The specification and testing of conformance in ODP systems

P.F. Linington, J. Derrick and H. Bowman
University of Kent
Computing Laboratory, University of Kent, Canterbury, CT2 7NF, UK
Phone: + 44 1227 764000, Fax: + 44 1227 762811,
Email: {P.F.Linington,J.Derrick,H.Bowman}@ukc.ac.uk.

Abstract

Open Distributed Processing (ODP) is a joint standardisation activity of the ISO and ITU. A reference model has been defined which describes an architecture for building open distributed systems. This paper introduces the key aspects of the reference model of open distributed processing, including the ODP conformance framework. We discuss how specific formal techniques are used in the ODP viewpoints, along with the implications for conformance assessment using such techniques. Particular attention is given to the role of consistency in the conformance assessment process. Finally, we review the current work on an ODP conformance testing methodology.

Keywords: Open Distributed Processing; Conformance; Formal Description Techniques.

1 THE OPEN DISTRIBUTED PROCESSING MODEL

Open Distributed Processing (ODP) [29] is a joint standardisation activity of the ISO and ITU. A reference model has been defined which describes an architecture for building *open* distributed systems [36, 33]. Central to this architecture is a *viewpoints* model. This enables distributed systems to be described from a number of different perspectives. There are five viewpoints: *enterprise*, *information*, *computational*, *engineering* and *technology*. Requirements and specifications of an ODP system can be made from any of these viewpoints.

The framework for the standardization of Open Distributed Processing is provided by the Reference Model for Open Distributed Processing (RM-ODP), the essential, normative parts of which were completed early in 1995. The RM-ODP gives a structure for a family of standards for common interfaces, reusable components and supporting notations needed for the establishment of a wide range of distributed systems.

The reference model is published as ISO 10746, and is in four parts [29]:

Part 1: Overview. This provides an introduction to the reference model, explaining its objectives and giving background to the concepts defined. Part 1 also includes a number of larger examples of ODP use. This part was completed somewhat after the main normative parts (2 and 3), in May 1996.

Part 2: Foundations. This part defines the basic concepts which make up the ODP Object Model, and supporting concepts needed to describe the specification techniques and architectures which are based on it. It also defines the ODP approach to conformance.

Part 3: Architecture. This part defines five viewpoints focussed on various major concerns in the creation of distributed systems, and identifies a set of functions needed to construct such systems.

Part 4: Architectural Semantics. This part provides a formal basis for the reference model. It does this by giving interpretations of the main ODP concepts in a number of standardized formal description techniques. Further detail is given later in section 3.2 of this paper. Part 4 is currently undergoing its final phase of balloting as an International Standard.

This paper introduces the key aspects of the reference model of open distributed processing, including the ODP conformance framework, in section one. Section two discusses the requirements on formal techniques that ODP makes. The use of specific techniques within the ODP viewpoints is considered in section three, along with the implications for conformance assessment using such techniques. Particular attention is given to the role of consistency in the conformance assessment process. We review the current work on an ODP conformance testing methodology in section four, and we conclude in section five.

1.1 The ODP Object Model

ODP is object based; the initial set of concepts used in the reference model define objects which are encapsulated and which interact only at well-defined interfaces. It is not object oriented, because the strong emphasis on implementation inheritance normally found in object oriented programming languages is not appropriate when defining objects which are to form the parts of a distributed system.

An object in ODP can have any number of interfaces, which may be of the same type or of different types. This ability to define objects with multiple interfaces, and to define the dynamic creation and deletion of interfaces, gives a powerful tool for describing the evolution of object behaviour and the development of system configuration. In particular, the ability to define the interaction of objects at an interface which is subsequently hidden forms the basis of many forms of object composition.

The behaviour at an interface is expressed in terms of a set of interactions, considered as atomic at the level of abstraction being used. The general object model does not place constraints on the nature of these interactions, but the more detailed architectural part concentrates on two kinds of interaction - operation invocation and stream flow - leading to two major categories of interface.

In addition to the basic object model, concepts are defined which relate to the specification languages which are used in defining object based systems. These include ideas of behaviour specification, type, class, instantiation and template. The definitions given are generic, capturing concepts found in slightly differing forms in a number of different specification languages. They indicate points where language specific rules are likely to be found associated with a particular notation, but the representation in the various languages of interest is left as part of the architectural semantics.

1.2 The ODP Viewpoints

One of the most important structuring principles in ODP is the definition of a set of viewpoints. The idea is to divide the complete system specification, which may be very large and complex, into a number of areas of concern appropriate to the different stakeholders in the design process. If the different viewpoints are well chosen, they can be developed with a fair degree of independence, simplifying the specification activity. Of course, the viewpoints remain interlocking views of a single system, and so they are not fully independent. Correspondences between them must be declared, even if only by identifying elements which are referred to in more than one viewpoint as equivalent.

The ODP architecture defines five viewpoints, as defined below. The rules associated with each viewpoint effectively define the grammar of a corresponding viewpoint language, and the viewpoint specifications produced for a system must be valid utterances in these languages.

The enterprise viewpoint

The starting point for a system design is the identification of the roles played by entities influencing or influenced by the system, the form of the agreement or contract relating these roles, and the policies which are to direct the detailed system design. In many respects the enterprise viewpoint establishes and documents the boundary conditions constraining the remainder of the design activities.

The information viewpoint

The interaction of different objects depends on the sharing between them of a sufficient interpretation of the parameters of the interaction for there to be meaningful communication. The objective of the information viewpoint is to provide a definition of the shared model used to interpret the information communicated. There will either be a single information model or a small number of overlapping information models, reflecting the domain structure in which the system is to operate.

The enterprise and information viewpoints are concerned with the environment in which the system is to operate, rather than with the distribution process itself. The individual elements which participate in the system's behaviour are not yet visible.

The computational viewpoint

The functional decomposition of the system into a collection of interacting objects forms the basis for distribution. Once a computational model of the system has been defined, the way is open for a statement of the mapping from this configuration to the available physical resources, and the communications requirements can then be deduced from the computational statement of the information flows needed.

The specific details of the object model implicit in the computational language effectively define a virtual machine for the execution of the computational specification. It is this virtual machine which forms the basis for portability and process migration.

The engineering viewpoint

The engineering viewpoint defines the mechanisms which support the various actions and interactions required in the computational specification. These engineering specifications can be seen as the templates which would be used when creating an infrastructure on which to interpret the computational specification.

