54

Formal Description Techniques for Object Management

J. Derrick, P. F. Linington and S. J. Thompson
Computing Laboratory, University of Kent, Canterbury, CT2 7NF, UK. (Phone: + 44 227 764000, Email: {jd1,pfl,sjt}@ukc.ac.uk.)

Abstract

Open network management is assisted by representing the system resources to be managed as objects, and providing standard services and protocols for interrogating and manipulating these objects.

Application of formal techniques can make the specifications more precise, reducing the ambiguity inherent in natural language, and can automate some or all of the process of implementation and testing. This paper examines the use of formal description techniques to the specification of managed objects. In particular we examine the relative merits of two formal languages, Object-Z and RAISE, which have been proposed as suitable for use in object management.

Keywords: Managed objects, Formal methods, Open Distributed Processing.

1 Introduction

Large scale open systems require open management to integrate their components, which may have been obtained from a number of sources; the cost of system administration will depend to a large extent on how easy it is to perform this management integration. The creation of open network management depends upon there being a common representation for the resources being managed. This can be achieved by the creation of a suitable family of managed object definitions.

Different implementations of these managed objects, the agents that give access to them and the managers that control them need to interwork. Confidence in these implementations can be increased by testing. However, this testing is expensive and time consuming, because it is labour intensive.

At present the nature of the resources to be managed and the behaviour they are expected to exhibit are expressed in natural language, structured and organized using a simple specification technique set out in the Guidelines for the Definition of Managed Objects (GDMO) [GDMO]. The informal nature of this technique makes the implementation and testing of managed objects expensive, because much skilled effort is needed to interpret the specifications and construct suitable tests.

A. S. Sethi et al. (eds.), *Integrated Network Management IV*

Formal description techniques offer the prospect of improved quality and cost reduction by removing errors and ambiguities from the specification and automating aspects of both implementation and testing. There are potentially large benefits to be gained from this. The number of managed objects already specified is large and can be expected to grow during the next few years until there are several thousand. These will range from objects whose behaviour is standardized internationally, through various levels of industry agreement to a wide range of vendor specific objects. Interworking will depend on specification and testing and product cost will depend on the efficiency of these processes.

However, the techniques and languages for formal description are not widely understood by the majority of implementors, and the choice of a suitable language for the application concerned is an important factor in their introduction and acceptance.

Two languages have recently been proposed for the specification of managed objects; they are Object-Z, based on the well-established Z language, and RAISE. They both have the necessary expressive power for such specifications, although they differ in the approaches taken in a number of areas. This paper examines their key features. It also reviews the tools available to support the languages, particularly with reference to the writing of managed object specifications and to the construction of tests from them.

However, the ultimate test of the acceptability of the techniques is the extent to which potential users are prepared to apply them. It is clear from consultation undertaken with the network management community that familiarity, perceived stability and relation to current practice are amongst the keys to success. Given that both languages have the necessary technical capabilities, selection should be based on the likely ease of uptake.

Action to promote the application of formal techniques in this area is timely; thousands of managed object specifications will be written and processed over the next few years, and the benefits of the formal techniques must be demonstrated before the bulk of the specification work takes place if they are to have the maximum impact.

The paper is structured as follows. The background and the requirements for the specification of managed objects are summarized in section 2. A review of RAISE and Object-Z is presented in sections 3 and 4. Tool provision is discussed in section 5. Language standardization and managed object requirements are discussed in sections 6 and 7. Section 8 discusses the testing process, and we present some conclusions in section 9.

2 Overview of network management

The OSI management framework and its associated standards have developed over a number of years [FormMan,GDMO]. The approach taken is object-oriented, involving the encapsulation of the resources to be managed as *managed objects*. These objects are manipulated in a unified way by using the common management information services and protocols and are defined in an informal way using the guidelines for the definition of managed objects and associated common object and attribute definitions.

The management framework can now conveniently be seen as a special case of the more general problem of distributed processing, and it is reviewed here in terms of the ODP terminology (from ISO/IEC 10746 parts 2 and 3), where appropriate.

