Richard O. Sinnott and Kenneth J. Turner. Applying the architectural semantics of ODP to develop a trader specification (pre-publication version). Computer Networks and ISDN Systems, 29(4):457-471, March 1997.

Applying the Architectural Semantics of ODP to Develop a Trader Specification

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**

October 2, 1995

Abstract

This paper provides an introduction to the role of Formal Description Techniques in the development of an architectural semantics for Open Distributed Processing (ODP). Following a brief overview of ODP and the Reference Model for ODP, an outline of the reasons for developing of an architectural semantics is given. The different approaches currently being taken for an ODP architectural semantics are presented. Finally, an ODP infrastructure component — the ODP Trader — is specified using the architectural semantics work.

1 Introduction

It is often the case that writing specifications proves to be difficult due to poor specification structure to begin with. Having a good [1] architecture upon which specifications can be based eases many of the difficulties involved in the actual writing of specifications. By a similar argument, specifications that are not based on a well-structured architecture tend to be not only difficult to write but also hard to understand, modify and extend. Having a good specification architecture is also very useful for problems that are not well defined, because it requires detailed consideration of the informal problem. Attempting to formalise "messy" problems directly may lead to "messy" specifications.

The advantages of developing a specification architecture are argued in detail in [17, 18] which focus on the architectural semantics for Open System Interconnection (OSI). This paper presents the development of an architectural semantics for Open Distributed Processing (ODP), focusing in particular on its application to develop an ODP common function — the ODP Trader.

The rest of the paper is structured as follows. Section 2 provides an overview of ODP and the reference model of ODP (RM-ODP). Section 3 focuses on Part 4 of the RM-ODP — the development of an architectural semantics for ODP. Section 4 introduces the ODP trader and shows how it may be specified in LOTOS using the architectural semantics work. Section

[1] The architecture must already have been applied to develop specifications which solved a similar problem successfully, *i.e.* they were clear, precise, easy to extend etc.

5 shows the relationship (or lack of one) between the formalisation of the basic concepts and those of the viewpoint language concept formalisations. Finally section 6 draws some conclusions on the advantages of developing the trader specification through the architectural semantics work.

2 Introduction to ODP and the RM-ODP

One property of a distributed system is that a user of the system is unaware of the differences in computers and operating systems in which their applications run. Such systems are inherently complex. Despite this distributed processing is growing rapidly, primarily due to the computer industry's ability to produce cheaper, more powerful computers. As a result of this growth, the need for the coordinated production of standards for distributed processing has been identified.

ODP is already a major effort between the International Organization for Standardization (ISO) and International Telecommunications Union (ITU-T) which will lead to significant product development in the coming years. The ODP work identifies and attempts to provide a framework for distributed systems. This has been set out in a Reference Model of ODP (RM-ODP). It defines an architecture through which distribution, interworking and portability can be achieved.

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 the RM-ODP is in effect 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 is based upon concepts derived from current distributed processing developments and, as far as possible, on the use of Formal Description Techniques (FDTs) to specify the architecture. The RM-ODP uses an object-oriented approach, where an object may be regarded as an identifiable encapsulated aspect of some real world entity. The advantages of this with regard to systems development generally are well documented in the literature, *e.g.* [11, 12, 13].

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

Part 1 - Overview and Guide to Use ([1]): contains an overview and guide to use of the RM-ODP.

Part 2 - Descriptive Model ([2]): contains the definition of concepts and gives the framework for descriptions of distributed systems. It also introduces the principles of conformance and the way it may be applied to ODP.

In effect Part 2 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 - Prescriptive Model ([3]): contains the specification of the required characteristics that qualify distributed system as open, *i.e.* constraints to which ODP systems

must conform. One particular part of Part 3 which is currently the focus of much of the recent architectural semantics work, is that of the viewpoint languages.

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. Each 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 as a whole 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.

These viewpoint languages will be considered in more detail in section 4.1, when they are used to develop a trader specification.

Part 4 - Architectural Semantics ([4, 5]) : 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. The architectural semantics is considered in detail in the next section.

3 An Architectural Semantics for ODP

An architectural semantics is the expression of a given architecture defined intuitively in a formal language, *e.g.* a formal description technique. The architecture considered here is the RM-ODP.

There are numerous advantages to be gained from "formalising" the architecture of the RM-ODP. Many of these advantages are argued in detail in [14, 15, 17]. The majority of advantages stem from the higher level of precision that is introduced through interpreting concepts formally, and the more detailed consideration of concepts required in doing so. This paper does not attempt to repeat an exposition of the advantages, but instead focuses on the practical advantages in applying the architectural semantics work to develop a specification.

An overview of the work on formalising the architecture of ODP is now given, and the approaches put forward so far are described.

3.1 The Scope of the Architectural Semantics Work

Ideally the architectural semantics work in Part 4 should formalise all of the RM-ODP. This, however, is not feasible due to both time and technical limitations. As a result the early

standardisation work focused on developing an architectural semantics for the basic modelling and specification concepts of Part 2. More recently, work has focused on formalising the viewpoint languages of the RM-ODP.

3.2 The FDTs Used in the Architectural Semantics Work

The FDTs currently being used in developing an architectural semantics for ODP are LOTOS [7], Z [8], SDL'92 [9] and ESTELLE [10]. These have been used as they satisfy the criteria for usage in ODP, namely that the FDT must be widely known or standardised. This raises questions over the use of new FDTs, *e.g.* Object-Z [20] or RAISE [19]. It may be the case that the introduction of new FDTs for developing an architectural semantics for ODP 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 to develop an architectural semantics for ODP, as the work is already well advanced in the standards making process.

