What Foundations does the RM-ODP Need?

Peter F. Linington.

University of Kent,
Canterbury, Kent, CT2 7NF, UK.
pfl@kent.ac.uk

Abstract

This position paper revisits the requirements for the set of Foundation Concepts for the ODP Reference Model and the approach originally taken to satisfying them. It then examines, in the light of experience, the areas where the Foundations have subsided, and areas where extensions need to be built. The aim is to provide a starting point for discussion on requirements to change the Foundations document.

1. Introduction

Even before the first draft of the Reference Model for ODP was produced, the group of experts working on it had found the need for a separate definition of a clear conceptual framework on which to base their work; almost ten years later, this became Part 2 of the published standard [1], and established the conceptual framework for the ODP Architecture [2] [3]. The need for the RM-ODP Foundations document was clear; experts from a number of different backgrounds had come together to work on ODP, and they brought with them a wide range of different vocabulary and usage, reflecting different assumptions about how systems should be structured and specified. Progress with a common reference model depended on the creation of a common conceptual framework.

At the same time, no single notation or descriptive technique could be expected to dominate. The broad scope of the work was such that different techniques would be needed to express different areas of concern. It was therefore necessary to express the Foundations in abstract terms, bringing together the common features of existing styles of usage, so that each concrete notation is seen as a refinement of the foundation concepts. This is particularly the case in the areas of interaction and behaviour, which are discussed below.

In reviewing how well the RM-ODP has stood the test of time, we must be aware of the constraints on modifying it. Not only must the Parts of the reference model maintain their internal consistency, but the documents that reference it must not be undermined. This applies both to the ISO standards within the framework the reference model has created, and to the work within bodies such as the ITU-T and the OMG, and also the usage within the wider community. We do not have a green field; we have a responsibility to perform restoration, not demolition and replacement. The Foundations may need to be strengthened, but not relaid.

2. Objects and Interactions

At the time that the Foundations was being debated, two formal description techniques were also being developed within the same parent standards committee. These were LOTOS [4] and ESTELLE [5]; they differ significantly in their representation of interaction and this coloured the discussions in ODP. In LOTOS, the processes interact at gates in a synchronous way, in that all the parties to an interaction would agree on when the interaction occurs (although basic LOTOS deals with action sequence, not timing). In ESTELLE, on the other hand, modules are linked by communication links, which contain umbounded queues, so that interaction is represented by distinct sending and receiving actions that are sequenced but not simultaneous.

This led to a need for a common modelling basis that was capable of unifying both approaches. Interactions in ODP are synchronous, but are defined between an object and its environment. If we then constrain the way objects are composed so that each object binds directly to another object in its environment, the resulting communication is synchronous, but if objects bind to link ends in their environment, the resulting communication is asynchronous.

One might feel that the asynchronous representation was slightly cumbersome, but the Foundations also provide the less closely coupled concept of communication, which represents a sequence of causally related interactions, so that channels between communicating objects can (but need not) be completely hidden in this style of notation.

Currently, the Foundations do not distinguish between different kinds of interaction, but leave it to the specification using them to refine the basic concept. This, together with the connotations in English of the work "interaction", may have given the impression that the intension is something like a method call. This could be avoided and the full generality demonstrated by including in the standard non-exclusive definitions for some common refinements of interaction, such as invocation, message transfer and event notification.[1] For brevity, these examples are taken from a computational domain, but the definitions added should be viewpoint-independent, with notes to clarify their application, including examples in at least the engineering and enterprise domains.

3. Interface

The Foundations define an interface in terms of a view of the behaviour of an object, resulting from taking a subset of the interactions of the object and hiding all the other interactions and behavioural constraints that involve them. This definition is basically sound, but there are some subtleties that need to be taken into account in order to understand it fully.

Let us consider a computational application of the definition to see some of the problems; assume a style of interaction in which there is a clear causal initiative, so that an object is invoked by its clients and may, as a result, invoke other objects providing services (see figure 1).

