Richard O. Sinnott and Kenneth J. Turner. Applying formal methods to standard development: The Open Distributed Processing experience. Computer Standards and Interfaces, 17:615--630, October 1995.

Applying Formal Methods to Standard Development: The Open Distributed Processing Experience

Richard O. Sinnott and Kenneth J. Turner,
Department of Computing Science and Mathematics,
University of Stirling,
Stirling FK9 4LA,
Scotland
email: **ros** || **kjt@cs.stir.ac.uk**

7th June 1994

Abstract

Since their introduction, formal methods have been applied in various ways to different standards. This paper gives an account of these applications, focusing on one application in particular: the development of a framework for creating standards for Open Distributed Processing (ODP). Following an introduction to ODP, the paper gives an insight into the current work on formalising the architecture of the Reference Model of ODP (RM-ODP), highlighting the advantages to be gained. The different approaches currently being taken are shown, together with their associated advantages and disadvantages. The paper concludes that there is no one all-purpose approach which can be used in preference to all others, but that a combination of approaches is desirable to best fulfil the potential of formal methods in developing an architectural semantics for ODP.

Keywords: formal methods, LOTOS, Z, standards, architectures, Open Distributed Processing, architectural semantics

1 Introduction

It is common knowledge that natural language is inadequate for giving precise specifications. Indeed this was the initial motivation behind the development of several *Formal Description Techniques* (FDTs) by the *International Organization for Standardization* (ISO) and the *International Consultative Committee for Telephony and Telegraphy* (CCITT now ITU-T). The standardised FDTs are LOTOS [9], ESTELLE [12] and SDL [11]. The aim of ISO was to produce precise and analyzable specifications of *Open System Interconnection* (OSI) standards which could act as definitive references.

FDTs allow for the unambiguous representation of requirements. The main advantages of their application with regard to standards development are in the improvement of quality and the effectiveness with which standards are produced. For example, FDTs can help to ensure that the concepts contained within the standard are well thought out and will not require major revisions at some later date, thereby putting in jeopardy any work that was based on these concepts.

FDTs arrived a little too late to have a major impact on the development of OSI standards. However, Open Distributed Processing (ODP) represents a fresh start in which the benefits of the precision available from FDTs can be achieved.

This paper gives an account of the current work on formalising the architecture[1] of ODP. Section 2 provides an outline of some of the past and present usage of FDTs in standards work. Section 3 gives a brief introduction to ODP and to the reference model of ODP (RM-ODP). Section 4 highlights the advantages to be gained in applying formal methods to ODP to develop an architectural semantics and also identifies the pre-requisites for FDTs to be used to develop an architectural semantics. Section 5 identifies the actual FDTs used at present in ODP along with the subset of the ODP architecture which is currently being formalised. Section 6 focuses on the formalisation of the basic modelling and specification concepts. Section 7 focuses on the formalisation of the viewpoint languages, including the different approaches possible and the advantages and disadvantages of each. Section 8 gives an example of the architectural semantics work in formalising a computational language concept. Finally section 9 draws some conclusions on the application of formal methods in standards work, and the development of an architectural semantics for ODP in particular.

2 Background to Formal Methods in Standards

Before proceeding with an account of the application of formal methods in standards making activities, it is worth considering what exactly is meant by the term "formal method". Natural language may be written in a semi-formal style through stylised English (or German, or...). However, English is not a formal method.

A formal method may be regarded as the use of mathematical techniques to aid the design of software or hardware. In particular, formal methods allow properties of a computer system to be predicted from a mathematical model of the system. A formal method is based on a formal language, *i.e.* a symbolic notation that uses unambiguous rules for developing legal expressions in that language and for interpreting the semantics of these expressions.

The mathematical techniques used need not be similar. Labelled transition systems, set theory, predicate logic, modal logics, algebra .... can all be used as the basis of formal methods.

Formal methods have been used, and are being used in varying degrees to aid the development of standards. A full account of the previous application of formal methods in standards may be found in [18]. An account of the application of formal methods generally may be found in [45, 46, 47].

Besides OSI communication standards as a realm of application, other standards which have, or are using/advocating formal methods, include many for safety [20], *e.g.* aviation [29], safety-critical systems [26, 27], space [28], defence [22, 23], railways [21] and nuclear power station software [24].

Formal methods have also been applied in attempts to understand graphics standards, *e.g.* the Graphical Kernel System (ISO 7942), the Programmers Hierarchical Interactive Graphics Standard (ISO 9592), Computer Graphics Metafile (ISO 8632), Computer Graphics Interface (ISO 9636) and GKS-3D (ISO 8805). Some of the results of these formalisation attempts are listed in [17].

[1] Rather a subset of the architecture is formalised. See section 5 for the identification of the parts of the RM-ODP which are currently in the process of being formalised.

Similarly, formal methods have been applied to Office Document Architecture [49]. This is a multipart international standard (ISO 8613) which standardises the structure and content elements of documents. A formal specification of the ODA (FODA) has been developed as an addendum to the standard.

A few of the current standardisation activities which employ formal methods include the following. Presentation Environments for Multimedia Objects (PREMO) [33] is using Object-Z [55] as the primary FDT. Standardization of Managed Objects [31] is also using formal methods to model a managed object's behaviour [32].

In fact current ISO/IEC JTC1 directives request that subcommittees investigate the use of formal methods in their work. These directives have resulted in further advocations of FDTs by various subcommittees, *e.g.* SC24. This work is documented in part in [53] and [50].

With regard to this paper, the application of formal methods to be considered is to ODP and in particular to Part 4 [4, 5] of the RM-ODP. Before proceeding to describe this work, an overview of ODP generally and an introduction to the RM-ODP will be given.

3 Introduction to ODP and the RM-ODP

One definition of a distributed system [39] states that it: "consists of multiple processors which do not share primary memory and which communicate by sending messages over a communications network". Inherent characteristics of a distributed system include properties such as:

Remoteness: components may be spread over space with both local and remote interactions possible;

Concurrency: components are likely to be executing in parallel;

Partial Failures: components may fail independently of others;

Asynchrony: global communications are not driven by a global clock;

Heterogeneity: different technologies may be used within the system;

Autonomy: different parts of the system may be owned and managed separately;

Evolution: the technologies in the system may change over time;

Mobility: sources of information in the system might be physically mobile.

The RM-ODP recognises that it cannot provide an infrastructure to meet all of the needs of distribution. Different systems will almost certainly have different demands on the infrastructure. The RM-ODP does, however, provide a framework for describing these infrastructure components and their configuration. Given applications may then select the components they need for their particular concerns. Thus in effect the RM-ODP is a framework for developing standards for distribution, where the standards to be developed reflect infrastructure components needed to overcome problems inherent in distribution.