3.3 Formalising the Basic Concepts

As stated in section 2, Part 2 of the RM-ODP introduces the basic concepts necessary for considering Part 3. These basic concepts may be of a fundamental modelling nature, *e.g.* object, action, interface, etc; specific to specification languages, *e.g.* type, class, behaviour compatibility, etc; or of an nature related specifically to issues of distribution, *e.g.* organizational concepts, naming concepts, etc.

Previous work in developing an architectural semantics for ODP focused on the modelling and specification concepts. However, recent work [21, 22] has also focused on those concepts related to issues of distribution, *e.g.* policy, domain, etc.

The formalisation of all of these concepts offer numerous advantages. For example, formalising the basic modelling concepts and concepts related to issues of distribution gives a precise understanding of the architectural concepts used in the RM-ODP. This then enables a solid architectural basis for writing specifications of ODP systems (and standards) to be achieved. Formalising the specification language specific concepts enables a comparison to be made of the suitability of FDTs to modelling concepts that arise in the RM-ODP. This then aids in the selection of the most apt FDT for the specification task at hand. Through these formalisations a uniform and consistent comparison of formal descriptions of standards written in different FDTs is possible, with the formal interpretation acting as a bridge between the ODP and FDT semantic models.

The approach taken towards formalisation in Part 2 is through interpretation: description is used to show how the architectural concepts of the RM-ODP may best be represented in a given formal method. There is no high level of prescriptivity in this approach; rather, a given concept representation might have many different formalisations in any one FDT. The following list summarises some of the basic concept definitions of Part 2 from the RM-ODP.

action: *something which happens; all actions are associated with objects and can be interactions or internal actions.*

- **interaction:** *an action that occurs with the environment of the object.*
- **internal action:** *an action that occurs without the environment of the object.*

environment: *the part of the model that is not part of the object.*

interaction point: *a location at which a set of interfaces exist.*

behaviour: *a collection of actions with a set of constraints on their occurrence.*

location in space: *an interval in space at which an action can occur.*

interface: *a subset of the interactions of an object.*

object: *a model of an entity characterised by its state and its behaviour; all objects are unique and encapsulated, i.e. state changes can only occur via internal actions or through interactions with the environment.*

composition (of objects): *a combination of two or more objects yielding a new object; the characteristics of the new object are determined by the objects being combined and the way they are combined.*

obligation: *A prescription that a particular behaviour is required.*

permission: *A prescription that a particular behaviour is allowed to occur.*

prohibition: *A prescription that a particular behaviour must not occur.*

The following list highlights briefly the way in which the above concepts are represented in the FDT LOTOS.

action: *a LOTOS event;*

- **interaction:** *an event that occurs via synchronisation on a common gate.*
- **internal action:** *an event at a hidden gate or the internal event symbol,* **i**.

environment: *all of the behaviour expressions that may synchronise with the object* [2].

interaction point: *a gate with a possibly empty list of associated values.*

behaviour (of an object): *determined by the behaviour expression associated with the object template.*

location in space: *a gate for interactions; hidden for internal actions.*

interface: *an abstraction of the behaviour of an object obtained by hiding* [3] *all observable actions that do not constitute part of the interface under consideration.*

object: *an instantiation of a LOTOS process definition which can be uniquely referenced.*

composition: *an object described through the application of one or more of the LOTOS composition operators, i.e.* |||, ||, |[]|, [], [> *and* >>.

[2] This may include behaviour not specified within the specification if the specification is parameterised with gates, *e.g.* through a simulator.

[3] Using the LOTOS **hide ... in** construction to make internal, gates (and hence the associated actions) not making up part of the interface.

permission: *an event offer which must occur before a particular behaviour of interest can occur. This is an inherent feature of the temporal ordering of events associated with every LOTOS specification. The evaluation of a guard to true, or the satisfaction of a selection predicate, may also be used to model a permission.*

obligation: *In LOTOS, behaviour is modelled directly through writing behaviour expressions. This behaviour either occurs or does not occur depending upon the interactions with the environment. However, if the specification is written so that at any one time only a single event can occur, then this event is obliged to occur (at some time).*

prohibition: *represented using the temporal ordering of events* [4]*. A prohibited behaviour may be placed after a permission event so the behaviour is prohibited until the permission event occurs. Alternatively a guard or selection predicate can be used to model a prohibition, i.e. the behaviour cannot occur until the guard or selection predicate is satisfied.*

As can be seen, these definitions and their interpretations in LOTOS offer guidance to the specifier without being overly prescriptive. For example, the definition of an object states nothing about the way in which the unique identity of an object can be established. This might be when the process is instantiated, as some value parameter which is used in all object interactions. Alternatively the identity might be some global value used directly in the behaviour expression associated with the object template (process definition). Another possible choice is for the object to have some initial behaviour 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.

3.4 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 into stream, operational and signal interfaces. An example of this specialisation relation is given in section 5. 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. One approach is based on interpretation, just as for the formalisation of the basic modelling and specification concepts. Another approach is based on providing specification templates. The third approach gives a direct formal semantics in mathematics for a subset of the computational language. These approaches have their own advantages and disadvantages, many of which are highlighted in [14]. For example, the direct formal approach captures the genericity of the computational language, but might be seen as overly mathematical. The interpretation approach brings a higher level of understanding of all concepts in all viewpoints, through requiring a detailed consideration of the semantics associated with the textual definitions. This

[4] Actually prohibited behaviours are more often simply not specified in LOTOS.

includes identifying terms which are not formally defined in the RM-ODP. For example, the term structure was used when considering objects in the enterprise viewpoint. Whilst this term had an intuitive meaning, it lacked precision. Instead, this term was replaced by the phrase, "the result of a composition", which had a precisely defined meaning, *i.e.* defined in the RM-ODP.

