the interface.

3.2.5.3 interface: dealloc

operation: locations_passive

returns: BOOLEAN

The effect of the locations_passive operation is identical to that of the block_sockets operation.

operation: deallocate

returns: BOOLEAN

The effect of the deallocate operation is identical to that of the block_sockets operation.

3.2.5.4 interface: inst

operation: reserve_locations

returns: BOOLEAN, (or interface_references)

This operation is invoked to carry out a check at the destination nucleus to ensure that quality attributes of services in a cluster will not be violated when the cluster is instantiated.

operation: install_cluster

returns: BOOLEAN (e.g. cluster_installed)

This operation is identical to the instantiate operation.

3.2.5.5 interface: getclub

operation: obtain_club_interface

returns: interface_references

This operation is invoked in order to obtain information on a suitable club service to be used by a migrating cluster.

3.2.5.6 interface: extend

operation: extend_club

returns: BOOLEAN

This operation is a request to allocate a suitable club service at a destination node of a migrating cluster.

3.2.6 Capsule management service

3.2.6.1 interface: add

operation: add_cluster

returns: BOOLEAN

On being instantiated, a passive cluster is placed in a memory space of a capsule. This

operation is used for updating the capsule management service after a cluster has been instantiated.

operation: add_club_locations

returns: BOOLEAN

This operation is a purely local update to the cluster management service as a special case for add_cluster operation.

3.2.7 Club entity

3.2.7.1 interface: join

operation: remote_sync

returns: BOOLEAN

This operation, issued by a migrating cluster, is a request to use the club service in creating an active replica of a migrating cluster in a remote capsule.

operation: request_join

returns: BOOLEAN

This operation completes the request to use the club service in creating an active replica of a migrating cluster. It is issued by services at the destination capsule.

operation: sync_one

returns: BOOLEAN

This operation effects club invocation after admitting a new club member. No result synchronization is however carried out, nor is invocation buffered.

operation: sync_invoc_buffers (at originating club interface)

returns: snap_one

This operation commences invocation at the originating club entity.

operation: start_buff

returns: buffer_name (name of invocation buffered)

This operation commences invocation buffering at the destination club entity.

operation: queue_length

returns: queue_size

This operation provides an estimate of the size of a queue of ordered invocations yet to be deleted at the destination club entity of a migrating cluster.

operation: dropout

response: leaving_club

This operation is invoked on the originating cluster by the destination club entity, on completion of migration.

3.3 Activating and passivating objects

3.3.1 The activate operation

The activate operation is used for retrieving a passive cluster's persistency bindings from a storage server and making it active at locations within a specified capsule.

It is assumed that relocators for service interfaces within the passive cluster point to the liveness transparency object of the passive cluster. Thus a consumer of these services performs a lookup operation on an appropriate relocator. In the case of the liveness transparency function, the interface reference of the liveness transparency object is obtained. In the case of the failure transparency function, the interface reference of the checkpoint holder is obtained. In each case, the interface is specified to be capable of receiving invocations on behalf of the services it represents.

The interface reference of the transparency object stored in the relocator is appropriately attributed such that the relocator knows whether to respond to a look-up operation directly back to an invoking service, or after it has invoked an activator. In the activate operation, the activator is invoked by the relocator to activate the passive cluster.

In both the liveness transparency function and the failure transparency functions, the activator supplies as arguments its management interface and the interface of the nucleus resource manager of the node where to cluster is to be reactivated. On receipt of the activate invocation from the activator the liveness transparency service replies indicating activation_in_progress. It then initiates an internal capability to send further progress responses when invoked by the activator. The progress response contains appropriate status indicators for use by the activator in monitoring progress of the activation operation.

The liveness transparency object invokes the nucleus resource manager to reserve_locations (i.e. service access points into end-to-end protocols) for the cluster that is being activated.

The nucleus resource manager records the status of the locations as pending activation of the cluster. This operation is performed in two stages. The second stage ensures that the passive cluster is not fetched from secondary storage in the exception case where the nucleus cannot allocate locations. This operation can thus be embedded inside the instantiation operation invoked by the storage server on the nucleus resource manager (described in the following paragraphs).

The liveness transparency object maintains policy information on where to put an activating cluster. It therefore invokes the selected nucleus resource manager to reserve unspecified locations supplying the properties of the kind of locations acceptable for the activating cluster. The nucleus resource manager returns the capsule management interface reference and the interface reference for locations to be occupied by the activating cluster. The liveness transparency object can therefore acquire knowledge of the destination capsule management interface reference and a set of interface references for the activating cluster.

The liveness transparency object invokes the retrieve_and_place operation on the storage service which holds the passive representation of the activating cluster in stable storage. The liveness transparency object knows the interface reference of the storage service holding the passive cluster since an activating cluster must have been passivated or checkpointed previously. Thus it is expected that the tuple consisting of the passive cluster's interface references and the storage service was previously registered with the liveness transparency object. The storage service invokes the nucleus resource manager to instantiate the cluster, sending the passive cluster's name.

The nucleus resource manager instantiates the cluster and then invokes the capsule management interface to add_cluster to the capsule. It then sends a cluster_instantiated to the storage service. The storage service then returns a cluster_placed_at_ location response to the liveness transparency object.

The liveness transparency object invokes the cluster management interface to update_relocator with the new set of interface references. The cluster management service performs a self-check on its services interfaces and then invokes the relocator with the register operation. On receipt of a response from the relocator, it then sends the relocator_updated response to the liveness transparency object.

The liveness transparency object invokes the activator with a service_activated operation on completion of activation. The activator then responds to the relocator updating the attribute of the services in the activated cluster as active. The relocator can then return the new interface reference of the service being looked up by the consumer.

The above specification of the activate operation is a serial activity. The activator invokes the liveness transparency object to activate the passive cluster. The services involved in the operation perform single operations. The liveness transparency object orders the operations ensuring that the nucleus resource manager reserves locations before the passive cluster is instantiated. The cluster is added to a capsule before the relocator is updated.

3.3.2 The passivate operation

The passivate operation is used for moving a cluster's persistency bindings from active storage to passive storage. It cannot be assumed that once initiated, passivation will continue to completion without being aborted. Thus provision needs to be made explicitly for the passivation operation to be aborted when necessary.

A passivator invokes a cluster's management interface to passivate the cluster. It is assumed that the cluster management service knows its liveness transparency object's interface reference. The cluster management service invokes the liveness transparency object to confirm passivation policy implied in the parameters of the invocation using the passivation_initiated operation. On receipt of an ok response, the cluster management service carries out a register operation on its relocator, providing the relocator with the interface reference of its liveness transparency object. It is assumed that at this point in the sequence, the cluster management service holds the reference to a storage service's interface reference, obtained from its liveness transparency object. On acknowledgement of registration of the liveness transparency object's interface with the relocator, the cluster management interface invokes the block_sockets operation on the nucleus resource manager. With this operation, any incoming invocation on service interfaces of the passivating cluster are ignored. Invocations on the cluster management interface are unaffected.

The cluster management service invokes the passivation_progress on its liveness transparency object and on receipt of ok, invokes a snapshot on its bindings. It is expected that the cluster's persistency bindings are the quiescent and internal administrative bindings. The onus is on the passivator to ensure that no transient bindings are being held by the cluster being passivated.

When the cluster is hosting activities, it is assumed that the passivator has the following options available. The passivator may invoke camp_on_existing / reject_new_invocations on the cluster management service. In this case, all the services inside the cluster discard invocations for new activities, only processing invocations for existing activities. Since the state: camp-on-existing activities could persist for a very long period of time, the onus is on the passivator to set a timer that guards against a protracted outage. Options open are: either the camp-on is lifted and passivation aborted, or existing activities are cleared down

in order for the passivate operation to make progress.

The cluster management interface invokes the storage service operation store, to store the persistency bindings of the passivating cluster. It then receives an acknowledgement from the storage service; included in the acknowledgement is the pair consisting of the passive cluster's interface reference and its extended reference in the context of the storage service.

The cluster management interface then invokes the register operation on its liveness transparency object to register its passive cluster's extended reference. On receipt of a response, it sends a response passive_ok to the passivator, and locations_passive to the nucleus resource manager in order that its socket is released. It is assumed that any invocation on the management interface of a passivated cluster can be looked up in the relocator as usual, the registered location being that of the passive cluster's liveness transparency.

3.4 Moving and migrating objects

3.4.1 The move operation

The move operation is used for moving a cluster's persistency bindings from one capsule to another. The cluster blocks incoming invocations while it moves. Its activity is a specification for the migration transparency function.

It cannot be assumed that once initiated, the move operation will continue to completion without being aborted. Thus provision needs to be made explicitly for the move operation to be aborted when necessary.

An invoker invokes a move operation on the cluster management interface. The cluster management service invokes the move_initiated operation on its liveness transparency object to confirm the policy for the move. It is assumed that the invoker provides the cluster management service with the interface reference of the destination nucleus resource manager. The originating cluster management service invokes a reserve_locations operation on the destination nucleus resource manager. It is expected that the destination resource manager can allocate sockets of appropriate attributes. Attributes state the quality of service expected of a migrating service. These are characterized by end-to-end communication protocols, and processor load at the destination node. If the destination nucleus resource manager cannot accept the move, it returns an exception and the move activity is aborted. If accepted, a positive response is returned to the originating cluster management interface.

The reserve_locations operation is provided explicitly to ensure that a move is not initiated only to be aborted due to the incompatibility between the specified locations provided as

an argument of the move operation, and the actual destination location. If such incompatibility is not likely, then reserve_locations operation does not need to be invoked. It is embedded in the install_cluster operation (described below).

The originating cluster management service registers the interface reference of its transparency object with its relocator and then invokes its nucleus resource manager to block_sockets of applications inside the moving cluster, except the socket of the cluster management interface. On receipt of a response, it sends a move_progress invocation on its transparency object. At this point, the transparency object can terminate the move operation.

The cluster management service ensures that all the outstanding invocations on services within the cluster have been processed. It then copies its persistency bindings and invokes the install_cluster operation on the destination nucleus resource manager. The destination nucleus resource manager sets up the necessary applications' functions and links to the persistency bindings for use by the incoming cluster. It is expected that a schedule (thread in technology terms) has been enabled successfully at the destination nucleus. If successful, the destination nucleus resource manager returns a cluster_installed to the originating cluster management service.

At this point, the originating cluster management service invokes a move_progress on its transparency object. The transparency object now has a chance to abort the move activity if necessary.

The originating cluster management service performs a register on its relocator to register the addresses of the newly instantiated cluster. It then invokes its nucleus resource manager to deallocate the originating cluster. On completion of de-allocation, the originating cluster management service notifies the invoker and the transparency object with moved_ok.

The originating cluster management service must ensure that a remote location is available before commencing the move. Also, the cluster is successfully installed and checked at the destination capsule before the addresses of its locations are registered with the relocator.

3.4.2 The migrate operation

The migrate operation is used for moving a cluster's persistency bindings from one capsule to another. Unlike the basic move operation, disruption in processing incoming invocations is kept to a minimum. This is achieved by not having to block any incoming invocations during the migration.

It cannot be assumed that once initiated, the migrate operation will continue to completion. Thus provisions need to be made explicitly for the migration operation to be aborted when

necessary.

The migration activity ordering consists of

(i) the validation phase

(ii) the initial synchronization phase

(iii) the update phase

(iv) the takeover phase

The validation phase is controlled by the originating cluster management service using the services of its transparency object. It then initiates the initial synchronization phase which invokes the club entity and the nucleus resource manager at both the originating and destination nodes. The initial synchronization ensures that the destination nucleus (and thus the capsule) can accept the migrating cluster.

The update phase involves the movement of bindings from the originating cluster to the destination cluster without blocking of incoming invocations. This sub-activity utilises a continuous update mechanism with a rule provided to ensure termination.

The takeover phase ensures that the relocator is updated with the new locations of the migrated cluster.

The invoker invokes a migrate operation on the cluster management interface supplying the interface reference of the destination nucleus resource manager. It is expected that the destination resource manager can allocate sockets of appropriate attributes. Attributes state the quality of service expected of a migrating service.

The originating cluster management interface invokes its transparency object to confirm the migration policy using the migrate_initiated operation.

On receipt of a positive acknowledgement from the transparency object, the originating cluster management service invokes its local resource manager to obtain an interface to a local club management service. This is the obtain_club_interface operation. It then joins the club by invoking a remote_sync operation on its local club management service. It supplies with this invocation the destination nucleus resource manager's interface reference. Where available, the set of interface references of the locations where the entities will migrate to are also provided.

The originating club entity invokes the destination nucleus resource manager to extend_club into locations (specified to the resource manager or otherwise supplied by the nucleus resource manager). The destination nucleus resource manager invokes the

destination capsule management service to add_club_locations. As part of the arguments of the invocation are the references to locations of services in the migrating cluster. References to the destination club management service are also provided.

The destination nucleus resource manager instantiates an empty cluster for the migrating cluster and then invokes on the elected capsule management service the add_cluster operation. An empty cluster is defined as a cluster which hosts services that do not have any persistency bindings.

When the empty cluster has been fully instantiated (i.e. added to capsule), the destination nucleus resource manager invokes an initiate_join operation on each empty service interface in the cluster. On receipt of the initiate_join invocation, each empty service interface within the cluster invokes its respective club service interface using the request_join operation. The empty service interfaces are extending the club initiated at the originating cluster. On completion of the club extension, the empty service interfaces send responses to their resource manager.

The destination nucleus resource manager then returns the destination club management interface to the originating club entity. Where necessary, interface references to the empty destination clusters are also returned.

The originating club entity invokes sync_one operation on the destination club management interface to initiate club protocols across club entities. This operation is a synchronization protocol to establish originating to destination cluster synchronization. On completion of synchronization, the originating cluster management service invokes migrate_progress operation on its transparency object. At this point, the migration activity can be aborted if necessary, under control of the club protocol. If migration is to proceed, the originating cluster management service invokes sync_invoc_buffers operation on the originating club entity. The originating club entity invokes the start_buff operation on its destination peer to initiate a takeover of the migrating cluster by buffering invocations. The destination club entity acknowledges the invocation by returning a club sequence number of the first entry in its buffer. This information element serves as a synchronization token for the buffering activity.

The originating club entity then responds to the sync_invoc_buffers invocation with the invitation to perform the initial migration snapshot operation, the snap_one operation. Bindings copied are the quiescent, transient and internal administrative bindings. A sequence number marker is provided to denote the last processed invocation. Once copied, further bindings are structured as incremental bindings according as a desired incremental step governed by both the sequence number and a timer.

The sequence number is used as an aid towards transfer of bindings (as described below). The timer is provided to guard against a failed takeover by the destination cluster.

On receipt of the snap_one response, the originating club entity invokes its cluster management service to carry out the send_isnap operation (initial snapshot). This operation results in bindings being sent to the destination cluster by invocation of the install_bindings operation on the destination cluster management interface.

