FDTs for ODP †

Howard Bowman, John Derrick, Peter Linington and Maarten Steen.
Computing Laboratory, University of Kent, Canterbury, CT2 7NF, UK.
(Email: {hb5,jd1,pfl,mwas}@ukc.ac.uk.)

Abstract

This paper discusses the use and integration of formal techniques into the Open Distributed Processing (ODP) standardization initiative.

The ODP reference model is a natural progression from OSI. Multiple viewpoints are used to specify complex ODP systems. Formal methods are playing an increasing role within ODP.

We provide an overview of the ODP reference model, before discussing the ODP requirements on FDTs, and the role such techniques play. Finally we discuss the use of formalisms in the central problem of maintaining cross viewpoint consistency.

Key words: Open Distributed Processing; Formal Description Techniques; Z; Consistency; Viewpoints.

1 Introduction

This paper discusses the use and integration of formal description techniques (FDTs) into the Open Distributed Processing (ODP) standard initiative.

The ODP standardization initiative is a natural progression from OSI, broadening the target of standardization from the point of interconnection to the end-to-end system behaviour. The objective of ODP [21] is to enable the construction of distributed systems in a multi-vendor environment through the provision of a general architectural framework that such systems must conform to. One of the cornerstones of this framework is a model of multiple viewpoints which enables different participants each to observe a system from a suitable perspective and at a suitable level of abstraction [27]. There are five separate viewpoints presented by the ODP model: Enterprise, Information, Computational, Engineering and Technology. Requirements and specifications of an ODP system can be made from any of these viewpoints.

Formal methods are playing an increasing role within ODP (Part 4 of the Reference Model outlines requirements for applying formal description techniques in the specification of ODP systems.). The suitability of a wide spectrum of FDTs is currently being assessed (eg LOTOS, Estelle, SDL, Z, Object-Z and RAISE). The Basic Reference Model of Open Distributed Processing (RM-ODP) recognises the need for formalism:

"The work of the RM-ODP is based on the use, as far as possible, of FDTs to give it a clear and unambiguous interpretation."

[0] † This work was partially funded by British Telecom Labs., Martlesham, Ipswich, U.K. and partially by the U.K. Engineering and Physical Sciences Research Council under grant number GR/K13035.

One of the consequences of adopting a multiple viewpoint approach to specification is that descriptions of the same or related entities can appear in different viewpoints and must co-exist. *Consistency* of specifications across viewpoints thus becomes a central issue. Similar consistency properties arise 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].

This paper outlines the issues surrounding the use of FDTs in ODP and focusses on the major problem of maintaining consistency of ODP viewpoint specifications.

In Section 2 we provide an overview of the ODP reference model. In Section 3 we discuss the use of formal description in ODP to date, and Section 4 discusses the issue of consistency of viewpoint specifications written in FDTs. We make some concluding remarks in Section 5.

2 Overview of the ODP Reference Model

2.1 Background to the ODP activity

The initiative which lead to the standardization of Open Distributed Processing came from a growing awareness that many of the communications-oriented standardization activities aimed at the provision of Open Systems Interconnection required a broader framework than was provided by the OSI Reference Model. Standardization of distributed applications such as interpersonal messaging or transaction processing requires a view of the way many components are linked into a distributed system, and of the resources and structures used by these components. A simple interconnection model is not powerful enough. What is needed is a model which can combine the description of system structure with statement of system-wide objectives and constraints, so that the adequacy of the solutions proposed can be judged against the system's original purpose. The ODP standardization initiative is a response to these issues.

The Reference Model for ODP consists of four parts. The first of these provides an introduction and is not formally normative, while the remaining parts are normative. The four parts are:

Part 1 (X.901) the tutorial introduction, which introduces concepts and gives guidance to the interpretation of the model;

Part 2 (X.902) a descriptive model; this part brings together a collection of modelling concepts which could be applied to the description of a wide range of distributed systems and to the many kinds of enterprise which they support. The approach taken is object-based, and the set of concepts defined constitute a precise basic object model, including the necessary definitions to construct type and class structures. This part also describes the basis for conformance to standards and for the associated testing methodologies.

Part 3 (X.903) a prescriptive model; this part contains the more detailed concepts and rules which must be observed for the system being described to be an ODP system. The rules provide the framework which unifies the subsequent ODP standards and which allows standard functional components to be constructed.

Part 4 (X.904) the architectural semantics; this part gives the detailed interpretation of the basic concepts from the descriptive model in a number of formal description techniques - currently LOTOS, Estelle, SDL and Z. The statement of a clear semantics for the architecture makes it possible to use formal descriptions of the various functions defined by the architecture in combination without ambiguity.

In addition to the Reference Model itself, standardization has started on one of the most important ODP functions, the Trader, which plays an essential role in coordinating the configuration of a distributed system by informing potential clients of the existence of instances of the services they require. Standardization of further functions is expected to start in the near future.

2.2 Scope of the Reference Model

The Reference Model of Open Distributed Processing aims to provide a unifying framework for the standardization of any mechanized distributed system, and of the supporting models, techniques and notations needed to describe such a system. Its scope is very broad, including support for all types of traditional data processing systems, networked personal computers, real-time systems and multimedia systems.

The specification of such systems starts with requirements capture and description of the environmental constraints the system must take account of. The model thus needs to be able to describe the enterprise in which the system is to operate at whatever level of detail is needed to express these requirements.

It should be clear that a stand-alone system is merely a special case of a distributed system. Indeed, since system evolution may lead to an initially isolated system later becoming distributed, either by enhancement or by federation with other systems, it is prudent to design all systems as potentially distributed. The RM-ODP therefore provides a framework for the specification of the functional aspects of any system in a way that emphasises potential for distribution and so maximizes the likely lifetime of the investment in its design and implementation.

2.3 The framework of abstraction

The complete specification of any non-trivial distributed system involves a very large amount of information. Attempting to capture all aspects of the design in a single description is generally unworkable. Most design methodologies aim to establish a coordinated, interlocking set of models each aimed at capturing one facet of the design, satisfying the requirements which are the concern of some particular group involved in the design process.

In ODP, this separation of concerns is established by identification of five *viewpoints*, each with an associated *viewpoint language* which expresses the rules relevant to a particular area of concern.

However, these viewpoints are not independent. They are each partial views of the complete system specification. Some items can, therefore, occur in more than one viewpoint, and there are a set of consistency constraints arising from the correspondences between terms in two viewpoint languages and the statements relating the various terms within each language. The checking of such consistency is an important part of demonstrating the correctness of the full set of specifications. There are five viewpoints defined in the ODP Reference Model: Enterprise, Information, Computational, Engineering and Technology.

Each viewpoint language consists of a set of definitions and a set of rules which constrain the ways in which the definitions can be related. The notion of language used here is an abstract one; the rules are, in effect, the foundations for the grammars of a set of possible detailed languages or notations.