The object, Obj, has four interfaces and its complete behaviour places constraints within and between them. Its environment contains four objects, A to D. The figure shows that:

a) object A can initiate interaction P at interface If_1 at any time;

b) object B can initiate interaction Q at interface If_2 at any time;

c) object B can initiate interaction R at interface If_2 but constraint C_1 requires Q to happen first; when R does occur, it is followed by Obj initiating interaction T with object D at If_4;

d) object B can initiate interaction S at interface If_2 at any time; when S does occur, it is followed by Obj initiating interaction U with object C at If_3; the behaviour of C results in an interaction V with Obj at interface If_2.

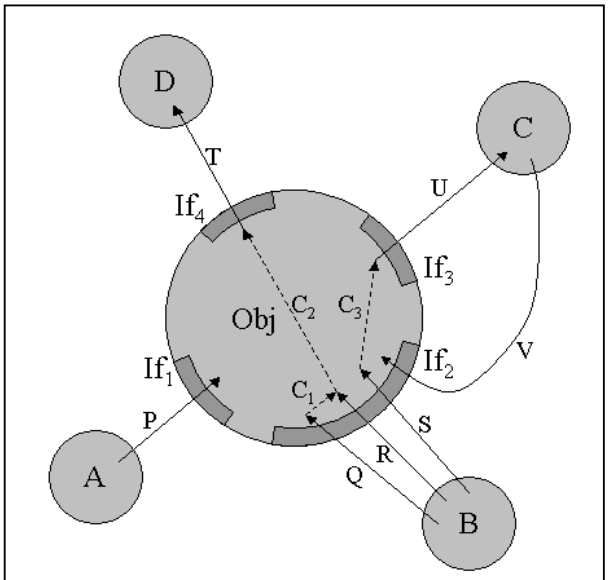

Figure 1 – Interfaces and Object Behaviour

Now let us see how the definition of the interface concept works. The first thing to note is that all the interactions of Obj are considered, and not just those in which it acts as a responder. Thus we are dealing with four interfaces, whereas a computational middleware might only consider interfaces 1 and 2 at which services are offered. Secondly, the interactions here are action occurrences, and not action types; actions P and Q could be of the same type, but the occurrences in the two interfaces are distinct. In fact, as a result of this, the sets of interactions are normally infinite, because most objects will have a behaviour that allows arbitrary repetition of smaller fragments of behaviour associated with some sort of thread or session; almost all notations simplify this by considering as a unit the sets of interactions with similar types and names (where the naming domain is associated with the interface). Note that this idea of equivalence sets implies that the identification of interfaces is a design activity – the interfaces cannot be deduced by examination of the overall computational behaviour (although, once the decision has been made, it is generally reflected by the naming structure used for interactions in the engineering viewpoint).

[1] The foundations originally avoided the use of the term event because of the connotations arising from the world of discrete event simulation; however, the use of the term event notification seems to avoid this.

Now let us consider the individual interfaces in the example. Interface 1 consists of occurrence of members of the set P without further constraint.

Interface 2 consists of the interaction sets Q, R, S and V, but subject to the constraint that an R must be preceded by an occurrence of Q. The constraints 2 and 3 are hidden in interface 2 because they involve interactions T and U, which are themselves hidden. The multi-step constraint between S and V is hidden for the same reason – it depends on U and also on the behaviour of C, which are themselves both hidden. These constraints only become apparent when the full behaviour of the object is considered; the link from S to V depends on specific behaviour in the environment, and so cannot be deduced from constraint 3 alone. However, a constraint between S and V could have been stated in interface 3 in terms of locally available properties, such as the presence of a correlation identifier as a data item in both the interactions.[2]

Interfaces 3 and 4 are again straightforward; apart from the placement of the initiative for interaction, they are structurally similar to interface 1.