The following is a generic set of operations for joining a club in satisfying the migration function stated in section 3.4.2. On receipt of bindings on the install_bindings operation, the destination cluster obtains its queue length at its destination club management interface. This is the queue_length operation. It then uses this information to decide the invocation sequence number (y) at which it can successfully join the club by processing invocations as a member of the club. Thus it can send responses to its peer at the originating cluster management service:

<installed_ok (x), send_bindings (q), joining (y)>.

The parameter x is the sequence number covered by the received incremental update to bindings. q denotes all the incremental bindings since x, that needs to be installed next.

As well as notifying its peer about joining (y), this operation also serves as an invocation from the destination cluster management service to the destination club management service to commence synchronization of club processing results.

The operations for copying across incremental bindings and the club operations for synchronization can be applied repeatedly in cases where synchronization cannot be achieved in one step, until synchronization is eventually achieved or the migration activity is aborted.

In the successful case, the destination club entity invokes the dropout operation on the originating club entity. It is assumed at this point that the originating cluster has successfully handed over processing to the destination cluster. The originating club entity invokes its local cluster management service with the handed_over (leaving club) operation. The originating cluster management service performs a register operation on its relocator, registering the destination locations. On receipt of acknowledgement, it then acknowledges the handed_over from the originating club entity. The originating club entity acknowledges the dropout invocation with the leaving_club_ok response.

The originating cluster management service cleans up itself by reporting to both its transparency object and its client that it has migrated_ok. It then invokes its local nucleus resource manager to deallocate itself.

4 The Interface Reference as a *type*

The operations specified in the storage functions have the following format:

Operation (Argument : Type)

→ Result (Argument : Type)

Value objects

pre - denote state variables and the values operated upon by the transition function

post - denoting state variables and their values resulting from enabling the state transition function induced by the operation.

Since computations are carried out by objects, the parameters supplied to a computation and the results of a computation can be hidden within a named object, if it is not necessary to examine these data items at a given level of specification. Thus by passing the references to object's interfaces, arguments and results of computations can be defined implicitly. The details of a computation under discussion should be clear, at a high level, from the description of the operation signature (as presented in the previous section); further description of an invocation's computations are provided in the definitions of bindings within the detailed specifications (as in the next section).

5 Detailed specification of storage operations

5.1 Introduction

This section presents a detailed specification of a set of operations in the storage functions. Example concurrency control policies are specified. Bindings generated by the operations are defined, supported by conditional operational statements. These statements serve as an aid to verification of the specification.

5.2 Specification of operations and statements for some concurrency requirements

5.2.1 Introduction

In the previous section, serial behaviours were described as a simple top level specification of storage activities: activate, passivate, move and migrate. These descriptions now serve as the starting point for a refined specification satisfying specific concurrency constraints.

In this refinement of the serial behaviours, the storage system is envisaged as a collection of sequences of operations. Each operation has a source and a sink. The sink carries out processing of an operation issued by a source. Thus the sink issues a response to each invocation from a set of possible responses. For each operation, the following is specified at the sink:

- definition of names and arguments of the operation and its responses
- definition of bindings before operation execution
- definition of bindings after operation execution

In the specification of each operation, it is necessary to allow for implementation flexibility, i.e. many implementation models that solve the same problem. Such flexibility arise due to the following:

- an operation may commute with another operation
- an operation may conflict with another operation
- an operation could be combined with another operation to produce the same observable effect

The execution of an operation results in the enabling and disabling of bindings. The enable/disable attributes of a binding can be further refined to cover other attributes. Meanings can then be given to these attributes in their use to reflect the state of a computation or activity. When no change to bindings' attributes occur after the execution

of an operation, such a special case can be described as an identity change in bindings' attributes.

In order to ensure that resources no longer required for computation can be reclaimed, each binding is attributed as starting, history or transient. In brief, this set of attributes reflect the closure state of an activity. Meaningful changes to an attribute of a binding are as follows:

- starting → starting
- starting → transient
- transient → history
- future → starting

The future attribute may be considered to be an abuse of the notion of a starting binding, when it is cumbersome to declare beforehand (i.e. provide a reference for) every future possible binding.

A binding can be typed as an IREF (Interface Reference) or an INAM (Internal Administrative). A binding of type IREF can be passed as an argument or a result of an invocation. The INAM is an internal administrative binding within an object. It is useful for reasoning about purely internal bindings within an object, or cluster. The ultimate effect of changes to such bindings is the IREF.

In the application of this set of refinement procedures, a diagram which illustrates the relationship among entities that constitute a system can be obtained. Figure 1 illustrates such a diagram for the storage system. The lines joining a pair of boxes can be interpreted as representing a set of shared operations.

5.2.2 Liveness transparency object's view of activator, passivator, mover and migrator

Service: liveness transparency policy

Activities: activate, passivate, move, migrate

Views: activator, passivator, mover, migrator

The activator invokes the activate operation on the liveness transparency service with the aim of activating a passive service. The passivator invokes a cluster's management interface to initiate passivation of the cluster. The liveness transparency service sees this invocation as the passivation_initiated operation, on being invoked by the cluster management service to confirm the cluster's policy on passivation. This same procedure applies to the move and migrate services where the liveness transparency service sees the move_initiated and migrate_initiated operations respectively.

Orderings and conflicts

activate and passivate conflict since an activate operation in progress may need to be aborted when a passivate operation is invoked. Also, a passivate operation in progress may need to be aborted when an activate operation is in progress. The activate and move operations are specified to commute, as are the following:

activate - migrate, passivate - move,

passivate - migrate, move - migrate.

Detailed generic specifications

Policy checking at liveness transparency object:

Each service interface has its attributes which can be used to check various invocation policies, e.g. passivation and activation policies. It is left open how a service's attribute is interpreted by a user. The liveness transparency service holds various storage policies that apply to services for which it is a transparency. It carries out policy checking by using attribute information supplied in an invocation as well as other information that it holds internally to compute a BOOLEAN result.

5.2.2.1 act INTERFACE of liveness transparency object

ACTIVATE operation is invoked by activator

```
Activate(PassiveClusterMgtRef:InterfaceRef
        ServiceInterfacesInCluster:IRefList
        DestinationResourceManager:InterfaceRef
        ConsumptionAttribute:STRING)
                → ActivationInProgress()
                → PassivationInProgress(cause:STRING)
```

Binding statements and persistency attributes

ActivationInProgress response

```
        pre-binding - ClusterPassive:Starting
                      ClusterActivating:Starting
```

```
post-binding -
                ClusterPassive:Transient

                ClusterActivating:Transient
```

Description of bindings

ClusterPassive and ClusterActivating are names for bindings to interface references of services in the cluster, and the cluster management service's interface reference. Included in each of the bindings is the passive cluster's name in the context of the storage service holding the passive cluster. The values of these bindings have meanings as suggested by their names.

Statements : The change in attribute of ClusterActivating is a leads-to relation towards activation of the cluster.

PassivationInProgress response

```
pre-binding  -  ClusterPassivating:Transient

post-binding  -  ClusterPassivating:Transient
```

Description of binding

ClusterPassivating is a name for a binding to interface references of services in the cluster, and the cluster management service's interface reference. Included in the binding is the passive cluster's name in the context of the storage service holding the passive cluster. The value of the binding has meaning suggested by its name.

Statements - Identity

5.2.2.2 policy INTERFACE of liveness transparency object

PASSIVATION_INITIATED operation is invoked by cluster management service

```
PassivationInitiated(ClusterMgtRef:InterfaceRef

                ServiceInterfacesInCluster:IRefList

                PassivateInitiatorAttribute:STRING)

                → PassivateAccept()

                → PassivateReject(cause:STRING)
```

Binding statements and persistency attributes

PassivateAccept response

```
pre-binding  -  ClusterActive:Starting
                ClusterPassivating:Future
post-binding -  ClusterActive:Transient
                ClusterPassivating:Transient
```

Description of bindings

ClusterActive and ClusterPassivating are names for bindings to interface references of services in the cluster, and the cluster management service's interface reference.The values of these bindings have meanings as suggested by their names.

Statements - unless AttributeAcceptable:FALSE

AttributeAcceptable is an INAM binding within the liveness transparency service. Its value is BOOLEAN,computed using the PassivateInitiatorAttribute supplied with the invocation.

5.2.2.3 servact INTERFACE of activator; description of liveness transparency service

SERVICE_ACTIVATED operation is invoked by liveness transparency service

```
ServiceActivated(PassiveClusterRefTuple:IRefList
        ServiceInterfacesInClusterTuple:IRefList)
        → Fine()
```

Description of IRefList

PassiveClusterRefTuple is a name for a list of old and new cluster management interface references. ServiceInterfacesInClusterTuple is a name for a list of old and new service interface references.

Binding statements and persistency attributes within liveness transparency service

Fine response

```
pre-binding -        ClusterActive:Transient

post-binding -       ClusterActive:Transient
```

Description of binding

ClusterActive is a name for the binding to interface references of services in the cluster, and the cluster management service's interface reference. An attribute Starting denotes (within the liveness transparency service) the potential of the cluster being active. The attribute Transient denotes that the liveness transparency service knows the cluster is active (as informed by the cluster management service).

Statements -

Identity. This means that the liveness transparency service has reached a fixed point with respect to the activation activity.Since an active state of a cluster consumes resources, the attribute of the cluster is left as Transient.

5.2.3 Liveness transparency object's view of storage server and cluster management service

Service: liveness transparency service

Activities: activate, passivate, move and migrate

Views: storage service, cluster management service

The liveness transparency service invokes on the storage service the retrieve_and_place operation on the storage service, in the activation activity. This operation is performed in its (i.e. liveness transparency's) capacity as the controller of the activity. Success of this operation, and absence of any conflicting activity, is followed by the invocation of the operation update_relocator on the cluster management service.

The liveness transparency service sees the cluster management service in checking policies for the activate, passivate, move and migrate functions. In this view, it can be invoked to abort an activity of any of these functions in cases of conflict described in the previous section.

Orderings and conflicts

The retrieve_and_place operation on the storage service returns cluster_placed_at_location. If the activation activity conflicts with a passivation activity, the serial behaviour described in section 5.4 requires that the activation activity progresses to completion before the

passivation activity is allowed to commence. After the issue of cluster_placed_at_location response, it is possible that the passive cluster stored in the storage service is garbage collected before it (the passive cluster) is needed again when the activation is followed by a passivation. This condition arises because once retrieved, a passive cluster inside a storage base can be given a history attribute. Thus where there is a need to allow an activation activity to complete before a conflicting passivation is initiated, the cluster_placed_at_location response is converted to an invitation: cluster_placed_at_location / give_passive_cluster_history_attribute. With this invitation, the liveness transparency service can invoke either a passive_cluster_still_needed or a passive_cluster_no_longer_needed, on the storage server.

If the activation activity is followed immediately by a passivation, the liveness transparency service invokes the newly created cluster management service to delete itself.

The passivate, move and migrate activities can be aborted by the liveness transparency object, at appropriate progress points. Thus the specification presented in this document can be extended to allow such abortions to proceed concurrently with other activities.

Detailed generic specifications

5.2.3.1 ret INTERFACE of storage server; description of liveness transparency service

RETRIEVE_AND_PLACE operation is invoked by liveness transparency service

```
Retrieve_And_Place(OldPassiveClusterMgtRef:InterfaceRef
                   ServiceOldReferencesInCluster:IRefList
                   DestinationResourceManager:InterfaceRef
                   DestinationLocationAttributes:STRING)
                   → ClusterPlacedAtLocation(
                         OldAndNewInterfaces:IRefList)
                   → RetrieveFailed(cause:STRING)
```

Binding statements and Persistency attributes

ClusterPlacedAtLocation response

```
pre- bindings - ClusterPassive:Transient
                ClusterActivating:Transient
post- bindings - ClusterPassive:Transient
                ClusterActivating:Transient
```

Description of bindings

Included in ClusterActivating is a binding used for indicating that progress is being made towards full activation.

Statements

The extension operation resulting in the extra binding in ClusterActivating is a leads-to relation towards activation of the cluster.

5.2.3.2 upd INTERFACE of cluster management service; description of liveness transparency service

UPDATE_RELOCATOR operation is invoked by liveness transparency service

```
UpdateRelocator(OldAndNewInterfaces:IRefList)
        → RelocatorUpdated()
        → UpdateRejected(cause:STRING)
```

Binding statements and persistency attributes

RelocatorUpdated response

```
pre- bindings - ClusterPassive:Transient
                ClusterActivating:Transient
post-bindings - ClusterPassive:History
                ClusterActivating:History
```

Description of bindings

Included in ClusterActivating is a binding used for indicating that progress is being made towards full activation.

Statements

On processing the response from the relocator, an extension of the extra binding is a leads-to relation towards a fixed point in the activation of the cluster.

5.2.3.3 progress INTERFACE of liveness transparency service

PASSIVATION_PROGRESS operation is invoked by cluster management service

```
PassivationProgress(ClusterMgtRef:InterfaceRef)

        → ContinuePassivation()

        → CeasePassivation(cause:STRING)
```

Binding statements and persistency attributes

ContinuePassivation response

```
pre- binding - ClusterPassivating:Transient

post-binding - ClusterPassivating:Transient
```

Description of binding

ClusterPassivating is a name for bindings to interface references in the cluster, and the cluster management service's interface reference. Included in ClusterPassivating is a passivation binding which has a BOOLEAN value continue/cease as TRUE/FALSE. This value is computed internally as a result of timer expiry or after processing of a conflicting external invocation.

Statements - unless passivation binding:FALSE

5.2.3.4 progress INTERFACE of liveness transparency service

REGISTER_PASSIVE_CLUSTER operation is invoked by cluster management service

```
RegisterPassiveCluster(PassiveClusterMgtRef:InterfaceRef
                       ServiceInterfacesInCluster:IRefList
                       PassiveClusterName:STRING)
                       → Registered()
                       → Rejected(cause:STRING)
```

Binding statements and persistency attributes

Registered response

```
pre- bindings - ClusterPassive:Future
                ClusterPassivating:Transient
post- bindings - ClusterPassive:Starting
               - ClusterPassivating:History
```

Description of bindings

ClusterPassive and ClusterPassivating are names for bindings to interface references of services in the cluster, and the cluster management service's interface reference. Included in ClusterPassive is the passive cluster's name in the context of the storage service holding the cluster. The values of these bindings have meanings as suggested by their names.

Included in ClusterPassivating is a passivation binding which has a BOOLEAN value continue/cease as TRUE/FALSE. This value is computed internally as a result of a timer expiry or after processing conflicting external invocations.

Statements - unless passivation binding:FALSE

5.2.4 Cluster management service's view of storage server and relocator

Service: cluster management service

Activities: activate, passivate, move, migrate

Views: relocator, storage service

The cluster management service invokes the register operation in support of the activate, passivate, move and migrate functions. In the activate function, the register operation is used to register the new references of a previously passivated cluster. In the passivate function, the interface reference of the cluster's liveness transparency service is registered. In the move function, there are two register operations: the first operation registers a temporary_out_of_service exception while the second operation registers the new interface references of the migrated cluster. In the migrate function, the register operation is used to register the new references of the migrated cluster.

