Richard O. Sinnott and Kenneth J. Turner. Modelling ODP viewpoints. In B. Cameron, C. Geldrez, A. Hopley, D. Howes, B. Mirek, and M. Plucinska, editors, Proc. OOPSLA 94 Workshop on Precise Behavioural Specifications in OO Information Modelling, pages 121-128, Portland, Oregon, USA, October 1994.

Modelling ODP Viewpoints

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

Abstract

This paper gives a brief insight into the current work on the development of an architectural semantics for Open Distributed Processing (ODP). It first provides an introduction to the work on the formalisation in LOTOS and Z of the basic modelling and specification concepts of Part 2, and then focuses on the viewpoint languages of Part 3 of the Basic Reference Model for ODP (RM-ODP). It also shows up the separation of concerns that is achieved through these viewpoints thereby enabling systems to be considered from aspects which might be of interest to different people. This paper also highlights the way in which conformance between these formalised viewpoints can be checked, thereby ensuring that the system as a whole is consistent with the system as a collection of abstract viewpoints of the system. Finally the paper concludes with a brief discussion on the advantages and disadvantages of LOTOS and Z for modelling viewpoint languages and the RM-ODP generally.

1 Introduction

In this paper it is assumed that the reader is aware of the work on ODP generally and that of the RM-ODP. Further information and an introduction to Part 4 of the RM-ODP can be found in [10] and [12]. It is also assumed that the reader is familiar with the FDTs LOTOS and Z. Further information can be found in [4], [5].

2 Basic Modelling and Specification Concepts

It is often the case that writing specifications proves to be difficult due to poor specification structures to begin with. Thus having a correct 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 not based on a well structured architecture tend, not only to be difficult to write, but 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 leads to "messy" specifications.

Formalising the basic modelling and specification concepts of Part 2 gives a precise understanding of the basic concepts used in the RM-ODP [1]. This includes identifying ambiguities or incorrect text in Part 2, *e.g.* interface and interface signature have been identified as inconsistent. Through this formalisation, a solid architecture for writing specifications of ODP systems (and standards) is achieved. This also enables a uniform and consistent comparison

[1] This formalisation is achieved by interpreting a given concept in terms of the constructs of a given FDT.

of formal descriptions of standards written in different FDTs to be achieved and it acts as a bridge between the ODP and FDT semantic models.

Thus when completed, the architectural semantics work should allow standards developers, in this case the standards that are to be generated from the RM-ODP framework, to have a clear unambiguous interpretation of the concepts contained within the RM-ODP. It should also provide specifiers with a library of definitions which they may use to specify ODP standards with.

2.1 Basic Modelling and Specification Concepts in LOTOS

The FDT LOTOS can interpret most of the basic modelling and specification concepts. Often this requires that a given style of specification (object-oriented) is followed. For example, having some means whereby an object may be uniquely identified, *e.g.* having a formal parameter associated with the process definition representing the object template which must, when instantiated, be unique in the system. From this, it may be seen that the imposing of a style represents a level of prescriptivity which users of Part 4 must follow. This prescriptivity has associated with it other issues which need to be considered, *e.g.* scalability and compatibility. These issues are discussed in [10]. This work also addresses some problems in the formalisation of the basic concepts of Part 2. For example the formalisation of the notion of type and hence class are not possible due to their generality.

2.2 Basic Modelling and Specification Concepts in Z

The FDT Z can also interpret most of the basic modelling and specification concepts. One area where Z is limited, however, is in the interpretation of concepts which requires more than a single object. For example, interaction and composition. The main problem with the FDT Z is that it does not provide for the object-oriented feature of encapsulation. See 5.2 for more details on this and possible solutions to this problem.

2.3 Formalising the Viewpoints

The relationship between Part 2 and Part 3 may be seen as a specialisation one. That is, Part 2 gives a basic interpretation of a given concept and Part 3 gives a more specialised version, *e.g.* an interface and a stream interface. One way of considering this specialisation is that Part 2 provides a vocabulary for consideration of Part 3. Thus 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.