What we have not yet considered is the dynamic lifecycle of an interface. Before interaction can take place, there are normally two steps to be taken. First, the object must be in a state where it is willing to interact (corresponding to the creation of the interface and its associated naming domain for interactions and initialisation of related internal state of the object) and second any interactions with the environment needed to establish preconditions for interactions at the interface must have been performed (corresponding to the establishment of a binding and associated communication and resultant state shared between the objects involved). After the first step, there is potential for interaction, but no specific partner for interaction has been selected, whilst after the second step interaction with specific partners can take place. Different kinds of object behaviour allow the description of one-to-one bindings or many-to-one client-server bindings. Similar considerations apply to the deletion of a binding and an interface.

When the Foundations were being drafted, a number of ways of modelling this process were considered, mostly based on including an intermediate concept for the unbound state, such as semi-interfaces or unbound interfaces, but none of them were successful. It would be worth revisiting this issue, but a better solution might be to consider a more general way of describing the potential behaviour of an object that is currently in a particular state (see section 8 below).

[2] This has different semantics, particularly with respect to the trust implications for C, but it may be equivalent for practical purposes.

4. Components

One of the significant changed in the last ten years has been the growth of interest in component-oriented architectures, so a natural question to ask is whether the Foundations should include a general definition of what a component is. If we consider, for example the CORBA Components 3.0 Specification (figure 2), and ask what the key properties of a component are, we find

a) encapsulation;

b) interactions at ports; the ports can be specialised as facets, receptacles, event sources, event sinks or attributes;

c) a component equivalent interface that provides metadata, navigation and control for the component;

d) an associated component home interface, representing a container in which components of the given type can be instantiated.

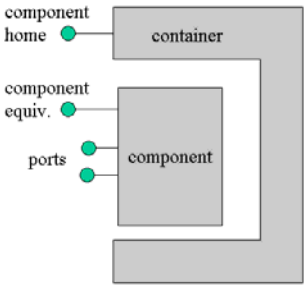

Figure 2 – A component model

In fact, all of these can be modelled using the existing Foundations. The general concept of interaction is rich enough to be refined into any of the defined port types, and the equivalent and home interfaces are just conventional computational interfaces. The container property requires a specialisation of the object concept to make explicit the relation to the instantiation of templates involved when it acts as a factory.

The ability to support both facets (server interfaces) and receptacles (client interfaces) as specialisations of the ODP concept of interface has already been mentioned above; it is primarily this generality, and the equivalent capabilities for interaction support, that make the foundation concept of object so flexible.

Perhaps a little more needs to be said about encapsulation. The Foundations explain encapsulation as the property of an object such that it can only have its state changed by interactions or internal actions expressed in the model. As such, the definition is more related to

completeness of description than to level of abstraction. Indeed, the inability to guarantee encapsulation on structural refinement is one of the problems in security analysis (see figure 3), for example, since structural refinements may introduce backdoors, and it is difficult to apply constraints to the refinement process that prevent this.

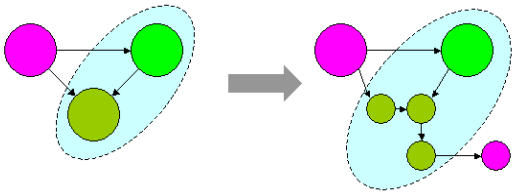

Figure 3 – Refinement breaks a security boundary

However, there would be nothing to choose between an object model and a component model built on the Foundations in this respect, even if our common intuition is that encapsulation of a component is a less abstract claim than encapsulation of an object.

As a result there is no need for extension of the Foundations if it was felt that there was a need to incorporate a computational or engineering component model into the architecture. However, there could be merit in adding derived definitions for component and factory, making it clear how the existing definitions can be used to model them.

5. Roles

The Foundations define a role as "an identifier for a behaviour, which may appear as a parameter in a template for a composite object, and which is associated with one of the component objects of the composite object". This definition has been much misunderstood, and some authors have tried to rework the definition in terms of static class structures, without much success. To attempt to do so is to miss the point of the definition, but clearly it does not, in its present form, convey the intent, which is indicated by the reference to templates and to the actualisation of parameters in its second paragraph (which is not quoted here).