Whilst an approach based on interpretation offers a means to check in detail the textual definitions, it does not offer a means to build ODP compliant specification directly. The template based approach is an attempt at achieving this. A template based approach enables ODP concepts to be structured in specifications thereby easing the specifier's task. However templates are not possible for all viewpoint languages or all concepts.

As identified in [14], the ideal approach would be to develop specification fragments (behaviour templates) for all concepts in all viewpoints. These could then be used to develop ODP compliant specifications and to ensure consistency between viewpoints. Two or more viewpoints must not offer behaviours (as given in the specification templates) which contradict one another.

The reality is somewhat different, however. Specification templates cannot be given for all viewpoint concepts due to their generic nature. For example, the information viewpoint gives very few prescriptions on legal behaviours of information objects. The approach taken there is to model behaviour through dynamic schemas (see section 4.3). The only prescription that is applied to dynamic schema occurrence is that they must satisfy any invariant schemas (see section 4.3) that might be given for this viewpoint. Hence it is not possible to be prescriptive at all when dealing with behaviour from the information viewpoint.

Other viewpoints, *e.g.* the computational viewpoint, are more prescriptive. However, it is not possible to be prescriptive enough to provide specification templates for all concepts. For example, interface templates require consideration of behavioural specifications and environmental constraints as well as interface signatures. However, the computational viewpoint language can only prescribe the form of the signature. This is not surprising as the computational viewpoint attempts to provide a generic framework whereby interworking of interfaces can be achieved. Thus it cannot prescribe a single behavioural specification and the language is necessarily generic.

As a result of this, the development of an architectural semantics is constrained to some extent in what can be achieved through prescription. Thus the ideal development of an architectural semantics as identified in [17, 18] of a specification library filled with formalisations of all of the architectural concepts seems very attractive, but is not realistic for ODP due to the necessary lack of prescription for some concepts and some viewpoints in the RM-ODP.

With regard to this paper and the development of a trader specification, an approach based on the interpretation of ODP concepts in LOTOS is employed. Whilst not as immediately useful for building specifications, this approach does generate a higher level of understanding of concepts, including how they may best be modelled. Hence it does simplify the specification development process.

4 Specifying the ODP Trader in LOTOS using the Architectural Semantics

4.1 Introduction to the ODP Trader

A trader [6] is an object that performs trading, which is an ODP common function. A trader therefore must therefore fit into the architectural framework of ODP.

ODP aims to provide distribution-transparent utilisation of services over heterogeneous environments. In order to use services, users need to be aware of potential service providers and to be capable of accessing them. Since sites and applications in distributed systems are likely to change frequently, it is advantageous to allow late binding between service users and providers. If this is to be supported, a component must be able to find appropriate service providers dynamically. The ODP trading function provides this dynamic selection of service providers at run time. The interactions that are necessary to achieve this are shown in figure 1.

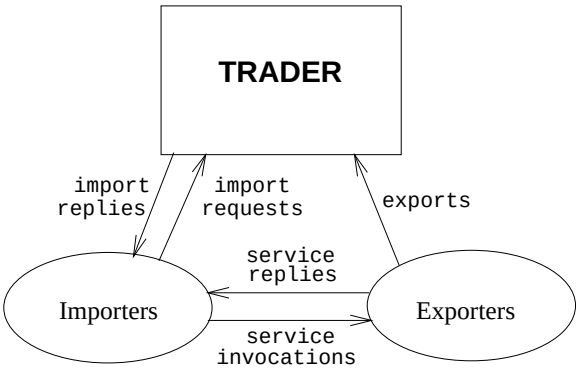


Figure 1: A Trader and its Users

Thus a trader accepts a *service offer* from an *exporter* wishing to advertise its services. A service offer contains the characteristics of a service that a service provider is willing to provide[5]. The trader then stores these service offers for use by importers.

A trader accepts service requests from importers of services. These represent requirements on available services that a trader may or may not have access to. Upon receipt of a request from an importer, the trader searches its store of service offers to see if any offers match the importers service request. If any matching offers are found they are returned to the importer, which may then interact directly with the service.

It should be pointed out that a trader might not itself have a service offer that matches an importer's request. In this case, a trader can check whether any other traders it "knows" can satisfy the import request. This is known as federated trading. The establishment of trading links between traders is not addressed in this paper, although an architectural approach can be applied to develop a trading federation.

[5]It should be noted that the service provider need not necessarily be the exporter. Similarly the service user need not necessarily be the importer.

4.2 A Trader from the Enterprise Viewpoint

In the enterprise viewpoint, the objectives, functional requirements and policy statements that govern the activities of a trading enterprise are identified.

From an architectural semantics viewpoint there is little that can be said directly regarding the scope and purpose of the trader enterprise viewpoint. That is, FDTs may not easily allow for such general statements to be made about a given system. These notions are normally given in the informal commentary associated with a specification.

Formal techniques can be applied to develop explicit policy rules, however. A policy is defined in [2] as "a set of rules related to a particular purpose". These rules may be: obligations; prohibitions; or permissions. The definitions of the concepts and their representation in LOTOS may be found in section 3.3.

As a result of the modelling consequences of these concepts in LOTOS, it can be seen that little can be written from an architectural semantics perspective regarding a generic trader. FDTs in general and LOTOS in particular allow for predicates on behaviour to be written. However, without explicit statements being made, nothing definite can be written from the enterprise viewpoint. Hence in order to consider the enterprise viewpoint as well as the other viewpoints in this paper, some explicit policy statements will be given.

Numerous policy statements can be considered when trading, *e.g.* security policy, service offer acceptance policy and resource consumption policy. Examples of these policies are represented in the statements:

> Only importers and exporters with the highest security level may trade services with service type "Secret Service". This service type will always have the service property, "very trustworthy" and service offer property "unlimited expiry date". There will be an unlimited resource consumption policy to satisfy all requests.