The rest of this paper gives a snapshot of the recent work in LOTOS and Z on formalising the computational and information viewpoint languages, and offers a first consideration as to the potentiality of formalising the enterprise viewpoint language. It also attempts to identify areas of conformance between these different viewpoints.

3 Enterprise Modelling

It is not possible to formalise anything without first having statements about the system under consideration. These statements often take the form of business rules, captured in ODP by the

notion of *policy*. A policy is defined in [1] as "a set of rules related to a particular purpose". These rules may be: *obligations* (a prescription that a particular behaviour is required); *prohibitions* (a prescription that a particular behaviour must not occur); and *permissions* (a prescription that a particular behaviour is allowed).

Most often these rules are given in natural language and as such are open to interpretation and possibly ambiguity. Formal methods offer precision and clarity in defining such rules.

It should be pointed out that it may not always be possible to formalise statements that might be made in the enterprise viewpoint. For example, statements of purpose are often informal by nature and cannot easily be formalised. Formalisation here would require a solution to the natural language processing problem [2]. Policy statements on the behaviour of the system might well be formalised however.

Ideally what is required from an FDT from the enterprise viewpoint is concise statements about the system under consideration. Statements of the form "this must happen" and "this must not happen" will be commonplace. To account for this the FDT should be both precise and yet abstract. That is, it should allow for precise statements to be made about the system without having to go into the details of the components of the system or how they are configured, *i.e.* it should be design independent as far as possible.

Having said that, it should be pointed out that it is not always possible to take a particular ODP viewpoint in isolation; viewpoints will to some extent effect others. For instance, considering a given system from the enterprise viewpoint requires rules to be made. These rules will almost certainly have an effect on the information in the system and the legal manipulations of this information, as well as the overall behaviour of the system. Thus the information and computational viewpoints of ODP have to be considered at the same time as the enterprise viewpoint. However, the enterprise viewpoint should take precedence over all other viewpoints. That is, one way of considering the enterprise viewpoint is as a framework for other viewpoints, *i.e.* the other viewpoints realise the statements made in the enterprise viewpoint.

3.1 Enterprise Modelling in LOTOS

As LOTOS is Turing complete, it should be possible to formalise any statement about the required behaviour of a system from the enterprise viewpoint. For example, one policy statement for a distributed database system might be that "the number of users logged on never exceeds 10". This might be modelled in LOTOS using ACT ONE data types representing users and some behaviour in the process algebra part representing the database. In the behaviour expression representing the database would then be some behaviour whereby users could log-on. There would likely be some constraint here in the form of a guard or selection predicate that no more than the 10 could log-on. Thus combinations of ACT ONE and process algebra behaviours are likely in capturing policy statements.

As can be seen from this simple example, policy rules can be modelled in LOTOS. However, it is fairly obvious that LOTOS is not ideally suited to modelling these rules. The problem with LOTOS is that there is no inherent property of the language which can be used to model rules which govern the behaviour of a system. Rather, the behaviour of the system is specified directly and rules are obeyed by the system providing the correct behaviour.

Thus what is required from an FDT modelling the enterprise viewpoint is the ability to

[2] Statements of purpose are usually supplied in the informal commentary accompanying every specification.

write high-level rules (invariants) for the system. In LOTOS, these rules are given at a lower level, *i.e.* in the actual behaviour itself. Thus they almost disappear in the specification itself.

It might be the case that specification styles can be used to capture rules on behaviour as succinctly as possible. For instance, a monolithic or constraint-oriented style of specification permits the "what" of a system to be captured. However, it is possible that a complete specification has to be written to show a single rule. Thus LOTOS can be considered as not abstract enough to capture policy statements ideally.

3.2 Enterprise Modelling in Z

The FDT Z is more suited to modelling systems from the enterprise viewpoint as it offers a higher level of abstraction and enables invariants for the entire system (policy rules) to be written down directly.

The ability to write down an invariant as a rule for the enterprise viewpoint defines what obligations and prohibitions must obey. Thus using the above example, for a distributed database policy rule, this can be written in Z as an invariant of the form $\#USERS \leq 10$. This could be part of the state schema or axiomatic descriptions in the specification, where the data type *USERS* has been previously defined. All operations in the specification must then satisfy this predicate (rule). Thus a non-total operation to log on another user would leave the system in a non-determinant state, if the number of users was already 10. Thus this invariant represents a boundary which separates obligations and prohibitions.