The discussion here is based on the explanation in [6]. The metaphor on which the role concept is based is theatrical. The text of a play is expressed in terms of lines and actions associated with various roles, which are declared initially in a cast-list. Putting the play on involves assigning actors to the various roles, although one actor may play several minor roles, and the actor playing a role may change during the run of the production. Identifying the roles rather than the actors obviously makes the script more reusable.

The key idea is that some constraints on system behaviour are associated with objects dynamically as a consequence of an earlier part of the behaviour, such as performance of a piece of negotiation. However, although the potential behaviour can be referenced (and hence the talk of an identifier in the definition) it is not associable with an actual object until the template is instantiated and the role bound to a specific object[3]. It is thus impossible to represent what is going on within a static class hierarchy.

The solution to the misunderstandings is to remove the idea of there being a parameter identifier in the template's behaviour to the explanatory note, and to focus the first paragraph of the definition on the idea of parameter substitution. Perhaps more importantly, though, we need to clarify the way the potential behaviour of an object is restricted when it is bound to a role, and this needs a proper framework for the discussion of potential behaviour of the kind described in section 8 below.

The second problem with the role concept at present is with usage rather than definition. There has been widespread discussion in ODP circles of community-roles in the Enterprise Language, but unfortunately this has been expressed the term using role without qualification, leading to an implicit assumption that saying role implies community-role. As shown above, role is defined as a parameterization mechanism for templates, and so can potentially be applied to anything for which a template can be defined. Indeed, there are other places where the role concept is not just useful but is vital to making necessary distinctions in the template definition.

Perhaps the clearest present need is in the definition of action templates, particularly interaction templates. In an interaction between, say, a client and a server object, it is essential to know which is which. We can do this by saying that the two objects in this example fill client and server roles in the interaction, and by associating necessary properties and constraints with these roles. At one point in a system's behaviour, an object A can fill the buyer role in a purchase interaction, while object B fills the seller role, and later the roles can be reversed, so that B is the buyer and A is the seller. The richer the interaction, the more useful this

[3] Depending on the nature of the template involved, some roles may be bound after the instantiation of the object defining them. This is particularly true of objects representing potentially long-lived structures like communities, where the behaviour will commonly include the dynamics of community-role bindings (e.g. changes of committee membership). However, the lifetime of the role binding is always within the lifetime of the defining object, so that the objects created by a factory are not simply filling roles in it.

approach is likely to be; it is particularly effective, for example, for expressing the capabilities and obligations associated with secure multi-way interactions, where capabilities or access permissions are clearly associated with a specific role. Another example is for distinguishing between actor and artifact roles in enterprise interactions. Users of the role concept should be encouraged always to qualify their use of role with the template type name, as in action-role and community-role.

6. Obligations

The Foundations define a number of deontic concepts, particularly obligations, permissions and prohibitions, but this part of the framework was produced before there had been much experience with their application in ODP. The result is that the definitions were taken directly from the Standard Deontic Logic (SDL), including a simple set of relations between the concepts, such as the assertion that a permission for something is an indication that there is not an obligation not to do it.

There is nothing wrong with these as statements from the SDL, but experience has shown that the SDL approach is somewhat brittle for enterprise modelling, and it would be better to take a less prescriptive approach in the Foundations, allowing, for example, a style of modelling based on Utilitarianism to be exploited if it proves effective. See [8] for a discussion of how obligations might be represented in this way.

7. The lifecycle of ODP specifications

It has always been a principle in the development of ODP that the reference model is neutral with regard to the methodologies to be applied, and maintaining this position gives the most broadly applicable framework.

However, It is clear that most systems will evolve over time, and the reference model needs to take this into account. The Foundations includes the concept of an epoch to describe the way in which objects or configurations of objects evolve through a series of stages. This concept can also be used to describe the evolution of the specification itself. This allows a new version of a specification to describe, for example, how it might be introduced as a staged transition from the previous version.

One area where there must be an expectation of evolution and statements of constraints on it is in the definition of policies. The current Foundations definition of a policy is very weak. It is just defined as “a set of rules related to a particular purpose”, with an indication that the rules are expected to be expressed in deontic terms. There has been a great deal of work on the use of policies, particularly in various styles of policy-based management, since the creation of the reference model, and the requirements are now much better understood. We can now identify at least two stronger requirements for a rule to be considered a policy.