Thus from these statements a security policy rule is prescribed limiting user access to certain service offers a trader may have. A service offer acceptance policy is prescribed which ensures that certain service offers have associated with them service properties and service offer properties. Finally, the resource consumption policy is explicitly given.

As stated, a trader offers importers and exporters a means to select and advertise services. It achieves this through import and export operations [6]. These operations have associated with them various parameters, *e.g.* the client identity. See section 4.4 for elaboration of these parameters.

An outline of the structure of a trader offering only import and export operations with some of the policy rules given previously is as follows:

```
process trader[imp,exp ...](formal parameters):noexit:=
  imp ?user_id: ID ? .... other parameters;
  ( [ security_level(user_id) eq highest ]=>
(* other behaviour with further checks on service type, etc *) (*1*)
    []
    [ security_level(user_id) ne highest ]=>
     imp !user_id !error;
     trader[imp,exp ...](formal parameters))
```

[6] A trader also offers several other operations, *e.g.* withdraw a service, obtain the details about a service, etc. but these are not considered here.

```
    []
    exp ?user_id: ID ? .... other parameters;
    ( [ security_level(user_id) eq highest ]=>
  (* other behaviour with further checks on service type, etc *) (*2*)
       []
       [ security_level(user_id) ne highest ]=>
       exp !user_id !error;
       trader[imp,exp ...](formal parameters))
endproc (* trader *)
```

This specification fragment satisfies some of the policy rules given previously. For example, guards are used [7] to represent prohibitions. If a guard is rewritten to a *true* value, then this may be seen as a permission. Thus following synchronisation at the gate *imp* say, an importer is either granted access (permitted) to the behaviour represented by (* 1 *) which was previously prohibited, or an error message is returned and this behaviour is still prohibited to that user. Following this, the behaviour represented by (* 1 *) becomes obligatory as it is the only behaviour that is possible in the specification; this is due to the nature of the choice operator. The use of this operator satisfies the resource consumption policy, *i.e.* no other behaviour can occur until this request has been satisfied [8].

Before going into detail about how the other policy rules can be specified, it is necessary to consider in more detail the information items in the policy statements and how they can be represented in LOTOS. Hence it is necessary to consider the information viewpoint of the trader.

4.3 Trader from the Information Viewpoint

The information viewpoint identifies, the information elements, the information structures and the requirements for information handling for a trader. Information is defined in [2] 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, [3] uses the notion of *invariant, dynamic* and *static schema*. These represent respectively, the properties of information that are independent of behaviour, dependent upon behaviour and give the state and structure of information at some particular time.

In LOTOS, exchange of data can only occur between behaviour expressions. It is not possible in LOTOS to exchange process definitions as data, hence only the ACT ONE part of LOTOS has been considered here for modelling information. Thus a given information item will be represented by a value in an ACT ONE sort with associated operations and equations.

Generally, there are many information items that need to be considered when trading. However, from the policy rules given in section 4.2 only service offers and the trading object itself will be considered.

[7] For brevity sake, the operation *security level* and sort *highest* have not been given. Their meaning here is expected to be obvious. In effect their modelling is dependent upon information and computational viewpoint considerations which are discussed in the following two sections.

[8] This is an unlikely scenario, but serves to show how such policy rules can be established in LOTOS.

A service offer describes an advertised service which is provided at a computational interface. It consists of a service type — which is itself represented by an interface type, a set of service properties, a set of service offer properties and an identifier at which the interface can be found. In effect service properties and service offer properties represent partial specifications of an interface giving more information than that conveyed by its interface signature.

The following LOTOS fragment shows how a service offer may be represented:

```
type service_offer is service, service_offer_properties, id_type
     sorts service_offer
     opns make_offer    : service, sop_set, id_sort -> service_offer
          get_service   : service_offer -> service
          get_sops      : service_offer -> sop_set
          get_id        : service_offer -> id_sort
     eqns forall st: service, sops: sop_set, id: id_sort
          ofsort service
            get_service(make_offer(st,sops,id)) = st;
          ofsort sop_set
            get_sops(make_offer(st,sops,id)) = sops;
          ofsort id_sort
            get_id(make_offer(st,sops,id)) = id;
endtype (* service_offer *)
```

The operations given here other than *make offer* will be applied to verify the example policy rules relating to service offers. The type *service* may be represented by the following LOTOS fragment:

```
type service is interface_type, service_properties
     sorts service
     opns make_service  : interface, sp_set -> service
          _IsSubtype_   : service, service -> Bool
          get_interface : service -> interface
          get_sps       : service -> sp_set
     eqns forall st1, st2: service, int: interface, sps: sp_set
          ofsort Bool
            st1 IsSubtype st2 = get_interface(st1) IsSubtype get_interface(st2);
          ofsort interface
            get_interface(make_service(int,sps)) = int;
          ofsort sp_set
            get_sps(make_service(int,sps)) = sps;
endtype (* service_type *)
```

It should be noted that the sort *interface* is generated through an approach whereby signature subtyping can be established. This approach is documented in [16].

The types *service properties* and *service offer properties* may be represented by the following LOTOS fragments:

```
type service_properties is Set
     actualizedby sp using
```

```
type service_offer_properties is Set
       actualizedby sop using
```

```
     sortnames sp_set for Set                    sortnames sop_set for Set
               sp for Element                              sop for Element
               Bool for FBool                              Bool for FBool
endtype (* sp_set *)                        endtype (* sop_set *)
```

Here the sort *Set* is taken from the LOTOS standard library. For brevity sake, the sorts *sp, sop* and *id_sort* are not given. It is sufficient to say that these are represented by sets of values which can be distinguished from one another.