2.1 The CMIS model

The Common Management Information Service (CMIS) describes the communication mechanism which links a manager to an agent, which in turn gives access to a set of managed objects. The CMIS model (derived, in part, from the Systems Management Overview) is essentially a computational model which describes the interaction between the manager and agent, and necessary aspects of the interaction between the agent and the managed objects to which it gives access. The agent provides for a degree of coordination of the management information within a system, allowing control of scoping, filtering and discrimination mechanisms. This interface includes operations invoked by the manager which allow it to: create new managed objects (subject to resource constraints); delete existing managed objects; get attribute values; set attribute values; perform actions; cancel an incomplete get operation; and for the agent to report events to the manager.

The emphasis on the CMIS as the basis for management standardization reflects the OSI emphasis on interconnection, rather than system structure. A distributed systems view of management would now de-emphasize this particular interaction in favour of a model giving equal visibility to the communication between agent and managed object (and between managed objects under a single agent).

2.2 Notations for specifying managed objects

The family of management standards includes an informal notation for specifying the behaviour of managed objects - the "Guidelines for the Definition of Managed Objects" (GDMO) [GDMO]. This notation provides a framework for the declaration of properties (based on options from the management object model) and structuring of informal statements of behaviour. The notation allows definition of one or more object classes which express the common properties of their members. The GDMO notation supports:

- identification of the class being defined;
- multiple inheritance from a set of superclasses, or structured specification by association of the definition with some packages of specification (which do not, of themselves, necessarily constitute complete objects);
- attributes or attribute groups, including the role of the attribute in the CMIS structure, the default values to be applied on creation and any restrictions on the way the attribute values may be modified;
- actions, enumerating all those possible and defining their parameterization;
- notifications, enumerating all those possible and defining their parameterization;
- behaviour, which indicates when actions or notifications are appropriate and the way in which their sequence is constrained; behaviour also defines the way that attribute values are updated by the occurrence of actions or notifications, and the way that parameter values carried by actions or notifications are determined by attribute values.

The specification may draw upon common object or attribute definitions, which also include informal descriptions of the corresponding types.

The management object model includes support for two hierarchies: an inheritance hierarchy supporting reuse and refinement of specifications, and a containment hierarchy associated with the interpretation of object creation and deletion actions. It also supports 'fairly arbitrary' assertions of compatibility called allomorphisms.

The ODP viewpoints [ISO10746] can be used to group together different concerns in managed object definitions. In the longer term this approach may simplify the relation of managed object definitions for OSI profiles.

2.3 The requirements for the formal description of managed objects

From the above discussion it can be seen that a formal description technique which is to be used for the specification of managed objects and of their manipulation will need to support:

1. the naming and name binding mechanisms for the managed objects;
2. the inheritance and containment relationships between objects and the ability to create templates representing these relations;
3. the definition of sets of actions and notifications, with their parameterization;
4. the definition of attribute types, including initial or default values and range restriction, matching rules and links to supporting abstract syntax definitions;
5. behaviour, in terms of rules for the occurrence of actions or notifications and the relation of their parameters to object attributes;
6. rules for the creation and deletion of objects;
7. rules for the concurrency constraints implicit in the use of CMIS.

In addition, future development may require statements of the interaction between managed objects which are independently defined. Capturing the full meaning of the linkage between the resources in a complex system will increasingly imply the statement of the effect that the changes applied to one managed object will have on others.

3 The Languages Z and Object-Z

The Z specification language [Spivey] has been developed over the past ten years, and is based upon set theory and first-order predicate calculus. It has proved (together with VDM) to be one of the most enduring formal description techniques, and has had significant industrial usage and support. Recently Z has been selected for the specification of the information model of the ODP Trader [Trader].

The development of Z has been supported through ZIP - A Unification Initiative for Z Standards, Methods and Tools. The project, [ZIP], had four main themes: standardisation of Z; methods support for Z; tools for Z and foundations of Z (for example, logic, proof rules). The project lasted for three years and finished at the end of January 1993.