The RM-ODP itself is divided into four main parts:

Part 1: Contains an overview and guide to use of the RM-ODP.

Part 2: Contains the definition of concepts and gives the framework for description of distributed systems[2]. It also introduces the principles of conformance and the way they may be applied to ODP. The modelling approach used in Part 2 — and the rest of the RM-ODP — is object-oriented. The advantages of this with regard to systems development generally are well documented in the literature, *e.g.* [34, 35, 36, 37]. There are three main sets of concepts used in Part 2:

Basic Modelling Concepts: these introduce the term object and other related terms, *e.g.* action, behaviour, interface, location in space/time, interaction point, object state, environment of an object...

Specification Concepts: these place requirements on specification languages, *e.g.* concepts such as composition, type/class, template, behaviour compatibility, creation/deletion, subtype/subclass...

Architectural Concepts: these may be seen as structuring concepts arising from considering issues of distribution and distributed systems, *e.g.* contracts, policies, binding, causal relationships...

Hence Part 2 in effect provides the vocabulary with which distributed systems may be reasoned about and developed, *i.e.* it is used as the basis for understanding the concepts contained within Part 3 of the RM-ODP.

Part 3: Contains the specification of the required characteristics that qualify distributed system as open, *i.e.* constraints to which ODP systems must conform. The main features of Part 3 include the viewpoint languages, conformance issues, functions and transparencies.

ODP uses the notion of a viewpoint as it recognises that it is not possible to capture effectively all aspects of design in a single description. A given viewpoint captures certain design facets of concern to a particular group involved in the design process. In doing so the complexity involved in considering the system is reduced. ODP recognises five viewpoints, each with its own associated language:

Enterprise Viewpoint: this focuses on the expression of purpose, policy and boundary for a given ODP system;

Information Viewpoint: this focuses on the information and information processing functions in a given ODP system;

Computational Viewpoint: this focuses on the expression of functional decomposition of a given ODP system, and of the interworking and portability of ODP functions;

Engineering Viewpoint: this focuses on the expression of the infrastructure required to support distributed processing;

Technology Viewpoint: this focuses on the expression of suitable technologies to support distributed processing.

Each viewpoint represents a different abstraction of same original system; however, there is likely to be common ground between the viewpoints.

[2] As well as broader areas where a clear understanding of object-oriented concepts is required.

Functions and transparencies help to overcome (hide) problems involved in distribution, *e.g.* hiding from users that their local application is interworking with remote applications to provide a given service. Transparencies can be applied selectively. All transparencies are realised in part (or totally) using the engineering viewpoint through the application of stubs, binders and protocol objects.

Part 4: Contains a formalisation of a subset of the ODP concepts. This formalisation is achieved through "interpreting" each concept in terms of the constructs of a given FDT. This interpretation is termed an architectural semantics and is the focus of the rest of this paper.

4 The Development of an Architectural Semantics

4.1 What is an Architectural Semantics?

It is often the case that writing specifications proves to be difficult due to poor initial choice of specification structures. Thus having a good architecture upon which specifications can be based removes many of the difficulties involved in the actual writing of specifications. By a similar argument, specifications written without a well structured architecture tend to be not only difficult to write but also hard to understand and difficult to modify and extend.

Having a good specification architecture is also very useful for problems that are not well defined by requiring detailed consideration of the informal problem statements. Thus attempting to formalise "messy" problems directly can lead to "messy" specifications.

As identified in [38, 40], the combination of formal methods and object-orientation are complementary techniques in promoting understanding of the software development process.

As a result of these considerations, ODP has identified the need for the development of an architectural semantics. An architectural semantics[3] may be regarded as the interpretation of given architectural concepts in a given FDT. For ODP the architecture may be regarded as "parts of" (See section 5) the RM-ODP. The theory is that by interpreting the most basic of concepts then more complex structures may be built. For example, interface, interaction, etc. may be used to build services, protocols, etc. Through this formalisation, concepts are no longer left open to interpretation. Thus, "intuitively clear" concepts which might be open to different interpretations are made more precise and any ambiguities are removed.

4.2 Who will use an Architectural Semantics?

Ideally the architectural semantics work will be used by anybody interested in ODP and the RM-ODP. These may include:

- developers of the RM-ODP themselves;
- developers of standards to be generated from the RM-ODP;
- implementers whose products comply with standards generated from the RM-ODP;
- testers of conformance to standards generated from the RM-ODP;
- end-users of products designed according to standards generated from the RM-ODP.

[3]This term was first used by Prof. Chris Vissers, University of Twente.

The ideal scenario would be if the people using the RM-ODP used the formal definition as given in Part 4, as opposed to the informal text given elsewhere in the RM-ODP. The likelihood of this occurring, however, is small due to the limited formal method literacy even within the computing science community generally.

4.3 Pre-requisites for FDTs

For a formal method to be used in developing an architectural semantics for ODP, certain criteria have to be fulfilled. Firstly that the FDT must be widely known or standardised. Thus the appropriateness of new FDTs are raised here, *e.g.* Object-Z [55] and RAISE [54]. It may be the case that the introduction of new FDTs is prohibited more by political reasons than technical reasons. It may also be the case that it might be too late already for new FDTs to be used in developing an architectural semantics for ODP, as the work is already well advanced in the standards making process. It should be noted that there is no insistence on a given FDT being able to model all (or any?) of the concepts of the RM-ODP. However, a given FDT must also keep up with the scope of the architectural semantics work. For instance when the documents, [2, 3], on which the architectural semantics are based are modified, then the text of the architectural semantics should be modified also. Also if a given FDT is used to describe a certain concept or viewpoint, then other FDTs should attempt to describe this also. Thus there should be a consistency of application between the FDTs. This also helps in identifying the FDT best suited to a particular problem, *i.e.* the best suited FDT to model a given viewpoint or concept. The result of this is that considerable work is required to keep an architectural semantics up to date.

4.4 What are the Advantages of an Architectural Semantics?

One of the main reasons for the development of an architectural semantics may be seen from the problems incurred by OSI. Formal specifications of OSI standards gave scope for different interpretations of architectural concepts. This was not in itself wrong, but simply reflected the generality of the architecture. Interpreting informal concepts in FDTs requires attention to how a given concept should be understood. Some of the problems identified included:

- Service primitives in OSI model interactions at services. Service primitives were not defined in the OSI Reference Model (OSI-RM) [7] but OSI service conventions [8]. It was not stated whether service primitives were atomic, instantaneous or synchronous. Thus specifiers could regard service primitives as procedure calls or asynchronous requests (in SDL [11], and ESTELLE [12]) or synchronous calls (LOTOS [9]). This was not just hair-splitting but led to radically different behaviours being specified, *i.e.* different implementations of the same standard.