A large part of the engineering specification is concerned with the definition of channels capable of supporting the various kinds of computational interaction at operational and stream interfaces. This includes the specification of recipes to achieve various transparencies - solving some of the common problems found in distributed systems, such as location or failure independence. Other aspects are concerned with the local behaviour of objects and with the management of resources needed to support it.

The technology viewpoint

The technology viewpoint adds little to the general specification of system behaviour. It is concerned with the collection together of any necessary pointers to pre-existing specifications, such as the specification of supporting protocol stacks or collections of data elements. The linkages thus established form an important step in the interpretation of system behaviour from test results, but the detailed conformance requirements will generally be found in the specifications referenced.

The tool chain

One of the aims of ODP is to provide a coordinated family of standards which encourages the creation of a powerful tool chain able to construct implementations from the set of viewpoint specifications. For any particular design, the computational viewpoint provides the application structure, the enterprise and information viewpoints provide common definitions, policies and constraints, and the engineering viewpoint provides standard templates for the reuse of established communication and resource management techniques. Such tool chains will play an important role in making the development of distributed systems a cost-effective process.

1.3 The ODP Functions

Having established the viewpoint languages, the architecture next defines a set of common functions needed to support distribution. Many of these are concerned with aspects of channel support, either directly or by provision of the additional information necessary to support channel establishment.

Examples of ODP functions are the trading function [10], used in locating suitable services by type when building configurations, and the type repository function [27], used in managing and federating the different type systems shared between many ODP components. Components performing these two functions are already being standardized, with the trader currently nearing publication.

The ODP functions are most closely related to the engineering viewpoint, but the whole set of viewpoints is used when standardizing them.

1.4 The ODP Conformance Framework

ODP defines a basic framework for conformance as part of the reference model, rather than it being retrofitted later, as in OSI. The framework identifies the three key roles as being those of the standardizer, implementor and tester.

The foundation concepts include the idea of a *reference point* as being a specific location at which one or more interfaces can be localized, so that interactions at those interfaces can be observed. A standardizer can then declare some subset of the possible reference points in the system configuration for the component being defined as *conformance points*, stating the set of conformance requirements to be met at each of these points.

There are four different kinds of reference point, corresponding to different interfacing techniques and different testing technologies. They are:

1. interworking reference points, at which interactions take place on an observable communication medium, so that communication events can be deduced from observation of the medium. Most OSI conformance testing is concerned with interworking reference points.

2. programmatic reference points, which represent boundaries between software components within a system. The boundary, for example, can be between an application and its library or middleware, or at the protection boundary of the system kernel. Events in such an interface can be observed by using a suitable software test harness, either to provide an artificial environment or to gain access to the interface within the operational system.

3. perceptual reference points, which represent interactions between the system and the physical world which thus need to be checked by direct observation. A perceptual reference point may be at a human to computer interface, or it may be at a remote sensor or physical actuator in a process control system or for a robot.

4. interchange reference points, which are points at which interchangeable media are read or written. Examples here might be the checking of the correctness of discs passed between otherwise unconnected systems, or testing of the correctness of physical audit or archive media.

Once the conformance requirements have been established by the standardizer, the implementor must create an implementation and make a claim of its conformance to the standard. This involves the implementor naming the standard and declaring which physical locations and test procedures correspond to the various reference points required. In general, this declaration will involve the implementor in declaring a suitable test and interpretation procedure which can be carried out to identify the primitive actions required by the standard.

This is no different in principle from the situation in traditional protocol testing, where the communication medium and the form of any supporting lower layers must be declared. However, it may be considerably more complex in practice because of the wider range of supporting mechanisms which may be available. Thus, for example, the testing of a network protocol requires the identification of data link service events, but there are only a limited number of commonly used network and datalink profiles. On the other hand, the checking of a subtype relationship in a federated type repository may require the testing of steps in an algorithm supported in different locations by a variety of standardized and proprietary interaction mechanisms, depending on the engineering domains involved.

From these requirements comes the idea of a general Implementation Conformance Statement (ICS) and a much fuller set of extra information for testing (IXIT) than in previous conformance testing activities.

Finally, the tester must use this collection of information to plan and execute a testing programme. The testing observations must be interpreted using the information in the IXIT to recognize (or not) the observations as being the events referenced in the standard, and then the behaviour, in terms of these events, must be checked for correctness. The broader scope and size of distributed systems makes exhaustive testing completely impractical. At the same time, however, the growing reliance on the software tool chain offers some hope of simplifications by certifying key components, such as Interface Definition Language [26] compilers and stub generators. It is becoming possible to validate that these tools generate correct engineering viewpoint channel components from correct IDL, thus eliminating a large body of potential run-time testing.

2 FORMAL TECHNIQUES IN ODP

The use and choice of formal description techniques (FDTs) within ODP has implications for both the development and conformance assessment cycles. A number of different techniques and combinations of techniques have been considered, and part 4 of the reference model provides a detailed interpretation of the basic concepts from the descriptive model in a number of formal description techniques. Within ODP, formal description is viewed as enabling precise, unambiguous, and abstract definition and interpretation of ODP standards, and FDTs have been widely used in a number of different contexts [16, 7, 39, 35].

Techniques being considered include those arising from considerations of communication and concurrency (in particular from protocol engineering) such as LOTOS [2], Estelle [21] and SDL [6], together with those arising from general software engineering concerns, for example Z [40], VDM [30] and Raise [18]. The architectural semantics currently considers the languages LOTOS, Estelle, SDL and Z.

Although ODP can be viewed as a natural progression from OSI, ODP includes a wider range of modelling concepts which are needed in order to encompass the concerns of ODP; these range from enterprise policy specification to the description of engineering infrastructures. These differing concerns place a number of requirements on FDTs in order that they may support specification and testing of modern distributed systems. Current languages (both informal and formal) do not support all of these requirements, and it is likely that combinations of languages will be used for the specification of ODP systems and standards.

One major departure from OSI is that ODP modelling is object-based: objects are encapsulated and they interact via a number of interfaces (potentially more than one). In addition, there is a requirement within ODP to be able to specify the composition of objects and for incremental specification using inheritance. The growth of object orientation as a modelling paradigm has led to the definition of a number of object-oriented flavours of FDTs, all of which are able to support the required concepts to various degrees. These flavours include SDL-92 which provides an upwardly compatible object-oriented version of the extended finite state machine notation SDL; Object-Z [11], an object-oriented version of the state based language Z and proposals on how to use LOTOS in an object-oriented style.