The first of these is that there must be some element of choice associated with any policy. Policies are identified in a specification wherever it is recognised that a rule may need to be changed during the lifetime of the specification; selecting a structure for the specification that emphasises the scope of the policy makes it easier to modify it without wholesale revision, and allows the likelihood of change to be reflected in the implementation. This is generally done by encapsulating associated decisions as the behaviour of a distinct computational or engineering object that can be replaced whenever the policy is changed.

Thus a rule that is universally true, and cannot be changed without wholesale replacement of the specification, is not a policy. The speed of light is not a policy, but the setting of interest on credit at a certain percentage above base rate is. Whether an organization operates with the status of a charity either might or might not be a policy, depending on whether the specifiers foresaw the possibility that the status might change and planned for it.

The second thing to be said about policy is that the specifiers who identify it will generally wish to limit the range of behaviour that would be acceptable; this gives rise to the idea of a policy envelope [7], which limits the range of behaviour any particular policy is allowed to specify. Knowledge of the policy envelope allows the verification of invariants on the specification that are independent of the particular policy in use at any particular instant.

Another aspect of the lifecycle of an ODP specification is the relationship between specification and instantiation, which is discussed in [6]. This paper identifies the need to enhance the ODP conformance model slightly so that it is able to distinguish between classes of use to which the specification is being put. The current ODP conformance model describes the relations between system the specifier, the implementor and the tester, and describes how conformance is deduced from observation during testing, confirming that the implementation is consistent with the original specification. A proposal made in [6] is to introduce the role of system owner, so that statements of rights to implement and use the design embodied in the specification can be made and then interpreted during the testing process to guide the interpretations made by the tester.

Making this comparatively minor extension opens the way to a more formal model of licensing and rights to use the design, and could help clarify the constraints on system

evolution involving reuse of design libraries or components.

8. Frames and unified semantics

Several of the areas examined so far have involved the need to describe the possibility of change or of the system responding to reaching a certain state of affairs or set of objectives. Constraints of the same basic kind are involved in basic behaviour like the interface and binding lifecycle, the definition of policies and the use or reuse of specifications. It would be a great aid to consistency of modelling if all these were based on a similar underlying model structure.

Most of the notations of practical interest here have their semantics defined denotationally, in terms of a mapping from notational elements to some mathematical target domain. For example, LOTOS is defined in terms of a labelled transition system. Other notations, such as UML, still lack a uniform and consistent semantic mapping. What is needed is a common target domain that is a natural extension of those in common use but with the power to be a target all the ODP-related notations, including the deontic aspects of enterprise languages.

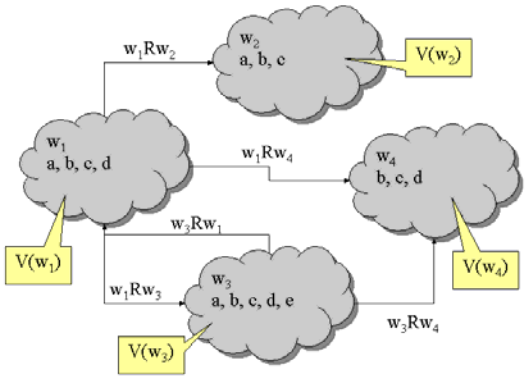

Figure 4 – A frame-based model

One possible direction would be to introduce a frame-based model such as one based on Kripke Frames (see [9] for an accessible review of these and other related systems able to support modal logics). The idea (see figure 4) is that the development or evolution of the system is represented by a model consisting of:

a) a set, W, of possible worlds of interest; the form of description of the world is not of interest here, except that it is decomposable into a set of attributes that carry Boolean values;

b) a dyadic relation, R, representing the reachability between members of W; R is true if w_2 is accessible from w_1, and false otherwise;

c) a value assignments, V, that assigns truth values to the attributes of each world in W.