There are three main reasons for extending Z to facilitate an object-oriented style. **Encapsulation** structures the specification. Data types and the operations upon them are declared together in classes. State is then local to a class as opposed to global state as in Z. **Inheritance** allows the inclusion of previously defined classes in class definitions. A hierarchy of classes and their subclasses can be developed as the Guidelines for the Definition of Managed Objects indicate. **Polymorphism** is the property that an object of a subclass can be substituted where an object of a superclass is expected.

Object-Z [Object-Z] is a specification language based on Z but with extensions to support an object-oriented specification style. Object-Z uses the concept of a class to encapsulate the descriptions of an object's state with its related operations. In addition, Object-Z provides support for inheritance, instantiation and polymorphism. Object-Z does not increase the expressive power of the Z notation, and both offer the same specification paradigm, which captures the relational aspects of state transitions within the system under study; it does, however, contain syntactic and semantic extensions to enable the object-oriented specification style to be supported explicitly.

Whilst Object-Z is not the only proposal to extend the Z language to support an object-oriented style, it is probably the most mature of the approaches; for a survey see [OOZ]. However, Object-Z is not currently in a stable form, and research is still being undertaken into the language and its semantics; this is in contrast to RAISE [RSL] which could be described as a finished product. There are clear disadvantages in using a language which is still in the process of evolving. However, by adopting a flexible approach, there is the possibility that the final version of Object-Z can be tailored to the needs of Managed Object specifications and ODP standards, [Cusack 92]. Indeed, this is the stated intent of some researchers in this area [Cusack 91].

A further factor to consider is the availability of tool support for Object-Z. RAISE has a clear advantage in this respect. Currently there is little or no tool support for Object-Z; tool support for Z exists, but the RAISE tools, coming from a single source, are better integrated.

Technical Assessment

Z specifications consist of schemas (to declare the state) and operations (which change the state). Like Z, Object-Z uses this state-based model to describe systems. This is the only model directly supported, in contrast to RAISE which offers a variety of styles to the specifier. Object-Z specifications use classes to encapsulate together the state and the operations on it. Object-Z provides direct support for expressing constraints and properties of an object's history, which makes temporal behaviour easier to describe and reason about. This can, for example, be used to express deadlock and liveness constraints.

Encapsulation, the definition of classes and objects, is achieved in Object-Z via a class definition mechanism. An Object-Z class is taken to represent a set of models; that is, a class is analogous to an ODP class type in which a class will determine the set of possible realizations that can implement it.

An object is then represented as a named member of a class. In Object-Z classes and objects have to be named, unlike RAISE where both a named and a nameless encapsulation mechanism are supported.

A visibility list in an Object-Z class nominates certain features of a class to be externally visible. In contrast RAISE uses hiding in classes to hide certain features. However, the expressive power is the same, although the specification style will obviously differ.

Object-Z supports incremental and multiple inheritance through class inclusion. The current definition of inheritance in Object-Z is compatible with that used for Managed Objects. A subclass incorporates all the features of its super-classes. As with RAISE renaming is possible for class entities upon inheritance.

There are a variety of proposed inheritance schemes that could be used in Object-Z, although no single one has dominated. Further work is needed in this area to define a notion of inheritance that directly supports ODP and Managed Object specification.

Object-Z supports ad hoc polymorphism, which is the property that an object of a subclass can be substituted when an object of a superclass is expected. Parametric polymorphism, as appears in languages such as Standard ML [SML], is not supported within Z or Object-Z; however, there is no explicit requirement from Managed Object specifications to support this.

The approaches taken by Z and RAISE with regards to subtyping are similar. RAISE supports subtyping, but defines maximal types to enable tool support for (static) type checking. There have been proposals from BT researchers to extend typing to define Object-Z classes as class types (thereby following ODP more closely), this approach would lead to subtyping. However, the issue of whether to extend Z to support subtyping appears not to have been fully resolved.

The development process of producing a new specification from an old one by adding more detail is known as refinement in Z and Object-Z [King]. There has been little work on the role of refinement in Object-Z. However, it is likely that an extension of the Z concept of refinement to Object-Z will be technically possible. In addition, using inheritance as the basis for a refinement relation is also a possibility (the subclass is the super-class with more constraints or with more detail added).