The cluster management service sees the storage server in the store operation which moves the bindings of a passivating cluster into a storage server.

Orderings and conflicts

Both the storage server and the relocator act as servers to the cluster management service in the provision of storage functions. Thus the register and store operations issued by a cluster management service are ordered according as the cluster's involvement in the activate, passivate, move and migrate functions. As long as the cluster management service does not violate these storage functions, no ordering conflicts arise.

When the store operation is invoked in the specification of various transparency functions, it is likely that a number of such operations would be serviced concurrently by the storage server. It is therefore necessary to name each store operation within the cluster management service in the storage service's context in order to ensure determinate referencing of passive clusters. Strict ordering of stored snapshots is however necessary in order to ensure that no vital information is lost on retrieval of a previously stored snapshot.

Detailed generic specifications

The specifications for register and store operations provided in the next sections are realized in the relocator and storage server respectively.

5.2.4.1 reglook INTERFACE of relocator; description of relocator

REGISTER operation is invoked by cluster management service

```
Register(OldAndNewInterfaces:IRefList

        RegistrationAttribute:STRING)

        → Registered()

        → Rejected(cause:STRING)
```

Description of arguments

OldAndNewInterfaces is a name for a pair of sets of interface references. Each component of the pair consists of a list of interface references. The Old- component of the pair is a list consisting of a previous interface reference to a cluster's management service and a set of interface references to services in the cluster. The New- component may be described in the same way.

The Old- component can be absent in the case where registration is carried out for the first time. It is possible that any, or both components of the pair consist of one and only one interface reference - that of a cluster management service.

RegistrationAttribute is a name indicating the type of service whose interface reference is denoted by the New- component of OldAndNewInterfaces. Two possibilities are service clusters and transparency clusters.

Binding statements and persistency attributes within relocator

Registered response

```
        pre-bindings  -  CurrentInterfaces:Starting

                         NextCurrentInterfaces:Future

                         NextNextCurrentInterfaces:Future

        post-bindings  -  CurrentInterfaces:Transient

                         NextCurrentInterfaces:Starting

                         NextNextCurrentInterfaces:Future
```

Description of bindings

CurrentInterfaces and NextCurrentInterfaces are names for bindings to sets of interface references. Included in these bindings are names for registration attributes of the services denoted by the interface references. NextNextCurrentInterfaces is similarly a name for bindings. The structure:...,Current,NextCurrent,NextNextCurrent,... is a notation used in the partial ordering of updates to a relocator's bindings.

Statements

The change in attribute of a binding from Future to Starting is an ensures relation in the context of the storage system. This means that the relocator always provides the latest information for accessing a service. Attribution of bindings as Transient etc. is for use by resource managers managing garbage collection within the system.

5.2.4.2 put INTERFACE of storage server; description of storage server

STORE operation is invoked by cluster management service

```
Store(ClusterMgtRef:InterfaceRef

        ServiceInterfacesInCluster:IRefList

        StoreInvocationName:STRING

        PassiceClusterPersistencyBindings:STRING)

        → Stored(StorageInvocationId:STRING)

        → Rejected(cause:STRING)
```

Description of argument

StoreInvocationName is a name denoting a specific invocation of the STORE operation in order to allow for concurrent commutative non-conflicting invocations of the storage server.

Binding statements and persistency attributes within the storage server

Stored response

```
        pre-bindings  -  InvocationIdTaggedObject:Future

        post-binding  -  InvocationIdTaggedObject:Starting
```

Description of binding

InvocationIdTaggedObject is a name for a stored persistency binding held within the storage service. Included in the named binding is the STORE invocation identity in the context of the storage server.

Statements

The change in attribute from Future to Starting is an ensures relation in the context of the storage system. This means that the storage server will make available a previously stored PassiveClusterPersistencyBinding.

5.2.5 Storage server's view of nucleus resource manager

Service: storage server

Activities: activate

Views: Nucleus resource manager

The storage server invokes on the nucleus resource manager the instantiate operation. Names of locations may be provided as part of the arguments of the operation in stipulating new locations for the cluster management service and associated application services.

Orderings and conflicts

In this computation, the storage server only sees the nucleus resource manager which can reject the invocation due to resource availability constraints.

Detailed specifications

5.2.5.1 inst INTERFACE of nucleus resource manager

INSTANTIATE operation is invoked by storage server

```
Instantiate(NewPassiveClusterMgtRef:InterfaceRef

        ServicesNewReferencesInCluster:IRefList

        PassiveClusterPersistencyBindings:STRING
```

```
LocationOption:BOOLEAN)

→ Instantiated(ActualNewClusterMgtRef:InterfaceRef

        ServiceActualReferencesInCluster:IRefList)

→ Rejected(cause:String)
```

Description of arguments

It may be optional that the New- cluster interface references suggested as arguments of the invocation be treated as hints, as indicated by LocationsOption. Where it is a hint, the Actual cluster interfaces may differ from the hinted interfaces.

Binding statements and persistency attributes

Instantiated response

```
        pre-bindings -
ActualLocationOptionAcceptable:Starting

        post-bindings -
ActualLocationOptionAcceptable:Transient
```

Description of bindings

ActualLocationOptionAvailable is a name for a binding to locations allocated by the nucleus resource manager. Included in this binding is a name for an internal administrative binding denoting that an internal computation using LocationsOption results in a value TRUE.

Statements

This is an ensures relation. The nucleus resource manager on performing an instantiation ensures that a service can be provided by the cluster.

5.2.6 Nucleus resource manager's view of capsule management service, cluster management service, and club entity

Service: Nucleus resource manager

Activities: activate, passivate, move, migrate

Views: capsule management service, cluster management service, club entity.

The nucleus resource manager invokes the add_cluster operation on the capsule management service once it (i.e. the nucleus) has instantiated the cluster. The objective of this operation is to put the newly instantiated cluster within the capsule's address space.

The nucleus resource manager sees the cluster management service's invocation to block_sockets in the passivation activity. This blocking operation only applies to application services inside the cluster, leaving the cluster management service's interface unaffected. However, on completion of passivation, the nucleus resource manager sees the cluster management service's commit response location_passive.

In the move activity, the nucleus resource manager sees the block_sockets operation. In this activity, the cluster management service invokes the deallocate operation on the nucleus resource manager at the migrating node since the sockets are no longer required after the move.

If a nucleus is specified to accept moved clusters, it is useful that the cluster management service being moved invokes a reserve_locations before initiating the move in order to ensure that there is no conflict in quality of service provision at the destination node. This operation is followed by the install_cluster operation from the cluster management service of the cluster being moved.

In the migration activity, a club service is required in order to minimize delay in processing of invocations during the migration. An explicit club service is required where the originating and destination locations are at different nodes. Thus a nucleus resource manager at a node from which a cluster migrates is invoked by a cluster management service using the obtain_club_interface operation. When serving as a node to which a cluster migrates, the nucleus resource manager is invoked using the extend_club operation, to add a set of its locations to the club interfaces created at the originating nucleus.

The nucleus resource manager elected to host a migrating cluster sees the capsule management service at its local node in the invocation to add_club_locations.

Orderings and conflicts

Operations supported by the nucleus resource manager are implicitly ordered by the execution sequence specified for liveness transparency service. Thus it can be stated that the nucleus resource manager's operations do not conflict.

In the move activity, it is acceptable to combine the reserve_locations with the install_cluster operations. However, the consequence of failure to satisfy necessary quality of service conditions for placement of a moved cluster at a new location needs to be judged against the benefits of combining the two operations.

Detailed generic specifications

The add_club_locations operation.

This operation is a special case of adding a cluster to a capsule. A response is effected only after the club has been successfully joined.

5.2.6.1 add INTERFACE of capsule management service

ADD_CLUSTER operation is invoked by nucleus resource manager.

```
AddCluster(ClusterMgtRef:InterfaceRef
        ServiceInterfacesInCluster:IRefList)
        → ClusterAvailable()
        → AddReject(cause:STRING)
```

Binding statements and persistency attributes

ClusterAvailable response

```
pre-binding  -  ClusterInterfacesAccessible:Future
post-binding  -  ClusterInterfacesAccessible:Starting
```

Description of bindings

ClusterInterfacesAccessible is a name for bindings to interface references of services in the cluster as seen by the capsule. The attribute Starting denotes that the cluster is available for service. An attribute Transient is used for a cluster state of interest to the capsule.

Statements

An instantiation of a cluster is effected by placing the cluster within the capsule; this operation is effectively an update operation on the information maintained within the capsule. In particular, such an update to internal administrative bindings ensures that incoming invocations are routed into appropriate services.

5.2.7 Cluster management service's view of club entity

Service: cluster management service

Activity: migrate

View: club entity

The cluster management service invokes a club entity local to its nucleus in order to effect

a synchronized hot migration. This is the remote_sync operation on the club entity. The cluster management service therefore delegates the hot migration set of operations to the club service.

The migrating cluster management service is invoked by the originating node's club entity to perform snapshots for migration. The first snapshot invocation is the snap_one (snapshot number one) operation.

The migrating cluster management service sees a generic migrate_bindings operation which stipulates from-to bindings according as the club's incremental steps of stored bindings. This operation is invoked by the originating club entity. A special case of this set of invocations is the send_isnap (send initial snapshot) operation.

The migrating cluster management service sees the handed_over operation, which effectively instructs it to cease processing of further service invocations.

The cluster management service at the destination node is provided with local operations to estimate its processing throughput during a given period of time, and to forecast the next set of throughput capabilities. Thus it can invoke a queue_length operation on its local (destination) club entity in its invocation of the < installed (), send_bindings (), joining () > operation.

Orderings and conflicts

Ordering of the interaction between the cluster management service and the club entity is governed by the last three of the migration phases:

- the initial synchronization phase
- the update phase
- the takeover phase

The remote_sync operation is issued during the initial synchronization phase. The set of snapshot and migration of binding operations are issued during the takeover phase. These phases cannot be allowed to commute or conflict since the ordering is specified to satisfy the migration function.

Detailed generic specifications

5.2.7.1 join INTERFACE of club entity; description of club service at migrating node.

EXTEND_CLUB operation is invoked by cluster management service

```
ExtendClub(DestinationNucleusInterface:InterfaceRef
        DestinationsDervicesAttributes:STRING
        MigratingClusterType:STRING
        LocationsAttributesOptions:BOOLEAN)
        → SynchronizedOK(ActualServicesInterfaces:IRefList
                        ActualClusterMgtRef:InterfaceRef)
        → RejectSync(cause:STRING)
```

Description of arguments

It may be optional that the destination cluster interfaces' quality of service attributes suggested as arguments of the invocation be treated as hints, as indicated by LocationsAttributesOptions. Where it is a hint, the actual cluster interfaces may differ from the hinted interfaces. The MigratingClusterType serves to specify its type to the destination nucleus service for use in the instantiation of the duplicate.

Binding statements and persistency attributes

SynchronizedOK response

```
        pre-bindings  -  JoiningInterfacesInClub:Future
                         MigrationClubInterface:Starting

        ActualDestinationServicesInterfaces:Future
                         ActualDestinationMgtInterface:Future
        post-bindings  -  JoiningInterfacesInClub:Starting
                         MigratingClubInterface:Transient

        ActualDestinationServicesInterfaces:Starting
```

```
ActualDestinationMgtInterface:Starting
```

Description of bindings

JoiningInterfacesInClub is a name for a binding to the interfaces references of services in the migrating cluster. MigratingClubInterface extends this binding to denote a club interface used for migration.

Statements

This operation is a leads-to relation. The club extension leads to buffering and snapshot computations for the migrating activity.

5.2.7.2 snap INTERFACE of cluster management service

SNAP_ONE operation is invoked by the club entity

```
SnapOne(IncrementalOption:STRING)

        → SnapTaken(Range:INTEGER)

        → RejectSnap(cause:STRING)
```

Description of arguments

IncrementalOption is a name that stipulates to the cluster management service the structure of incremental bindings. This structure can be on a per-timer basis, or on a per-invocation count. Irrespective of the invocation option, the range denotes the (ordered) last invocation processed before the first migration snapshot (SNAP_ONE) was taken.

Binding statements and persistency attributes

SnapTaken response

```
        pre-bindings  -  BindingIncrementalStructure:Starting

                         BindingInvocationFromRange:Starting

                         BindingInvocationToRange:Starting

        post-bindings  -
BindingIncrementalStructure:Transient

                         BindingInvocationfromRange:Transient
```

BindingInvocationToRange:Transient

Description of bindings

BindingIncrementalStructure is a name for an internal administrative binding denoting the structuring policy for migrating bindings.

Statements

Attribution of BindingIncrementalStructure to Transient ensures structuring of bindings in a way that allows for an agreed continuous incremental update to the service at the new location.

Generation and naming of bindings within a specified range leads-to the computation steps satisfying migration of bindings.

APPENDIX B

A graph model specification of the congestion avoidance function

B.1 Introduction

This appendix provides a detailed description of the evolution of important system properties which characterise the congestion status of shared resource pools. The important entities whose attributes change over time in the specification of constrained system evolution states are the input buffer of traffic admission stream and the utilisation level of each shared resource pool along the path of a stream. Relations are defined over states; sequences of relations over time are also defined. However, the specific values of clock ticks when a specified relation holds is not specified since such values can be provided as simulation configuration.

The description of state evolutions provided in this section complements the operational framework described in section 7.3 of chapter 7. The state transitions are effectively the internal workings of the equipment behaviour automata, implemented as interacting objects. The specification presented here has been refined by the author, and implemented by the author and a co-researcher, Philip Aranzulla, on two modelling environments. The modelling environment Mil 3's OPNET ™ supports the notion of object interactions through message passing. The modelling environment Mathwork's STATEFLOW ™ incorporates Mathwork's MATLAB™ and SIMULINK™ but needed to be enhanced by the author to support the notion of object interactions. These enhancements have been implemented by the author and his co-researchers Philip Aranzulla and Jonathan Pitts. The experiments reported in Chapter 7 were carried out on the enhanced STATEFLOW™ simulation environment.

B.2 State transitions from the geometry of behaviours

B.2.1 Evolution bounds of system entities

Predictions of plausible behaviours of user demands for resources can be represented as geometric figures which can then form the structural basis for generating control policies in the allocation of resource pools to the competing users. Figures B.1 and B.2 illustrate fragments of such geometric patterns that can be used to represent plausible

state evolution values. Definitions for names of the state variables shown on the diagrams are given in table B.1.

Figure B.1 illustrates a fragment of a possible geometric trajectory of snapshot values at an input stream's buffer. The scheduler at this buffer would normally be designed in such a way that while the system is operating at a committed transmission rate of its stream, the buffer is emptied as quickly as it is being filled. When the source generates bursts and transmits packets of information above the committed rate, there could result a build up of packets up to a MARKER level.