A permission might be modelled as the (re-)setting of the binding of a variable to a certain value when a certain postcondition becomes satisfied. For instance, having an operation which sets the number of users in this example to zero when it was previously 10 would be a permission.

4 Information Modelling

In order to consider how information is modelled in the FDTs LOTOS and Z, it is first necessary to consider what exactly is meant by information. Information is defined in [1] as: *Any kind of knowledge about things, facts, concepts and so on, in a universe of discourse that is exchangeable amongst users. Although information will necessarily have a representation form to make it more communicable, it is the interpretation of this representation (the meaning) that is relevant in the first place.*

To give a representation-free way of modelling information, [2] uses the notion of *invariant, dynamic* and *static schema*. These represent the properties of information that are independent of behaviour, dependent upon behaviour and give the state and structure of information at some particular time respectively.

4.1 Information Modelling in LOTOS

Generally, in LOTOS the information to be modelled in a specification is represented in ACT ONE. This is manipulated and used in the process algebra part. Information might also be modelled using the process algebra part of LOTOS. However, the above definition of information in ODP states that the information must be exchangeable amongst users. In LOTOS, exchange of data can only occur between instantiations of process definitions using data modelled in ACT ONE. It is not possible in LOTOS to exchange process definitions

between process definitions, hence only the ACT ONE part of LOTOS has been considered here for modelling information. Thus a given information item will be represented by an instance of an ACT ONE sort with associated operations and equations.

From this it can be seen that the whole ACT ONE part of a LOTOS specification represents an ODP *invariant schema* that instances of sorts and their manipulations declared in the process algebra part of LOTOS must satisfy. This is slightly strange (but correct!) in that the equations and operations in ACT ONE are not "watchdogs" looking over illegal information manipulations; they define the manipulations and so cannot be wrong in themselves!

An ODP *static schema* is represented by the binding (value) associated with the name of an instance of an ACT ONE sort at some point in time, *e.g.* an ACT ONE sort *Queue* might be initially bound to the value *empty*.

An ODP *dynamic schema* is given by the ACT ONE expressions used to modify information objects (ACT ONE sorts) in the process algebra part of a LOTOS specification. It should be pointed out that it is ACT ONE expressions that are associated with the action denotations of the behaviour expressions that manipulate the information. For example, consider the event offer *g !push(val,empty queue)* that might exist as part of the behaviour of a given specification. It is not the event offer itself that manipulates the information item, in this case an empty queue, but the ACT ONE expression (*push(val,empty queue)*) that manipulates the information. Thus it is only ACT ONE that manipulates the information; however, the ACT ONE can only be used in LOTOS when it is associated with behaviour, *i.e.* the process algebra. Thus the notion of information processing activity in LOTOS is linked both to the ACT ONE information modelling, and the ACT ONE information processing contained in the process algebra part of LOTOS.

One way of considering this is that the process algebra represents a behavioural framework on which information can be hung and manipulated. Further information modelling considerations in LOTOS can be found in [3].

4.2 Information Modelling in Z

With regard to information in the FDT Z; information is given through mathematical data types. This has the advantage when used to develop an architectural semantics for the information language of being precise yet not fixed to any technology or computer language. Thus information may be modelled so that it can always be interpreted unequivocally.

These data types may be either: basic types; set types; Cartesian product types; schema types; or more complex types composed of combinations of these types, *e.g.* functions and relations etc. The application of these types categorise the data types into sets. The type (*ODP notion*) of any given data type is then determined by set membership.

An ODP *invariant schema* can be written explicitly in Z. It corresponds to any invariants (predicates) that may be present in the state schema or axiomatic descriptions associated with a Z given specification. These predicates are independent of behaviour, or rather they govern the possible behaviours of the given Z specification, where behaviour in Z is modelled as the performance of operations defined in operation schemas.