There is no explicit process algebra support for communication or concurrent aspects in either Z or Object-Z. Object-Z deals with the specification of concurrent properties by using linear temporal logic [Duke]. Temporal logic allows the intended execution sequences of objects of a class to be constrained in abstract ways. The resulting style of specification of concurrent and distributed systems is different from that using process algebras or the concurrent aspects of RAISE. Concepts needed for managed object specifications which cannot be provided by the use of linear temporal logic include dynamic aggregation of objects and sharing of one object by other objects. In addition, the duration of methods would need to be modelled via the use of an explicit attribute of a class which gives the current time and time constraints on it. Further work is needed to address these areas.

4 RAISE and the Specification Language, RSL

RAISE (Rigorous Approach to Industrial Software Engineering) is the name of an EC funded ESPRIT project (315) which ran from 1985-1990; research in this direction is currently continuing in the LaCoS (5383) and MORSE ESPRIT projects. The major results of the project include: definition of the (wide-spectrum) specification language,

RSL [RSL]; a formal (denotational) definition of RSL and a set of proof rules for reasoning about RSL specifications and designs; a methodology for program development and design in RAISE; and a set of tools to support formal development within RAISE.

RAISE has been used on a small number of pilot projects and by a number of the partners in LaCoS at a larger scale, and courses to disseminate information about various aspects of RAISE program development are offered by a number of bodies.

RSL is in a reasonably stable form, resulting as it does from a process intended to produce a standard. An early aim of the LaCoS project was to review the language, and apart from a small number of minor changes, it was deemed to need no modification. (Some work in the MORSE project is directed towards adding real-time information to the system, but this is not relevant to the specification of managed objects.)

RSL allows specification in three different paradigms: declarative (a style close to programming in Standard ML [SML], a strict functional programming language); imperative, using expressions which can cause side-effects; and concurrent, using an amalgam of CCS [Milner] and CSP [Hoare].

As might be expected, it has the advantages and disadvantages of a committee design: three programming paradigms are addressed, and both model-oriented and algebraic (property-oriented) specifications can be written.

The design appears successfully to have integrated the three programming paradigms. The language is expression-oriented, with a 'pure' functional core. On top of this are added expressions which can read or write to variables, and take input from and give output to communication channels. Sufficient checks (or imprecations) are made to ensure that side-effects and communications are restricted to appropriate parts of the language - axioms are expected not to have side-effects, for instance.

The development relation for the language contrasts with the notion of refinement familiar from VDM and Z; development is a stricter relation, requiring as it does theory extension, but on the other hand it makes modularisation of development easier to achieve, an aspect which may also carry over to test generation.

Certain aspects of language design are questionable. For example, the logical 'and' and 'or' operations are not symmetric since they are lazy in their evaluation of a second argument, a property which leads to a distressing lack of symmetry in the rules of proof for the language. In addition, the notion of concurrency differs subtly from both of those familiar from CSP and CCS, which makes intuitive understanding of its behaviour more difficult for the non-specialist user. However, without doubt it is suitable for specifying substantial systems.

Since the language is for specification and not for direct execution, it is possible for the type system to incorporate undecidable logical assertions: an object can be in a type if (and only if) it meets a particular logical property. Mechanical type checking for programming languages is essential if certain sorts of trivial error are to be found, and the same would apply to specifications. The language has a system of maximal types to which the richer types can be reduced: adherence to the maximal-type system is machine checkable. (A similar approach is used for Z.)

Technical Assessment

Classes in RSL are intended to denote sets of models, each of which may be described as objects; schemes are named classes. At its simplest, a class introduces

- a collection of type names and named types;
- a collection of variables;
- a collection of names of specified type (a signature, in other words); specifications will include variable (and channel) access descriptions;
- a collection of axioms which describe properties of the named values.

The axioms may completely specify a value, either explicitly in a declarative definition or implicitly through a set of algebraic axioms, or only specify some of its properties.