Having decided how service offers (and other necessary information items used to model service offers) are represented from an information viewpoint, the next consideration is how to represent a trader object from an information viewpoint. A trader with the functionality of import and export is represented in the following LOTOS fragment:

```
type trader_object is service_offers, match_constraint
     sorts trader_object
     opns make_trader: -> trader_object
          export: service_offer, trader_object -> trader_object
          import: trader_object, service, mc ->  so_set
     eqns forall to: trader_object, some_mc: mc,
                  so: service_offer, st: service
          ofsort so_set
           import(make_trader,st,some_mc) = {};
           ((not(get_service(so) IsSubtype st)
            or (not(get_mc_sps(some_mc) IsSubsetOf get_sps(get_service(so))))
            or (not(get_mc_sops(some_mc) IsSubsetOf get_sops(so)))))=>
            import(export(so,to),st,some_mc) = import(to,st,some_mc);
           ((get_service(so) IsSubtype st)
            and (get_mc_sps(some_mc) IsSubsetOf get_sps(get_service(so)))
            and (get_mc_sops(some_mc) IsSubsetOf get_sops(so))) =>
            import(export(so,to),st,some_mc) = Insert(so,import(to,st,some_mc));
endtype (* trader_object *)
```

The equations here state that importing from a newly created trader returns an empty set of service offers. Importing from a trader with previously exported service offers requires a check on the service type, service properties and service offer properties associated with the service offer. This results in either the service offer matching the constraints and hence being added to a set to be returned to the user, or the offer being ignored and other offers checked.

The type *service_offers* is used to model sets of service offers. It has the following structure:

```
type service_offers is Set actualizedby service_offer
     using sortnames so_set for Set
                    service_offer for Element
                    Bool for FBool
endtype (* service_offers *)
```

Type *match_constraint* represents the matching constraints; these are predicates on service properties and service offer properties which act as a filter for service offers. This type has the following structure:

```
type match_constraint is service_properties, service_offer_properties
     sorts mc
     opns make_mc         : sp_set, sop_set -> mc
          get_mc_sps      : mc -> sp_set
          get_mc_sops     : mc -> sop_set
     eqns forall sps: sp_set, sops: sop_set
            ofsort sp_set
              get_mc_sps(make_mc(sps,sops)) = sps;
            ofsort sop_set
              get_mc_sops(make_mc(sps,sops)) = sops;
endtype (* match_constraint *)
```

Thus when matching constraints are applied to satisfy the policy rules given previously in section 4.2, the *sp_set* and *sop_set* associated with importing an acceptable service offer would contain the service property *very_trustworthy* and service offer property *unlimited_expiry_date* respectively.

It should be pointed out that there are also likely to be scope criteria that have to be considered, *i.e.* which traders have to be searched if the request cannot be satisfied locally, and selection preferences, *i.e.* order the service offers returned according to some criteria, such as in order of cheapest cost. For simplicity sake, these have been ignored here.

The type definitions supplied so far gives the static and invariant schemas which determine the legal states and structures of the information elements *service_offer* and *trader_object*, *e.g.* one invariant schema is that a *trader_object* contains no service offers initially, hence an import request returns an empty set of service offers. It is also necessary to consider the behavioural aspects that are associated with these information elements. Thus dynamic schemas need to be considered. To deal with dynamic schemas, the computational viewpoint needs to be considered.

4.4 Trader from the Computational Viewpoint

4.4.1 Specification

As identified in [5], dynamic schemas are represented in value expressions that are present in the process algebra, *i.e.* the process algebra represents a behavioural framework on which information (sorts) can be hung and manipulated.

It is argued in [5] that the computational language is best represented through the process algebra part of a LOTOS specification. The computational language deals with behaviours and the interfaces that exist between them to enable interworking.

From a trading perspective, only a subset of the interfaces identified in the computational viewpoint are considered: the operational interfaces. Operational interfaces consist of interactions between binding objects and computational objects, where a binding object may be regarded as an object through which interactions between other computational objects can occur. They are used, amongst other things, to ensure quality of service. Thus it is likely that a given importer or exporter will interact with a trading object via a binding object. These interactions may be either announcements or interrogations. Announcements and interrogations are modelled as sequences of signals, where a signal may be regarded as an atomic interaction. A signal signature is an action template for a signature consisting of: a name for

the signal; the number names and types of parameters for the signal; and an indication of causality.

An announcement consists of an invocation, where an invocation is a sequence of interactions comprising two signals. The first called *invocation submit* occurs between a client object and binding object and the second called *invocation deliver* occurs between the same binding object and a server object.

An interrogation consists of an invocation followed by a termination, where a termination is a sequence of interactions comprising two signals. The first called *termination submit* occurs between a server object and binding object and the second called *termination deliver* occurs between the same binding object and a client object.

The specialisation relationship between the formalisation of the basic concepts, as given in section 3.3, and those of the computational viewpoint may be seen clearly here. A higher level of prescription is required when modelling the basic concepts of interface and objects. Instead of dealing with generic interfaces and objects, the computational viewpoint is more specific with regard to the form that these concepts should take. A further illustration of the relationship between Part 2 and the computational viewpoint language may be found in section 5.

From a trading point of view, interrogations are predominantly used. That is, importers and exporters *request* a trading service from the trader, which in turn gives an appropriate *response* depending upon whether it can satisfy their request. Thus the notion of announcements is not as applicable for trading.