- Service data units had different interpretations. It was not clear whether they were atomic or not. Hence different behaviours were possible, *e.g.* protocol data units could be sent off before a given service data unit was completely received by a protocol entity.

- Service access points had different interpretations.

  * Did they reflect a structural concept, *e.g.* an interface between two protocol entities?

* Were they active agents, *e.g.* did they have a dynamic aspect through which connections could be established?
* Could they be represented by processes which could be further decomposed?
* Could connection-less and connection-mode services be supported at the same service access point?
* Were endpoints necessarily associated with endpoints or were they a more general concept?

A fuller account of the historical reasons for the development of an architectural semantics for OSI may be found in [48].

4.5 Direct Advantages of an Architectural Semantics

An architectural semantics provides clear and concise statements in a given FDT – a formalisation of concepts which then acts as a more precise definition of the given ODP concepts. In doing so it requires a more in-depth consideration of the textual definition of each concept than might otherwise have been achieved.

Developing an architectural semantics also assists in the sound development of formal descriptions of standards for ODP systems. That is, it offers a more structured approach to specification of ODP systems (and standards), enabling software reuse to be achieved. An analogy here would be an electronic engineer who works at an architectural level. The engineer does not have to re-specify the most basic of components such as flip-flops and NAND gates, but rather may use these as building blocks to create more complex components. An approach using LOTOS to do exactly this may be found in [41] and [42].

In defining an architectural semantics, the developers of the architecture itself may have confidence in their architecture if it can be interpreted in an FDT. The architectural semantics acts as a bridge between the concepts of a given architecture and the semantic model of a given FDT. It should not be assumed, however, that because a given concept cannot be interpreted in a given FDT then it is necessarily wrong. It might simply mean that this concept is not well matched by the semantic model of the given FDT.

An architectural semantics also offers the basis for comparison of different FDTs when used to provide formal descriptions of the same standard. Hence it also helps in identifying which FDT is most suitable for a given problem domain.

Notions such as conformance, consistency and compliance may also be addressed through the architectural semantics work. Advantage can be taken of existing tool support, *e.g.* [13, 14, 15], to check these aspects for specifications developed from the architectural semantics work.

4.6 Indirect Advantages of an Architectural Semantics

There are several indirect advantages that arise out of the development of an architectural semantics. Perhaps the most important of these is in clearing up the text of the architecture under consideration. In the case of the RM-ODP this means removing any ambiguities or misleading text contained within the relevant standards. It could even be argued that this is one of the main direct (visible) advantages of applying FDTs to develop an architectural semantics for ODP.

By developing an architectural semantics the limitations of the FDTs used are also identified and documented. These can then be used by FDT developers to extend and improve existing FDTs. Taking the work of Part 4 specifically, useful extensions identified for the FDTs LOTOS and Z include object-oriented concepts, temporal logic for Z, and dynamic configuration and time for LOTOS.

4.7 What an Architectural Semantics is Not!

It should be pointed out that an architectural semantics is not about showing that two arbitrary specifications written in different FDTs are the same (though, of course, the equivalences defined for the specification language should help here). It is also not about redefining architectural concepts in a form more suitable for FDTs, or adding/removing architectural concepts that can/cannot be interpreted in given FDTs. An architectural semantics might, however, result in the last of these three by making the architecture developers reconsider the existing concepts.

Having identified the need for an architectural semantics, the rest of this paper will focus in detail on the practical realities of developing one for ODP.

5 Developing an Architectural Semantics for ODP

The first questions that arise when considering an architectural semantics for ODP are: what parts of the reference model should be formalised, and what FDTs should be used? So far the FDTs considered have been LOTOS [9], SDL'92 [11], Z [10], ESTELLE [12] and a direct formalisation in mathematics, all of which have their own particular advantages and disadvantages in formalising the architecture of ODP. All of these FDTs also satisfy the prerequisites identified in section 4.3. Almost as important as the choice of FDTs is the expertise that is immediately available in these FDTs. With regard to the work on the architectural semantics, the international community draws on a rich vein of expertise in all of the chosen FDTs.

Ideally the architectural semantics work should cover all of the RM-ODP. This is not feasible, however, due to time limitations[4] (and possibly technical limitations). The original scope of the architectural semantics work was the basic modelling and specification concepts of Part 2[5]. Whilst this brought a more thorough understanding of the more elementary[6] concepts, it was identified that an architectural semantics could be more useful than in just this role. It was identified that formalising the viewpoint languages would be useful also. In effect this extends the basic idea of interpreting the elementary concepts to create more complex components. That is, by interpreting the viewpoint languages it is not simply the basic building blocks that are being formalised, but the more prescriptive building blocks of the viewpoint languages; the level of prescription required to interpret a given concept is thus increased. In doing this the development of an architectural semantics requires much more work. However, the benefits of the architectural semantics are increased dramatically also; some of these benefits are listed in section 7.

[4] The architectural semantics work for the basic modelling and specification concepts is expected to become an international standard by October 1996. The architectural semantics for the viewpoint languages is expected to become an international standard by June 1997.

[5] Much of this work tended to be of a tutorial nature and can now be found in [51, 52].

[6] This term does not imply the concepts are trivial, but that they are more fundamental.

The immediate question which now arises is: what viewpoint languages should be formalised? Once again, all of them should ideally be formalised although it is unlikely that some, *e.g.* the technology viewpoint, may be realistically formalised. So far, initial work has been carried out on formalising: the computational viewpoint [56] and information viewpoints [57] in LOTOS; the information language in Z [60]; and the computational language in ESTELLE [61], SDL'92 [62] and a direct formalisation in mathematics [64]. Recent work has also been carried out on formalising the enterprise viewpoint language in LOTOS [58] and Z [59]. Work was done previously [63] using LOTOS to develop an architectural semantics for the viewpoint languages — including the engineering viewpoint language. However, this is now out of date with regard to the technical content of the RM-ODP. This due to the relatively fluid condition of Parts 2 and 3 especially of the RM-ODP.

It should be pointed out that in formalising the viewpoint languages, more formalisations from Part 2 of the RM-ODP are required, particularly those concepts dealing with architectural and organisational issues, *e.g.* policy, binding, etc. As a result of this the architectural semantics work has been extended further to cover more of the RM-ODP. Care has been taken to ensure that the scope of the architectural semantics work is not extended so far that it is not practicable to complete the work before the intended deadlines.