The definition of a class may be deemed to extend one or more classes, thereby giving a multiple inheritance mechanism. Inheritance is, by default, strict, but a non-strict version can be modelled by means of hiding and renaming. Classes can be defined parametrically over one another, which gives, as a special case, parametric polymorphism (as in SML and other functional languages, and in the templates of ANSI C++).

Types in the language are flexible, and not restricted to statically-checkable types. This allows, for instance, range restrictions to be type specifications.

Object creation and deletion have to be dealt with rather inelegantly using object arrays, which allow the specification of a collection of objects of unbounded size. Creation and deletion are themselves modelled by the setting of the appropriate boolean flag in an object.

RSL supports synchronous concurrency explicitly. Asynchronous communication can be modelled in standard ways.

Behavioural descriptions are possible in a number of styles. Pre- and post-conditions allow conditions to be placed on when actions take place and on their effects. Higher-level algebraic specifications allow the identification of sequences of actions which have congruent effects.

The module system and development relation allow separation of concerns within program development - it is envisaged that this may also facilitate test generation from specifications.

5 Tool Support

The tool support associated with the two languages differs in approach. The RAISE tool set is mature and powerful and could be seen as an industrial specifiers' tool set, but needs a workstation to run it. It provides proof facilities which could be an advantage when investigating automatic test generation. The tool set includes a structure-oriented editor (including a (maximal-)type checker); pretty printers generating LaTeX; translators for the constructive part of the language into Ada and C++; justification (i.e verification) tools.

The structure editors, which allow interactive construction of schemes, objects etc. are impressive. The justification editor supports interactive construction of proofs using

a menu/mouse style interface. However, it is slow, and the tool is clearly not as mature as the structure editors. Support for larger-scale developments is very limited.

In contrast to RAISE, there exist a number of other sources of tools to support the specification process in Z. These include type checkers, syntax checkers and proof support tools, however, none are integrated in the same manner as the RAISE toolset. The ZIP project contains an overview of the available tools [ZIP]. ICL, for example, supply a verification environment for Z in their ProofPower system. The Formaliser specification tool, developed by Logica, is a generic tool (which is not tied to one specific language, although the bias is towards supporting Z specifications) to create and type check specifications via use of a structure editor. Unlike the RAISE toolset, these are not integrated into one system, and thus the tool support will appeal to different constituents in each case.

6 Language Standardization

Z has recently passed a work item ballot in ISO, and so will move towards standardisation through this body. There have been a variety of extensions proposed to the Z language which are claimed to be object-oriented, [OOZ]. Object-Z is one of the most mature of the object-oriented extensions to the Z language in terms of the number of applications written in the language and the international take-up of the language. However, there can be no guarantee that it will remain in the forefront or that it will be an appropriate language for standardization. It is extremely unlikely that standardization of Object-Z will begin within the next three years.

Standardising RSL is a work package in the LaCoS project, with two man years of effort devoted to it. The aim is to achieve ISO standardisation in about five years time. However, progress depends on support from other ISO National Bodies.

7 ODP and Managed Object Requirements

Both Object-Z and RAISE satisfy the general requirements made of formal description techniques supporting ODP specifications. One weakness is that Object-Z is still unstable and does not have a full semantics, although it can be translated into Z and that does have a stable semantics. Consequently there are no introductory texts nor is there a wide range of examples available.

There are more specific modelling concepts needed in ODP and Managed Object definitions. The main ODP modelling concepts include template, type, subtype, class type, polymorphism and inheritance. All of these are supported or can be supported in Object-Z.

There is work to be done on Managed Object concepts of conditional packages, atomic synchronization and allomorphic classes, and it is possible that extensions to Object-Z will be needed or desired.

Z and Object-Z offer less built-in design paradigms than RAISE. However, all the concepts provided by RAISE can also be modelled in Z and Object-Z. For example, RAISE supports concurrency by use of channels and concurrent combinators. Object-Z does not offer these, but communication can be modelled by the unification of inputs and

outputs to classes and operations. Behavioural descriptions are possible through via pre- and post-conditions in a style similar to that available in RAISE.