This model captures in a mathematical way the intuition that we can describe any situation of interest for ODP as a set of worlds with local states of affairs and a set of statements as to whether any world can evolve into any other. Something is possible in a world if there exist worlds reachable from it in which that thing would be true, and something is necessary in a world if that thing would be true in all worlds reachable from it. Clearly, this kind of structure is capably of defining concepts like obligation, which is the deontic version of necessity.

From such a model we can go a step further by dividing the model into a set of related frames <W, R> and a separate set of value assignments, or markings, V. This is not quite as flexible as the general related worlds model, but does separate the structural aspects of evolution from the constraints on variables, making it easier to reason about. Frames of this kind are called Kripke Frames.

It would be possible to add such a frame definition to the clauses of the Foundations on basic interpretation concepts (putting mathematical detail in an annex, if necessary, to avoid possible intimidation of some readers) and thereby establish a common framework for supporting the basic behavioural and deontic aspects, and possibly the broader conformance and evolutionary issues as well.

Let us consider some of the problems discussed earlier in this light. First, consider the lifecycle of interfaces and bindings. The ability of an object to be involved in a particular kind of action or interaction can be represented as a marking in any particular world, and the ability to create an interface or perform a binding interaction is a special case of this; the actual performance of the interaction would result in changes to the markings between a world and any of its successors that are reached by performing the interaction.

Now, the existence of an interface can be deduced in a particular world from the existence of paths of succession between that world and worlds that are marked as allowing the associated interactions without, in so doing, traversing any successor step corresponding to interface creation – that is, if there is no creation between the point considered and some point of use. Essentially, what this is saying is that the property describing the existence of an interface is shared by all the worlds that can reach a world in which an interaction at that interface takes place (interaction is possible) without passing between a pair of worlds whose linking relationship corresponds to the creation of the interface. Although this may sound trivial, it gives a basis

for determining precisely the circumstances under which an interface exists, and so interactions at it can happen. It also implies some consistency conditions, in that, if two worlds are reachable by multiple paths, either all of the paths must include a step representing interface creation or none of them can.

Similar conditions can be applied to the existence of bindings. In this way we can describe the way in which objects are characterised by their potential to engage in bindings or specific interactions with an unbounded set of candidate peer objects, without requiring detail of the behaviour involved in their doing so. Clearly, the set of worlds that are equivalent in having the property that a binding exists must be a subset of the possible worlds that are equivalent in having the property that all the interfaces to be bound exist. However there is not a subset relationship between the sets in which each of the individual interfaces exist.

In a similar approach, we can capture the way in which the potential behaviour of an object is modified by creation of a community and by the object filling a community-role; filling the role results in a modification of the set of accessible worlds. The performance of an action commits the objects involved to playing their action-roles, and the preconditions for so doing can be derived from the accessibility relations.

Before leaving this issue, it should be stressed that what is being discussed here is a sketch of the definition of semantics for the basic modelling and specification concepts. There is no intention that the average user of these concepts would be involved in such considerations, but the unified underpinnings would give the basis for reasoning about the consistency of the framework and the correctness of interpretations by tools in difficult or potentially ambiguous cases.

9. Gaps and omissions

Finally, there are a number of areas in which the current Foundations standard omits material on the grounds that it is self evident or sufficiently obvious to be taken as read. Experience has shown that it is worth making even apparently well-understood concepts explicit if they are to be used in a formal way.

One example is the omission of terms in common technical usage, such as *relationship* or *association*, definitions of which should be included on the basis of significant usage in ODP specification, even if they seem obvious. The Foundations should include generic definitions consistent with, but less detailed and restrictive than those in the ISO General Relationship Model [10]. These generic definitions would then need to be related to the specific relationships that are already defined in the Foundations, such as the *subtype* and *subclass* relationships.