The **enterprise viewpoint**, which is concerned with business policies, management policies and human user roles with respect to the systems and the environment with which they interact; the use of the word enterprise here does not imply a limitation to a single organization; the model constructed may well describe the constraints placed on the interaction of a number of distinct organizations.

The **information viewpoint**, which is concerned with information modelling; by factoring an information model out of the individual components, it provides a consistent common view which can be referenced by the specifications of information sources and sinks and the information flows between them. The information language defines concepts for information schema definition. The language distinguishes between an instantaneous view of information (a static schema), a statement of information which is necessarily unchanged by the system (an invariant schema) and a description of information reflecting the behaviour and evolution of the system (a dynamic schema).

The **computational viewpoint**, which is concerned with the algorithms and data flows which provide the distributed system function; this viewpoint specifies the individual components which are the sources and sinks of information flows. The computational language represents the system and its environment in terms of objects which interact by transfer of information via interfaces. This does not necessarily imply that the computational objects will be realized in the eventual system by separate components, but indicates which are the candidate boundaries when components are chosen.

The **engineering viewpoint**, which is concerned with the distribution mechanisms and the provision of the various transparencies needed to support distribution. The engineering language defines a number of functional building blocks which can be combined together to provide the requested transparencies (e.g. distribution, failure or migration transparencies). The engineering language lists a large number of supporting functions which are candidates for standardization (or for which there are already standards) and gives initial definitions of them.

The **technology viewpoint**, which is concerned with the detail of the components and links from which the distributed system is constructed.

Since the aim of the ODP Reference Model is to support a very wide range of distributed applications, the degree of prescription in the five languages varies. The computational and engineering languages impose clear design choices in order to reduce unnecessary variety in the range of infrastructure components and so to achieve successful portability and interworking. On the other hand, the enterprise and information languages are primarily intended to describe the environment in which the system is to be used, and undue prescription would effectively limit their scope by ruling out some possible styles of use.

3 Formal Description in ODP

Formal description has been extensively employed in Open Distributed Processing, [16, 10, 37, 31]. Within ODP, formal description is viewed as enabling precise, unambiguous, and abstract definition and interpretation of ODP standards. This is the familiar motivation for employing FDTs in standardisation activities. However, the spectrum of FDT usage in ODP is both extensive and diverse. Which FDT should be employed for each particular role is a central issue. The spectrum of available FDTs also offers significant diversity. For example, LOTOS [2, 19], Estelle [20] and SDL [7] are targeted at issues of explicit concurrency and interaction (specifying ordering and synchronisation of abstract events). Communication protocols are a typical example of this class of application. In contrast, approaches such as Z [39] and VDM [22] address specification of software systems in terms of data state change. Importantly, none of these FDTs fully address the specification requirements of modern distributed processing and Open Distributed Processing in particular. Such systems are extremely broad, encompassing, for example, both information modelling and description of engineering infrastructures.

We will clarify the requirements for formal description in ODP in the following subsections. We first consider general requirements for ODP specification and then highlight three of the most important areas of application of formal description within ODP.

3.1 General Requirements

Typical requirements of FDTs are: expressiveness, compositionality, that they be verifiable, have clear semantics and tool support; these all apply within the ODP setting. However, there are a number of additional requirements which arise from the specific characteristics of ODP systems.

- **Object Oriented Specification**
  ODP modelling is object oriented [23] - objects are encapsulated and they interact only via interfaces. There is support for composition of objects and incremental specification using inheritance. Thus, formal descriptions must support this paradigm.

- **Dynamic Reconfiguration**
  Open Distributed Processing offers a flexible model of configuration; ODP systems can be modified and extended during their lifetime. This is a very important requirement since user and system needs may alter dynamically. 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. The dynamic reconfiguration requirement is prompting some of the most significant current research in concurrency theory.

- **Non-functional Requirements**
  Broadly speaking, a requirement is non-functional if it cannot be directly represented computationally, i.e. the requirement is not identified in terms of a sequence of interactions between communicating objects. Examples of non-functional requirements arise in the area of quality of service and security. Such requirements are important in supporting multimedia interaction in Open Distributed Systems. The expression of real-time quality of service constraints, such as latency, throughput and jitter, is of particular significance. The provision of support for continuous media, through stream bindings and real-time synchronisation, imposes demanding requirements upon specification notations for ODP, see [3] for a discussion.

- **Co-existence of Multiple FDTs**
  It became clear early that a single FDT would not have the generality or expressiveness to support the full range of ODP specifications. Even wide spectrum FDTs such as Raise [18] are not able to embrace all needs. 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. The importance and demanding nature of this requirement will become evident and provides the motivation behind the section 4 of this paper.

- **Support for Formal Reasoning**
  It is essential that formal reasoning can be applied to the FDTs used in ODP. Thus, rigourous and usable semantic definitions must be provided by the FDT. Relations between specifications such as refinement and behavioural compatibility are defined in the reference model, and corresponding FDT semantic relations need to be available for instantiation of these concepts in particular FDTs.

- **Abstraction**
  By its very nature a reference model must not be overly prescriptive and must define a framework for building compliant systems which is sufficiently abstract to support all relevant realisations. This is particularly true of the RM-ODP which seeks to embrace a huge

spectrum of target systems. Thus, it is essential that interpretation of the ODP model in a particular formal notation does not compromise this level of abstraction.

- **Standardised FDTs**
  A further, more pragmatic, requirement is that FDTs must be sufficiently mature that re-definition of the FDT will not render existing specifications obsolete. This generally means that a suitable FDT must itself have been standardised or that standardisation of the FDT must be sufficiently advanced that the language definition is stable.

The last of these requirements effectively reduces the choice to a handful of FDTs. In fact, LOTOS [19], Estelle [20], SDL [7] and Z [39] are seen as the main contenders. Although, if the final requirement is wavered a whole host of object oriented dialects [10, 35], process calculi supporting mobile processes, [30] and real-time FDTs [26, 28] could be considered.

	SDL	SDL-92	Estelle	LOTOS	Z
OO	$\times$	$\surd$	$\times$	$\times$	$\times$
Dynamic Reconfiguration	$\surd$	$\surd$	$\surd$	$\surd/\times$	$\times$
Non-functional requirements	$\surd/\times$	$\surd/\times$	$\surd/\times$	$\times$	$\times$
Standardised	$\surd$	$\surd$	$\surd$	$\surd$	$\surd/\times$
Support for formal reasoning	$\times$	$\times$	$\times$	$\surd$	$\surd$

Table 1: Relating FDTs to General ODP Requirements

A classification against a subset of these requirements is shown in table 1. We have not considered the issues of co-existence or abstraction because the former is primarily a relationship between FDTs and the latter is too subjective to make a binary choice. The classification of SDL, SDL-92 and Estelle in the non-functional requirements category arises because all the languages support a model of quantitative time, but it is far from clear that these are appropriate for definition of quality of service constraints [3]. In addition, the classification of LOTOS in the dynamic reconfiguration category is made to reflect the fact that some dynamic reconfiguration can be supported, e.g. [29], but such an approach is not expressive enough to meet the full configuration requirements of ODP, e.g. [30, 25]. The support of invariants is an additional important FDT characteristic, and because of this SDL, LOTOS, and Estelle are not particularly suitable for the enterprise and information viewpoints (see below).