Figure B.2 illustrates a fragment pattern of constrained allowable occupancy states for pooled resources. The X-X level is an example occupancy level of the resource pool. The goal of a good resource allocation algorithm would normally be to push the occupancy level to the lower end of the state ENTERING CONGESTION. Thus the provisioned resource is 'well used' at high levels of occupancy. However, control procedures are required to detect congestion at a resource pool and invoke appropriate users to reduce their levels of resource consumption. Where parameters of such schemes are not well chosen, packet discard is allowed when the snapshot value of the resource's occupancy level reaches a threshold value.

B.2.2 State transitions graph automata

The equipment behaviour automata placed at various locations of a network are configured to observe each other, and develop views of the system state evolution at remote peer entities. These views and their encapsulation as states operated upon by a timed automaton implement the notions of product form representation and the graph automata described in this thesis. This subsection is a description of three graph automata: the source module automaton, the shared resource pool automaton and the destination module automaton. Since the source module invokes the necessary congestion avoidance mechanism, a refinement of this automaton is also presented.

B.2.2.1 The source module's automaton

Figure B.3 illustrates state transitions as seen by the originating module. A transmission resource pool traversed by a path is uncongested if all the users of the path are not concurrently sending bursts of packets. Congestion condition at a resource pool is detected jointly by the automaton at the pool, and the destination modules of all the paths that traverse the pool. Thus each destination module raises an indication (operation 1 on figure B.3) when congestion is detected. Under congestion, source

modules transmitting at rates above committed information rate (CIR) reduce their stream rate thereby making the resource pool go into the LEAVING CONGESTION state. If control is relaxed, the system's state for this resource could oscillate between the CONGESTED and LEAVING CONGESTION states. It is important to note that the state ENTERING CONGESTION is not seen explicitly by the source module automaton because it is remote from the location of the congestion, and does not take part in the distributed agreement function carried out at the destination modules of the relevant streams jointly with the automaton at the congested resource pool. Thus operations 5 and 2 on figure B.3 which can also cause an oscillation, are transparent to the source module.

Figure B.4 illustrates the automaton at a resource pool implementing the geometric evolution structure shown in figure B.2. Figure B.5 illustrates the state of the automaton at each destination module of each transmission path. Since the destination module co-operates with the resource pool module to carry out statistical analysis of congestion state information, this state transition is identical to that of B.4 except for the absence of the TRAFFIC DISCARD state.

B.2.2.2 A refinement of the source module's automaton

The source module's scheduler takes snapshots of its input buffer and empties any waiting packets according as the level of credits accumulated at the credit buffer. Thus the credit buffer's level of a stream can be defined in terms of the offered traffic information rate variables. Using this approach a credit level can be defined in terms of a stream's Committed Information Rate (CIR), maximum transient-burst information irate (CMAX), or other intermediate values. When a source stream is not transmitting information into a network, its credits are being awarded at the CIR, and can be saved for future use, the saved credits limited to a maximum value CMAX. When a source module has a build up of packets, and has saved credits, and the path is not congested, it can transmit at a rate greater than its CIR because it *must* be invited by the destination module to increase its transmission rate. Note that the destination module does not know that there is a packet build up at the source module; the invitation is issued because the path is not congested. Any free capacity is to be used up if there is traffic available to exploit the slack in user demand for resources. Thus the source module can spend saved credits at a rate greater than CIR. When the source module has a build up of packets and the path is not congested, it is invited by the destination module to transmit the packets through an increase in the credit award rate, up to an equivalent rate of CMAX. When there is congestion along a shared path, all streams are invoked with respect to credit award rates so that those being awarded rates greater

than CIR are then offered decreasing rates. Reduced credit award rates translate to reduced credit levels which in turn translate to reduced offered packet rate.

Figure B.6 illustrates how a permutation on four important system state attributes captures the specification of states of paths as such paths experience resource consumption bounds. The attributes are:

a) buffer status
b) credit level
c) credit level derivative curve (direction)
d) path status.

Nine composite states are described; these states are inter-related due to the possibility of twenty five operations. These operations are described next.

1: This transition occurs because the stream has remained dormant and so credit build up is operational.

2: This transition occurs because packets are arriving at the input buffer of the stream at a rate approximating the CIR.

3: This transition occurs because packets are arriving at the input buffer of the stream at a rate less than the CIR. The credit level is increasing because credits are being accumulated at a rate faster than the expenditure rate.

4: This transition occurs because packets are arriving at the input buffer of the stream at a rate greater than the CIR. The credit level is decreasing because credits are being spent at a rate faster than the rate that credits accumulate.

5: This transition is similar to transition 3, except that the previous buffer state is EMPTY/FILLING, not EMPTY. The transition sequence is <A-B-C> as compared with <A-K-C>.

6: This transition occurs because the source stream rate is persistently higher than CIR, thus accumulated credits are being spent quicker than the arrival of new credits.

7: This transition occurs because the source stream rate is persistently higher than CIR. All accumulated credits have been spent. If path is not congested, any increase in credit award rate is consumed due to high input stream rate.

8: This transition occurs because accumulated credit is being consumed faster than awarded credit. Credit award rate stays at CIR since there is no accumulation of packets at input buffer.
However, the high input traffic results in the credit level dropping to approximately CIR.

9: This transition occurs because packet arrival at the input buffer far exceeds the rate at which accumulated credits can be spent. Thus there is a build up of packets at the input buffer.

10: This transition occurs because a resource pool along the path of the resource has been detected to be congested.

11: This transition occurs because a resource pool along the path of the stream which has been congested, thereby keeping the credit level at CIR, has become uncongested.

12: This transition occurs because the stream has been notified of congestion, as the path moves into state H. The stream has received invitations to increase its traffic rate and has thus responded, resulting in the buffer level being EMPTY/FILLING. The input stream rate is less than the credit award rate hence credit level is greater than CIR.

13: This transition occurs because the source module has been notified of congestion at a resource pool along its path. Since this source module's credit level is higher than CIR, this source module could be one of the contributors to the congestion at the resource pool.

14: This transition occurs because the resource pools that were congested have now recorded the level LEAVC ~ leaving congestion.

15: This transition occurs because of one of the following:

a) the control parameters of the congestion management algorithm have not been configured satisfactorily, thereby allowing streams being flow-controlled to transmit packet rates that can cause congestion along the path.

b) another resource pool along the path has entered the CONGESTED state.

16,17,18: These transitions occur because the resource pools that were congested have now recorded the level LEAVC ~ leaving congestion. The sequence of states are as follows:

a) transition 16, sequence <F-H-G-E>
b) transition 17, sequence <D-H-G-D>
c) transition 18, sequence <C-H-G-C>

Other transitions such as <C-H-G-E-F> are possible, depending on the intended dynamics of the flow control mechanism.

19: This transition occurs due to a pause in the input traffic stream. When the path is uncongested in state A, credit build up is indicated after a time out. The transition <A-K> or <A-B> are invoked if the traffic streams remain dormant or resume transmission before the time out expires, as the case may be.

20: This transition occurs because when the path is congested, packets build up at the input buffer.

21: This transition occurs because even though the source module is not transmitting any packets, a resource pool along the path is congested due to the volume of traffic offered by the other transmitting streams.

22,23: These transitions occur because the resource pools that were congested have now recorded the level LEAVC ~ leaving congestion.

24: This transition occurs because the path has been uncongested, the build up of packets at the source module has been cleared, and the credit level is building up because traffic arrival is less than the credit award rate.

25: This transition occurs because due to the uncongested status of the path, the credit level has dropped to approximately CIR for this source module.

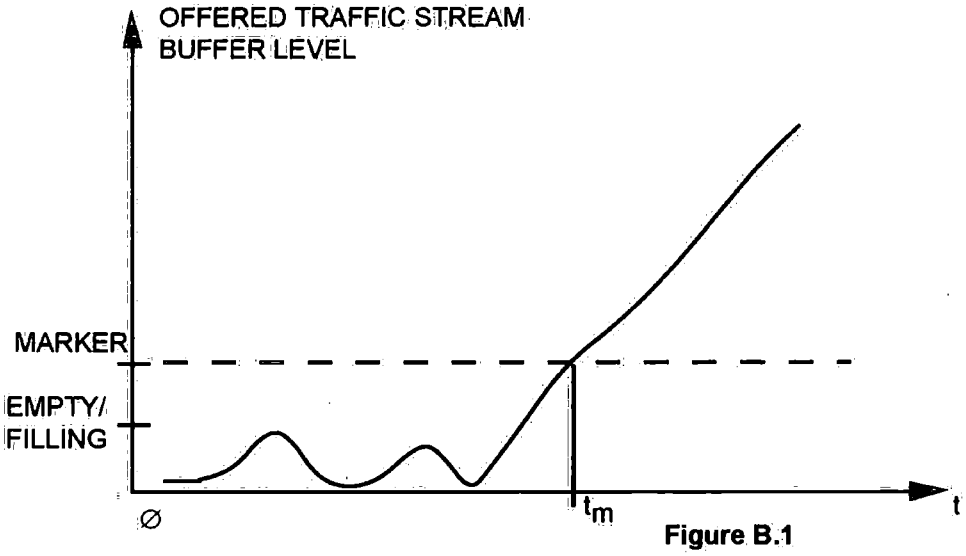


Figure B.1

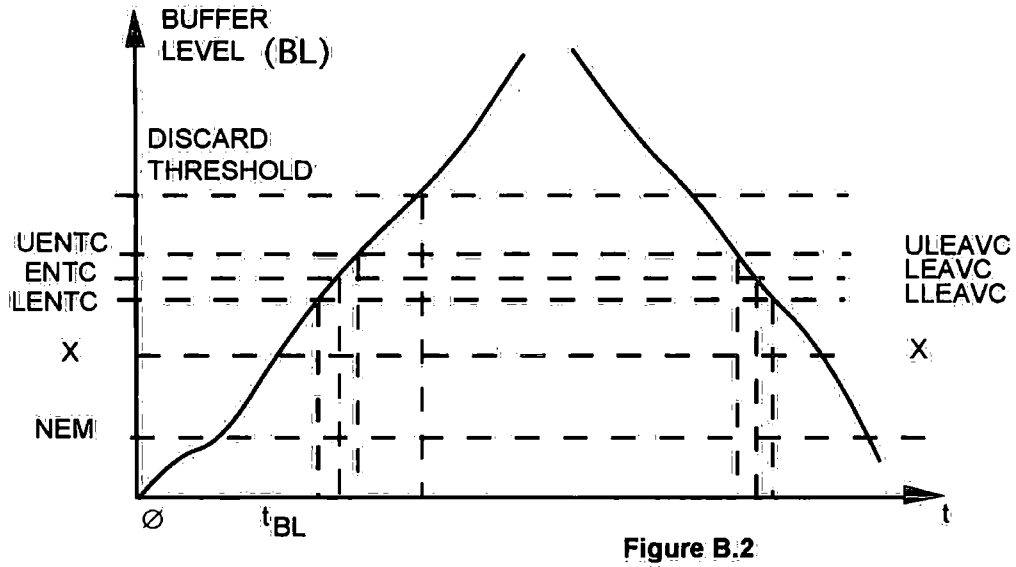


Figure B.2

Figure B.1: Input buffer evolution bounds

Figure B.2: Pooled transmission resource evolution bounds

Table B.1: Names of state variables

MARKER	-	A snapshot value at an input stream's buffer indicating that a build up of packets for transmission has occurred.
EMPTY/FILLING	-	A statistically stable value at an input stream's buffer indicating that a build is being emptied as quickly as it is being filled.

NEM - A statistically stable value at a transmission resource indicating that the occupancy of the resource is negligible (NEARLY EMPTY).

LENTC, UENTC - A statistical LOWER BOUND at a transmission resource indicating that the occupancy of the resource is entering congestion. The UPPER BOUND is UENTC.

ENTC - A statistically stable state at a transmission resource indicating that the occupancy of the resource is ENTERING CONGESTION.

DISCARD THRESHOLD - A snapshot value at a transmission resource indicating that information packets can be discarded.

ULEAVC, LLEAVEC, LEAVEC - Statistically stable states at a transmission resource indicating UPPER BOUND, LOWER BOUND and AVERAGE values for the occupancy of the resource being in the state LEAVING CONGESTION.

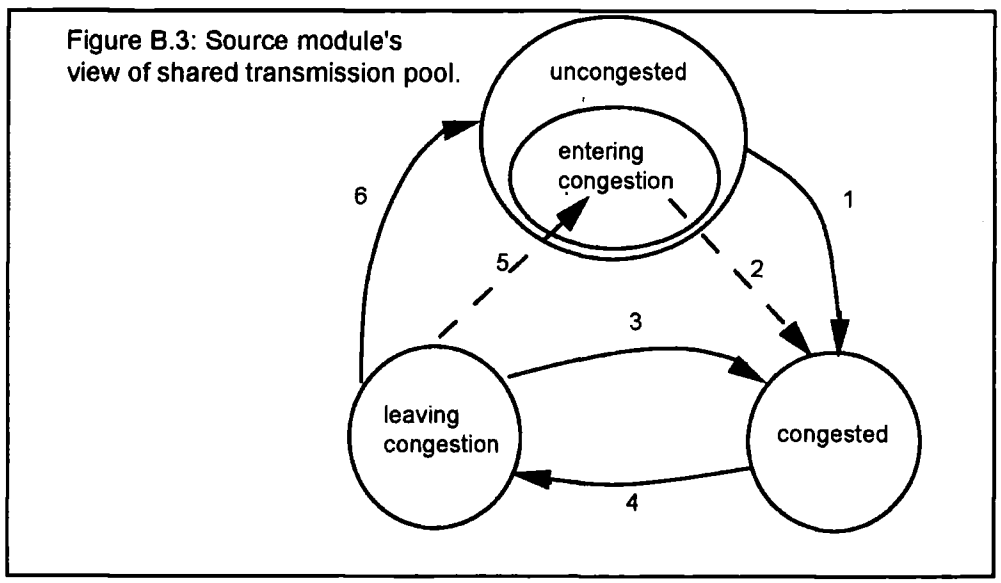


Figure B.3: Source module's view of shared transmission pool.