6 Formalising the Basic Modelling and Specification Concepts

Formalising the basic modelling and specification concepts of Part 2 gives a precise understanding of the basic concepts used in the RM-ODP. This formalisation is achieved by taking a given definition from Part 2 of the RM-ODP and writing in English how that definition may be represented in a given FDT. The approach to formalising the basic modelling and specification concepts is not prescriptive.

It is often the case that there might be more than one way in which a given concept can be represented. For example, an object is defined in Part 2 of the RM-ODP as: "A model of an entity. An object is characterised by its behaviour and, dually, by its state. An object is distinct from any other object. An object is encapsulated, *i.e.* any change in its state can only occur as a result of an internal action or as a result of an interaction with its environment". The terms behaviour, state, interaction and environment are defined elsewhere in Part 2 of the RM-ODP.

The foregoing is a general definition of an object which allows for several choices to be made when modelled in a given FDT. For example, for LOTOS the interpretation given in the architectural semantics work is: "An instantiation of a LOTOS process definition which can be uniquely referenced".

It may be seen that the interpretation in the Part 4 work is still very general. For instance, it states nothing about the way in which the unique identity of an object can be established. This might be when the process is instantiated through some ACT ONE[7] data type in the value parameter list associated with the process definition which is used in all object interactions. Alternatively it might be through some global data type modelled in ACT ONE used directly in the behaviour expression associated with the object template (process definition). Another possible choice is through the object having some initial behaviour

[7] See [9, 43, 44] for more information on ACT ONE and LOTOS generally.

which establishes its identity. Thus the specifier is left with choices as to the best way to model an object in LOTOS. The choice that is made should, however, be consistent with the architectural semantics work.

In effect the formalisation of the Part 2 concepts offers guidance to the specifier as to how to specify a given concept. In some cases, a concept may be modelled only through being very prescriptive in the style of LOTOS used. For example, inheritance may be modelled in LOTOS provided a restrictive style of specification is used, *i.e.* one in which the inherited process has **exit** functionality. This means that any process having **noexit** functionality may never be inherited from. This is a severe restriction upon the specifier and one which is clearly not scalable. An account of the modifications necessary to the LOTOS language to enable inheritance to be dealt with in all cases is given in [65]. As an example, therefore, the architectural semantics work should provide guidance on how specifiers may specify concepts such as inheritance. It should also provide warnings of the problems that the specification of these concepts may induce.

As the approach to modelling taken in the RM-ODP is an object-oriented one, the formalisation of many of the concepts becomes a task in formalising object-oriented concepts in FDTs that may not be object-oriented. This requires that an object-oriented style of specification is imposed. The question might be asked as to whether it is valid to restrict the users of the architectural semantics work to a certain style of specification. This will have repercussions in that it may not be easy to take an arbitrary specification and identify the architectural concepts contained within it. This can be countered, however, since it is up to the specifiers of ODP systems to use the architectural semantics work. Hence the restriction to a particular style of specification may be seen as a valid restriction.

7 Formalising the Viewpoint Languages

The relationship between Part 2 and Part 3 of the RM-ODP may be seen as specialisation. That is, Part 2 gives a basic interpretation of a given concept and Part 3 gives a more specialised version. For example, Part 2 introduces the concept of an interface and Part 3 specialises this basic concept into stream, operational and signal interfaces. Thus one way of considering this specialisation relationship is that Part 2 provides the vocabulary for consideration of Part 3 concepts. Whilst it is essential to have a precise definition in Part 2, it is likely that ODP systems developers and standards writers will, in practice, use the viewpoint languages of Part 3 to develop their systems. Hence FDTs should be applied — where possible — to the viewpoint languages.

In formalising the viewpoint languages three main approaches have been put forward:

- a direct formal semantics in mathematics [64].
- an approach based on interpretation [56, 57, 58, 59, 60].
- an approach based on providing specification templates [63].

Each of these approaches has both advantages and disadvantages which will now be discussed.

7.1 Direct Formal Semantics in Mathematics

This approach is based on giving a direct mathematical interpretation of ODP concepts. This mathematical interpretation takes the form of transition rules which characterise valid behaviours of configurations of computational objects. Doing this realises several advantages:

- it enables the semantics of ODP concepts to be defined directly with mathematics, as opposed to an FDT meta-language based on mathematics.
- it is claimed[8] to offer a means to compare the consistency of different FDTs when used to compare the same computational behaviour. This may be achieved through a mapping of the semantics of a given FDT onto this direct formal semantics. In doing so, problem areas such as ensuring when different FDTs represent the "same" behaviour are alleviated. This has repercussions on notions such as ODP conformance.
- it also captures the genericity of the computational language.

The approach is not without its drawbacks however. The greatest drawback is that it is very (overly?) mathematical and hence might tend to scare away possible users of the architectural semantics work. Formal methods often offer a symbolic meta-language which to a great extent hide their mathematical foundations. With this approach, however, the mathematical foundations are blatantly visible and hence not as accessible to people with a non-formal background.

This approach does not offer any means to develop specifications. It simply represents the generic mathematical interpretation of a subset of the computational language behaviour. Specifically, it deals with the operational interactions of the computational language. It is unlikely that such an approach could be extended to other viewpoint languages, *e.g.* the information viewpoint language, due to the lack of prescription in defining concepts in these languages.

7.2 An Approach Based on Interpretation in an FDT

This approach is a continuation of the one taken in formalising the concepts of Part 2 of the RM-ODP. That is the formalisation is based on interpreting given concepts in FDTs.

The advantages in taking such an approach are that it enables an in depth comparison of all ODP viewpoint language concepts in all FDTs. Thus the semantics of all of the concepts may be checked against the semantic models of the FDTs. In doing so, it brings more understanding of the ODP concepts to users of the RM-ODP. The approach also gives specifiers guidance without being prescriptive as to how they should specify certain ODP concepts.

This approach is not without its drawbacks, however. For instance, as it not prescriptive it is not possible to identify immediately whether any given specification is ODP compliant. With this approach, notions such as cross viewpoint consistency may also not be established directly.

7.3 An Approach Based on Providing Specification Templates

This approach is based on providing specification templates for ODP concepts. At present this work has been done only in LOTOS. Through this approach, a structuring of concepts can be achieved which can then be used to build ODP compliant specifications. It should be pointed out that the specification templates given in [63] only represent a structuring of the

[8] So far, little work has been done in showing that this pivotal mapping exists, even for simplistic specifications.

computational viewpoint language concepts, as opposed to a direct behavioural specification of the concepts. This is because the computational viewpoint language is too generic to be directly formalised in a constructive FDT such as LOTOS.