The structure of signals suitable for importing from a trader and exporting to a trader are given by the following LOTOS fragments:

```
imp ?user_id: id_sort
    ?serv_type_imp: service
    ?ip: import_policy
    ?some_mc: match_constraint
    ?sel_pref: selection_preference
    ?serv_prop_of_int: sp_set
    ?serv_off_prop_of_int: sop_set
    !inv_deliver;
```

```
exp ?user_id: id_sort
    ?serv_type_exp: service
    ?sp_values: sp_set
    ?sop_values: sop_set
    ?serv_id: id_sort
    ?serv_off_eval_int: id_sort
    !inv_deliver;
```

Here the gate name *imp*/*exp* acts as the signal name. The value *inv deliver* indicates the causality of the signal. *inv deliver* is used here as opposed to *inv submit* as it is likely that the initial request (*inv submit*) from an importer or exporter will be received by a binding object which in turn passes (*inv deliver*) the request to the trader. It should also be noted that in LOTOS this is only an informal notion of causality. It is not the case that one event offer "causes" another event offer. Event offers which synchronise on a single event do so simultaneously and instantaneously. Thus attaching a causality label can only be done informally. This is often achieved through interpreting the value-passing form of synchronisation in LOTOS as a client/server role or, as done here, through a parameter which indicates informally a client or server role.

The other parameters are those specifically identified as mandatory in the trader standard [6]. For the import operation: *user id* is used to distinguish between different users; *serv_type_imp* describes the type of service required by the importer; *ip* represents an importer's policy requirements which can restrict the scope of a search; *some mc* represents the importer's matching criteria, *i.e.* the constraints on service properties and offer properties for

an acceptable service offer; *sel_pref* represents a selection preference which orders matching service offers by some means, *e.g.* cheapest; *serv_prop_of_int* and *serv_off_prop_of_int* represent the service properties and service offer properties of interest to an importer, *i.e.* the values of these will be returned to an importer along with the selected offer. For the export operation: *serv_type_exp* describes the type of service on offer by the exporter, with *sp_values* and *sop_values* representing the values of service properties and offer properties associated with this service; *serv_id* represents an identifier for an interface at which this service can be found; and *serv_off_eval_int* represents an identifier for an interface at which additional properties (likely to change frequently) for a service can be established.

Due to the complexity involved in dealing with a fully operational trader, only a subset of these parameters have been dealt with, *i.e.* those that are required to satisfy the policy rules given in section 4.2. Thus import policy constraints, selection preferences, service offer evaluation interfaces, values of service properties and service offer properties of interest which are returned with acceptable offers have not been considered in depth here. For simplicity, only those properties necessary for matching acceptable services offers have been considered.

From these considerations of computational parameters, the basic outline of a trader given in section 4.2 needs to be extended. This extension should incorporate both the information viewpoint considerations, *e.g.* the dynamic schemas which manipulate the information items, and the computational viewpoint considerations, *e.g.* the operational interfaces. An outline of such a trader extension is given through the following LOTOS fragment.

```
process trader[imp,exp](trader_info_object: trader_object):noexit:=
   imp ?user_id: id_sort ?serv_type_imp: service
       ?ip: import_policy ?some_mc: match_constraint
       ?sel_pref: selection_preference ?serv_prop_of_int: sp_set
       ?serv_off_prop_of_int: sop_set !inv_deliver;
   ( [ (security_level(user_id) eq highest) and
      (serv_type_imp eq Secret_Service) ] ->
      imp !user_id
          !import(trader_info_object,serv_type_imp,some_mc)
          !term_submit;
      trader[imp,exp](trader_info_object)
    []
    [ (security_level(user_id) ne highest) and
      (serv_type_imp eq Secret_Service) ] ->
      imp !user_id !error !term_submit;
      trader[imp,exp](trader_info_object)
    []
    [ (serv_type_imp ne Secret_Service) ] ->
      imp !user_id
          !import(trader_info_object,serv_type_imp,some_mc)
          !term_submit;
      trader[imp,exp](trader_info_object))
   []
   exp ?user_id: id_sort ?serv_type_exp: service
       ?sp_values: sp_set ?sop_values: sop_set
       ?serv_id: id_sort ?serv_off_eval_int: id_sort !inv_deliver;
```

```
( [ (security_level(user_id) eq highest) and
    (serv_type_exp eq Secret_Service) and
    (sp_values eq Insert(very_trustworthy,{})) and
    (sop_values eq Insert(unlimited_expiry_date,{}))]->
     exp !user_id ?new_id: id_sort !term_submit;
     trader[imp,exp](export(make_offer(serv_type_exp,sop_values,serv_id),
                            trader_info_object))
  []
   [ (serv_type_exp eq Secret_Service) and
     ((security_level(user_id) ne highest) or
     (sp_values ne Insert(very_trustworthy,{})) or
     (sop_values ne Insert(unlimited_expiry_date,{}))) ]->
     exp !user_id !error !term_submit;
     trader[imp,exp](trader_info_object)
  []
   [ (serv_type_exp ne Secret_Service) ] ->
     exp !user_id ?new_id: id_sort !term_submit;
     trader[imp,exp](export(make_offer(serv_type_exp,sop_values,serv_id),
                            trader_info_object)))
endproc (* trader *)
```

4.4.2 Discussion

Some general comments should be noted about this specification outline:

Service Offer Identity: When an export offer is accepted, an identifier is returned to the exporter. This should be unique within a trading system. For simplicity, the explicit generation of this unique identifier has not been made here.

Computational Errors: Different computational errors have not been catalogued here. Rather, the approach has been to return the response *error* with terminations, without exposing the cause of the failure. This has been done for simplicity.

From this specification outline, various points specific to the viewpoint languages should be noted:

Enterprise Viewpoint Considerations: The enterprise rules are satisfied through a combination of guards and the LOTOS choice composition operator. For example, the restriction of only certain users (importers/exporters) to trade the service type *Secret_Service* has been achieved through guards. Similarly the policy rule that the service type *Secret_Service* should have the service property *very trustworthy* and service offer property *unlimited expiry date* has been satisfied in the guards.