This table indicates that the extended finite state machine techniques, SDL, SDL-92 and Estelle, support a lot of the required features; this is particularly true of SDL-92. However, there are also good reasons for prefering LOTOS or Z. Arguments for LOTOS and Z focus on the elegance, simplicity and usability of their approach. Importantly, both offer more tractable semantic foundations than the extended finite state machine approaches; this is reflected in the categorisation under support for formal reasoning. For example, semantically well founded notions of refinement and equivalence have been defined for both LOTOS, see [6] [26], and Z, see [32], while corresponding definitions for Estelle, SDL or SDL-92 are mathematically problematic. Furthermore, Z and LOTOS have been argued for on grounds of their superior support for abstract specification. In addition, particular areas of application of FDTs in ODP reveal a richer set of requirements than those highlighted in table 1. The next three sections will clarify these requirements by considering three of the most important areas of application of FDTs in ODP.

3.2 Viewpoint Languages

As indicated earlier, viewpoint languages 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 as each of the viewpoint languages, by supporting the concepts and rules defined in the RM-ODP.

How then do FDTs relate to viewpoints? It is now well accepted that different FDTs are applicable to different viewpoints, because the features of the FDTs variously support the required abstraction levels and modelling concepts of each particular viewpoint:

- **Enterprise**
  None of the four FDTs, Z, LOTOS, SDL or Estelle, are seen as suitable candidates for the enterprise viewpoint language. Enterprise modelling entails statements of policy, of organizational objectives and obligations which must be discharged. Most current Enterprise modelling is performed in 'informal' diagrammatic notations; see [17] for a discussion of such notations. However, the semantics of the informal diagrammatic notations is usually not precisely specified, leading to incomplete, ambiguous, and unverifiable specifications. However, a logical approach may be applicable to the type of abstract statements of system constraints that are required in the enterprise viewpoint. It would be interesting to consider some form of extended Z for this purpose.

- **Information**
  Z is recognised as highly appropriate for information modelling, e.g. [36, 15]. Z is able to specify the format of information and operations to access and manipulate information without prescribing a particular implementation.

  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. In particular, many of the information viewpoint concepts suggest an interpretation which uses both the process and data part of LOTOS and SDL. Furthermore, it has been suggested that the ADT definitions are too concrete and force over specification in information modelling (although the Object Z approach may be considered as based on ADTs, and this does not force over specification), e.g. see [15] for a comparison of an information model of the ODP trader in Z and LOTOS.

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

- **Computational**
  The need to specify interaction and synchronisation prevents Z from being a suitable choice for computational viewpoint specification, although, one of the object-oriented dialects of the language, e.g. [10], may be more applicable. In contrast, the FDTs LOTOS, SDL and Estelle, all offer considerable support for computational viewpoint specification.

- **Engineering**
  The requirements for engineering viewpoint specification have many similarities to those for the computational viewpoint. Thus, from the four main candidate languages it is reasonable to consider LOTOS, SDL and Estelle as reasonable 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. 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.

3.3 Conformance assessment

Conformance assessment of ODP systems has been considered from very early on in the work on Open Distributed Processing. This is in contrast to the experience of OSI where consideration of conformance assessment was left late in the standards work. Hence, the meaning of conformance has been built into the ODP reference model.

The very wide range of component specifications and standards that ODP must support and, in particular, the accent on the incorporation of existing technology means that an ODP conformance assessment methodology must describe a method to assess *de jure* and *de facto* standards and specifications that may not specifically be labelled "ODP". The work in PROST project [9] has defined a suitably general architecture, and through this methodology conformance to ODP systems can be asserted.

PROST divides conformance assessment for ODP into two parts: *specification checking* and *conformance testing*. Specification checking focusses on specification to specification relationships (such as equivalence); it aids conformance by supporting verification of ODP specifications.

FDTs underpin the work on conformance assessment within ODP. They do so by enabling rigourous system development to be undertaken from formal specifications. A formal approach to conformance facilitates the development of tools to support the automated checking of relevant conformance relationships. This applies both to the specification checking phase of conformance assessment (where the specification to specification relationships of particular FDTs can be used) and to conformance testing (where automatic test case generation can be applied).

3.4 Architectural Semantics

Interest in architectural semantics arose during the work on formal description of the protocol layers of the OSI-RM. Specifically, it was realised that specifications of protocol entities in different FDTs could not easily be combined. This was caused by the totally different interpretations of the OSI concepts, such as service access point, in different FDTs. Most importantly, LOTOS uses synchronous communication, while Estelle and SDL uses an asynchronous model. See [40] for a good discussion of the origins and motivation for architectural semantics. Architectural semantics seek to provide a resolution of this variety of interpretation, by tying the different FDTs to a single set of architectural concepts. Specifically, interpretations of these architectural concepts are made in each FDT, thus providing an unambiguous intermediate between the FDTs.

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. Thus, this work will act as a bridge between the ODP model and the semantic models of the FDTs and will enable formal description of standards for ODP systems to be developed in a sound and uniform way. A further important objective of the architectural semantics work is to subject the definitions in the RM-ODP to a rigourous examination. In this sense the application of formalism forces standards writers to think deeply about the definitions they are making. Inconsistencies or ambiguities found in the definitions have then been fed back into the work on part 2 and 3 of the reference model.

Currently, the formal interpretation of the part 2 reference model concepts is relatively stable and is an ISO Committee Draft; however, interpretation of the part 3 concepts is still under active investigation. This reflects the fact that part 3 of the reference model is only now becoming stable itself. There is considerable interest in how to provide fully abstract interpretations of viewpoint language concepts, such as stream binding in the computational viewpoint. See [41] for a discussion of computational viewpoint interpretation using LOTOS.

A further, more general and more fundamental problem with the architectural semantics definition is that each FDT interpretation imposes its own prescription on the basic ODP concepts. The RM-ODP by its nature defines an extremely abstract and generalised architectural framework and each FDT interpretation constrains this in some way. Thus, it is not clear that fully abstract descriptions of the ODP architecture have been made. Important work that seeks to address this issue, at least in the context of interpretation of the ODP computational language, is [8]. This contribution seeks to provide a fully general interpretation of the computational language by defining transition rules that characterize the valid behaviours of configurations of computational objects. These transition rules provide a weakest constraint approach to viewpoint language definition by leaving as many issues of behaviour open as possible. It is suggested that FDTs can be related to this direct semantics relatively easily. Thus providing the single, fully general, intermediate required of an architectural semantics.