The approach is very prescriptive, however. It is also not possible to provide templates for all concepts in a given FDT for a given viewpoint language. It is unlikely that a similar style of specification could be adopted in different kinds of FDT, *e.g.* Z, due to their fundamentally different natures. It is also unlikely that this approach can be taken for all viewpoint languages, *e.g.* information, enterprise or technology, as it is only really in the computational and engineering viewpoint languages that the RM-ODP is prescriptive enough to be able to support specification templates. One other drawback with this approach is that it is not possible to take any arbitrary specification and check whether it is ODP compliant or not.

7.4 Conclusion on Approaches

The best possible approach that could be taken in developing an architectural semantics for ODP would consist of:

- providing (behavioural) templates for all of the concepts contained within all of the viewpoint languages in all relevant FDTs;
- being able to show (or prove) consistency between different (all?) viewpoints;
- being able to check arbitrary specifications for ODP compliance.

In reality, however, this is not the case. There is no one all-purpose FDT. Different FDTs are suited for different viewpoint languages. Templates are only possible to a limited extent, *i.e.* not all concepts can be interpreted (modelled) through a template. It is also not possible to provide templates for the information, enterprise or technology viewpoint languages due to their very nature, *i.e.* they place very few prescriptive constraints on the modellers.

As a result of this, the best solution with regard to developing an architectural semantics is through a combination of all of the above approaches. That is, templates should be provided where possible. These should be accompanied by an approach based on interpretation. The limitations of the template based approach should be identified and documented. Finally, (if possible) a complete direct formal semantics should be made of the viewpoint languages where possible and transformation (mapping) rules supplied to enable consistency of FDTs to be determined.

Much of this work remains to be done. So far the approaches have been made predominantly in isolation; however, it is clear that a composite approach is beneficial. The following section illustrates briefly in some detail one of these approaches: the interpretation based approach. This in turn leads in a natural way to an approach based on specification templates.

8 Example of the Architectural Semantics

The RM-ODP uses a hierarchical approach in defining its concepts. Indeed this is one of the main reasons that an architectural semantics is so useful: through defining formally the more basic concepts, the more complex concepts can be developed. One of the consequences of this, however, is that attempting to show even a simple example of the architectural semantics work is quite a laborious task. Hence the example chosen to illustrate the architectural semantics work is amongst the simplest possible: that of an *invocation*[9].

An invocation is defined in Part 3 of the RM-ODP as: "*a sequence of actions comprising two signals:*

- *the first, called invocation submit, between a client object and a binding object, followed by*
- *the second, called invocation deliver, between the same binding object and a server object.*"

The term *signal* used here is defined in Part 3 as: "*an atomic interaction consisting of a single atomic action between a basic computational object and a binding object.*"

A *signal* itself has a signature defined as: "*an action template for a signal comprising*

- *a name for the signal;*
- *the number, names and types of parameters for the signal;*
- *an indication of causality.*"

The term template (action template) used here is defined in Part 2 of the RM-ODP as: "*the specification of the common features of a collection of* $< X >$*'s in sufficient detail that an* $< X >$ *can be instantiated using it.*" Here an $< X >$ may be an object, interface or action. A note is added indicating that action instantiation is deprecated and should be replaced by action occurrence.

Action (and interaction) themselves are defined in Part 2 of the RM-ODP as: "*something which happens. Every action of interest for modelling purposes is associated with at least one object. The set of actions associated with an object is partitioned into internal actions and interactions. An internal action always takes place without the participation of the environment of the object. An interaction takes place with the participation of the environment of the object.*"

The environment of an object is defined in Part 2 of the RM-ODP as: "*the part of a model that is not part of that object.*"

As may be seen, even in this very basic concept of an invocation there is a large collection of sub-concepts required in its definition. Specifically, the concepts required in formalising the notion of an invocation are: *action, template (action), object, environment of an object, signal* and *signal signature.*

Further concepts which are used in the definition of invocation are *type, atomicity, causality, name, model, instantiation, signature, basic computational object, binding object, client object* and *server object.* For the sake of simplicity in this example, *i.e.* limiting the explosion of concepts required to define invocation, these terms will not be formally defined here; their formal definition may be found in [4, 5].

8.1 Formalising "Invocation" in LOTOS

To formalise the concept of invocation as given here requires that all of the subconcepts are also defined. Each of the above identified concepts may be interpreted in LOTOS, either

[9] The following italicised text represents text taken directly from Parts 2 and 3 of the RM-ODP.

directly in the semantics of LOTOS, or through imposing a specification style as will be shown. Architecturally, the most basic concept represented here is that of an action. The following definitions are taken from Part 4 of the RM-ODP.

8.1.1 Action

An internal or observable event. All events in LOTOS are atomic. An internal action may be given explicitly by the internal event symbol, **i**, or by an event occurrence whose associated gate is hidden from the environment.

An interaction is represented in LOTOS by a synchronisation between two or more behaviour expressions associated with objects at a common interaction point (gate). Interactions may be of the kind:

- pure synchronisation on a common gate with no offer: no passing of values between objects occurs;
- **!** and **!** for pure synchronisation: no values are exchanged between the objects;
- **!** and **?** for value passing provided the **?** event contains the **!** event: another way of considering this is that the **!** event selects a value from a choice of values for the **?** event;
- **?** and **?** for value establishment: here the effect is an agreement on a value from the intersection of the set of values. If the intersection of the values is the empty set then no synchronisation and hence no interaction occurs.

8.1.2 Object

The definition of an object is given in section 6.

8.1.3 Environment of an Object

The environment of an object within a LOTOS specification at a given time is given by the environment of the specification and the other behaviour expressions that are composed with that object in the specification at that time.

8.1.4 Action Template

An action denotation which may be either an internal-event-symbol, a gate-identifier or a gate-identifier followed by a finite sequence of value and/or variable declarations. It should also be pointed out that the definition of an action template is not really supported in LOTOS. That is, in LOTOS possible behaviours are specified by giving action denotations combined in some form. To relate a template to an action denotation is the closest that can be achieved in LOTOS. However, the text of Part 2 requires an action template to group the characteristics of actions. This is not part of LOTOS as event offers (action denotations) exist in isolation and it is not possible to collect them and apply a template to characterise them.

8.1.5 Signal Signature

An event offer which consists of a name for the signal; the number, names and types of parameters for the signal; and an indication of causality.