Information Viewpoint Considerations: The static and invariant schemas of the information viewpoint are given through the type definitions of a given specification, whilst the dynamic schemas are represented through the value expressions present in the process algebra. As an example, consider the expression given in the recursive call of the trader: *export(make_offer(serv_type_exp,sop_values,serv_id), trader_info_object))*. Here, a service offer details (service type, service offer properties and identifier of an interface

at which the service can be found) are being registered with the trader information object, *i.e.* they are being exported to the trader *trader_info_object*. It is this expression that is dynamically manipulating (processing) the information items in the system. This dynamic manipulation must satisfy the ACT ONE part of a specification. Hence the invariant and static schemas must always be satisfied implicitly [9]. Following this recursive call, the static schema associated with this trader information object has been modified, *i.e.* the state of the information object has been modified through the acceptance of this service offer.

Computational Viewpoint Considerations: The specification outline above shows how operational interactions may be represented. It should be noted that the causality identifiers *inv_deliver* and *term_submit* are used for the reasons put forward in section 4.4. This is not the only way in which computational signals may be represented in LOTOS, however. It seems likely that if a template-based approach is to be used to develop specifications (as opposed to an interpretation-based approach), then much of the action denotation associated with operational event offers can be replaced with a parameter list. For example, the import invocation signal given above could be replaced by the event offer *imp !parameter_list !inv_deliver*, where *parameter_list* represents a list of those parameters identified as mandatory. Through this approach, different trading systems could be assured of interactions which did not deadlock immediately. Whether these interactions were meaningful would then depend upon the parameters associated with a given parameter list.

4.5 Trader from the Engineering and Technology Viewpoints

From the engineering point of view, the main focus is on the realisation (implementation) of computational objects. As a result, it could be argued that this is outside the scope of specification. Thus this viewpoint has not been addressed in the architectural semantics work will not be discussed in detail here.

It is unlikely that very much should be said from an architectural semantics perspective regarding the technology viewpoint, so the technology viewpoint of a trader will not be considered here.

4.6 Combining the Viewpoint Specifications

The approach given in this paper is not the only one that could be taken when developing specifications using the viewpoint languages. Another alternative might be to specify systems from the different viewpoints independently and then use a constraint-oriented style of specification to combine them. The problem with this approach though, is that the viewpoints are not independent of one another. Viewpoints are inherently linked to one another, *e.g.* a single behaviour can be used to satisfy a policy rule, to satisfy an information processing constraint, and to represent a computational activity. Hence the necessity is to consider all of the viewpoints simultaneously.

Ideally there should be a mechanism whereby a proof of consistency between different viewpoints could be achieved. In this paper, this has been largely avoided. The simple rules

[9] This is a strange (but correct) interpretation, as the ACT ONE specification is not a "watch-dog" looking over illegal manipulations; it defines the manipulations and so cannot be "wrong".

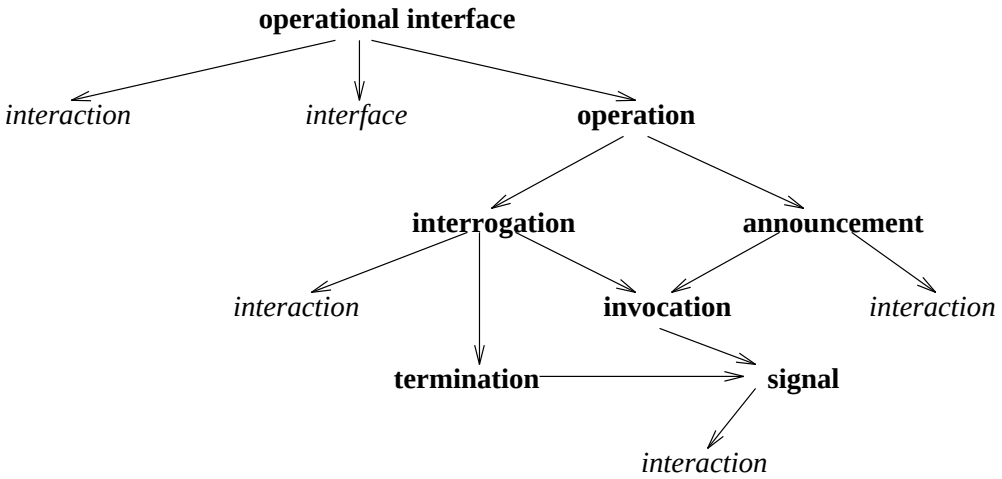
put forward in the enterprise viewpoint were realised through an initial specification structure which was "filled out" using the computational and information viewpoints.

There are several advantages of the approach taken to modelling the information viewpoint predominantly through data types and the computational viewpoint in the process algebra has several advantages. For example, the notion of behaviour as an information processing activity (dynamic schema) and computational activity is catered for directly. The process algebra acts as a behavioural framework on which information can be hung and manipulated, offering an approach whereby information processing and computational activities can be accommodated.

5 Relationship of Viewpoint Languages to Basic Concepts

As identified in [17, 18] developing an architectural semantics should be based on indirect mappings of architectural concepts to their formalisations. Thus rather than formalise each concept independently, a hierarchical relationship should be established — where possible — between the concepts. This hierarchical relationship in many cases already exists between the concepts in the RM-ODP.

Having established in section 3.4 that the viewpoint languages are a specialisation of the basic concepts. It might be expected that a hierarchical relationship should exist between the formalisation of the basic concepts and their specialised formalisations in the viewpoint language. Currently, however, this is not always the case. When dealing with the computational viewpoint, it is the case that the concepts contained there are specialisations of the basic concepts. For example, an operational interface is an interface whose interactions are restricted to operations, *i.e.* interrogations or announcements which have given signatures. One hierarchy of concepts both from the RM-ODP definitions and their formalisations might be:



Here the relationship between the computational concepts and their use of the basic concepts is clearly illustrated, with the computational concepts in bold text and basic concepts in italics. This hierarchical structure is reflected in both the RM-ODP concept definitions and their formalisation in LOTOS.