However, of these variants only SDL-92 is as yet standardized, and clearly languages or methods advocated in a standard must be sufficiently mature and stable, and therefore should be standardized. Thus it is likely that consideration of FDTs in the ODP reference model will be restricted to LOTOS, SDL-92, Estelle and Z as the standardised or near standardised languages.

Two further ODP modelling requirements which are not in general supported by standardised formal languages are dynamic re-configuration and the ability to express non-functional requirements.

Because ODP systems can be modified and extended during their lifetime, ODP offers a flexible model of configuration. For example, faulty components may need to be replaced or it may be desirable to enable components to migrate in order to enhance performance and availability. The majority of semantic models of distribution and concurrency, e.g. labelled transition systems, finite state machines, event structures or petri nets, only allow static configuration. As a result the ability of LOTOS to model dynamic reconfiguration properly is limited and rather clumsy, [34, 31] discuss the support for dynamic reconfiguration in LOTOS, and the extensions needed to support it fully. On the other hand SDL and Estelle lack a formal semantics, a necessary pre-requisite for convincing modelling of dynamic reconfiguration.

The term non-functional refers to properties not identifiable in terms of a sequence of interactions between communicating objects; for example, quality of service and security issues are usually deemed non-functional. The provision of support for multi-media in ODP means that such requirements are important for the specification and testing of ODP systems. For example, the expression of real-time quality of service constraints, such as latency, throughput and jitter, is significant. [3] discuss the demands made upon

ODP modelling notations by the use of stream bindings and real-time synchronisation in the specification of continuous media.

Of the available notations, SDL, SDL-92 and Estelle all support a model of quantitative time, but it is far from clear that these are appropriate for definition of quality of service constraints [3]. LOTOS supports the expression of the temporal ordering of actions, but not the expression of real-time which is needed for quality of service constraints. A number of real-time extensions to LOTOS have been proposed, and one such variant, E-LOTOS, is currently undergoing standardisation. In addition to the real-time aspects, the new standard contains improved means to model mobility and reconfiguration [20]. The specification of non-functional requirements for an ODP system raises significant issues concerning automatic test generation from FDTs which involve real-time.

As a general purpose language, Z has the ability to support the specification of real time systems without extending the syntax. Work on real time specification using Z includes [12, 14, 32, 37]. However, as time is not built into the language as a pre-defined modelling concept, any real time specification using Z has to model time explicitly in a rather ugly way by using clocks and ticks etc.

As can be seen all four languages LOTOS, Estelle, SDL and Z provide support for modelling in ODP in a number of ways. However, no one language has all the facilities required for the specification of modern distributed systems.

This consideration, and others, point towards the use of multiple languages for the specification of a single ODP system. Single languages do not have the generality or expressiveness to support the full range of ODP specifications across all the viewpoints. Even wide spectrum FDTs such as Raise [18] are not able to embrace all needs (Raise, for example, has no support for real-time modelling). Thus, it is now accepted that a multiple language specification paradigm must be employed and that mechanisms must be provided in order to enable these FDTs to co-exist. This has implications for both the consistency checking and testing aspects of ODP conformance assessment which we describe below.

3 VIEWPOINT LANGUAGES

As indicated earlier, viewpoint languages are used to express specifications in each of the ODP viewpoints. The RM-ODP defines the concepts and rules of the viewpoint languages. However, the reference model is not prescriptive in the choice of specification notation; rather, the intention is that particular existing notations will be instantiated for each of the viewpoint languages, by supporting the concepts and rules defined in the RM-ODP.

Because the modelling concepts and abstraction levels of particular viewpoints vary, different FDTs are applicable to different viewpoints. A testing methodology for ODP then has to encompass the testing of a system potentially specified from a number of different viewpoints each employing a different notation at a different level of abstraction. We review here the specific requirements made by each viewpoint, and discuss the consequences for conformance assessment within ODP.

Enterprise

Enterprise modelling entails statements of policy, of organizational objectives and obligations which must be discharged. None of the concurrency based FDTs are thus seen as suitable candidates for the enterprise viewpoint language. Current enterprise modelling is performed in informal diagrammatic notations [17]; however, the semantics of the informal diagrammatic notations is usually not precisely specified, and thus not suitable for automatic test generation. A logical approach may be applicable to the type of abstract statements of system constraints that are required in the enterprise viewpoint. Therefore a combination of temporal logic and Z is a possibility for a formally based approach.

Because the enterprise viewpoint specifies organizational objectives and policy constraints it is not clear to what extent current automatic test suite generation techniques are applicable to it. However, since this viewpoint might specify key enterprise objectives, it would be instructive to see to what extent enterprise specifications can be used to offer guidance in the selection of test cases.

Information

Z is recognised as highly appropriate for information modelling, e.g. [38, 15]. Z is able to specify the format of information and operations to access and manipulate information without prescribing a particular implementation. Furthermore, many of the information viewpoint concepts have a natural interpretation in Z, which facilitates the formalisation of the information viewpoint in Z in the architectural semantics (see below).

The abstract data typing (ADT) languages incorporated into LOTOS and SDL are also possible vehicles for information specification. However, the correspondence between such ADT notations and the information language concepts is not as natural as it is for Z. This is because the definition of information in ODP centres on the notion that information must be exchangeable amongst users, which relates more directly to the behavioural part.

In LOTOS, information is modelled in ACT ONE along with operations that can act upon them. The behaviour expression part of the specification may then use and manipulate instances of these items. However, it is not the behaviour expressions themselves that change the information items, rather it is the ACT ONE expressions that are associated with the action denotations of the behaviour expressions that manipulate the information. As an example, consider the event offer $g!push(val, empty\_queue)$ in a behaviour expression. It is not the event offer that manipulates the information, but the ACT ONE expression $push(val, empty\_queue)$. Thus, only ACT ONE manipulates the information; however, ACT ONE can only be used within the behaviour expression of LOTOS.

This has implications for both the location of conformance points in an information viewpoint and automatic test suite generation from this viewpoint. Current automatic test suite generation techniques based upon LOTOS (and other process algebras) generate tests from the process algebraic part of the language, and are not capable of independent ACT ONE generated tests. Hence these techniques could only be used in conjunction with other viewpoints (for example, the computational viewpoint), and therefore the viewpoints could not be assessed separately for conformance against an implementation.