3.5 Discussion

In summary, FDTs are being employed extensively in ODP and have proved valuable in supporting precise definition of reference model concepts, but the requirements of ODP have revealed a number of problems with the state of the art of formal description. Specifically, there is a need to support non-prescriptive, fully general interpretation of the ODP architecture. Additionally, there is a need to support object-oriented concepts, dynamic reconfiguration and expression of non-functional properties within the context of an abstract FDT which supports formal reasoning. There are further more pragmatic requirements, such as the need to stabilize FDT definitions through standardization and the need to provide excellent and portable tool support, which can enable formal interpretations to be checked during standards meetings. This, in particular, is essential if FDT interpretations are to be kept up to date with changes made to the reference model.

Probably the most central requirement on FDTs arising from ODP is the need to support multiple paradigm specification. Providing mechanisms to demonstrate that multiple viewpoint specifications are consistent is seen as essential to ODP. This is a very demanding requirement, because different FDTs are applicable to different viewpoints. Thus, consistency checks must be made between different languages with different underlying semantics. The next section discusses this issue.

4 Consistency of Viewpoint Specifications

Consideration of the issues raised by the preceding sections indicates that to use FDTs effectively within ODP we should target specific languages in particular viewpoints. This raises the issue of how to ensure consistency of viewpoint specifications written in different FDTs. As noted in the introduction, this issue of consistency has applications in other standardization activities outside of ODP.

Before one can provide a mechanism to check the consistency of specifications, one must ask the question: what does consistency mean? Unfortunately there is more than one answer, so we must seek to reconcile different possible interpretations of consistency. In this section we discuss the consistency relationships currently defined in the ODP standard, and explain how different definitions are applicable to different FDTs. We then outline a consistency checking mechanism for viewpoint specifications written in Z, which we illustrate with an example.

4.1 Defining Consistency in ODP

The RM-ODP contains three different interpretations of the meaning of consistency, two in Part 1 (clause 12.2) and the third in Part 3 (clause 10). Each of these notions of consistency is intuitively reasonable. However, as all definitions are informal, it is not clear how the different interpretations relate to one another; in particular, are these definitions of consistency themselves consistent?

The different definitions have arisen because each FDT has a "natural" interpretation of consistency. 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. Thus, there is a need to give a precise definition of exactly what consistency of viewpoint specifications means; once formally defined, their differences can be determined and possibly be reconciled. This can be seen as one of the advantages of applying formal methods to a standardization activity.

The different interpretations are likely to be applicable in different settings. For example, the first interpretation of consistency is relevant in a logical setting, e.g. in an FDT such as Z which is based on first order logic. The latter interpretations are applicable in a behavioural setting (such as with LOTOS, SDL or Estelle).

The problem, then, is to use formalization to reconcile the interpretations and thus clarify the standard; this then helps to realize the standard by applying formal techniques to produce consistent specifications. In [5] we have shown how the definitions of consistency can be formalized, and how they can be interrelated. This was achieved by defining a consistency checking mechanism which just involves specification checking; i.e. specification to specification relationships (e.g. equivalence, refinement) are used, as opposed to any notions of logic, testing or behavioural compatibility. The starting point is the most general definition of consistency from the RM-ODP which is, informally:

> "Two specifications are consistent if and only if it is possible for at least one example of a product (or implementation) to exist that can conform to both of the specifications."

This definition of consistency hinges on the notion of conformance. *Conformance* is a relation between an implementation and a specification, that holds when the implementation passes all the tests that can be derived from the specification. As an implementation is not a formal object, conformance can only be determined through physical testing. This is unsatisfactory, because, to reap the benefits of FDTs fully we must be able to establish consistency/inconsistency at specification time before real implementation(s) have been produced.

Therefore 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 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 $[n! : \mathbb{N} \mid n! = 5 \wedge n! = 3]$ 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*.

We can then define a consistency checking mechanism via specification checking relationships as follows. Using the formal conformance relation *conf* between specifications (for example, the *conf* relation in LOTOS), 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. Thus, an implementation of the unification of two specifications will implement both specifications that were unified. A natural specification checking definition of consistency is that *two specifications are consistent if their unification is internally valid.* Obviously, the non-existence of such a unification means that the two specifications are inconsistent under the notion of refinement applied. In addition to the existence of the unification, verification of the internal validity of the unification is needed. The internal validity condition guarantees that a conformant implementation of the unification exists.

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. Naturally, we require the specification and its translation to identify the same set of conformant implementations (this is the meaning of the phrase *information preserving* as used in RM-ODP). Once a translation mechanism has been provided, it is possible to extend the specification checking consistency mechanism to a multi-FDT environment.

This specification checking definition of consistency can then be used to relate the definitions given in the RM-ODP, [5]. This definition is stronger than the definition of consistency from the RM-ODP, this is appropriate however, since the extra knowledge available during specification checking enables system developers to apply consistency with more discrimination.

To summarize, we have defined a mechanism to check for consistency which allows the differing aspects of FDTs to surface within the mechanism in appropriate ways. The definition is applicable to different FDTs 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).

4.2 Unifying Viewpoint Specifications in Z

In this section we describe a general strategy for unifying two Z specifications. In order to increase its applicability, it is not specific to any particular ODP viewpoint, nor is it tied to any particular instantiation of the architectural semantics. We then show how to use this unification to check the consistency of two viewpoints written in Z. This is illustrated with an example of an information viewpoint specification from OSI Management.

As described above the unification of two specifications is the least refinement of both viewpoints. Unification of Z specifications will therefore depend upon the Z refinement relation, which is given in terms of two separate components - data refinement and operation refinement, [32]. Two specifications will thus be consistent if their unification is *internally valid*, and for Z this holds when the unification is free from contradictions (assuming the specifications that were unified were both internally valid). Thus to check the consistency of two specifications, we check for contradictions within the unified specification.

Z is a state based FDT, and Z specifications consist of informal English text interspersed with formal mathematical text. The formal part describes the abstract state of the system (including a description of the initial state of the system), together with the collection of available operations, which manipulate the state. One Z specification refines another if the state schemas are data refinements and the operation schemas are operation refinements of the original specifications state and operation schemas. We assume the reader is familiar with details of the language and its refinement relation, introductionary texts include [32, 39].

The unification algorithm we describe is divided into three stages: normalization, common refinement (which we usually term unification itself), and re-structuring. This algorithm can be shown to be the least refinement of both viewpoints, [11].

Normalization identifies commonality between two different viewpoint specifications, and re-writes each specifications into a normal form suitable for unification in the following manner. Clearly, the two specifications that are to be unified have to represent the world in the same way within them (e.g. if an operation is represented by a schema in one viewpoint, then the other viewpoint has to use the same name for its (possibly more complex) schema too), and that the correspondences between the specifications have to have been identified by the specifiers involved. These will be given by mappings that describe the naming, and other, conventions in force. Once the commonality has been identified, normalization re-names the appropriate elements of the specifications. Normalization will also expand data-type and schema definitions into a normal form. Examples of normalization are given in [32, 33].