An ODP *static schema* corresponds to the binding associated with the variables declared within the state schema of a Z specification at any point in time. Here time is an abstract notion which is related to the state of the specification. Time is thus dependent upon the state changes that have occurred and those that can occur within the specification.

An ODP *dynamic schema* corresponds to the possible behaviours that might be exhibited

by a Z specification. A dynamic schema is given by the operation schemas present in a Z specification. All operation schemas must satisfy any global predicates (invariants) that may limit the bindings that can be taken by variables declared in the state schema or in axiomatic descriptions associated with the specification. Further information modelling considerations in Z can be found in [3].

5 Behavioural Modelling

The computational viewpoint language of [2] gives the functional decomposition of an ODP system. That is, it is concerned with the different behaviours that can exist in a distributed system and how these may be configured to achieve interoperability. The essential concept is that of an interface. Three kinds of interface are identified: stream, operational and signal. The computational language then gives rules which govern the way in which these interfaces may be configured, *e.g.* only interfaces in a subtype relation may be bound to one another with interactions only taking place through binding objects.

It should be pointed out that the information viewpoint deals with behavioural modelling also, but this is more concerned with behaviour as an information processing activity. It does not attempt, for example, to provide details as to how interoperability of behaviours can be achieved. Rather, it gives the legal manipulations of the information in the system.

5.1 Behavioural Modelling in LOTOS

As far as LOTOS is concerned, behaviour is given by writing down expressions directly in the process algebra part of LOTOS. Although LOTOS can model most of the concepts of the computational language provided a certain specification style is chosen, *e.g.* using identifiers to distinguish the different forms of actions, there exists the problem of applying structuring rules to these concepts.

Generally speaking in LOTOS (and elsewhere!), it is not the case that two arbitrary interfaces can be connected (synchronised) such that "correct" interaction occurs, *e.g.* one free from deadlock or livelock. The problem is primarily caused by the inability to apply a type predicate to an arbitrary behaviour. However, when dealing with interfaces from the computational viewpoint, the interfaces are no longer arbitrary. They have defined structures which ODP compliant specifications from the computational viewpoint must adhere to. Hence it is possible to connect interfaces and ensure a form of correct behaviour. It should be pointed out that checking this behaviour can only be done outside a LOTOS specification. See [3] for more details on the formalising the computational language in LOTOS.

5.2 Behavioural Modelling in Z

As stated earlier, behaviour is modelled in Z by the performance of operations defined in operation schemas. In effect there is no difference in technique between modelling ODP information schemas and modelling behaviour more generally in Z. This is unlike LOTOS where two separate languages, ACT ONE and the process algebra, can be used to specify the system from different considerations.

It remains to be seen whether Z can model all of the features of the computational viewpoint language. At first sight Z is restricted in that, generally speaking, a Z specification represents a single object: there being no means in Z to guarantee object encapsulation.

Various object oriented flavours of Z have been introduced, *e.g.* [7], [8], which permit the specification of more than one object in a single specification along with other features, *e.g.* inheritance. However, the use of an object-oriented version of Z is limited in the ODP work as such versions are not as stable or widely known as Z. There is also the question of which is best suited to ODP work. Other work [6] attempts to use Z in an object-oriented way, however the modelling of more than one object here depends on the definition of an object.

One possible solution to these problems of object oriented versions of Z might be found in ZEST [9]. Work on GDMO [11] is using ZEST, despite the fact that it is not currently standardised. The reason is that a ZEST specification may be flattened into a Z specification. Therefore it is argued that it is only the Z language being used.

6 Conformance Issues

6.1 Viewpoint Conformance Consideration in LOTOS

LOTOS does not readily allow for the conciseness required to model enterprise policy statements. However, if these statements are modelled using a combination of ACT ONE and process algebra behaviours, it should be possible to check for conformance with the information and computational viewpoint languages. Conformance here might well be assessed by some form of behavioural equivalence relation between the system described from the enterprise viewpoint, *i.e.* some basic behaviours which sets out the policies of the system, and the more complex behaviours of the system represented by information and computational viewpoint considerations. This is a non-trivial task however.