Another example is the omission of ODP specific detail or logical consequences of the existing definitions. Here one might consider the explicit definition of the concept of an *inter-viewpoint correspondence*, which is not discussed when *viewpoint* is defined. It should, indeed, be obvious to everyone that a system is only defined if both the viewpoints and the correspondences between them are detailed to a sufficient level to unify the overall specification, but sets of viewpoint specifications are often published without clear statements of correspondences, and a more balanced set of definitions would help just a little to get this message across.

Finally, the clause on specification concepts should be reviewed to check whether residual restrictions on usage are necessary. Many of the concepts, such as type and class, can be applied to a wide range of basic concepts. Thus, for example, the concept *type (of an <X>)* can be applied to any <X>. An individual specification can then declare which kinds of terms in it can have types. At present, however, composition only applies to objects, and refinement only applies to specifications. Discussion will be needed to determine whether these two concepts should be *of an <X>* or applied indirectly as relating specification fragments; either way, there would be a minor incompatible change to one or other of the concepts.

10. Conclusions

It seems that, in general terms, the ODP Foundations document has stood the test of time quite well. There is a need for some clarification of the definitions of roles and policies, and for addition of clear definitions of a number of concepts originally assumed to be well known, such as relation.

The most pressing need is for more explicit definitions relating to evolution in time, relating both to system behaviour (for example the lifecycle of interfaces) and, more generally, of a set of ODP specifications to reflect changes of requirement and policy. This can best be done by having an explicit representation of the possible worlds a specification applies to; this would need to be referenced as part of the most fundamental support for modelling used to define the basic and specification concepts.

Although the concept definitions are quite stable, they are not always used to best advantage in related standards. In particular, because the ODP Architecture was developed in parallel with the Foundations, there are places where usage in Part 3 is incorrect or where rules could be expressed more precisely by making best use of the foundation concepts. Part 1 could be improved by making

the usage of the concepts from the Foundations more consistent and complete. It would also be possible to improve Part 4 by interpreting some of the concepts directly in terms of the semantic domain for the formal description techniques rather than writing them in the techniques directly. However, it seems unlikely that there is enough expert effort available to attempt so major a task on Part 4, and the refinement of Parts 2 and 3 of the reference model should be the primary target at present.

References

[1] ISO\IEC IS 10746-2, *Open Distributed Processing – Reference Model: Foundations*, 1996.

[2] ISO\IEC IS 10746-3, *Open Distributed Processing – Reference Model: Architecture*, 1996.

[3] P. F. Linington, *RM-ODP: The Architecture*, In K. Raymond and E. Armstrong, editors, *Open Distributed Processing: Experience with Distributed Environments*, pages 15-33. IFIP, Chapman and Hall, February 1995.

[4] ISO\IEC IS 8807, *Information processing systems -- Open Systems Interconnection – LOTOS: A formal description technique based on the temporal ordering of observational behaviour*, 1989.

[5] ISO/IEC IS 9074, *Information technology -- Open Systems Interconnection -- Estelle: A formal description technique based on an extended state transition model*, 1997.

[6] P. F. Linington and W. F. Frank, Specification and implementation in ODP, In J. Cordeiro and H. Kilov, editors, *Proceedings of the 1st Workshop on Open Distributed Processing: Enterprise, Computation, Knowledge, Engineering and Realisation*, pages 69-80, Setubal, Portugal, July 2001. ICEIS Press.

[7] P. F. Linington and S. Neal, *Using policies in the checking of business to business contracts*, In H. Lutfiyya, J. Moffat, and F. Garcia, editors, *Fourth IEEE International Workshop on Policies for Distributed Systems and Networks*, pages 207-218, Lake Como, Italy, June 2003. IEEE Computer Society.

[8] P. F. Linington, Z. Milosevic and K. Raymond, *Policies in Communities: Extending the ODP Enterprise Viewpoint*, In *Proceedings of 2nd International Workshop on Enterprise Distributed Object Computing (EDOC98)*, San Diego, USA, November 1998.

[9] G.E.Hughes and M. J. Cresswell, *A Companion to Modal Logic*, Methuen, London, 1984

[10] ISO/IEC IS 10165-7, *Information Technology – Open Systems Interconnection – Structure of management information: General relationship model*, 1996.