Work on test generation from Z specifications includes [43, 8, 5, 42, 19], however, little of this work is specifically targeted towards distributed systems or ODP in particular. Exceptions to this include [8, 42]. [8] generalises the CO-OP method of extracting test suites from LOTOS specifications and applies this to Z specifications. Work done under the Prost project in the UK on the testability of managed object specifications is also relevant to the information viewpoint. In [42] an object-oriented variant of Z is used to specify managed objects, and an inheritance hierarchy is constructed which facilitates the construction of a sound and complete test suite. Importantly, though, the test generation aims to supply heuristics and is not automatic. The heuristics provide a collection of tests together with a residual component which makes explicit the functionality not covered by the test suite. The tests generate form an independent and orthogonal collection of tests.

Because of the inheritance hierarchy, the reuse of tests between related specifications is maximised. A prototype tool-set developed by Logica provides organisational support for the collection of test specifications as they are generated.

The use of Pascal as the Estelle data language prevents Estelle from being an appropriate vehicle for information modelling.

Computational

The computational objects and their interfaces become visible in this viewpoint. The computational viewpoint also identifies the candidates for distribution (the choice being resolved in the engineering viewpoint); hence languages used in this viewpoint need to support the specification of interaction and synchronisation. Languages such as LOTOS, SDL and Estelle all offer considerable support in this respect; however, the encapsulation into objects is not directly supported (except in SDL-92). The work on automatic test suite generation in the languages LOTOS, SDL and Estelle is directly relevant to this viewpoint, where the concerns are similar to those found in OSI. Relevant work is surveyed in [24].

Z lacks explicit support for interaction and synchronisation, although one of the object-oriented dialects of the language, e.g. [7], may be more applicable. The object-oriented versions of the language offer encapsulation and define mechanisms to specify interaction and communication, either by adopting conventions or by definition of appropriate operators. Test generation methods for Z outlined above are again relevant to conformance in this viewpoint. However, aspects of concurrency and hiding have traditionally had less emphasis in Z, and this is reflected in the automatic test generation research using the language.

Engineering

The requirements for engineering viewpoint specification have many similarities to those for the computational viewpoint. Thus, from the potential candidate languages it is reasonable to consider LOTOS, SDL and Estelle as suitable choices and Z as less appropriate.

Technology

Specification in this viewpoint is primarily concerned with referencing appropriate standards and technologies to use in order to realise the specifications of the other viewpoints. This could involve referencing conformance standards and choices and combinations of standards as specified in a profile. Thus, extensive FDT specification is not a major requirement of this viewpoint, although it should be noted that the appropriate standards and technologies are not always rigourously specified, so FDTs may be useful for this purpose as well.

3.1 Consistency

The consistency of multiple viewpoint specifications is an aspect of conformance assessment that is new in ODP. The separation of concerns provided by the viewpoints can simplify the conformance assessment process, but it adds an additional obligation to be discharged, namely to show that all the viewpoints are consistent. Providing mechanisms to demonstrate that multiple viewpoint specifications are consistent is seen as essential to ODP. Furthermore, to use FDTs effectively within ODP, different FDTs are applicable to different viewpoints. This raises the issue of how to ensure consistency between different languages with different underlying semantics. This issue of consistency also arises outside ODP. For example, within OSI two formal descriptions of communication protocols can co-exist and there is no guarantee that, when the two protocols are implemented on the basis of these specifications, processes which use these two protocols can communicate correctly [13].

Consistency checking is relevant to conformance assessment, because even though the check is applied before any implementation is produced, clearly no implementation can hope to conform to all the viewpoints if they are inconsistent. Thus consistency checking can be viewed as part of ODP conformance assessment which is necessary before product testing can begin.

There are a number of different interpretations of the meaning of consistency [4]: one interpretation is to view consistency in terms of whether specifications impose contradictory requirements. Another interpretation is in terms of finding a common implementation, and as such is based on a notion of conformance. The final interpretation is in terms of behavioural compatibility of specifications. We adopt the most general of these possible interpretations, and define that:

> "A collection of viewpoints is consistent if and only if it is possible for at least one example of an implementation to exist that can conform to all viewpoints."

This definition of consistency hinges on the notion of conformance. Therefore, as usual, we divide conformance testing into two parts.

Firstly, we consider formal conformance up to implementation specifications (a relation *conf* between specifications is used for this purpose) and then we consider conformance testing of implementation specifications (essentially a very detailed specification

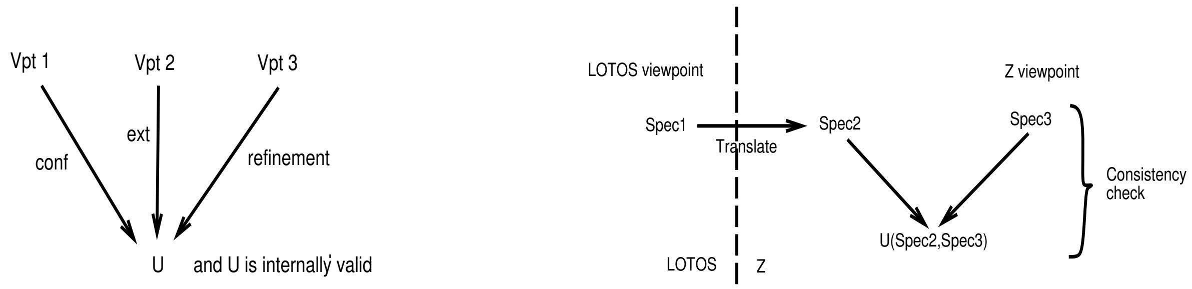

Figure 1: (a) Consistency checking in general, and (b) between LOTOS and Z.

that won't be refined further) to real implementations. The latter is needed because implementation specifications relate to real implementations in different ways for different FDTs and, in particular, for some FDTs not all implementation specifications are implementable. For example, a Z specification that contains an operation $[n! : \mathbb{N} | n! = 5 \wedge n! = 3]$ has no real implementation. Since the implementability of a specification is a property that depends on the FDT used, we will capture this property in our model by an assertion Ψ, which we call *internal validity*.

Then we can view a collection of viewpoints as consistent whenever an internally valid implementation specification exists which conforms to each of the viewpoints. Notice that because different (formal) languages have different notions of conformance, different relations are likely to be used with respect to the different viewpoints (e.g. we might use conformance in one viewpoint and functionality extension in another), see Figure 1 (a).