There is direct conformance between the computational and information viewpoints in LOTOS. That is, ACT ONE is used to model the information and the process algebra is used to model the behaviour which manipulates the information. Conformance between the computational and information viewpoints is thus given when the information is used correctly. Thus, the process algebra must not violate the ACT ONE information modelling. It should be pointed out that in LOTOS there exists the possibility to specify a system so that it is not possible for information to be used in an "incorrect way". This may be achieved by extending the interface to the ACT ONE sorts, *i.e.* adding operations to the signature of the ACT ONE sort so that all possible modifications to the ACT ONE sort are given. Which style of information modelling is chosen depends upon whether there are likely to be limitations on the information to be modelled. That is, it may not be desirable to extend the interface of the ACT ONE sort. An example of this is given in [3].

6.2 Viewpoint Conformance Consideration in Z

Z allows for concise statements to be made from the enterprise viewpoint. These statements represent the invariants of a given specification. Conformance between the enterprise and other viewpoints, *e.g.* the computational and information viewpoints, is realised when the invariants representing the enterprise statements are not violated.

Conformance between the information and computational viewpoints is likely to be given by the correct use of information. Thus operations which manipulate data must satisfy preconditions and postconditions given in the ODP schemas representing information items.

7 Conclusions

Formal methods have a very important role to play in ODP. Apart from the additional advantages to the RM-ODP in clearing up any ambiguous (or incorrect) text, formal methods offer guidance to specifiers or systems developers using the RM-ODP. It might also be considered that formal methods represent the first real test of the RM-ODP, with the formalisation of the viewpoints and the checking of conformance between them.

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 on from the ODP work. Object-oriented features in Z and dynamic configurability features in LOTOS represent two such examples.

As has been seen here, the FDTs LOTOS and Z have particular advantages and disadvantages in formalising the architecture of ODP. Z appears to be more suited to the enterprise viewpoint modelling whilst LOTOS appears more suited to the computational viewpoint modelling. They both offer facilities to model information directly, although it could be argued that Z is better suited for this due to its higher level of abstraction.

Following the work on the formalisation of the viewpoint languages, it has been identified that the scope of Part 2 should be extended. For example, concepts such as binding and policy which are defined elsewhere in Part 2 are used in the interpretation of the viewpoint languages. As a result of this, the workload on formalising the RM-ODP has increased further, hence collaborative international work is required immediately to provide a sound and apposite architectural semantics for ODP.

References

[1] Basic Reference Model of ODP — Part 2: [Recommendation X.902 - ISO 10746-2].

[2] Basic Reference Model of ODP — Part 3: [Recommendation X.903 - ISO 10746-3].

[3] Basic Reference Model of ODP — Part 4: [Recommendation X.904 - ISO 10746-4] *Working Draft — Post-Southampton document*

[4] *The Formal Descriptive Technique LOTOS*, P.H.J. van Eijk, C.A. Vissers, M. Diaz, North-Holland, Amsterdam, 1989.

[5] J.M. Spivey. The Z Notation, A Reference Manual, Second Edition, 1992. Prentice Hall International Series in Computer Science, C.A.R. Hoare Series Editor.

[6] Using Z as a Specification Calculus for Object-Oriented Systems, J.A. Hall in VDM'90: VDM and Z - Formal Methods in Software Development, LNCS 428, pp 290-318.

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

[8] Object-Oriented Process Specification, S.A. Schuman, D.H. Pitt, P.J. Byers in Specification and Verification of Concurrent Systems, Workshops in Computing, Stirling (Springer-Verlag 1990), pp 21-70, ed C. Rattray.

[9] ZEST — Z Extended with Structuring, G-H. Bagherzadeh Rafsanjani, British Telecommunications Deliverable, CNS/9/DELIV/0001, February 1993.

[10] The Development of an Architectural Semantics for ODP, R O Sinnott, TR-122, Department of Computing Science and Mathematics, March 1994, University of Stirling.

[11] ISO/IEC JTC1/SC21/WG4 10165-1 to 4, *Management Information Model*, 1991.

[12] *Computer Standards and Interfaces*, Vol 15, Nos 2-3.