Unification itself takes two normal forms and produces the least refinement of both. Because normalization will hide some of the specification structure introduced via the schema calculus, it is necessary to perform some re-structuring after unification to re-introduce the structure chosen by the specifier. We do not discuss re-structuring here.

4.2.1 State Unification

The purpose of state unification is to find a common state to represent both viewpoints. The state of the unification must be the least data refinement of the states of both viewpoints, since viewpoints represent partial views of an overall system description.

The essence of all constructions will be as follows. Given two viewpoint specifications both containing the following fragment of state description given by a schema D:

$$\begin{array}{|l}
\multicolumn{1}{l}{D} \\ \hline
x : S \\ \hline
pred_S \\ \hline
\end{array}
\qquad
\begin{array}{|l}
\multicolumn{1}{l}{D} \\ \hline
x : T \\ \hline
pred_T \\ \hline
\end{array}$$

we unify as follows

$$\begin{array}{|l}
\multicolumn{1}{l}{D} \\ \hline
x : S \cup T \\ \hline
x \in S \implies pred_S \\
x \in T \implies pred_T \\ \hline
\end{array}$$

whenever $S \cup T$ is well founded. (Axiomatic descriptions are unified in exactly the same manner.)

4.2.2 Operation Unification

Once the data descriptions have been unified, the operations from each viewpoint need to be defined in the unified specification. We assume all renaming of names visible to the environment has taken place. Unification of schemas then depends upon whether there are duplicate definitions. If an operation is defined in just one viewpoint, then it is included in the unification (with appropriate adjustments to take account of the unified state).

For operations which are defined in both viewpoint specifications, the unified specification should contain an operation which is the least refinement of both, w.r.t. the unified representation of state. The unification algorithm first adjusts each operation to take account of the unified state in the obvious manner, then combines the two operations to produce an operation which is a refinement of both viewpoint operations.

The unification of two operations is defined via their pre- and post-conditions. Given a schema it is always possible to derive its pre- and post-conditions, [24]. Given two schemas A and B representing operations, both applicable on some unified state, then the unification of A and B is:

$$\begin{array}{|l}
\hline
\mathcal{U}(A, B) \\
\vdots \\
\hline
pre\ A \lor pre\ B \\
pre\ A \Longrightarrow post\ A \\
pre\ B \Longrightarrow post\ B \\
\hline
\end{array}$$

where the declarations are unified in the manner of the preceding subsection. This definition ensures that if both pre-conditions are true, then the unification will satisfy both post-conditions. Whereas if just one pre-condition is true, only the relevant post-condition has to be satisfied. This provides the basis of the consistency checking method for object behaviour which we discuss below.

4.3 Example

The application of Z in the ODP information viewpoint to the modelling of OSI Management has been investigated by a number of researchers, [36, 42]. We show here how unification and consistency checking can be used with such modelling techniques by considering viewpoint specifications of sieve managed objects and their controlling CME agent.

To illustrate some of the techniques in this paper we shall consider two viewpoint specifications of an event reporting sieve object together with a third viewpoint which describes a CME agent and its manipulation of the sieve objects. In this simplified model we have not considered the relationships between managed objects, although a complete presentation would include their specification.

Within ODP, an information object template is modelled by a Z specification. An information object instance is then modelled as a Z specification instance (i.e. a specification complete with initialization of variables), and an ODP action is described by a Z operation.

The variable declarations in a state schema represent the attributes of a managed object. The state schema also describes the state invariant which constrains the values of the attributes. The initialization schema (e.g. *InitSieve*) constrains the initial values of the state schema.

A managed object definition cannot include a *Create* operation, since before it is created a managed object cannot perform any operation, including *Create* itself. However, by including a *Create* operation in the CME agent viewpoint as we do below, we can describe formally the interaction between *Create* and the sieve managed object definition.

We have not considered any particular flavours, or design considerations, to differentiate between the first two viewpoints. Their purpose here is to represent to view of the system from similar standpoints.

4.3.1 Viewpoint 1 : Sieve object

To describe the sieve object, we first declare the types. *SieveConstruct* is used in the event reporting process, its internal structure is left unspecified at this stage, hence it is defined as a given set.

$$[SieveConstruct]$$

The remaining types are declared as enumerated types.

$$\begin{array}{lcl} Operational & ::= & disabled \mid active \mid enabled \mid busy \\ Admin & ::= & locked \mid unlocked \mid shuttingdown \\ Event & ::= & nothing \mid enrol \mid deenrol \\ Status & ::= & created \mid deleted \end{array}$$

Status models the life-cycle of the sieve object, and is used as an internal mechanism to control which operations are applicable at a given point within an object's existence. The state schema defines the attributes of the sieve object, here there are no constraints upon them; and the initialization describes their initial values.

$$\begin{array}{|l} \textit{Sieve} \\ \hline opstate : Operational \\ sico : SieveConstruct \\ adminstate : Admin \\ status : Status \\ \hline \end{array} \qquad \begin{array}{|l} \textit{InitSieve} \\ \hline Sieve \\ \hline opstate = active \\ adminstate = unlocked \\ \hline \end{array}$$

We describe two of the operations available within a sieve object (for a full description of operations see [36]). The first is an operation to delete a sieve. Upon deletion a sieve sends a *deenrol* notification to its environment, and moves into a state where no further operations can be applied.

$$\begin{array}{|l} \textit{Delete} \\ \hline \Delta Sieve \\ notification! : Event \\ \hline status = created \\ notification! = deenrol \\ status' = deleted \\ \hline \end{array}$$

We define a relation *filter* to represent criteria to decide which events to filter out and which to pass on

$$\begin{array}{|l} filter : Event \leftrightarrow SieveConstruct \end{array}$$

and the *Filter* schema represents the operation to perform the filtering.

$$\begin{array}{|l} \textit{Filter} \\ \hline \Xi Sieve \\ event? : Event \\ notification! : Event \\ \hline status \neq deleted \\ opstate = active \land adminstate = unlocked \\ (event?, sico) \in filter \Rightarrow notification! = event? \\ (event?, sico) \notin filter \Rightarrow notification! = nothing \\ \hline \end{array}$$

4.3.2 Viewpoint 2 : Sieve object

To illustrate some of the unification techniques we now describe a second view of the same sieve object. First of all we declare the types

$$[SieveConstruct]$$

$$\begin{array}{lcl} Operational & ::= & disabled \mid active \mid enabled \mid busy \\ Admin & ::= & locked \mid unlocked \mid shuttingdown \\ Event & ::= & nothing \mid enrol \mid deenrol \\ Status & ::= & being\_created \mid created \mid deleted \end{array}$$

Notice that in this viewpoint $Status$ includes an additional value, $being\_created$. The state schema and its initialization are then declared.

$$\begin{array}{|l} \hline Sieve \\ opstate : Operational \\ sico : SieveConstruct \\ adminstate : Admin \\ status : Status \\ \hline opstate \in \{active, disabled\} \\ adminstate \in \{locked, unlocked\} \\ \hline \end{array}$$

$$\begin{array}{|l} \hline InitSieve \\ Sieve \\ \hline status = being\_created \\ opstate = active \\ adminstate = unlocked \\ \hline \end{array}$$