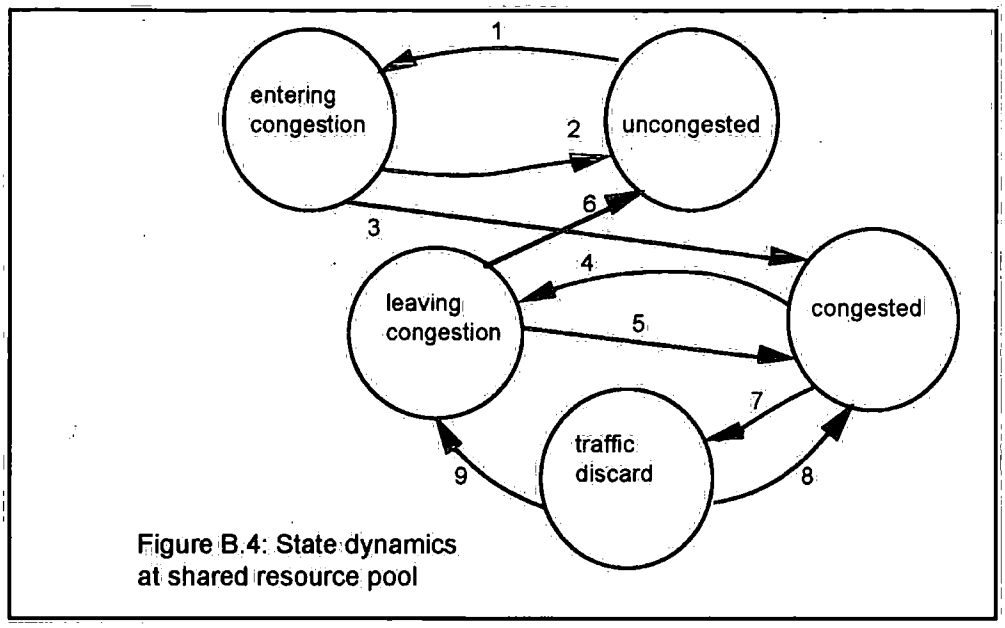


Figure B.4: State dynamics at shared resource pool

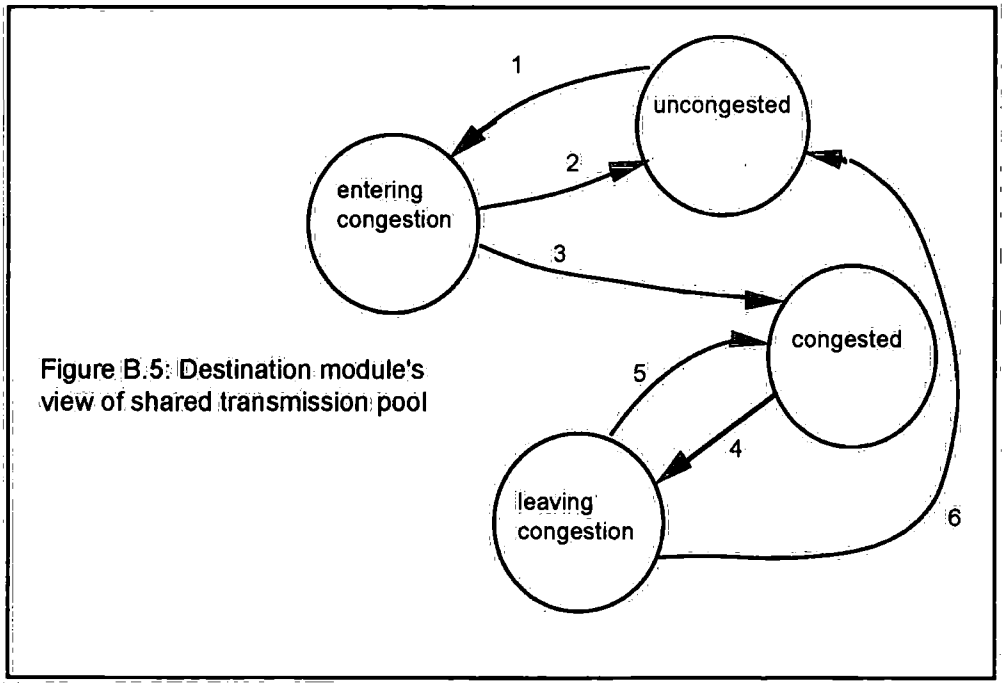


Figure B.5: Destination module's view of shared transmission pool

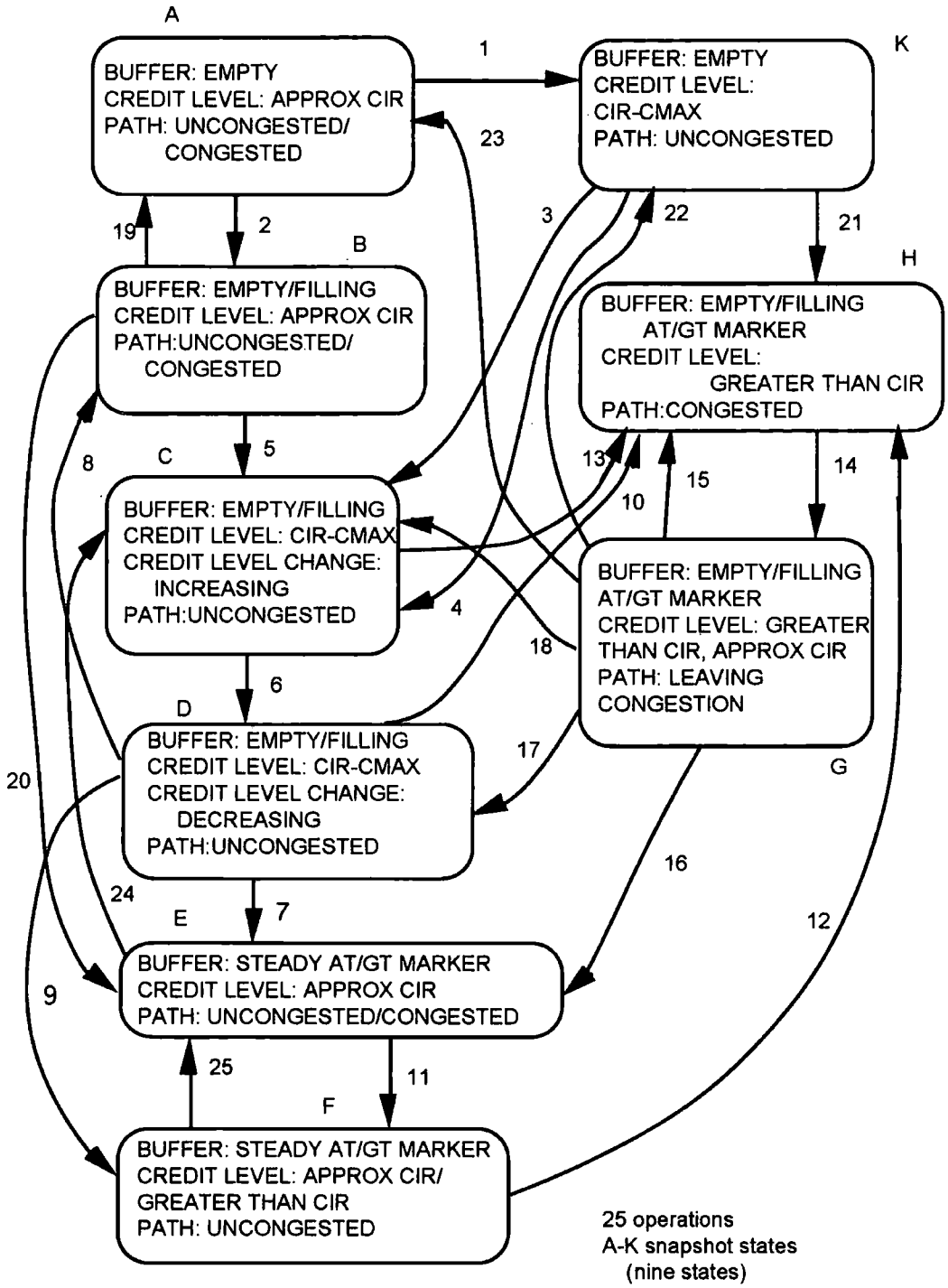


Figure B.6: Refinement and permutative operations on source module's view

Appendix C

A generic characterization of system function realizations

C.1 Introduction

An operational framework describes how algorithms that animate system functions fit together. An algorithm is characterised by both the networking problems it addresses, and generic mathematical semantics which serve as meta-algorithms for the problem class. An operational framework has been developed in the previous chapters; the framework has been used for the specification and implementation of the system functions presented in appendices A and B. Using the two case studies as a starting point, this appendix is a description of the constituents of a meta-algorithm for system functions. The interplay between algebra and analysis is exploited using the very important concept of **direct limits** to describe how system property evolution sequences can be represented as convergence preserving transactions.

This appendix makes a contribution in the construction of system function transactions as limit preserving processes.

C.2.1 Introduction to the specification of characteristic relations

In generating a holistic specification of a collection of system functions, any linguistic framework adopted needs to be sufficiently flexible so that representations of all system evolution states are expressible in the language. As presented in chapter 2 section 2.3.1, and developed in chapter six, system evolution properties can be represented as n-ary functions satisfying a collection of relational constraints. Geometric mappings have been defined to represent admissible characteristic paths of system property evolution. Such mappings have been given algebraic and limit preserving representations. However, the specification of a large system function is often carried out piece by piece, the constituent parts being put together incrementally as the design progresses. This means that representation techniques for system functions should be amenable to compositional specification approaches. Before embarking on the definitions, it is useful to digress in mentioning how the approach being presented here has been exploited successfully in the area of algebraic geometry.

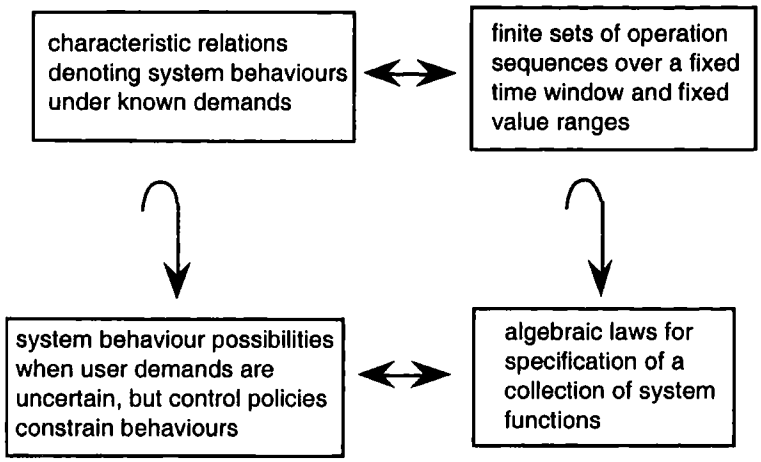


Figure C.1 Generating rules for system behaviours

A very powerful approach often adopted for solving non-linear problems in number theory and geometry is to develop a characteristic solution for a problem class. Such a solution should lie within a bounded set of values; the set of values representing a solution must be reachable after the execution of a finite set of operations. This analogy can be carried over to the realization of system functions satisfying non-deterministic behaviours of users. Figure C.1 illustrates the analogy.

The top two sets on figure C.1 denote simulation of networking scenarios where the user demands are known precisely. Once defined, the finite solution can be generalised by inclusion into a generic solution framework spanning finitely many, possibly infinite models of system behaviour scenarios. The bottom two sets on figure C.1 are supersets of the top two things. However, by careful selection of appropriate algebraic laws, a small specification of the problem class can be generated as denoted by the bottom two sets. Such a generic specification can be instantiated as necessary, in an expansive way when the possibilities of value permutations taken by a system under study is very large. A comprehensive treatment of this concept in connection with finding zeros of polynomials in several indeterminates is presented in Eisenbul D. and Harris J. [1992] under the subject of SCHEMES. Thus the work on algebraic geometry employing the notion of **Universal Constructions** is a good prototype for the correspondence described in this appendix.

C.2.2 Observations and attribute values

Specifications of system functions for modelling and realization of realistic networking systems are often very large (i.e. a large number of operations; state variables and state value possibilities are often generated). The inclusion scheme introduced in the previous

subsection provides a characterization procedure for large quantities of operations and data items since algebraic laws are generally stated as a small number of axioms. The suite of storage functions described in appendix A is an example of a large specification. The appendix is used as a prototype to illustrate how a large specification can be given a concise and perspicuous formalism.

Figure C.2 illustrates a generic interconnection structure of the automata presented in this thesis. The user demand automata are inherently non-deterministic as per Salomaa A. [1973]. Thus the start state is not unique except that it is a member of a set contained in the state signatures. Moreover, state transitions though contained in the transition productions, are not unique when the automaton is in a known state.

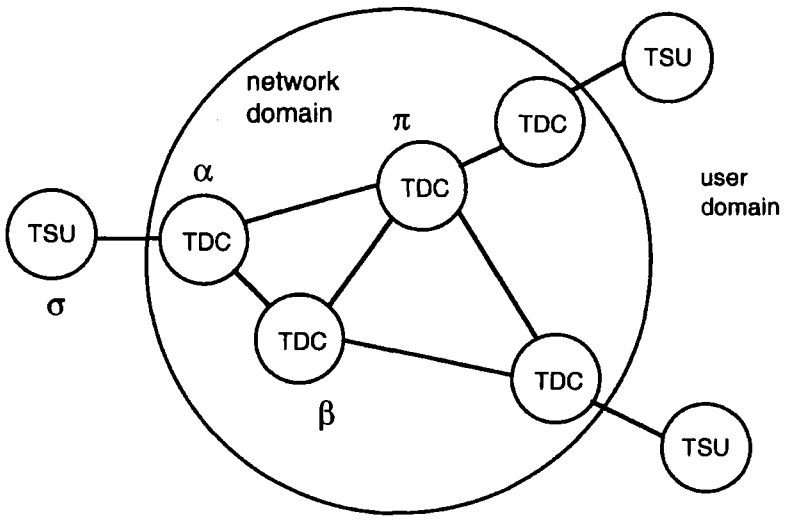


Figure C.2 Inter-connecting user and control automata

TSU (Transaction source unit):

Non-deterministic automaton with recurrent behaviour patterns which can be described in a statistical way, and classified over a large time horizon into a number of pattern types:

- bounded state value ranges,
- non-deterministic actions,
- finite recognizable sequences as sub sequences within a non-deterministic sequence.

TDC (Transactions 'direct limit-preserving' controller):

Deterministic control automaton

- bounded state value ranges,
- deterministic actions,

- finite number of operation sequences,
- desirable small intervals of clock ticks as object interaction schedules,
- potentially infinite values of state denotations.

The deterministic control automata are embedded within the network domain. It should be noted that the number of states spanned by the deterministic automata can be very large - as large as is desired by the system designer's specification of the representation of the parts of system functions encapsulated within the automaton. Refer to section 3 of appendix A ~ Operations of the storage functions. Objects within the storage system are instances of the TDC shown on figure C.2. Views described in the storage system specifications correspond to the interconnections of the deterministic control automata. Operations of the storage functions correspond to mathematical operations (more precisely permutations) on the storage system state space declared to capture all system evolution states and properties. The *activate, passivate, move* and *migrate* functions can be interpreted as *transactions* each of which is a collection of sequences of operations.

Section 5 of appendix A addresses the requirement that transaction threads need to be joined together as nested system functions executed by distributed deterministic control automata. Co-ordination of transaction executions are carried out by distributed agreement automata e.g. the liveness transparency service (automaton). Conditional rules specified as ordering and conflict resolutions are basically policy constraints on resource access.