This methodological framework is compatible with current standardisation work on formal methods in conformance testing [25]. [25] defines a framework for the use of formal methods in conformance testing; in particular, it defines the meaning of conformance if formal methods are used for the specification of a communication protocol or service. In the framework, $MODS$ is used to represent all internally valid implementation specifications under the *testing assumption* that any real implementation can be modelled by an element of $MODS$ (which might be a labelled transition system, finite state machine etc). Conformance is asserted by means of a relation $\underline{\text{imp}}$ between $MODS$ and the set of specifications $SPECS$ (i.e. $\underline{\text{imp}} \subseteq MODS \times SPECS$). Then consistency is defined in [25] as finding a common element of $MODS$ conforming to each viewpoint. The work at Kent diverges from [25] because to define consistency across viewpoints we consider $MODS$ to include possibly unimplementable specifications, and therefore place an additional check on the implementable specification, that it be internally valid.

With this addition in place, a framework such as [25] can be extended to provide a methodology for consistency checking of multiple ODP viewpoint specifications. Mechanisms needed to support consistency checking can then be defined in terms of *refinement*. One specification is said to be a refinement of another if it restricts the set of conformant implementation specifications.

One way of determining whether two specifications are consistent is to unify them. A *unification* of two specifications is their least common refinement. We then check for consistency by determining whether the unification is internally valid, which guarantees that a conformant implementation of the unification exists.

The definition of consistency allows the differing aspects of FDTs to surface within the mechanism in appropriate ways because it involves two distinct parts: firstly the construction of a unification, and secondly verification of internal validity. Which part is appropriate depends on whether the behavioural or logical aspects are dominant in the FDT used. For example, consistency checking in Z and in LOTOS have a very different character. With LOTOS the central issue is finding a unification, while with Z the central issue is demonstrating that a unification does not contain any contradictions and can thus be implemented (assuming the specifications to be unified were themselves implementable).

Consistency checking between viewpoint specifications in LOTOS can involve a number of different refinement relations (**red, ext** etc). It is possible to obtain syntactic definitions of various kinds of unifications, some of these are guaranteed to be least unifications, and classified consistency induced by different refinement relations, [41].

In Z, finding a least unification of two viewpoints is an almost syntactical operation. For any two viewpoints, it is possible to construct a candidate unification, which is the least common refinement if one exists, [1]. Two conditions characterise whether the candidate is a refinement.

Unification combined with verifying internal validity for the unification forms a suitable method of consistency checking in a single FDT environment. However, since specifications in different FDTs cannot be unified, a translation mechanism is needed to transform a specification in one language to a specification in another language. In order to support consistency checking between viewpoints written in Z and LOTOS, a translation between the two languages has been defined in [9]. This can be combined with the Z unification mechanisms to support consistency checking between LOTOS and Z, as shown in Figure 1 (b).

An important aspect in deriving unifications is the correspondence between the viewpoints. Naming alone is insufficient to determine which parts of which viewpoint refer to the same object. Correspondence relations (which are similar to retrieve or abstraction relations [40]) document the dependency between the viewpoints, and the unification is relative to this correspondence relation.

A framework for consistency checking such as that just described has important implications for test case generation. In fact, a major consequence of the move from Open Systems Interconnection to Open Distributed Processing is the influence that viewpoints modelling has on generating tests. Specifically, the tests for an ODP system must be appropriate with regard to all the ODP viewpoints. There seem to be two alternative approaches to enhancing test generation strategies in order to fulfil the requirements of ODP viewpoints modelling.

- **Generation from Multiple Viewpoints.** In this approach, classic methods of deriving tests from a single specification are enhanced to embrace multiple viewpoints. For example, test cases could be derived from each viewpoint in turn and then composed in order to reduce the tests to only those that are appropriate for all viewpoints. Clearly, an explosion in the number of test cases when tests are generated from each viewpoint would be an important issue here.

- **Generation from a Unification Specification.** Central to the framework for consistency checking is the concept of a unification. Such a unification can be viewed as an "implementation specification" for the five viewpoints, i.e. a single specification that reflects all the viewpoints (by being a refinement of them all) and is as developed a description of the system under consideration as can be expressed in the specification domain. So, an alternative approach is to first unify the viewpoint specifications and then derive tests from this unification using classic test case generation techniques. It should be noted thought that it is still not clear that a manageable unification can always be derived from an ODP specification.

There is a clear trade-off between these two approaches. In the first approach the specific process of generating tests is hard and must handle the multiple viewpoints problem, but no explicit unification of the ODP viewpoints is required. In contrast, in the second approach the process of generating tests is unchanged from that traditionally considered, but an actual unification must be derived and this may be difficult. Thus, the former approach puts the emphasis on extending test case generation technology, while the latter emphasizes extending software development technology for multiple viewpoints architectures.

3.2 The ODP Architectural Semantics

The need for an architectural semantics was recognised from the start of the work on the ODP reference model and is reflected in the inclusion of the architectural semantics as Part 4 of the standard. This provides an interpretation of the ODP modelling and specification concepts in LOTOS, Estelle, SDL and Z. The aim of this work is to enable formal description of standards for ODP systems to be developed in a sound and uniform way by providing a link between the ODP modelling concepts and each of the different FDTs (with differing semantics).

Architectural semantics grew out of work on formal description of the protocol layers of the OSI reference model, where it was realised that specifications of protocol entities in different FDTs could not easily be combined. The problem had arisen because OSI concepts were being given totally different interpretations in different FDTs. By defining an architectural semantics, each architectural concept (interaction point, service access point etc) is given a fixed interpretation in different FDTs.

The formalization of the ODP modelling concepts in its architectural semantics consists of formalizing parts 2 and 3 of the reference model (i.e. the prescriptive parts). The RM-ODP defines a set of basic modelling concepts which define an interpretation for a vocabulary together with a collection of viewpoints which specify different aspects of a distributed system. The architectural semantics respects this distinction and comprises an interpretation of modelling concepts in a number of different FDTs together with a formalization of the viewpoint languages (directly and in particular FDTs). Conformance assessment of an ODP system written using a formal technique begins with the architectural semantics, both as a means to interpret the specification, and hence to provide for meaningful test generation, and to define the location of conformance and reference points, i.e. at which locations the testing will take place. As an illustration we discuss the approach taken in the semantics using LOTOS and Z.