Thus one example of a signal in LOTOS which might be offered by a basic computational object to a binding object is: *g !sig_id !parameter_list !initiate*. The sort *sig_id* gives the signal name; *parameter_list* gives the details of the number, name and type of the parameters of the signal which includes the identifiers for the basic computational object and binding objects involved in the interaction amongst other things; and *initiate* gives an idea of the causality of the signal. Here LOTOS is being used in a stylised way to represent signals. For example, the sort *initiate* which is used to represent causality can only do so informally. There is no notion of this event offer causing the signal event to occur, *i.e.* there are just two event offers from the basic computational object and the binding object. The two event offers enter into the signal event simultaneously, or not at all. Thus in reality there is no causality associated with either event offer.

It should also be pointed out here that a signal must be represented by two event offers in LOTOS, *i.e.* an event offer associated with the basic computational object and an event offer associated with the binding object. Thus a single event offer which may occur without participation from the environment is not a signal. Two event offers which might occur through synchronisation will only represent a signal if the event offers are associated with a basic computational object and a binding object. It should be noted that this should never happen if the interaction rules for computational structuring are followed. That is, computational objects cannot simply interact with one another directly, as this will cause an infrastructure failure. They may interact only after being bound.

8.1.6 Signal

There is no inherent feature of LOTOS which can be used to distinguish between a signal, a stream and an operation; they all use LOTOS events. Consider an arbitrary LOTOS event without some idea of the context in which it occurs, *i.e.* which objects were involved in the synchronisation and what other events have occurred through synchronisations between the same objects. Without having some sort of restrictions on the modelling of event offers, it is not possible to state that it was a signal or one part of an operation or stream. It may be the case, however, that a style of LOTOS can be used to distinguish between signals, streams and operations. Thus all signals might have similar formats for their event offers. An example of one possible format is given in section 8.1.5.

8.1.7 Invocation

An invocation is modelled in LOTOS as a sequence of interactions between a client object and a binding object and the same binding object and a server object. These interactions consist of two events with signal signatures. In order to distinguish between an invocation and any other LOTOS event, it may be useful in LOTOS to use a special label to note it as such. This may take the form of a gate name or a sort used in the action denotation of the signal signature. Thus the events *g !sig_id !parameter_list !invocation_submit* and *g !sig_id !parameter_list !invocation_deliver* could represent a given invocation between a client and a binder object and a binder and a server object respectively. Here the sorts *invocation_submit* and *invocation_deliver* represent both the causality of the signal and the label used to identify it as an invocation (within the limitations of LOTOS as put forward in section 8.1.5.)

8.2 Discussion

Subsection 8.1 illustrates all aspects of the architectural semantics work. For example, a comparison of the semantic basis of the ODP concepts to the LOTOS model is discussed, and an outline of the more prescriptive modelling choices required to reflect a given concept is given. There is a close semantic relationship between the informal text and the formalised LOTOS text. That is, as LOTOS is based on a labelled transition system the notion of an action is provided for directly.

When the more prescriptive concepts are modelled, however, such as signals, it is necessary for the architectural semantics to provide guidance as to how the concept may best be specified. This may be by providing specification fragments — in this case an event offer consisting of several parameters identified as being necessary from the informal text of the RM-ODP. It is also possible for this guidance to be more prescriptive, *i.e.* explicitly give specification templates to prescribe the form of a signal, say, as opposed to simply offering suggestions to the specifier. This is a natural follow on from the interpretation based approach, however, and hence the two approaches are, in many respects quite similar. It should also be pointed out that not being prescriptive gives specifiers more room for their own style of specification. For example, instead of having *parameter_list* as given in section 8.1.5 represent the details of the number, name and type of the parameters of the signal, these might be given individually in a single event offer. This might be done with: *e.g. g !sig_id !1 !myNumber !myNat !initiate* where *myNumber* represents the name of the parameter and *myNat* its sort.

Another advantage in not being prescriptive too early on in the development of an architectural semantics is that the limitations of the different approaches may be highlighted and comments given which may help the specifier to model a particular concept.

The issue of a template based approach becomes more convincing when higher levels of prescriptivity are given as is the case in this example, *e.g.* identifying all of the parameters for a given event offer (signal). This is not always the case though and so the application of specification templates in ODP is limited. Similarly, even when all of the required concepts are identified it may not always be possible to model them, *e.g.* an environmental contract may not be modelled fully in LOTOS.

9 Conclusions

Formal methods have a very important role to play in ODP with the development of an architectural semantics for the RM-ODP. Apart from the advantage of clearing up any ambiguous (or incorrect) text, formal methods offer guidance to specifiers or systems developers using the RM-ODP.

Formal methods themselves benefit from their application to ODP. The usage and publicity they gain from the ODP work as well as the identification of possible limitations and hence suggestions for improvements of the methods themselves are all additional advantages following from the ODP work.

It could be argued that the first real test of the RM-ODP is through the development of an architectural semantics. That is, the development of an architectural semantics represents a thorough "work-out" of the architecture of the RM-ODP. Through this approach, notions such as consistency between viewpoints can be assessed and conformance of distributed systems developed using the RM-ODP checked.

Work on the formalisation of the viewpoint languages has identified that formalisation of more of Part 2 of the RM-ODP is necessary, *i.e.* concepts other than the basic modelling and specification concepts. As a result of this the workload on formalising the RM-ODP has increased further. Whilst the architectural semantics work is currently having a boom time with many contributions from many of the member bodies involved in the standards making process, more work is required in the international community to develop a full architectural semantics for the RM-ODP.

There is no all-round FDT or approach best suited for the work on formalising the architecture of ODP. It seems that the best approach is through a composite approach which uses all of the techniques suggested so far. Hence it is likely that future contributions on formalising the viewpoint languages should proceed along these lines.

The success or failure of the architectural semantics work also depends to a large extent on its tutorial nature. At present the Part 4 work is very terse and difficult to read. Hence the need exists for many small examples illustrating the concepts being modelled. Similarly, the need exists for a large example illustrating the application of the architectural semantics work. It is likely that this work can only be attempted when the formalisation of the viewpoint languages is more mature.

Time and politics may play a large role in the success or failure of this application of FDTs. If this is not going to be a case of FDTs yet again not being fully exploited, then further collaborative international work is required immediately to provide a sound and apposite architectural semantics for ODP.

To help speed up the progress of developing an architectural semantics for ODP an email discussion group has been set up. Interested readers should contact the first named author.

10 Acknowledgements

This work was supported by a joint grant from the United Kingdom Engineering and Physical Sciences Research Council (EPSRC) and the Department of Trade and Industry (DTI) as part of the Formalisation of the ODP Systems Architecture (FORMOSA) project in which British Telecom (BT) is prime contractor. Special thanks are given to individual members of the FORMOSA project team from BT including Steve Rudkin and Pete Young. Thanks are also given to those co-workers in ISO involved in the architectural semantics work.