Managed objects have already been specified in RSL, VDM, Z and Object-Z, and no overriding problems have been found [North], [SimMar], [Rudkin]. In Britain, British Telecom's (BT) Conformance Test Laboratory has done work on developing automatic test generation from process algebras, and has undertaken work on how this can be integrated into an object-oriented Z environment. In addition, there is current research (at the National Physical Laboratory (NPL) in Britain) on the development of test generation using Prolog and LOTOS [Ashford].

8 The Testing Process

Most of the current expertise in formalized conformance testing is based on the testing of communications protocols, which emphasises the procedural aspects of behaviour, in terms of the sequence of observable events.

This aspect of managed object behaviour is important, but it is also necessary to test the consistency of the management information model used to define the managed object state, and to test longer term consistency between periods of communication and between different managed objects. Doing this requires a flexible approach to testing and the combination of a variety of test techniques.

For OSI, the testing methodology and framework is defined in ISO 9646, [ISO9646]. This defines a series of test configurations, procedures and tools for test definition. The tools defined provide structure but are not truly formal, so that the scope for tool support is limited.

The number of test steps involved in a non-trivial management application will be very large. Figures of thousands or tens of thousands of test steps are typical. When operating on this scale, the cost per step of test realization must be kept very low; the process must be made as automatic as possible. It is here that the use of formal specifications for the managed objects can have major pay-backs. The current specifications use a semi-formal framework to organize information, but the heart of the behaviour specification is based on natural language, and so requires human interpretation to create the test steps.

However, the techniques for automated test generation are still being explored. Work transferred from the protocol theatre can be adopted, but is not necessarily the most effective way to create cost effective testing of all aspects of managed objects.

Formal Methods in the Testing Process

The ultimate aim of using formal methods in the testing process is to develop tools which will assist with the generation of sensible tests from formal specifications. There are currently two drawbacks to this approach.

First, fully automatic techniques generate too many tests, and hence test selection and test structure become necessary for the output to be usable. Secondly, automatic techniques do not acknowledge the relative importance of different parts of the specification.

Test generation and selection from formal specifications are active research topics in

the UK; with representative work coming from both the commercial sector (eg BT) and government institutions (eg NPL).

One thread of BT's work has been to extend its LOTOS-based CO-OP work to managed objects [CusWez]. The object-oriented specifications are described by a labelled transition system, which allows general techniques, developed by Brinksma and others, to be applied.

Work at NPL has focussed on a number of areas. In aiming to generate tests for the Transport class 4 protocol [Ashford], a formalisation of test purposes as well as of the specifications themselves has shown promising results. More speculatively, there is discussion of exploiting the different description styles available in RSL to derive tests at different levels of abstraction. Related work, using the proof obligations generated during formal development to guide the search for tests is also under way.

A major manufacturer has introduced a testing methodology internally, with some degree of success. It is based on augmenting an IDL (Interface Definition Language) with pre- and post- conditions, whilst the user specifies separately which 'interesting' sets of parameters should form part of the tests. This gives some weight to the view that formal specifications of managed objects should take the form of augmented GDMO descriptions.

9 Conclusions and recommendations

The formal specification of managed objects is feasible using existing languages; the specifications produced are likely to be more precise than the existing informal or semi-formal techniques, which depend, in the last analysis, on the interpretation of English text.

However, there is considerable resistance to the use of such techniques in industry, and a lack of information about the languages and their application. An education campaign would be needed to ensure that the necessary information is made available to those involved in the specification, implementation and testing of managed objects. Due account needs to be taken of both the *de jure* and the *de facto* standardization mechanisms, and any actions taken need to be coordinated with other initiative in Europe, America and Japan.

The technical assessment of the languages (Object)-Z and RAISE indicates that either of these two languages could be used to produced specifications of managed objects. The styles would be different, reflecting the capabilities of the two languages, but the essence of the existing informal specifications could be captured. The choice of language thus rests primarily on non-technical factors such as user familiarity and degree of standardization, and on the quality of the tools available to support each language.