Modelling concepts

Rather than provide a mapping of all the ODP modelling concepts, the architectural semantics focusses on the most basic, leaving interpretation of higher level architectural concepts to be made indirectly through their definition in terms of the most basic ones. The focus on object-based modelling means that many of the basic concepts centre around these definitions, and provide interpretations of Object, Interface, Composition of objects, Type, Class etc. In addition, there are interpretations of notions of Action, Behaviour, Communication, Refinement, Pre- and Post- condition etc.

The architectural semantics also defines which relation in a language is to be used to assert conformance. For example, in LOTOS conformance and refinement are then identified with the relations **conf** for the former, and **red** or **ext** for the latter.

Refinement has a well defined interpretation in Z, although importantly there is no general non-transitive relation that corresponds to conformance. Thus Z supports a series of refinements from abstract to more concrete specifications, but not the ability to determine conformance to an implementation specification (that might be written in another formalizations, e.g. labelled transition systems). The definition of a suitable conformance relation is an area that needs further study.

Formalization of Viewpoint Languages

Concepts from the viewpoint languages can be represented directly in particular formal techniques, we illustrate this by describing the information viewpoint in Z, and part of the computational viewpoint in LOTOS.

Formalization of information viewpoint in Z

An information specification corresponds to a Z specification, with the data being represented by the types and the information processing activities given by the possible behaviours as described by the operations in the specification. A static schema then corresponds to the bindings of the variables within the state schema at any point in time. An invariant schema corresponds to predicates given in the state schema and axiomatic descriptions within a specification (i.e. the predicates that the specification always satisfies).

Finally, dynamic schemas correspond to the possible behaviours of a Z specification. Since behaviour is given by the performance of the Z operation schemas, a dynamic schema corresponds to one or more operation schemas in the Z specification. Note that within Z the representation of invariant schemas and dynamic schemas are combined. Any

Z operation must satisfy the invariants within state schemas and axiomatic descriptions, thus any behaviour represented by the Z operations is constrained by the invariant schemas by default. The initialization schema of a Z specification is an example of a information viewpoint static schema, as such an initialization describes the values of variables (i.e. a binding) at a particular point in time, namely initially.

Because behaviour is represented by the Z operation schemas, a conformance statement in an information specification corresponds to one or more operation schemas. If a conformance point in the engineering or computational viewpoints has behaviour that uses information, a check on the use of the information is required to ensure conformance. This behaviour is said to conform if the post-condition and invariant predicates of this information manipulation are satisfied in the associated Z schemas.

A reference point will occur at an interface where tests can be applied to check for conformance. In Z, interfaces are associated with collections of operation schemas (the rest being hidden as described above), so reference points can be considered to reside at the pre-conditions of the operation schemas. However, a Z specification will not in itself identify which reference points are programmatic, perceptual, interworking or interchange. Such identification would accompany the specification as informal commentary.

Formalization of computational viewpoint in LOTOS

The mapping between computational viewpoint concepts and LOTOS is less complete than the Z formalization of the information viewpoint. This is because the computational viewpoint contains a number of modelling concepts for which a direct representation in LOTOS is not obviously available. For example, modelling streams and environmental contracts in LOTOS is problematic because of the lack of real-time support in LOTOS.

However, a subset of the computational viewpoint modelling concepts can be reflected. For example, a computational object is represented by an instantiation of a LOTOS process with the behaviour of the object modelled as the behaviour of the LOTOS process. ODP operations are modelled by LOTOS actions and the interface of an ODP object is the behaviour of a LOTOS process after gates have been hidden. Subtyping relationships between computational interfaces are viewed in ODP in terms of a concept called behavioural compatibility. This concept has in turn been identified with the LOTOS *conf* relation. However, there is now some doubt over this choice.

The exact relationship between conformance concepts and LOTOS constructs is not precisely specified in the current version of the architectural semantics. However, in a general sense, reference points will be identified with interfaces at which tests can be applied. The behaviour of specific actions from amongst those offered by an interface would be monitored during testing. Again, identification of the nature of the identified reference points, i.e. programmatic, perceptual, interworking or interchange, would have to be added as informal commentary.

4 AN ODP CONFORMANCE TESTING METHODOLOGY - CURRENT WORK

The RM-ODP provides a basic framework for describing conformance requirements and testing procedures for distributed systems. However, considerable detail needs to be added before workable conformance requirements can be specified. At the same time there is a need for a consistent approach across a number of areas of standardization, including ODP. In particular, there are closely related requirements arising in work originating from the programming language community to define open system environments. This has lead to requirements for OSE profile definition in part 3 of ISO TR 10000 [28].

At the same time, there is increasing emphasis on development of cost effective standards and support for product testing within ISO. A recent JTC1 Interoperability Policy Statement indicated that individual subcommittees should be given authority and responsibility for the development of an assessment methodology for their areas of activity. The ITU-T is expected to define a question on OSE conformance assessment in the next study period.

On a practical level, the final editing of the ODP Trader standard has indicated an urgent need for a consistent ODP style for the drafting of conformance statements for system components, and for a common understanding of the way middleware requirements and assumptions are to be expressed. More compact representation of test requirements and test purposes are needed to avoid the combinatorial explosion.

These considerations have lead to a general agreement within the ODP community that work on an ODP Conformance Testing Methodology is needed. SC21/WG7 has responded to this need by using the well established SC21 question procedures to clarify scope and direction of work before starting any necessary standardization activity. The question was formulated in the SC21 meeting in Ottawa in July 1995, and work started on it in the meeting in Kansas City in May 1996.

4.1 The WG7 Conformance Testing Question

The question posed was as follows (the text of the question quoted here and a summary of discussion to date can be found in [23]):

"Are any new standardization activities needed to support the conformance testing of ODP systems, components and supporting tools? To what extent can ODP requirements be met by standards which already exist or are under development, and are new liaison activities needed to ensure ODP requirements are taken into account in other work?"

Issues to be addressed:

1. JTC1 standards should avoid unnecessary constraint and proliferation of options; are there any features of the ODP architecture which lead to unnecessary complexity in the conformance testing process?

2. Both the RM-ODP and the work on OSE profiling define four kinds of reference point - in ODP terms, programmatic, interworking, perceptual and interchange

reference points. In OSE, these are considered as interfaces to systems, while in ODP they are interfaces to objects. The implication of this fine grain structure and the associated object encapsulation for testing methodology requires study.