References

[1] Basic Reference Model of ODP — Part 1: *Overview and Guide to Use of the Reference Model*, Draft International Standard 10746-1, Draft ITU-T Recommendation X.901, 1994.

[2] Basic Reference Model of ODP — Part 2: *Descriptive Model*, Draft International Standard 10746-2, Draft ITU-T Recommendation X.902, 1994.

[3] Basic Reference Model of ODP — Part 3: *Prescriptive Model*, Draft International Standard 10746-3, Draft ITU-T Recommendation X.903, 1994.

[4] Basic Reference Model of ODP — Part 4: *Architectural Semantics*, ISO/IEC JTC1/SC21 N9035, 1994.

[5] Basic Reference Model of ODP — Part 4: *Architectural Semantics Amendment*, ISO/IEC JTC1/SC21 N9036, 1994.

[6] ISO/IEC. *Information Processing Systems — Data Communications — Network Service Definition.* ISO-IEC, April 1987, IS8348.

[7] ISO/IEC. *Information Processing Systems — Open Systems Interconnection — Basic Reference Model.* ISO/IEC 7498. International Organization for Standardization, Geneva, Switzerland, 1984.

[8] ISO/IEC. *Information Processing Systems — Open Systems Interconnection — Conventions for the Definitions of OSI Services.* ISO/IEC TR 10731. International Organization for Standardization, Geneva, Switzerland, 1992.

[9] ISO/IEC. *Information Processing Systems — Open Systems Interconnection — LOTOS — A Formal Description Technique based on the Temporal Ordering of Observational Behaviour.* ISO/IEC 8807, International Organization for Standardization, Geneva, 1989.

[10] J.M. Spivey, *The Z Notation: A Reference Manual*, 2nd Edition, International Series in Computer Science, Prentice-Hall International, 1992.

[11] ITU-T. *Specification and Description Language*, CCITT Z.100, International Consultative Committee on Telegraphy and Telephony, Geneva, 1992.

[12] ISO/IEC. *Information Processing Systems — Open Systems Interconnection — ESTELLE — A Formal Description Technique based on an Extended State Transition Model.* ISO/IEC 9074, International Organization for Standardization, Geneva, 1989.

[13] Editors M. Caneve, E. Salvatori. *LOTOSPHERE LITE User Manual*, ESPRIT Ref: 2304, Lo/WP2/N0034/V04, LOTOSPHERE Consortium, 1991.

[14] UK-Verilog Ltd. *The GEODE Reference Manual*, Hampden House, Hampden Road, Chalfont St. Peter, Bucks, SL9 9DP.

[15] J. M. Spivey. *The FUZZ Manual*, 1991.

[16] B. Meek, *Language Standards Committees and Revisions*, SIGPlan Notices, December 1988, pp 134-142.

[17] D. A. Duce, P. J. W. ten Hagen, R. van Liese, *Components, Frameworks and GKS Input*, Proceedings of the Eurographics 89 Conference, North-Holland, Amsterdam, 1989.

[18] C. Ruggles, *Formal Methods in Standards*, Springer-Verlag, BCS, 1990.

[19] ISO/IEC/JTC1 *Statement of Policy on Formal Description Techniques.* ISO/IEC JTC1 N145 and JTC1 N1333, International Organization for Standardization, Geneva, Switzerland, 1988.

[20] Jonathan Bowen, Victoria Stavridou, *Safety-Critical Systems, Formal Methods and Standards*, PRG-TR-5-92, Oxford University Computing Laboratory, 11 Keble Road, Oxford, OX1 3QD.

[21] BRB/LU Ltd/RIA. *Safety Related Software for Railway Signalling*, Technical Specification no. 23, Consultative Document, Railway Industry Association, 6 Buckingham Gate Road, London, SW1E 6JP, UK, 1991.

[22] Ministry of Defence. *The Procurement of Safety-Critical Software in Defence Equipment.* (Part 1: Requirements, Part 2: Guidance). Interim Defence Standard 00-55, Issue 1, Ministry of Defence, Directorate of Standardization, Kentigern House, 65 Brown Street, Glasgow, G2 8EX, UK, April 1991.

[23] Ministry of Defence. *Hazard Analysis and Safety Classification of the Computer and Programmable Electronic System Elements of Defence Equipment.* Interim Defence Standard 00-56, Issue 1, Ministry of Defence, Directorate of Standardization, Kentigern House, 65 Brown Street, Glasgow, G2 8EX, UK, April 1991.

[24] David L. Parnas. *Proposed Standard for Software Computers in the Safety Systems of Nuclear Power Stations.* Final Report for contract 2.117.1, for the Atomic Energy Control Board, Canada, March 1991 (By David L. Parnas, TRIO, Computing and Information Science, Queen's University, Kingston, Ontario K7L 3N6, Canada. Based on IEC Standard 880 [25].)

[25] IEC. *Software for Computers in the Safety Systems of Nuclear Power Stations.* International Electrotechnical Commission, IEC 880, 1986.

[26] IEC. *Software for Computers in the Application of Industrial Safety Related Systems.* International Electrotechnical Commission, IEC 65A (Secretariat) 122, Version 1.0, August 1991.

[27] IEC. *Functional Safety of Programmable Electronic Systems: Generic Aspects.* International Electrotechnical Commission, IEC 65A (Secretariat) 123, February 1992.

[28] European Space Agency. *ESA Software Engineering Standards.* European Space Agency, 8-10 rue Mario-Nikis, 75738 Paris Cedex, France, ESA PSS-05-0 Issue 2, February 1991.

[29] Radio Technical Commission for Aeronautics. *Software Considerations in Airborne Systems and Equipment Certification.* DO-178A, RTCA, 1 McPherson Square, 1425 K Street N.W., Suite 500, Washington DC 20005, USA, March 1985.

[30] J. Rushby. *Formal Methods and the Certification of Critical-Systems.* Technical Report SRI-CSL-93-07, SRI International, Computer Science Laboratory, 1993.

[31] ISO/IEC. *Information Technology — Open Systems Interconnection — Structure of Managed Information — Part 4: Guidelines for the Definition of Managed Objects*, IS 10165-4, July 1994.

[32] ISO/IEC. *UK Contribution on WG4 N1836: Use of Z for Managed Object Behaviour*, ISO/IEC JTC1/SC21/WG4 N1965, 20 June 1994.

[33] D. A. Duce, D. J. Duke, P. J. W. ten Hagen, G. J. Reynolds. *PREMO — An Initial Approach to a Formal Definition*, Computer Graphics Forum, 1994, *in press.*

[34] B. Meyer, *Object Oriented Software Construction*, Prentice-Hall International Series in Computing Science, C.A.R. Hoare Series Editor, Prentice-Hall, 1988.