In section 5 of appendix A is provided a relational partitioning of system states as follows:

future: a data entity has this relational attribute stating that it is installed and recognised by the system of TDCs, but that the data entity is not as yet enabled to hold as part of a transaction. It can in future, under appropriate conditions bootstrapped by the system administrator, get into the state relation *starting*.

starting: a data entity is given this relational attribute to make it ready to take part in a transaction. The TDC hosting data items with this attribute can expect to receive an invocation at an appropriate time, in the commencement of a transaction.

transient: a data entity with this relational attribute is taking part in a transaction which must complete either successfully, or terminate as an aborted transaction.

history: a data item with this relational attribute carries a history of a transaction that had executed either successfully, or had failed to accomplish its goal.

A precise mathematical formulation of this relational structure is as follows.

C.2.3 Generic state structures

This subsection describes how the notion of constrained observation paths and attribute values presented in chapter six can be represented in a generic way by specifying state evolution paths as loci on hyperplanes. Since plausible paths evolve over time, a hyperplane bearing curves denoting such paths can be envisaged as evolving over time, generating system attribute values on intervals of real lines. The latter denote system behaviour properties. A detailed description of this representation is as follows.

C.2.3.1 Illustrative Diagrams

An introduction to vector algebra with values on hyperplanes is presented in Binmore K. G. [1982]. The important connection between algebraic operations and the sets of values operated upon in presenting the operations can be found in Baker C. W.[1991]. Figure C.3(i) illustrates how the notion of evolving hyperplanes over time animate an attribute value sequence $<1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow E_{\infty}>$ while a curve on the hyperplane is traversed as a path. Thus one hyperplane becomes another as follows, in this example:
$<(-1) \rightarrow (2) \rightarrow (1) \rightarrow (-3) \rightarrow (-2)>$. In computations which seek to reach a convergence point, E_{∞} would be such an attribute value.

Referring back to figure C.2, each automaton maintains a collection of <hyperplane curves, attribute values> tuples. The practical meaning of such tuples were illustrated in chapter seven while analysing the case study of appendix B. Using the illustrations of figure C.2, the automaton TDC_{α} interacts with automata TSU_{σ} , TDC_{β} and TDC_{π}. Assume that the automaton can handle concurrently a number of transactions (e.g. activate, passivate, move and migrate). Depending on the requirement for attribute sets of a *particular* transaction, the attribute *transient* would be further decomposed into one or more value ranges.

As an example, when automaton TDC_{β} invokes automaton TDC_{α} to request the execution of an operation, TDC_{β} has its local view of the system state maintained at TDC_{α}, of all the transactions affected by the invocations. The TDC_{β} invocation contains information

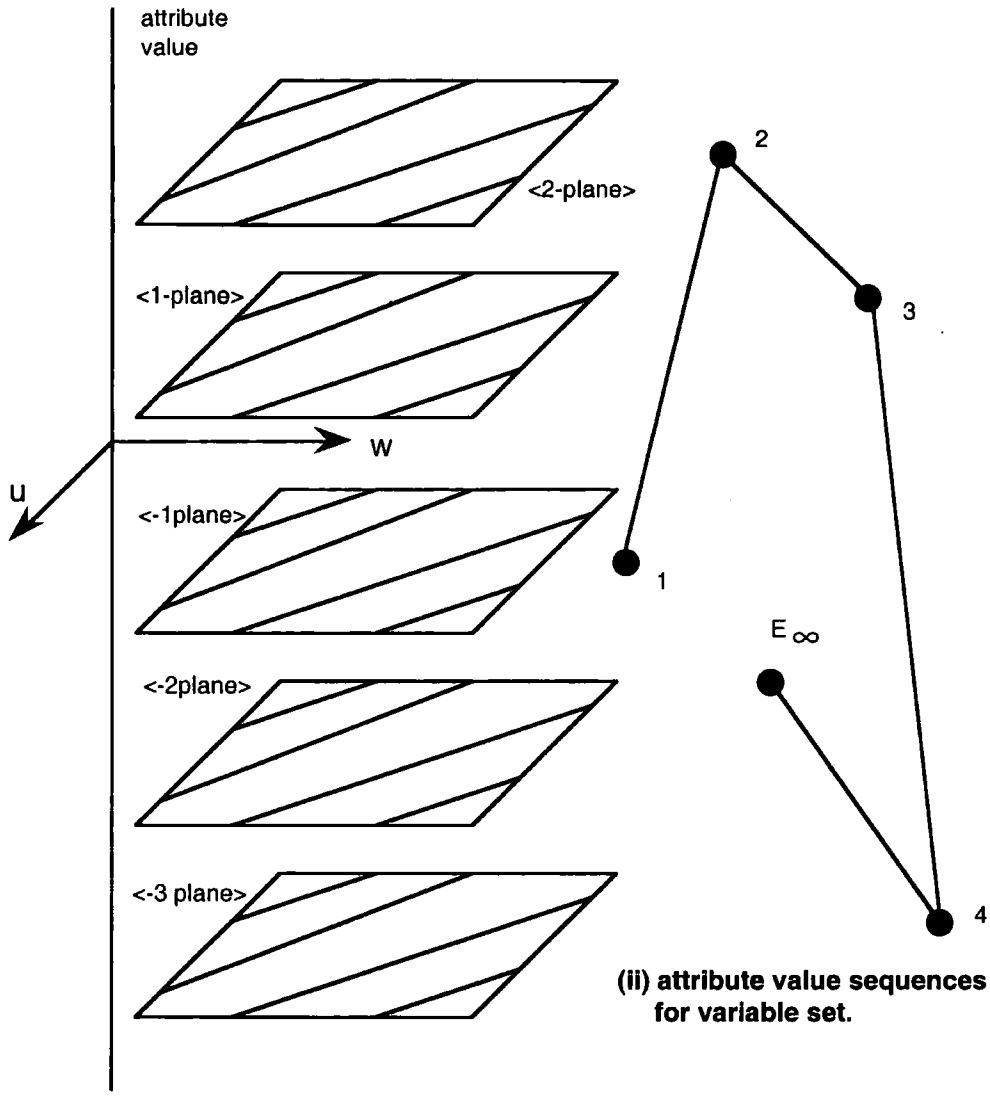
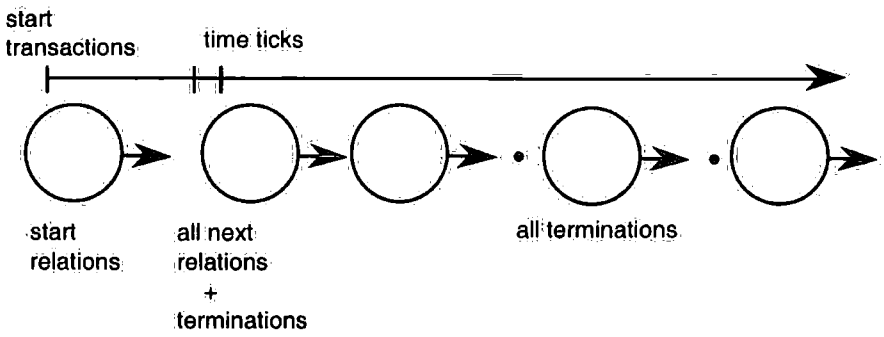


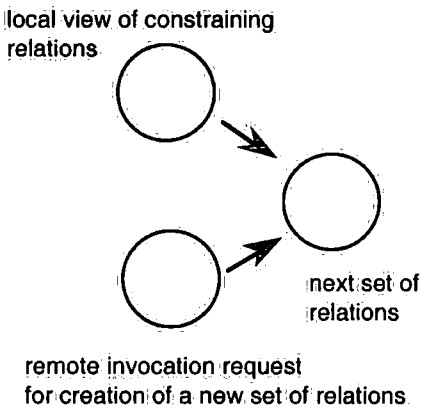
Figure C.3 (I) Hyperplanes within R^3 as value ranges

elements denoting its knowledge of the relevant system state (fragments of hyperplanes and attribute values) at the time of the invocation. Since TDC_α has approximate knowledge of the delay bounds of invocations from all its peers, it can make judgement regarding how inaccurate an invocation is, due to the peer's remoteness. Moreover, an automaton may not have access to the same system state information as another automaton because the two automata are not connected to the same source of information. Thus an automaton may not be party to privileged information within a system of interacting automata.

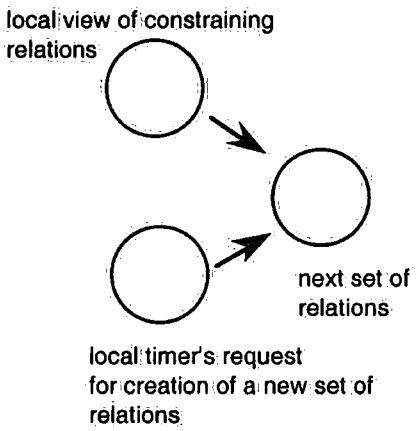
A refinement of the *transient* attribute means that a TDC manages a collection of hyper planes and attribute value sequences that capture the behaviours of all the possible transactions that it is specified to host concurrently. Such specifications are carried out in various stages. It is useful to start from an informal description of the finished product, and



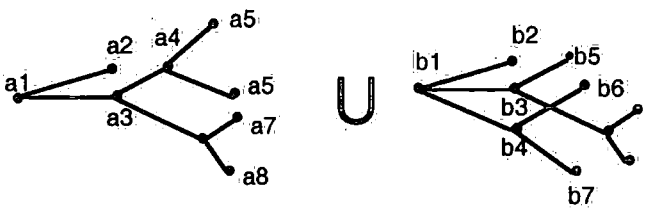
(i) Sequences of relations denoting system property evolution



(ii) remote invocations



(iii) timer induced invocations



(iv) products of transactions a and b

Figure C.4 Structuring transaction sets.

then introduce gradually details of the formal procedure. Figure C.4 illustrates various informal structures of structural relational sequences.

Figure C.4(i) illustrates the evolution of relations within a TDC as time passes. This relational evolution captures all the properties of all the transactions that the automaton can see as it controls its actions relating to the transactions. It should be noted that the

granularity of the clock tick can be as small as desired. Some transactions in this sequence may terminate while others are still operational. The sequence denotes a characteristic concurrent set of transactions which interact among themselves, but eventually all terminate.

Figures C.4(ii) and (iii) illustrate how a TDC constructs the next state from a current state after being invoked by another TDC or TSU, or being invoked by a timer engine. A TDC resolving an invocation has access to all the possible options to select from as the *next state* denoting the state reached by a set of coexisting transactions. In its current state, it has a set of preferences and constraints regarding the most desirable next state. The arrival of an invocation provides the information elements required for narrowing and selecting the most desirable next state.

In many cases, it is easier to generate specifications of individual transactions as if such a transaction does not interact with other coexisting transactions. After specifying the individual transactions, a merging re-writing can be carried out to collapse coexisting transactions into commuting and conflict-resolved threads. Figure C.4(iv) illustrates that trees of relations denoting transactions that need to be joined together to form a single bi-transaction. Such merged specifications were carried out in appendix A.

C.2.3.2 Formalisms

(i) Categories and the structuring of *Sets*

An approach to defining the axioms of set theory is to build up large sets from smaller components called collections. Such a constructive approach provides a generic framework for defining (using simple logical constructs as presented in chapter six) functions and relations among collections. The presentation in Potter M. D. [1990] is an exposition of this category theoretic approach to modern set theory. In particular the distinction between Sets and Classes is thoroughly described. A class provides the limit to a universe within which set constructions are carried out.

For completeness, the axioms of category theory are presented as follows (see Wyler O. [1991]):

A category **C** consists of two classes: the class of *objects of* **C** and the class of *morphisms of* **C**. Four operations and seven formal laws specify **C**.

- two operations assign to every morphism f of **C**, two objects of **C**, called the *domain* dom f of f, and the *co-domain* cod f of f. Often written:

$f: A \rightarrow B$

- the third operation assigns to every object A of **C**, the *identity* morphism id_A in **C**.

- the fourth operation is the *composition,* a partial binary operation on morphisms. It assigns a morphism g of **C** to every pair (f, g) of morphisms of **C** such that dom g = cod f.

  often written $g \circ f$ or gf

- four of the seven laws assign domains and codomains to each other, given morphisms and associated objects:

  $$\text{dom } (id_A) = A = \text{cod } (id_A)$$

  $$\text{dom } (g \circ f) = \text{dom } f$$
  $$\text{cod } (g \circ f) = \text{cod } g$$

  for morphisms f and g such that dom g = cod f.

- two of the remaining three formal laws are:

  $f \circ id_A = f = id_B \circ f$

  for $f: A \rightarrow B$ in **C**.

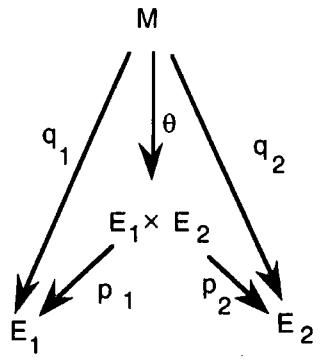
- the third formal law is:

  $h \circ (g \circ f) = (h \circ g) \circ f$

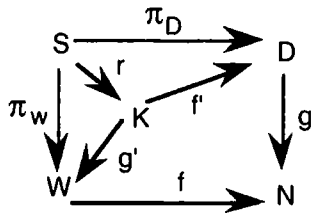
  for morphisms f, g, h such that dom g = cod f and dom h = cod g.

This theory provides a very powerful logical framework generating a structure that holds in the specification of any sequence denoting system property evolution.

Figure C.5 illustrates how the category of sets can be presented to generate the system evolution relations described in figure C.4. All the objects in figure C.5 are sets; all the arrows are functions.



(i) Products of sets E1 and E2 with unique mapping θ.



K is the subspace $W \underset{N}{\times} D$ of W and D.

It is the pullback of D via f

(ii) Fibre products of sets W and D over N:

Figure C.5 The construction of relations within a generic structure of sets

Figure C.5(i) states in a generic way; if E_1 and E_2 are objects in a category, then their *product* is an object $E_1 \times E_2$ together with morphisms $p_i : E_1 \times E_2 \rightarrow E_i$; for i = 1, 2, called *projections*, such that for every object M with morphisms $q_i : M \rightarrow E_i$, there exists a unique morphism $\theta : M \rightarrow E_1 \times E_2$ making the diagram commute. θ is denoted by (q_1, q_2).

A detailed discussion on products for the category of sets is presented in Blyth T.S. [1986].

Figure C.5(ii) illustrates how the generation of fibre products are structured within the category of sets. K is the fibre product of W and D over N:

$$K = \{ (w,d) \in W \times D ; f(w) = g(d) \}$$

This construction generates sets (such as N) with relations as members; these relations are constructed from functions involving products of sets.

(ii) Composition of relations

Within the category of sets, the construction of relational composition is of great importance. In connection with the specification of sequences of relations denoting system property evolution, the composition of relations serves as a fundamental principle. This is because system state observations and attribute values are grouped together into sets; system properties are relations that hold within and across these sets.