When dealing with the information viewpoint language however. There is no clear relationship between the formalisation of the information viewpoint concepts in LOTOS and

the LOTOS formalisation of the basic concepts of Part 2. This relationship does, however, exist in the text of the RM-ODP. For example, the Part 2 terms: behaviour, state, object, etc. are used in the information viewpoint language. The reason for this discrepancy is that the current formalisation of the basic concepts in LOTOS contained within Part 2 takes a predominantly process algebra dominated approach to modelling the concepts, *e.g.* see the definition of an object in section 3.3. This is not the only way in which objects may be modelled in LOTOS, however. The data typing part of LOTOS might also be used to model many of the basic concepts. This is not surprising, as identified in [17], developing an architectural semantics requires choices to be made in choosing the best modelling approach. Whilst the process algebra approach is well suited to modelling most of the concepts of the RM-ODP, it is not as directly useful when information modelling is considered. The consequences of this are that further work is required to extend the current formalisation of the basic concepts in LOTOS, to give alternative modelling approaches, *e.g.* based on ACT ONE.

6 Conclusions

This paper has demonstrated through practical application how an architectural approach can be used to develop specifications. Following an outline of ODP and the RM-ODP, an overview of the work on formalising the architecture of ODP has been given. This included the basic modelling concepts and how they can be represented in LOTOS, as well as a discussion on the formalisation of the ODP viewpoint languages and the different approaches taken.

LOTOS was then applied through an approach based on interpretation to develop a trader specification. This approach highlighted the interrelation between the viewpoint languages and how they need to be considered together when developing specifications. The paper has exposed the myth that viewpoint languages can be applied in a "top-down" manner. This is especially true when behaviour is considered which must satisfy enterprise policy rules as well as information and computational viewpoint constraints. Thus the abstraction achieved through a viewpoint language often becomes blurred when behaviour is considered, *i.e.* the viewpoint languages have to be considered at the same time when building specifications.

The limitations in attempting to specify anything without having enterprise policy rules clearly identified has been apparent. These rules represent predicates on the allowed behaviour of the system. Hence without explicit rules being made, a given specification would be overly generic. Thus simple statements were made in the enterprise viewpoint which were then realised through an initial specification structure. This structure was then expanded upon through consideration of the information and computational viewpoints.

Issues of consistency are likely to play a large part in developing specifications of ODP systems using the architectural semantics work. In this paper, a simplistic approach based on satisfying basic rules was taken to ensure consistency between the enterprise policy statements and the information and computational viewpoint considerations. It is likely that a more rigorous approach is required, however, which can be adapted to a much wider selection of enterprise policy rules.

The checking of consistency between the information and computational viewpoints is alleviated to a great extent through this approach. That is, the information viewpoint may be specified predominantly separately from the computational viewpoint. It is only in the area of information processing (through dynamic schemas) that the two viewpoints overlap. It is always the case that the computational viewpoint must be consistent with the information

viewpoint, *i.e.* computational behaviour cannot occur which will manipulate information items in such a way that they are being violated. The reverse of this might not be the case, however. For example, a computational operation to export a service offer may fail if the trader object has reached a maximum level of offers stored. Computationally, this offer might be acceptable (if all of the parameters are "correct"), but from the information viewpoint it would fail; rewriting the expression denoting the export operation would result in the previous trader object. The consequences of this are that computational interfaces should take into account all information and enterprise viewpoint considerations, *e.g.* have a guard restricting the export operation if the trader information object is full.

The work on formalising the architecture of ODP is not yet finished. The expected dates for standardisation are: October 1996 for the formalisation of the basic modelling and specification concepts; and June 1997 for the formalisation of the viewpoint languages. As identified in section 5, further work is required in formalising the basic concepts in LOTOS through a data type based approach, as opposed to a predominantly process algebra based approach. Thus considerable work is still required. Also, since the work is based on Parts 2 and 3 of the RM-ODP, the architectural semantics will never be finished until these documents are in a very stable condition. Currently, however, this is not the case.

An email discussion group has been set up which debate many of the ideas involved in developing an architectural semantics for ODP. The interested reader may contact the first named author for more information.

7 Acknowledgements

The first author is supported by a joint grant from the United Kingdom Engineering and Physical Sciences Research Council and the Department of Trade and Industry, as part of the FORMOSA (The Formalisation of the ODP Systems Architecture) project. This is a joint project between the University of Stirling and British Telecommunications (BT). The authors would thus like to thank the chief collaborators in this project from BT, namely Steve Rudkin and Pete Young.

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, August 1994.

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

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

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

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

[6] ISO/IEC. *Information Technology - Open Distributed Processing - ODP Trading Function*, ISO/IEC JTC1/SC21/N9122, August 1994.

[7] 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.

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

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

[10] 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.

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

[12] P. Coad, E. Yourdon. *Object-Oriented Design.* Second Edition, Yourdon Press, Prentice Hall Building, Englewood Cliffs, NJ 07632, 1991.

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

[14] R.O. Sinnott, Kenneth J. Turner. "Applying Formal Methods to Standard Development: The Open Distributed Processing Experience". *Accepted for publication in Computer Standards and Interfaces Special Edition*, 1995.

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

[16] Richard O. Sinnott, Kenneth J. Turner. Exploring the Specification of ODP Types in LOTOS, *submitted to PSTV'95, Warsaw, Poland.*

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

[18] Kenneth J. Turner. Specification Architecture in a Communications Context. *Submitted to Special Issue of Computer Networks and ISDN Systems on Specification Architecture.*

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

[20] 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.

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

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