3. Current work on conformance testing has concentrated on the testing of single interfaces. In ODP, there is a need for testing which involves multiple interfaces:

   (a) ODP objects can have multiple interfaces , and assessment of conformance to the object's behaviour may require correlation of tests at a number of interfaces.

   (b) One ODP computational interface can correspond to a number of engineering interfaces of different kinds within the supporting channel (e.g. an interworking reference point and one or more programmatic reference points).

4. The ODP computational model includes stream interfaces; a methodology is required for testing the conformance of stream interfaces.

5. The ODP concept of viewpoint is intended to support separation of concerns, and it is expected that the specifications in the various viewpoints will be combined by suitable tools (as, for example, when an information or computational declaration of need for a transparency results in incorporation of mechanisms based on an engineering specification). The conformance testing methodology may need to distinguish validation of this tool chain from detailed testing of the individual implementations produced.

6. The ODP concept of environment contract allows performance constraints to be placed on implementations. There is a need to study the methodology for testing conformance to these performance requirements.

7. The behaviour of an ODP computational object may require that, in defined circumstances, the object should initiate an invocation at one of its client interfaces. Current conformance testing methodology has concentrated on the testing of objects acting as servers. Study is needed to determine if extensions to the methodology are required to cover testing of client behaviour.

4.2 Constructing an answer

The initial discussions on this question have focussed on removing any architectural uncertainty and on establishing priorities for the work.

One of the key aspects seems to be the need to clarify what kind of standard will include statements of conformance requirements. Consideration needs to be given to component standards, component composition standards and notational standards. These different classes of standard will have quite different forms of assessment, with corresponding style of ICS and IXIT. ODP conformance will place emphasis on aspects of cooperative, multiparty operation, and dynamic configuration not addressed in protocol testing. On the other hand, some traditional concerns can be de-emphasised as a result of the introduction of more powerful notations; static conformance virtually disappears as a result of the widespread use of notations such as IDL.

On the other hand, there is now a level of agreement that apparent conflict in (2) above is not real; the system is, at some level of abstraction, an object, and there is only a problem if the system is seen as an absolute, rather than relative, structuring concept.

Other inputs to date have primarily been concerned with clarifying particular issues in the question text, and further input is being sought. The Kansas City meeting defined priorities for the work, identifying three areas as the most urgent. These were (again, quoting from [23]):

1. the creation of a model conformance statement, using the ODP Trader as an example, which can be used as a pattern for other component standards being developed. The Implementation Conformance Statement and Test Cases for the ODP Trader are to be published as a separate part [22], and so can be developed further without delaying the main part of the technical specification. It was noted that much duplication might be avoided by the definition of implied requirements from an IDL specification.

2. the study of certification of the tool chain as a way to simplify the assessment of conformance of complex systems, rather than exhaustive testing of the actual system.

3. initial development of a methodology for the testing of conformance of stream implementations. There is, as yet, little consensus in this area and thus development of an initial draft is urgent.

5 FUTURE ACTIVITY

This paper has outlined a wide area of work in which there is currently considerable activity. It has also indicated that there is now a well-established body of theory associated with specification and conformance testing. Over the next year or two a draft answer to the ODP question should be agreed and published, and work started on the standardization of the standard for an ODP conformance methodology.

It is to be hoped that conformance testing experts and ODP experts will work together to establish the necessary framework for the creation of reliable and cost effective distributed systems in the future.

References

[1] E. Boiten, J. Derrick, H. Bowman, and M. Steen. Consistency and refinement for partial specification in Z. In M.-C. Gaudel and J. Woodcock, editors, *FME'96: Industrial Benefit of Formal Methods, Third International Symposium of Formal Methods Europe*, volume 1051 of *Lecture Notes in Computer Science*, pages 287–306. Springer-Verlag, March 1996.

[2] T. Bolognesi and E. Brinksma. Introduction to the ISO Specification Language LOTOS. *Computer Networks and ISDN Systems*, 14(1):25–59, 1988.

[3] H. Bowman, L. Blair, G. S. Blair, and A. Chetwynd. Formal description of distributed multimedia systems; an assessment of potential techniques. *Computer Communications*, 18(12):964–977, 1995.

[4] H. Bowman, E.A.Boiten, J. Derrick, and M. Steen. Viewpoint consistency in ODP, a general interpretation. In E. Najm and J.-B. Stefani, editors, *First IFIP International workshop on Formal Methods for Open Object-based Distributed Systems*, pages 189–204, Paris, March 1996. Chapman & Hall.

[5] D. Carrington and P. Stocks. A tale of two paradigms: Formal methods and software testing. In J.P. Bowen and J.A. Hall, editors, *ZUM'94, Z User Workshop*, pages 51–68, Cambridge, United Kingdom, June 1994.

[6] CCITT Z.100. *Specification and Description Language SDL*, 1988.

[7] E. Cusack. Object oriented modelling in Z for Open Distributed Systems. In J. de Meer, V. Heymer, and R. Roth, editors, *IFIP TC6 International Workshop on Open Distributed Processing*, pages 167–178, Berlin, Germany, September 1991. North-Holland.

[8] E. Cusack and C. Wezeman. Deriving tests for objects specified in Z. In J. P. Bowen and J. E. Nicholls, editors, *Seventh Annual Z User Workshop*, pages 180–195, London, December 1992. Springer-Verlag.

[9] J. Derrick, E.A.Boiten, H. Bowman, and M. Steen. Supporting ODP - translating LOTOS to Z. In E. Najm and J.-B. Stefani, editors, *First IFIP International workshop on Formal Methods for Open Object-based Distributed Systems*, pages 399–406, Paris, March 1996. Chapman & Hall.

[10] Draft Rec. X.950 — ISO/IEC DIS 13235-1. *Open Distributed Processing - Trading Function - Part 1: Specification*, May 1996.

[11] R. Duke, G. Rose, and G. Smith. Object-Z: A specification language advocated for the description of standards. *Computer Standards and Interfaces*, 17:511–533, September 1995.

[12] M. Engel. Specifying real-time systems with Z and the duration calculus. In J.P. Bowen and J.A. Hall, editors, *ZUM'94, Z User Workshop*, pages 282–294, Cambridge, United Kingdom, June 1994.