Freyd P. J. and Scedrov A. [1990] present a comprehensive treatment of the constructions for the composition of relations. A summary of this construction is provided as follows:

In categorical language, the notion of a *monic* morphism can be translated into set theoretic language, by saying that the map is injective. Thus to construct the relational product

$$\text{Rel}(A,B) \times \text{Rel}(B, T) \rightarrow \text{Rel}(A, T)$$

the following morphisms are monic

$$A \rightarrow B\ ,\ B \rightarrow T.$$

Given two binary relations $\alpha \times \beta = \{\langle x, y\rangle : \exists\, z\ (\langle x, z\rangle \in \alpha \text{ and } \langle z, y\rangle \in \beta)\}$

Thus α, β, x, y and z are appropriately inserted within the product and pullback diagrams which obey the categorial laws.

The composition of relations over sets is another way of looking at the generation of semigroups operations over sets (see Higgins P. M. [1992] page 5). Thus transactions can be generated as transformation semigroups. The category of sets provides all the flexibility required in the construction and presentation of permutations over sets, and semigroup amalgams.

The L-model is the top level specification structure for system functions as proposed in this research. This structure can be realized within the category of sets; it is called an ordered set. Two relations within the structure permute if they obey the rule:

$$\alpha \times \beta = \beta \times \alpha\ .$$

(iii) Direct limits

In composing relations and associated semigroups, it is important to build up larger structures from primitive ones. This constructive approach enables the specification of any collection of system properties and their evolution in a logically complete way. Completeness means that given the theory of relations, for every closed formula Γ, either Γ holds in the theory or (not Γ) holds in the theory. Such logical conclusions are purely local within an object hosting an automaton since relations, interpreted as properties of remote system entities and non-deterministic events, are only predictive. The notion of convergence to a system property set is very useful in the characterisation of system behaviours due to the presence of a lot of statistical data in the observations data used for predictions. It is therefore necessary to embed the notion of discrete ordered relations into the notion of convergence to limit relations.

A complete set of theorems for the composition of relations has been generated by McKenzie R. N. et al. [1987], built on primitive concepts of the notion of a binary relational structure given by <A, R> where A is the carrier set, and R is a single relation. To every binary structure is associated four binary relations ($\leq_l$, $\leq_r$, $\leq$ and $\equiv$) with the following properties

$$x \leq_l y \leftrightarrow (\forall z \in A)\ (zRx \rightarrow zRy)$$

$$x \leq_r y \leftrightarrow (\forall z \in A)\ (xRz \rightarrow yRz)$$

$$x \leq y \leftrightarrow x \leq_l y \text{ and } x \leq_r y$$

$$x \equiv y \leftrightarrow x \leq y \text{ and } y \leq x$$

(where l and r are left and right relational orders respectively).

These relations provide a context independent enumeration of relations. Relational products can then be gathered into semigroups. Semigroups are collected together into an ordered structure which can be characterised as a directed set Λ with a relation $\leq$ as follows: Ignore for the present the need to expand the meaning of relations as $\leq_l$, $\leq_r$,and $\equiv$.

Definitions:

a) if $\lambda \leq \mu$ and $\mu \leq \nu$ then $\lambda \leq \nu$,

b) $\lambda \leq \lambda$ for all λ ,

c) if $\lambda \leq \mu$ and $\mu \leq \lambda$ then $\lambda = \mu$,

d) for all λ and μ there is some ν with $\lambda \leq \nu$ and $\mu \leq \nu$.

Note that the set of all sub- semigroups of a semigroup Ω (i.e. all finitely generated sub-semigroups) is a directed set under inclusion.

For all $\lambda \in \Lambda$, let Ω_λ be a semigroup. For all λ and μ in Λ with $\lambda \leq \mu$,

$\varphi_{\lambda\mu} : \Omega_\lambda \rightarrow \Omega_\mu$ is a homomorphism such that $\varphi_{\lambda\lambda}$ is the identity mapping for all λ such that $\varphi_{\lambda\mu}\varphi_{\mu\nu} = \varphi_{\lambda\nu}$ whenever $\lambda \leq \mu \leq \nu$.

The collection of semigroups Ω_λ and homomorphisms $\varphi_{\lambda\mu}$ is a direct system of semigroups and homomorphisms indexed by Λ .

Characteristic mapping:

Given Θ to be a sub semigroup with homomorphisms

$\pi_\lambda : \Omega_\lambda \rightarrow \Theta$ for all λ .

$(\Theta, \{\pi_\lambda\})$ is called the **direct limit** of the system if $\pi_\lambda = \varphi_{\lambda\mu}\pi_\mu$ for all $\lambda \leq \mu$

and if for any semigroup H and homomorphisms $f_\lambda : \Omega_\lambda \rightarrow H$ such that $f_\lambda = \varphi_{\lambda\mu}f_\mu$ for all $\lambda \leq \mu$, there is a unique homomorphism

$f : \Theta \rightarrow H$ such that $f_\lambda = \pi_\lambda f$ for all λ .

Θ is the limit relational structure.

(See Cohen D.E.[1989] for an example of the use of direct limits in group theory).

In this thesis, the notion of homomorphism of structures hold as follows:

Given two structures M = <A,R> and N = <B,S>, a homomorphism f: M$\rightarrow$ N is a function

f : A $\rightarrow$ B such that if a R b, then f(a) S f(b).

In the practical case of the deterministic control automaton, homomorphisms occur when snapshots change system state. Thus a system state is fully characterized by the structure M at an instant say t_1. During a snapshot at an instant say $t_1+\delta$, the automaton receives an invocation and carries out a sequence of rule-based computations. The result of this computation is the structure N, and a number of atomic invocations sent by this automaton to another automaton, as declared in the rule of the computation being executed. It should be noted that the state transition operation enabled by an automaton does not always result in the invocation of another automaton. This is because an automaton can be invoked to update its internal data representing some state of the system.

(iv) Complexity characterization of system function representations

Judging from the development carried out so far in this appendix, and the large specifications in appendices A and B, the task of specifying a system function is clearly quite involved and challenging. It is therefore useful to have a framework for characterization of system function representations with respect to complexity measures. The first issue to be addressed is how the formalism presented in this appendix lends itself to complexity classification. Appendix C-annex, extracted from Nyong O.D.O.[1995] illustrates how the classical complexity measure of function sequences instantiated over inductively defined sets fits into the formalism described in this appendix.

Semigroups are basically relational graphs. Sub semigroups of the directed semigroup described in the previous subsection create sequences which may satisfy system goals, and are isomorphic to optimal semigroup sequences; optimality is defined in terms of permutations on sequences that solve a resource allocation problem. Selecting an appropriate semigroup sequence is therefore a complexity problem similar to the one described in Cook S. A.[1971]. A detailed investigation of complexity classification of system functions is for further investigation.

C.2.4 Summary

This appendix presents a constructive procedure for building system property relations from atomic relational structures. A procedure of deriving potentially infinite state spaces from finite algebraic laws is described using elementary set theoretic techniques. This is akin to a meta-algorithm. Since the meta algorithm is constructive, it can be applied to the design of large scale system functions where sub components of a specification are joined together to make a logically complete system. Automata with non-deterministic actions can exist within the system of interconnected automata provided that statistics based prediction concepts and rules are incorporated within the deterministic automata.

Appendix C-annex

Classical complexity measures

1 Introduction

Classical complexity measures are defined with respect to a very general model of a computing machine, and also a very general model of a computation. A number of generic formulations have been developed under the subject of computability. The definition presented in this section is an overview adapted from Cutland N.J.[1980]. This summary is carried out in four steps as follows:

a) partial functions and convergence of computations
b) minimalization for computable functions
c) numbering computable functions; universal programs
d) description of computational complexity

1.1 Partial functions and convergence of computations

Define an infinite sequence $a_1, a_2, \ldots,$ from natural number N. P denotes a program.

Assume that a computing machine possess an unlimited set of registers for carrying out a computation which is a function

$$f : N^n \rightarrow N \quad (n \geq 1)$$

In stating that f is computable is equivalent to saying that a model can be exhibited which has in its universe the computed value $f(a_1, \ldots, a_n)$ say b generated by a program whose initial configuration is $a_1, a_2, \ldots, a_n . . 0, 0, \ldots$ Since such a computation may not stop, the definition of computability is extended to cover partial functions (i.e. f from N^n to N whose domain may not be all of N^n). Thus when the computable function f which is a partial function is exhibited by a model with a computed value b and the computation terminates, the computation is said to converge to b, denoted as

$P(a_1, a_2, \ldots, a_n) \downarrow b$

where the $\downarrow$ denotes that the program terminates.

1.2 Minimalization for computable functions

A technique often adopted for generating simulations of dynamical systems is known as unbounded minimalization in recursive function theory. Suppose a function f(x,y) which is not necessarily total simulates a function g(x) obeying the rules:

$g(x)$ = the least y such that $f(x,y) = 0$; if f is computable, so then so is g.

In the general case where $f(x,y)$ is defined as a partial function, the following unconstrained behaviours can exist:

For some x, there may not be any y such that $f(x,y) = 0$. Given a model for computing y, and given that $f(x,y)$ exists, there can be present in the model a sequence (i.e. an ordered set of relations) which never terminates if it is executed to compute y.

It is therefore useful to define an operator which is more constrained than f, as the minimalization operator μy for f, obeying the following rules:

$\mu y \ (f(x,y) = 0)$ is defined as the least y such that:

i) when $f(x,z)$ is defined, all $z \leq y$; and also $f(x,y)=0$ if such y exists.
ii) μy is undefined if there is no such y.

This set of rules is useful for decomposing the model of a computation into smaller models.

1.3 Numbering computable functions; universal programs

Given an arbitrary set M, and the set N of natural numbers, M is said to be denumerable if there is a mapping $f : M \rightarrow N$ such that for each $q \in N$ there is there is exactly one $m \in M$ such that $q = f(m)$, i.e. a bijection.This definition means that it is possible to exhibit a collection of models which satisfy a rule based specification, each model being denumerable as:

P_q for each $q \in N$, $n \geq 1$; and the n-ary function computed by P_q as $\phi_q^{(n)}$.

P_q can thus be regarded as a program identified by q. By referring to the language from which models (and hence programs) are constructed, a universal program which embodies all the computable functions of the models can be defined. It is therefore not decidable whether every relation in the language holds when an identifiable program is simulated by such a universal program. However, given a set of constraining rules, a computable function of a denumerable program could terminate when the arity of the

function is sufficiently large. It can therefore be said that relations defined for such a sequence holds.

1.4 Description of computational complexity

Based on the definitions outlined in the preceding three subsections, the following constituents of a computational complexity measure are defined.

i) given a problem defined as a logical language specification, a number of programs for solving the problem (i.e. models of a logical language) can be generated; each program numbered as P_e.

ii) for a numbered program, $t^n(x)$ is the function which computes the number of steps required by the program to compute $f^n(x)$ if such $f^n(x)$ is defined. $t^n(x)$ is undefined otherwise. It is also assumed that $t^n(x)$ is the minimalization operator for the program P_e computing $P_e(x)$ and terminating in a finite number of steps.

iii) given all P_e defined for a problem, a complexity measure is a collection of $t^n(x)$ such that for each of the programs P_e, the domain of $t^n(x)$ coincides with the domain of $f^n(x)$.

iv) all of the relations defined for sequences executed by the minimalization operators $t^n(x)$ hold (i.e. are decidable) when computing values of x.

It is a challenging task to formulate the behaviour and complexity if a communications system function by extending the definitions i) to iv) above.

Appendix D

Selected publications by the author

[1] Nyong O.D.O. [1993]: 'Incorporating Time Into Specifications of Open Distributed Processing Architectures', in 10th. IEE Teletraffic Symposium, Martlesham Heath, UK., pp.17/1-8.

The use of addressing as a structuring framework for system functions is presented in this paper. Concepts for defining distributed objects are also presented.

[2] Nyong O.D.O., Patel P.P. [1994]: 'Signalling Interworking and Timer Issues in a Heterogeneous Frame Relay and ATM Networks', in 11th. IEE Teletraffic Symposium, Moller Centre, Cambridge, UK., pp.25/1-4.

The definition of the logical unit for system specification comprising declarative operations of resource users, resource access arbiters, and resource allocation optimizers is presented in this paper.

[3] Nyong O.D.O. [1994]: 'Specification and Characterization of Timed Distributed Algorithms as Canonical Mappings', in 11th. IEE Teletraffic Symposium, Moller Centre, Cambridge, UK., pp. 1-7.

The notion of composition of relations for system functions is introduced in this paper. The presentation on constructive composition of relations within limit processes, described in appendix C, is an enhanced development of the ideas introduced in this paper.

[4] Nyong, O.D.O. [1995] : Complexity Classification for Performance Characterisation of System Functions within ATM Sub-networks', 12th. IEE Teletraffic Symposium, Beaumont Centre, Old Windsor, March 1995, pp. 5/1 - 5/12.

The use of semigroups for defining the timed automaton and an initial investigation of complexity measures are presented in this paper.

[5] Nyong O.D.O., Aranzulla P., Cosmas J., and Pitts J. [1998]: 'Resource Based Policies for Design of Interworking Heterogeneous Service Networks', Journal of Interoperable Communication Networks, Vol. 1/2-4, July 1998, pp. 571-580.

Policy based mappings and related simulations using timed automata are presented in this paper.

References

Abiteboul, S., Vardi, M.Y. and Vianna V. [1997]: 'Fixpoint Logics, Relational Machines, and Computaional Complexity', Journal of the ACM, Volume 44, No. 1, pp. 30-56.

Aliprantis C.D. and Burkinshaw O. [1990]: 'Theory of Measure', Chapter 3, in **Principles of Real Analysis**, 2nd. ed., Academic Press, London. ISBN 0-12-050255-0.

Baker C.W. [1991]: 'Product Spaces - Continuity of Algebraic Operations on R', in **Introduction to Topology**, Wm. C. Brown Publishers, Dubuque, IA, USA.
ISBN 0-697-05972-2.

Bertsekas D.P. [1982]: 'Optimal Routing and Flow Control Methods for Communication Networks', in LNCS vol. 44, Springer Verlag, London, pp. 615-643.

Bertsekas D.P. [1983]: 'Distributed Asynchronous Computation of Fixed Points', Mathematical Programming 27, pp. 107-120.

Bhattacharyya G.K. and Johnson R.A. [1977]: 'Nonlinear Relations and Linearization Transformations', in **Statistical Concepts and Methods**, John Wiley, Chichester,
pp. 379-384. ISBN 0-471-07204-4.

Binmore K.G. [1982]: 'Vectors - Hyperplanes', in **Mathematical Analysis**, Cambridge University Press, Cambridge, UK. ISBN 0-521-24680-6.

Blyth T.S. and Robertson E.F. [1986]: 'Permutations', in **Sets and Mappings**, Chapman and Hall, London. ISBN 0-412-27880-4.