The change of state of a sieve object from $being\_created$ to $created$ is governed by an internal operation, which can occur spontaneously. This change in $status$ allows other operations to be invoked subsequently apart from the $Enrol$ operation itself.

$$\begin{array}{|l} \hline Enrol \\ \Delta Sieve \\ notification! : Event \\ \hline status = being\_created \\ status' = created \\ notification! = enrol \\ \hline \end{array}$$

In this viewpoint a relation $newfilter$ defines the filtering criteria, and the operation $Filter$ performs the filtering.

$$\begin{array}{|l} newfilter : Event \leftrightarrow SieveConstruct \end{array}$$

$$\begin{array}{|l} \hline Filter \\ \Xi Sieve \\ event? : Event \\ notification! : Event \\ \hline status = created \\ opstate = active \land adminstate = unlocked \\ (event?, sico) \notin newfilter \Rightarrow notification! = nothing \\ \hline \end{array}$$

4.3.3 Viewpoint 3 : CME agent

The final viewpoint is a description of a controlling CME agent. For our purposes here we present a very simplified version of an agent which consists of a number of sieve managed objects. We then show how we can promote the *Delete* operation defined on individual sieve objects, and define a *Create* operation to instantiate sieve objects as required.

This viewpoint has a number of schemas from the other viewpoints as parameters, these are given as empty schema definitions. Upon unification the under-specification of these parameters in this viewpoint will be resolved by the other viewpoint specifications, and thus unification will allow functionality extension of these parameters. The parameters we require are:

$$\begin{array}{|l} \hline \mathit{Sieve} \\ \hline \end{array} \qquad \begin{array}{|l} \hline \mathit{InitSieve} \\ \hline \end{array}$$

$$\begin{array}{|l} \hline \mathit{Delete} \\ \hline \Delta \mathit{Sieve} \\ \hline \end{array}$$

We declare types to represent the set of object classes and set of object identifiers respectively. A *CMEagent* is then modelled as a collection of sieve objects, and initially no sieve objects have been created, so the range of *sieves* cannot include a state described by *Sieve*

$$[\mathit{Class}, \mathit{Id}]$$

$$\begin{array}{|l} \hline \mathit{CMEagent} \\ \hline \mathit{sieves} : \mathit{Class} \times \mathit{Id} \nrightarrow \mathit{Sieve} \\ \hline \end{array} \qquad \begin{array}{|l} \hline \mathit{InitCMEagent} \\ \hline \mathit{CMEagent} \\ \hline \nexists\, \mathit{Sieve} \bullet \theta \mathit{Sieve} \in \operatorname{ran} \mathit{sieves} \\ \hline \end{array}$$

Here we use promotion (i.e. the θ operator) in the structuring of viewpoints, which allows an operation defined on an object in one viewpoint to be *promoted* up to an operation defined over that object in another viewpoint. As we can see, this can be used effectively to reference schemas in different viewpoints without their full definition.

In order to define CME agent operation, we define a schema $\Phi \mathit{CMEagent}$ which will allow individual object operations to be defined in this viewpoint. See [33] for a discussion of the use of promotion.

$$\begin{array}{|l} \hline \Phi \mathit{CMEagent} \\ \hline \Delta \mathit{CMEagent} \\ \Delta \mathit{Sieve} \\ \mathit{objectclass?} : \mathit{Class} \\ \mathit{sieveid?} : \mathit{Id} \\ \hline \mathit{sieves}(\mathit{objectclass?}, \mathit{sieveid?}) = \theta \mathit{Sieve} \\ \mathit{sieves}' = \mathit{sieves} \oplus \{\mathit{sieves}(\mathit{objectclass?}, \mathit{sieveid?}) = \theta \mathit{Sieve}'\} \\ \hline \end{array}$$

An agent operation to delete a specific sieve object can now be defined by promotion of the *Delete* parameter specified in another viewpoint. The other managed object operations are promoted in a similar fashion.

$$\mathit{DeleteSieve} \mathrel{\widehat=} (\Phi \mathit{CMEagent} \land \mathit{Delete}) \setminus (\Delta \mathit{Sieve})$$

Finally the *Create* operation can be defined. Notice this is not part of the sieve specification, so we have preserved the concept that *Create* must occur before any operation in the sieve specification can be applied.

$$
\begin{array}{|l}
\hline
\textit{Create} \\
\Delta CMEagent \\
\Delta Sieve, \Delta InitSieve \\
objectclass? : Class \\
sieveid? : Id \\
\hline
sieves(objectclass?, sieveid?) \neq \theta Sieve \\
sieves' = sieves \oplus \{sieves(objectclass?, sieveid?) = \theta InitSieve'\} \\
\hline
\end{array}
$$

4.3.4 Unification of Viewpoints

To describe the unification of viewpoints, we decorate with subscripts, so for example $Filter_1$ is the schema *Filter* from the first viewpoint. To unify viewpoints 1 and 2 we first unify the state. The only conflict in the declarations are due to differing types $Status_1$ and $Status_2$. To resolve this conflict, the type *Status* in the unification is taken as the least refinement of $Status_1$ and $Status_2$ (i.e. $Status_1 \cup Status_2$), and state unification is applied to the schema *Sieve*. Hence, in addition to the declarations which are not in conflict, the unification will contain the following:

$$Status ::= being\_created \mid created \mid deleted$$

$$
\begin{array}{|l}
\hline
\textit{Sieve} \\
opstate : Operational \\
sico : SieveConstruct \\
adminstate : Admin \\
status : Status \\
\hline
status \in \{created, deleted\} \Rightarrow true \\
status \in \{being\_created, created, deleted\} \Rightarrow \\
\qquad (opstate \in \{active, disabled\} \land adminstate \in \{locked, unlocked\}) \\
\hline
\end{array}
$$

Upon simplification the schema *Sieve* becomes

$$
\begin{array}{|l}
\hline
\textit{Sieve} \\
opstate : Operational \\
sico : SieveConstruct \\
adminstate : Admin \\
status : Status \\
\hline
opstate \in \{active, disabled\} \\
adminstate \in \{locked, unlocked\} \\
\hline
\end{array}
$$

In a similar fashion we unify $InitSieve_1$ and $InitSieve_2$, which simplifies to

$$
\begin{array}{|l}
\hline
\textit{InitSieve} \\
Sieve \\
\hline
status = being\_created \\
opstate = active \\
adminstate = unlocked \\
\hline
\end{array}
$$

The *Delete* and *Enrol* schemas are defined in just one viewpoint. Hence, both these schemas are included in the unification (with adjustments due to the unified state schema *Sieve*). Similarly the unification contains both relations *filter* and *newfilter*.