[35] G. Booch. *Object-Oriented Analysis and Design with Applications.* 2nd Edition, Benjamin-Cummings, 1994.

[36] P. Coad, E. Yourdon. *Object-Oriented Design.* Second Edition, Yourdon Press, 1991.

[37] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen. *Object-Oriented Modelling and Design.* Prentice-Hall International Editions, 1991.

[38] H. Kilov, J. Ross. *Information Modeling: An Object-Oriented Approach*, Prentice-Hall, 1994.

[39] G. Blair, J. Gallagher, D. Hutchison, D. Shepherd. *Object-Oriented Languages, Systems and Applications.* Pitman Publishing, 1991.

[40] S. Cook, J. Daniels. *Designing Object Systems: Object-Oriented Modelling with Syntropy*, Prentice-Hall, 1994.

[41] R. O. Sinnott. *The Formally Specifying in LOTOS of Electronic Components*, M.Sc Dissertation, University of Stirling, March 1993.

[42] R.O. Sinnott, Kenneth J. Turner. *DILL: Specifying Digital Logic in LOTOS.* In R.L. Tenney, P.D. Amer, Ü. Uyar, eds *Proceedings of Formal Description Techniques VI*, pages 71-86. North-Holland, Amsterdam, Netherlands, 1994.

[43] Kenneth J. Turner. *Using Formal Description Techniques: An Introduction to ESTELLE, LOTOS and SDL.* John Wiley and Sons, 1993.

[44] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1.* Springer-Verlag, Berlin, 1985. EATCS Monographs on Theoretical Computer Science 6.

[45] G.I. Parkin, S. Austin. *Overview: Survey of Formal Methods in Industry*, Proceedings of Formal Description Techniques VI, pages 189-204, North-Holland, Amsterdam, Netherlands, 1994.

[46] D. Craigen, S. Gerhart, T. Ralston. *An International Survey of Industrial Applications of Formal Methods: Volume 1 – Purpose, Approach, Analysis and Conclusions.* Technical Report NISTGCR 93/626, National Institute of Standards and Technology, Gaithersburg, USA, March 1993.

[47] D. Craigen, S. Gerhart, T. Ralston. *An International Survey of Industrial Applications of Formal Methods: Volume 2 – Case Studies.* Technical Report NISTGCR 93/626, National Institute of Standards and Technology, Gaithersburg, USA, March 1993.

[48] Kenneth J. Turner. *Relating Architecture and Specification. Submitted to Special Issue of Computer Networks and ISDN Systems on Specification Architecture*, February 1995.

[49] ISO/IEC. *Office Document Architecture (ODA) and Interchange Format*, ISO 8613, Geneva, 1988, 1989.

[50] ISO/IEC. *Report of the ISO/IEC JTC1/SC24 Special Rapporteur Group on Formal Description Techniques.* ISO/IEC JTC1/SC24 N1152, 1994.

[51] ISO/IEC. *Use of Formal Description Techniques for ODP.* ISO/IEC JTC1/SC21/WG7 N753, 1991.

[52] ISO/IEC. *Z and Object-Oriented Z in ODP.* ISO/IEC JTC1/SC21/WG7 Arles, 1991.

[53] ISO/IEC. *The Use of Formal Description Techniques in SC24 Standards.* ISO/IEC JTC1/SC24 N1197, 1994.

[54] RAISE Language Group. *The RAISE Specification Language*, Prentice-Hall, Englewood Cliffs, New Jersey, USA, 1992.

[55] D. A. Carrington, D. Duke, R. Duke, P. King, G. A. Rose, G. Smith. *Object-Z: An Object-Oriented Extension to Z*, in Formal Description Techniques FORTE'89 (North Holland, 1987) pp 313-341, ed S. Vuong.

[56] BSI *Formalisation of the Computational Viewpoint Language in LOTOS.* ISO/IEC/JTC1/SC21/WG7 N941, July 1994.

[57] BSI *Formalisation of the Information Viewpoint Language in LOTOS.* ISO/IEC JTC1/SC21/WG7 N914, July 1994.

[58] BSI *Formalisation of the Enterprise Viewpoint Language in LOTOS*, BSI Input document for New Jersey meeting, December 1994. *Number to be assigned.*

[59] BSI *Formalisation of the Enterprise Viewpoint Language in Z*, BSI Input document for New Jersey meeting, December 1994. *Number to be assigned.*

[60] BSI *Formalisation of the Information Viewpoint Language in Z.* ISO/IEC JTC1/SC21/WG7 N915. July 1994.

[61] DIN *Formalisation of the Computational Viewpoint Language in ESTELLE.* ISO/IEC JTC1/SC21/WG7 N930. July 1994.

[62] DIN *Formalisation of the Computational Viewpoint Language in SDL'92.* ISO/IEC/JTC1/SC21/WG7 N932, July 1994.

[63] A. Vogel. *Entwurf, Realisierung und Test von ODP-Systemen auf der Grundlage formaler Beschreibungstechniken,* submitted as PhD Thesis, Humboldt-Universität zu Berlin (1993). *In German.*

[64] AFNOR *Direct Formalisation of the Computational Viewpoint Language* ISO/IEC JTC1/SC21/WG7 N936, July 1994.

[65] S. Rudkin. *Inheritance in LOTOS*, Fourth International Conference on Formal Techniques (FORTE'91), pages 409-424, Sydney, November 91.

About the Authors:

Richard Sinnott graduated in Theoretical Physics from the University of East Anglia (Norwich) in 1988. He obtained his Masters (M.Sc) in Software Engineering from the University of Stirling in 1993. Since then he has been working as a research fellow at Stirling on the Formalisation of Open System Architectures (FORMOSA) project. This work is funded by a joint grant from the Department of Trade and Industry and the Engineering and Physical Sciences Research Council. He is currently acting as the editor of the ISO/ITU-T standardisation activity of the formalisation of the Reference Model of Open Distributed Processing. His research interests lie in formal methods (LOTOS and Z in particular) and distributed systems.

Ken Turner graduated in Electrical Engineering from the University of Glasgow in 1970. He was awarded a Ph.D from the University of Edinburgh in 1974 for his research on Pattern Recognition. Until 1986 he was mainly employed by International Computers Ltd. as a data communications consultant. During this period he specialised in systems architecture, data communications and formal methods, leading to his appointment as Professor of Computing Science at the University of Stirling in 1987. His research interests lie in formalising systems architecture using the ISO Formal Description Technique LOTOS.