Blyth T.S. [1986]: **Categories**, Longmans, London. ISBN 0-582-98804-7.

Brechtken-Manderscheid U.[1991]: 'Variational Problems with Side Conditions', in **Introduction to Calculus of Variations**, Chapman & Hall, London. ISBN 0-412-36700-9.

Burckert H- J., Herold A., Schmidt-Schauss M. [1990]: 'On Equational Theories, Unification, and (Un)Decidability', in **Unification**, ed. Kirchner C., Academic Press, London, pp. 69-115. ISBN 0-12-409590-9.

Chandy K.M. and Misra J. [1988]: 'A Programming Logic', in **Parallel Program Design**, Addison and Wesley. ISBN 0-201-05866-9.

Chang C.C. and H.J. Keisler [1990]: **Model Theory**, North Holland, Oxford. ISBN 0-444-88054-2.

Cohen D.E. [1989]: 'Direct Limits', **Combinatorial Group Theory**, Cambridge University Press, Cambridge, UK, pp. 70-73. ISBN 0-521-34133-7.

Cohn P.M. [1981]: 'The Division Problem for Semigroups and Rings', in **Universal Algebra**, D. Reidel Publishing Company, London, pp. 263-277. ISBN 90-277-1254-9.

Cohn P.M. [1991]: 'Fields with Valuations', Ch.1 of **Algebraic Numbers and Algebraic Functions**, Chapman and Hall, London. ISBN 0-412-36190-6.

Cook S.A. [1971]: 'The Complexity of Theorem Proving Procedures', in Proceedings of the 3rd. Annual ACM Symposium on the Theory of Computing, New York, pp. 151-158.

Cosmas J., Pitts J., Bocci M., Luo Z., Nyong D., Rai S. [1997]: 'Prediction of Resource Utilization within Irregular Topology ATM Networks', 14th. IEE UK Teletraffic Symposium, Manchester University, Manchester, pp.5/1-7.

Craven B.D. [1978]: 'Local Solvability', in **Mathematical Programming and Control Theory**, Chapman and Hall, London. ISBN 0-470-26407-1.

Cutland N.J. [1980] : **Computability**, Cambridge University Press, Cambridge, UK. ISBN 0-521-29465-7.

deBakker J.W. and Rutten J. [1991]: 'Concurrency Semantics based on Metric Domain Equations', in **Topology and Category Theory in Computer Science**, ed. G. M. Reed, A.W. Roscoe and R.F. Watchter, OUP, pp. 113-151. ISBN 0-19-853760-3.

Degano P. and Montanari U. [1984(a)]: 'Liveness Properties as Convergence in Metric Spaces', Proceedings of the 16th. ACM Symposium on Theory of Computing, April 30 - May 2, 1984, pp. 31-38.

Degano P. and Montanari U. [1984(b)]: 'Distributed Systems, Partial Ordering of Events, and Event Structures', in **Control Flow and Data Flow Concpets of Distributed Programming**, ed. M. Broy, Springer Verlag, London, pp. 6-106. ISBN 3-540-17082-0.

Dineen S. [1995]: 'Directed Curves', in **Functions of Two Variables**, Chapman and Hall, London. ISBN 0-412-70760-8.

Doets, K. [1994] : 'Semantics', Section 2.2 - **From Logic to Logic Programming**, MIT Press, London, pp. 15-17. ISBN 0-262-04142-1.

Ehrich H.D., Goguen J.A., Sernadas A. [1990]: 'A Categorical Theory of Objects as Observed Processes', in Foundations of Object Oriented Languages, REX School/Workshop, Noordwijkerhtout, The Netherlands, May 28 - June 1, LNCS 489, Springer Verlag, London, pp. 203-228.

Eisenbud D. and Harris J. [1992]: 'Affine Schemes', **SCHEMES - The Language of Modern Algebraic Geometry**, Wadsworth and Brooks/Cole, California, pp. 4-39.
ISBN 0-534-17606-2.

Fekete A., Lynch N., Mansour Y., and Spinelli J. [1993]: 'The Impossibility of Implementing Reliable Communication in the Face of Crashes', JACM, Vol. 40, No. 5, pp. 1087-1107.

Freyd P.J. and Scedrov A. [1990]: **Categories and Allegories**, North Holland, Amsterdam, Netherlands. ISBN 0-444-70368-3.

Gallager R.G., Humblet P.A., and Spira P.M.[1983]: 'A Distributed Algorithm for Minimum-Weight Spanning Trees', ACM Transactions on Programming Languages and Systems, Vol. 5., No.1, January, pp. 66-77.

Gelenbe, E. and Pujolle, G. [1987]: 'Performance of a Computer Network with Virtual Circuits', Section 5.8 - **Introduction to Queueing Networks**, John Wiley and Sons, Chichester, pp. 140-143. ISBN 0-471-90464-3.

Gelman A., Carlin J.B., Stern H.S., and Rubin D.B.[1995]: 'Frequency Evaluation of Bayesian Inferences', in **Bayesian Data Analysis**, Chapman and Hall, pp. 94-116.
ISBN 0-412-03991-5.

Goguen J. [1990]: 'Sheaf Semantics for Concurrent Interacting Objects', Programming Research Group, Oxford University, England.

Goldstein H. [1995]: **Multilevel Statistical Models**, 'Nonlinear Multilevel Methods', Edward Arnold, London, UK. ISBN 0-340-59529-9.

Grimmett G. and Welsh D. [1986]: **Probability**, An Introduction, Clarendon Press, Oxford, England. ISBN 0-19-853264-4.

Harel D. [1987]: 'STATECHARTS: A Visual Formalism for Complex Systems', Science of Computer Programming, 8, pp. 231-274.

Higgins P.M. [1992]: **Techniques of Semigroup Theory**, Oxford Science Publications, Oxford, UK. ISBN 0-19-853577-5.

Higman G. [1952]: 'Ordering by Divisibility in Abstract Algebras', Proceedings of London Mathematical Society (3) 2, pp. 326-336.

Hogg R.V. and Craig A.T. [1978]: 'Bayesian Estimates', in **Introduction to Mathematical Statistics**, Hogg R.V. and Craig A.T., Collier Macmillan, London. ISBN 0-02-355710-9.

Hurewicz W. and Wallman H. [1969]: Dimension Theory, Princeton University Press, Princeton, NJ.

ISO 8073[1986]: Connection Oriented Transport Protocol Specification.

ITU E.500[1992]: 'Measurement and Recording of Traffic', Telephone Network and ISDN, International Telecommunication Union, Recommendation E.500.

ITU E.711[1992]: 'User Demand Modelling', Telephone Network and ISDN, International Telecommunication Union, Recommendation E.711.

ITU E.716[1995]:'User Demand Modelling in Broadband ISDN', International Telecommunication Union, Recommendation E.716.

Jantzen, M. [1988]: 'Genreal Reduction Systems', section 1.1 - **Confluent String Rewriting**, Springer-Verlag, London, pp 7-26. ISBN 3-540-13715-7.

Kashiwara M., Kawai T., Kimura T. [1986]: **Foundatiuons of Algebraic Analysis**, Princeton University Press, Princeton, New Jersey. ISBN 0-691-08413-0.

Knight F.B. [1991]: **Foundations of the Prediction Process**, Oxford Science Publications, Oxford, UK. ISBN 0-19-853593-7.

Korfhage R.R. [1966]: 'Polish Notation and the Tree of a Formula', in **Logic and Algorithms**, John Wiley, London. ISBN 0-471-50365-7.

Lancaster G. [1992]: 'System Functions', in **Introduction to Fields and Circuits,** OUP, Oxford, UK. pp. 286-310. ISBN 0-19-853931-2.

MacLane S. [1971]: **Categories for the Working Mathematician**, Springer Verlag, London. ISBN 0-387-90035-7.

Mane R. [1987]: 'Measure Theory', in **Ergodic Theory and Differentiable Dynamics**, Springer Verlag, London, pp. 1-14. ISBN 3-540-15278-4.

Mars P., Chen J.R. and Nambiar R. [1996]: **Learning Algorithms**, Theory and Applications in Signal Processing, Control and Communications, CRC Press, London. ISBN 0-8493-7896-6.

MATLAB™[1995]: High Performance Numerical Computation and Visualisation Software, Prentice-Hall International, London.

McKenzie R.N., McNulty G.F. and Taylor W.F.[1987]: **Algebras, Lattices and Varieties**, Vol.1, 'Some Refinement Theorems', Wadsworth&Brooks, Monterey, California. ISBN 0-534-07651-3.

Merrit M., Modungo F., and Tuttle M.R. [1991]: 'Time Constrained Automata', in Concur'91, LNCS 527, Springer Verlag, London, pp. 409-423.

Meseguer J. [1992]: 'Conditional Rewriting Logic as a Unified Model of Concurrency', Theoretical Computer Science 96, pp. 73-155.

Mortimer H.[1988]: 'The Inductive Rule of Acceptance', in **The Logic of Induction**, Ellis Horwood, John Wiley, Chichester, England. ISBN 0-7458-0312-1.

Narendra K.S. and Parthasarathy K. [1990]: 'Identification and Control of Dynamical Systems Using Neural networks', IEEE Transactions on Neural Networks, Vol.1, No.1, pp.4-27.

Narendra K.S. and Thathachar M.A.L. [1989]: **Learning Automata**, An Introduction, Prentice Hall International, London. ISBN 0-13-527011-1.

Nyong O.D.O. [1993]: 'Incorporating Time Into Specifications of Open Distributed Processing Architectures', in 10th. IEE Teletraffic Symposium, Martlesham Heath, UK., pp.17/1-8.

Nyong O.D.O. [1994]: 'Specification and Characterization of Timed Distributed Algorithms as Canonical Mappings', in 11th. IEE Teletraffic Symposium, Moller Centre, Cambridge, UK., pp. 1-7.

Nyong O.D.O., Patel P.P. [1994]: 'Signalling Interworking and Timer Issues in a Heterogeneous Frame Relay and ATM Networks', in 11th. IEE Teletraffic Symposium, Moller Centre, Cambridge, UK., pp.25/1-4.

Nyong O.D.O., Aranzulla P., Cosmas J., and Pitts J. [1998]: 'Resource Based Policies for Design of Interworking Heterogeneous Service Networks', in Interworking '98, International Symposium on Interworking, July 6-10, Ottawa, Canada., listed in Journal of Interoperable Communication Networks, Vol. 1/2-4, July 1998, pp. 571-580.

Nyong, O.D.O. [1995] : Complexity Classification for Performance Characterisation of System Functions within ATM Sub-networks', 12th. IEE Teletraffic Symposium, Beaumont Centre, Old Windsor, March 1995, pp. 5/1 - 5/12.

O'Hagan A. [1994]: 'Sequential Decisions and Experiments', in **Bayesian Inference**, Kendall's Advanced Theory of Statistics, Vol 2B, Edward Arnold, London, pp. 90-93. ISBN 0-340-52922-9.

O'Neill B. [1966]: **Elementary Differential Geometry**, 'Curves', Academic Press, London. ISBN 0-12-526750-9.

Pezze M., Taylor R.N. and Young M.[1995]: 'Graph Models for Reachability Analysis of Concurrent Programs', ACM Trans. Software Engineering and Methodology, Vol.4, No.2, pp.171-213.

Potter M.D. [1990]: **Sets**, An Introduction, Clarendon Press, Oxford, UK. ISBN 0-19-853388-8.

Ramakrishnan K.K., Jain R. [1988]: 'A Binary Feedback Scheme for Congestion Avoidance in Computer Networks with a Connectionless Network Layer', Proc. SIGCOMM'88, Vol.18, No.4, pp.139-156.

Rosenbatt M. [1974]: **Random Processes**, Springer Verlag, London. ISBN 0-387-90085-3.

Saito, H. [1993]: 'CDV Impact on Bandwidth Management', Chapter 5 - **Teletraffic Technologies in ATM networks**, Artech House, London, pp.123-134.
ISBN 0-89006-622-1.

Salomaa A. [1973]: 'Hierarchy of Automata', in **Formal Languages**, Computer Science Classics, Academic Press, London. ISBN 0-12-6157-50-2.

Salomaa, A. [1985(b)]: 'Computational Complexity', Section 6 - **Computation and Automata**, CUP, Cambridge UK, pp. 139-185. ISBN 0-521-30245-5.

Salomaa, A. [1985a]: 'Languages and Rewriting Systems', Section 2.1 - **Computation and Automata**, CUP, Cambridge UK, pp. 5-14. ISBN 0-521-30245-5.

Soparkar N.R., Koth H.F. and Silbercharltz A. [1996]: **Time Constrained Transaction Management**, Kluwer, London. ISBN 0-7923-9752-5.

Tarjan R.E. [1983]: **Data Structures and Network Algorithms**, Society for Industrial and Applied Mathematics, Philadelphia, Pa. ISBN 0-89871-187-8.

Thathachar M.A.L and Phansalkar V.V.[1995]: 'Convergence of Teams and Hierarchies of Learning Automata in Connectionist Systems', IEEE Transactions on Systems, Man and Cybernetics, Vol. 25, No.11, pp.1459-1469.

Therrein C.W.[1989]: **Decision Estimation and Classification**, John Wiley, Chichester. UK.
ISBN 0-471-83102-6.

Tsitsiklis J.N. and Bertsekas D.P.[1986]: 'Distributed Asynchronous Optimal Routing in Data Networks', IEEE Transac. Automatic Control, Vol. AC31, No.4, pp. 325-332.

Varaiya P. and Kumar P.K.[1986]: **Stochastic Systems** - Estimation, Identification and Adaptive Control, Prentice Hall International, London. ISBN 0-13-846684-X

Vossen G. [1991]: 'The Entity-Relation Model', in Data Models, **Database Languages and Data Management Systems**, Addison-Wesley, Wokingham, pp. 35-50.
ISBN 0-201-41604-2.

Wechler W. [1992]: **Universal Algebra for Computer Science**, Springer Verlag, london. ISBN 3-540-54280-9.

Weihl W.E. [1988]: 'Commutativity Based Concurrency Control for Abstract Data Types', IEEE Trans. on Computers, Vol. 37, No.12, pp. 1488-1505.

Weihrauch K. [1987]: **Complexity**, Springer Verlag, London. ISBN 3-540-13721-1.

Welch J.L., Lamport L., Lynch N. [1988]: 'A Lattice-Structured Proof Technique Applied to a Minimum Spanning Tree Algorithm', Proceedings of ACM SIG ACT - SIG OPS, Toronto, Ontario, pp. 28 - 43.

Werlang S.R.daC. [1989]: 'Common Knowledge', in **Game Theory**, ed. Eatwell J. et al., Macmillan Reference Books, London, UK., pp. 74-85. ISBN 0-333-49537-3.

Wyler O. [1991]: **Lecture Notes on Topoi and Quasi Topoi, World Scientific**, London. ISBN 981-02-0153-2.