To unify $Filter_1$ and $Filter_2$ we first adjust $Filter_1$ due to the unified state schema. The predicate part of $Filter_1$ is then

$$status \in \{created, deleted\} \Rightarrow (status \neq deleted \wedge opstate = active \wedge adminstate = unlocked)$$

Calculation of the pre-conditions $preFilter_1 \vee preFilter_2$ then simplifies to

$$(status = created \wedge opstate = active \wedge adminstate = unlocked)$$

Thus the unification of $Filter_1$ and $Filter_2$ is then given by:

$$
\begin{array}{|l}
\hline
\textit{Filter} \\
\Xi Sieve \\
event? : Event \\
notification! : Event \\
\hline
status = created \wedge opstate = active \wedge adminstate = unlocked \\
(event?, sico) \in filter \Rightarrow notification! = event? \\
(event?, sico) \notin filter \Rightarrow notification! = nothing \\
(event?, sico) \notin newfilter \Rightarrow notification! = nothing \\
\hline
\end{array}
$$

To complete the unification we must unify this specification with the third viewpoint which specified the CME agent. The parameters in the third viewpoint have their functionality extended upon unification. For example, the schema *InitSieve* defined in the third viewpoint is just a parameter from the other viewpoints, and consequently its unification will just be:

$$
\begin{array}{|l}
\hline
\textit{InitSieve} \\
Sieve \\
\hline
status = being\_created \\
opstate = active \\
adminstate = unlocked \\
\hline
\end{array}
$$

The complete unification is then achieved in the obvious manner, by expanding *Sieve*, *InitSieve* and *Delete* and including *Enrol* and *Filter* along with the CME agent operations.

4.4 Consistency Checking of Viewpoint Specifications in Z

The mechanism for unifying two Z specification yields a consistency checking process. In terms of the ODP viewpoint model, consistency checking consists of checking both the consistency of the state model and the consistency of all the operations. Consistency checking of the state model ensures there exists at least one possible set of bindings that satisfies the state invariant, and the Initialization Theorem (see below) ensures that we can find one such set of bindings initially.

Because a conformance statement in Z corresponds to an operation schema(s), [38], we require operation consistency. Thus a given behaviour (i.e. occurrence of an operation schema) conforms if the post-conditions and invariant predicates are satisfied in the associated Z schema. Hence, operations in a unification will be implementable whenever each operation has consistent post-conditions on the conjunction of their pre-conditions.

Thus a consistency check in Z involves checking the unified specification for contradictions, and has three components: State Consistency, Operation Consistency and the Initialization Theorem.

State Consistency : Consider the general form of state unification given in Section 4.2.1:

$$
\begin{array}{|l}
\multicolumn{1}{l}{D} \\
\hline
x : S \cup T \\
\hline
x \in S \Longrightarrow pred_S \\
x \in T \Longrightarrow pred_T \\
\hline
\end{array}
$$

This state model is consistent as long as both $pred_S$ and $pred_T$ can be satisfied for $x \in S \cap T$.

Operation Consistency : Consistency checking also needs to be carried out on each operation in the unified specification. The definition of operation unification means that we have to check for consistency when both pre-conditions apply. That is, if the unification of A and B is denoted $\mathcal{U}(A, B)$, we have:

$$pre\ \mathcal{U}(A, B) = pre\ A \vee pre\ B, \quad post\ \mathcal{U}(A, B) = (pre\ A \Rightarrow post\ A) \wedge (pre\ B \Rightarrow post\ B)$$

So the unification is consistent whenever $(pre\ A \wedge pre\ B) \Rightarrow (post\ A = post\ B)$.

Initialization Theorem : The Initialization Theorem is a consistency requirement of all Z specifications. It asserts that there exists a state of the general model that satisfies the initial state description, formally it takes the form:

$$\vdash \exists\, State \bullet InitState$$

For the unification of two viewpoints to be consistent, clearly the Initialization Theorem must also be established for the unification.

The following result can simplify this requirement: Let $State$ be the unification of $State_1$ and $State_2$, and $InitState$ be the unification of $InitState_1$ and $InitState_2$. If the Initialization Theorem holds for $State_1$ and $State_2$, then state consistency of $InitState$ implies the Initialization Theorem for $State$. In other words, it suffices to look at the standard state consistency of $InitState$.

If, however, $InitState$ is a more complex description of initiality (possibly still in terms of $InitState_1$ and $InitState_2$), the Initialization Theorem expresses more than state consistency of $Initstate$, and hence will need validating from scratch. An example of this is given below.

4.4.1 Example

State Consistency : Consider the state schema $Sieve$. The unified schema across all three viewpoints was given by:

$$
\begin{array}{|l}
\multicolumn{1}{l}{Sieve} \\
\hline
opstate : Operational \\
sico : SieveConstruct \\
adminstate : Admin \\
status : Status \\
\hline
status \in \{created, deleted\} \Rightarrow true \\
status \in \{being\_created, created, deleted\} \Rightarrow \\
\qquad (opstate \in \{active, disabled\} \wedge adminstate \in \{locked, unlocked\}) \\
\hline
\end{array}
$$

From this it can be seen that both predicates $true$ and $(opstate \in \{active, disabled\} \wedge adminstate \in \{locked, unlocked\})$ can be satisfied for $status \in \{created, deleted\} \cap \{being\_created, created, deleted\}$, which is the requirement for consistency for this state schema.

Operation Consistency : Consider the unification of the $Filter$ operation schema. From the unification we found that

$$preFilter_1 = preFilter_2 = (status = created \wedge opstate = active \wedge adminstate = unlocked)$$

Thus to show operation consistency we have to show that under this pre-condition, we have

$$(((event?, sico) \in filter \Rightarrow notification! = event?) \wedge ((event?, sico) \notin filter \Rightarrow notification! = nothing))$$
$$= ((event?, sico) \notin newfilter \Rightarrow notification! = nothing)$$

It is easy to show that a necessary, but not sufficient, condition for the consistency of this operation is

$$\forall (event?, sico) \in Event \times Event \bullet (event?, sico) \in filter \wedge (event?, sico) \notin newfilter \Rightarrow event? = nothing$$

Thus the consistency of *Filter* requires this condition to be maintained. This type of consistency condition should probably fall under the heading of a *correspondence rule* in ODP, [21], that is a condition which is necessary but not necessarily sufficient to guarantee consistency.

By giving specifiers explicit notification of which relationships between objects in the viewpoints need preserving, this constraint can then be used by the individual viewpoint specifiers to ensure that any further refinement of the viewpoints do not violate consistency.

Inspection of the unification of the CME agent with the sieve object shows that both state and operation consistency carry over from the state and operation consistency of the unification of viewpoints 1 and 2. Hence, consistency will follow once we establish the Initialization Theorem for the unification of all three viewpoints.