[13] A. Fantechi, S. Gnesi, and C. Laneve. Two standards means problems : A case study on formal protocol descriptions. *Computer Standards and Interfaces*, 9:11–19, 1989.

[14] C.J. Fidge. Specification and verification of real-time behaviour using Z and RTL. In J. Vytopil, editor, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, Lecture Notes in Computer Science, pages 393–410. Springer-Verlag, 1992.

[15] J. Fischer, A. Prinz, and A. Vogel. Different FDT's confronted with different ODP-viewpoints of the trader. In J. C. P. Woodcock and P. G. Larsen, editors, *FME'93: Industrial Strength Formal Methods*, LNCS 670, pages 332–350. Springer-Verlag, 1993.

[16] R. Gotzhein and F. H. Vogt. The design of a temporal logic for Open Distributed Systems. In J. de Meer, V. Heymer, and R. Roth, editors, *IFIP TC6 International Workshop on Open Distributed Processing*, pages 229–240, Berlin, Germany, September 1991. North-Holland.

[17] J. J. Van Griethuysen. Enterprise modelling, A necessary basis for modern information systems. In J. de Meer, V. Heymer, and R. Roth, editors, *IFIP TC6 International Workshop on Open Distributed Processing*, pages 29–68, Berlin, Germany, September 1991. North-Holland.

[18] The RAISE Language Group. *The RAISE Specification Language.* Prentice Hall, 1992.

[19] H-M. Horcher. Improving software tests using Z specifications. In J. P. Bowen and M. G. Hinchey, editors, *Ninth Annual Z User Workshop*, LNCS 967, pages 152–166, Limerick, September 1995. Springer-Verlag.

[20] ISO, Kansas City. *Minutes of the ISO/IEC JTC1/SC21/WG7/E-LOTOS meeting*, May 1996.

[21] ISO 9074. *Estelle, a Formal Description Technique based on an extended state transition model*, June 1987.

[22] ISO/IEC DIS 13235-2. *Open Distributed Processing - Trading Function - Part 2: Implementation Conformance Statements and Test Cases*, May 1996.

[23] ISO/IEC JTC1 SC21/WG7 N1163. *Working Document on Question 7/003 on an ODP Conformance Testing Methodology*, 1996.

[24] ISO/IEC JTC1/SC21. *FMCT guidelines on "Test Generation methods from Formal descriptions"*, December 1995. ISO/IEC ITU-T Interim meeting on FMCT.

[25] ISO/IEC JTC1/SC21. *Formal methods in conformance testing - Part 1: Framework*, December 1995. ISO/IEC ITU-T Interim meeting on FMCT CD 13245-1.

[26] ISO/IEC JTC1/SC21 N10389. *Open Distributed Processing - Interface Definition Language*, April 1996.

[27] ISO/IEC JTC1/SC21 N10389. *Open Distributed Processing - Type Repository Function*, May 1996.

[28] ISO/IEC TR 10000-3. *Information Technology - Framework and Taxonomy of International Standardized Profiles - Part 3: Principles and Taxonomy for Open System Environment Profiles*, 1995.

[29] ITU Recommendation X.901-904 — ISO/IEC 10746 1-4. *Open Distributed Processing - Reference Model - Parts 1-4*, July 1995.

[30] C. B. Jones. *Systematic Software Development using VDM.* Prentice Hall, 1989.

[31] T. Koch, B. Kramer, and N. Volker. Modelling dynamic ODP-configurations with LOTOS. In J. de Meer, B. Mahr, and O. Spaniol, editors, *2nd International IFIP TC6 Conference on Open Distributed Processing*, pages 346–351, Berlin, Germany, September 1993.

[32] L. Lamport. TLZ. In J.P. Bowen and J.A. Hall, editors, *ZUM'94, Z User Workshop*, pages 267–268, Cambridge, United Kingdom, June 1994.

[33] P. F. Linington. RM-ODP: The Architecture. In K. Raymond and L. Armstrong, editors, *IFIP TC6 International Conference on Open Distributed Processing*, pages 15–33, Brisbane, Australia, February 1995. Chapman and Hall.

[34] E. Najm, J-B. Stefani, and A. Fevrier. *Introducing Mobility in LOTOS.* ISO/IEC JTC1/SC21/WG1 approved AFNOR contribution, July 1994.

[35] P. F. Pinto and P. F. Linington. A language for the specification of interactive and distributed multimedia applications. In B. Mahr J. de Meer and O. Spaniol, editors, *IFIP International Conference on Open Distributed Processing*, pages 217–234, Berlin, Germany, September 1993. North-Holland.

[36] K. Raymond. Reference model of open distributed processing (RM-ODP): Introduction. In K. Raymond and L. Armstrong, editors, *IFIP TC6 International Conference on Open Distributed Processing*, pages 3–14, Brisbane, Australia, February 1995. Chapman and Hall.

[37] A.R. Ruddle. Formal methods in the specification of real-time, safety-critical control systems. In J. P. Bowen and J. E. Nicholls, editors, *Seventh Annual Z User Workshop*, pages 131–146, London, December 1992. Springer-Verlag.

[38] S. Rudkin. Modelling information objects in Z. In J. de Meer, V. Heymer, and R. Roth, editors, *IFIP TC6 International Workshop on Open Distributed Processing*, pages 267–280, Berlin, Germany, September 1991. North-Holland.

[39] M. Van Sinderen and J. Schot. An engineering approach to ODP system design. In J. de Meer, V. Heymer, and R. Roth, editors, *IFIP TC6 International Workshop on Open Distributed Processing*, pages 301–312, Berlin, Germany, September 1991. North-Holland.

[40] J. M. Spivey. *The Z notation: A reference manual.* Prentice Hall, 1989.

[41] M. W. A. Steen, H. Bowman, and J. Derrick. Composition of LOTOS specifications. In P. Dembinski and M. Sredniawa, editors, *Protocol Specification, Testing and Verification, XV*, pages 73–88, Warsaw, Poland, 1995. Chapman & Hall.

[42] S. Stepney. Testing as Abstraction. In J. P. Bowen and M. G. Hinchey, editors, *Ninth Annual Z User Workshop*, LNCS 967, pages 137–151, Limerick, September 1995. Springer-Verlag.

[43] P. Stocks and D. Carrington. Deriving software test cases from formal specifications. In *6th Australian Software Engineering Conference*, pages 327–340, July 1991.