Z has a wide user base, and has a successful history of use. As an extension of Z, Object-Z would benefit from the position of Z in the market place. RAISE on the other hand is relatively untried; however, it is clearly powerful and offers the specifier several different design paradigms, as opposed to the single one supported by Object-Z. The most significant factor in selecting a language is its acceptability to the intended user community. From this point of view, traditional Z is the clear winner, although no formal technique is really widely established with implementors.

References

[Ashford] Automatic Test Case Generation using Prolog", S.J. Ashford, NPL Report DITC 215/95, 1993.

[Cusack 91] Object Oriented Modelling in Z For Open Distributed Systems", E Cusack, BT, 1991.

[Cusack 92] Using Z in Communications Engineering", E Cusack, BT, 1992.

[CusWez] Deriving tests for objects specified in Z", E. Cusack, C. Wezeman, in Proceedings of Z User Meeting, December 1992, Springer Verlag, 1992.

[Duke] Towards a semantics for Object-Z", David Duke and Roger Duke in VDM'90: VDM and Z, Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1990.

[FormMan] Liaison to CCITT SG VII concerning the use of Formal Techniques for the specification of Managed Objects", ISO/IEC JTC1/SC21/WG4 N1644, December 1992.

[GDMO] Information Technology - Open Systems Interconnection - Structure of Management Information - Part 4: Guidelines for the Definition of Managed Objects" ISO/IEC 10165-4 (X.722).

[Hoare] Communicating Sequential Processes", C A R Hoare in Prentice Hall International Series in Computer Science, 1987.

[ISO9646] Information Technology - Open Systems Interconnection - Conformance Testing Methodology and Framework, Parts 1-5", ISO/IEC 9646.

[ISO10746] Basic Reference Model of Open Distributed Processing - Part 2: Descriptive Model, Part 3: Prescriptive Model", ISO/IEC 10746, July 1994.

[King] Z and the refinement calculus", S King in D Bjorner, C A R Hoare and H Langmaack (eds) VDM'90: VDM and Z, LNCS, Springer-Verlag, Berlin, 1990.

[Milner] Communication and Concurrency", R Milner in Prentice Hall, 1989.

[North] RSL specification of the Log Managed Object", N D North, NPL Report, 1992.

[Object-Z] Object-Z: An object oriented extension to Z", D. Carrington et. al., in S Vuong (ed), Formal Description Techniques 1989, North Holland, 1990.

[OOZ] Object Orientation in Z", S. Stepney et. al. (eds.), Springer Verlag, 1992.

[RSL] The RAISE Specification Language", The Raise Language Group, Prentice-Hall, 1992.

[Rudkin] Modelling information objects in Z", Steve Rudkin in J de Meer (ed) International Workshp on ODP, October 1991, North Holland 1992.

[SimMar] Using VDM to specify OSI managed objects", Linda Simon and Lynn S Marshall in K R Parker and G A Rose (eds), Formal Description Techniques 1991, North Holland 1992.

[SML] The Definition of Standard ML", Robin Milner, et.al., MIT Press, 1991.

[Spivey] The Z Notation, A Reference Manual", J. M. Spivey, Prentice Hall, 2nd Edition, 1992.

[Trader] Working Document on Topic 9.1 - Trader", ISO/IEC JTC1/SC21/WG7 N743, November 1992.

[ZIP] ZIP Project Final Report" in Bulletin of EATCS, 54, October 1994.

Biography

Peter Linington has been Professor of Computer Communication in the University of Kent at Canterbury since 1987. His research interests span networks and distributed systems, currently concentrating on distributed multimedia systems exploiting audio and video information. In ISO, he is currently involved in the standardization of Open Distributed Processing. He chairs the BSI panel on ODP and leads the UK delegation to the international meetings. He also chairs the internal technical review committee for the Esprit ISA project (previously ANSA).

John Derrick has been a Lecturer in Computer Science at the University of Kent since 1990. His research interests include applications of formal techniques to ODP and distributed computing. His current projects include developing techniques for the use of FDTs within ODP and formal definitions of consistency and conformance.

Simon Thompson has lectured in Computer Science at the University of Kent since 1983. His interests include functional programming, constructive type theory and the application of formal and logical methods in computing science.