The **Initialization Theorem** for the unification is

$$\vdash \exists\, CMEagent \bullet InitCMEagent$$

which expands to

$$\vdash \exists\, sieves : Class \times Id \rightarrowtail Sieve \bullet \nexists\, Sieve \bullet \theta Sieve \in \operatorname{ran} sieves$$

Upon simplification this becomes

$$\vdash \exists\, sieves : Class \times Id \rightarrowtail Sieve \bullet \langle status, opstate, adminstate, sico\rangle \notin \operatorname{ran} sieves$$

The final term describes a set of bindings, and it is clear that such a function *sieves* exists. Hence, the viewpoint descriptions given for the CME agent and sieve objects are indeed consistent.

The consistency checking mechanism works well for small to medium sized Z specifications. For larger specifications additional structure is needed in order that the consistency checking strategy can be scaled up. [12] shows how support for this can be provided by using object oriented variants of Z. These object based methodologies provide sufficient structure for the consistency checking to remain feasible.

4.5 Translation

Above we have shown how consistency checking may be performed within a single FDT, viz. Z., however, the real challenge lies in checking for consistency across language boundaries, and this requires translation between FDTs. There has been some success in relating formal languages that have similar underlying semantics, e.g. [34, 1]. However, the common semantics used in these approaches is typically very ugly. ODP consistency checking requires translation across FDT families. Some directions that could be pursued to make such translations possible are discussed below.

Syntactic translation

Translation based upon a direct relation of syntactic terms in one FDT to terms in another FDT is one possible approach. However, it is difficult to envisage how such an approach could offer a general solution. In particular, a lot of semantic meaning will certainly be lost in such a crude translation of FDTs. Partial syntactic translations may, however, be feasible.

Common underlying semantics

Another, more promising, approach is to define a common underlying semantic model for the required FDTs. Specifications could then be translated into the common semantic domain, in which a consistency check can be performed. Such translation could either use the semantics of one of the FDTs as the intermediate semantics or use a third semantics. The former of these is not fully general; for example, Z and LOTOS are so fundamentally different that relating one to the others semantic model is very difficult to envisage. Relating FDTs using a third intermediate form is a more likely approach.

A sufficiently general semantic model has been developed in [44]. One-sorted first-order predicate logic is proposed to capture the semantics of specifications in different formalisms. The proposed semantical model is very general, but it does not necessarily have the same properties as the standard semantics of the FDTs. Also, there are certain features of specification languages which cannot be captured in first-order logic. Nevertheless, the proposed method presents a promising step towards a general technique for consistency checking.

There have been several other attempts to relate different formal languages:

- A link between model based action systems (and thereby Z) and CSP has been made by showing that refinements (forwards and backwards simulation) in an action system are sound and jointly complete with respect to the notion of refinement in CSP [43]. Ongoing research is focussing on extending this work to the general setting of Z and LOTOS.
- The requirement for highly expressive intermediate semantics suggests that logical notations may be appropriate. [14] and [4] consider logical characterisations of LOTOS in temporal logic. However, relating temporal logic to the Z first order logic remains an open issue. Categorical approaches and the theory of institutions offer a possible solution [4].
- A final alternative which has the benefit of being ODP specific is suggested by the work of [8]. This work offers a direct denotational semantics for the computational viewpoint language. This semantics could, theoretically, be used to relate different FDT interpretations of the computational viewpoint language. Clearly, this work does not give a complete solution to consistency as the semantics are restricted to a single viewpoint. However, it may be possible to extrapolate this approach to a general solution.

A further issue affecting translation is the role of the ODP architectural semantics. Specifically, part 4 should provide a basis for relating FDTs. ODP concepts, in particular viewpoint languages, are defined in different FDTs in the architectural semantics. Thus, when relating complete viewpoint specifications in different FDTs these definitions can be used as components of a consistency check. However, it is important to note that the architectural semantics will only provide a framework for consistency checking. Actual viewpoint language specifications will extend the ODP architectural semantics, which are non-prescriptive by nature, with FDT specific behaviour. There is then a need to combine the framework provided by the architectural semantics with actual consistency checking relationships arising from FDTs.

It is clear, though, that a usable translation mechanism is likely to represent a pragmatic, compromise solution. In particular, complete preservation of semantic meaning during translation will not be possible. In addition, different viewpoints describe different sets of features and thus may not be directly translatable between each other.

5 Conclusion

Formal description techniques are being employed extensively in ODP and have proved valuable in supporting precise definition of reference model concepts, but the requirements of ODP have revealed a number of problems with the state of the art of formal description. There is a need to support non-prescriptive, fully general, interpretation of the ODP architecture. In addition, support is needed for object-oriented concepts, dynamic reconfiguration and expression of non-functional properties within the context of an abstract FDT which supports formal reasoning.

However, probably the most central requirement on FDTs arising from ODP is the need to support consistency checking of multiple viewpoint specifications of ODP systems. We discussed the consistency relationships currently defined in the ODP standard, and explained how different definitions are applicable to different FDTs. We have shown how consistency checking may be performed on viewpoints written in Z. This was illustrated with an example from OSI Management.

References

[1] D. Bert, M. Bidoit, C. Choppy, R. Echahed, J.-M. Hufflen, J.-P. Jacquot, M. Lemoine, N. Levy, J.-C. Reynaud, C. Roques, F. Voisin, J.-P. Finance, and M.-C. Gaudel. Operation SALSA: Structure d'accuil pour specification algebriques. Technical report, Rapport final, PRC Programmation et Outils pour l'intelligence Artificielle, 1993.

[2] T. Bolognesi and E. Brinksma. Introduction to the ISO Specification Language LOTOS. *Computer Networks and ISDN Systems*, 14(1):25–29, 1988.

[3] H. Bowman, L. Blair, G.S. Blair, and A. Chetwynd. Formal description of distributed multimedia systems; an assessment of potential techniques. *To Appear in Computer Communications*, 1995.

[4] H. Bowman and J. Derrick. Formalizing conformance in ODP. In R. Wieringa and R. Feenstra, editors, *Working papers of IS-CORE'94, International Workshop on Information Systems - Correctness and Reusability*, pages 357–370, September 1994.

[5] H. Bowman, J. Derrick, and M. Steen. Some results on cross viewpoint consistency checking. In *IFIP International Conference on Open Distributed Processing*. Chapman Hall, 1995. To appear.

[6] E. Brinksma, G. Scollo, and C. Steenbergen. Process specification, their implementation and their tests. In B. Sarikaya and G. V. Bochmann, editors, *Protocol Specification, Testing and Verification, VI*, pages 349–360, Montreal, Canada, June 1986. North-Holland.

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

[8] AFNOR cont. *A direct computational language semantics for Part 4 of the RM-ODP*. ISO/IEC JTC1/SC21/WG7 approved AFNOR contribution, July 1994.

[9] G. Cowen, J. Derrick, M. Gill, G. Girling (editor), A. Herbert, P. F. Linington, D. Rayner, F. Schulz, and R. Soley. *Prost Report of the Study on Testing for Open Distributed Processing*. APM Ltd, 1993.

[10] 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.

[11] J. Derrick, H. Bowman, and M. Steen. Maintaining cross viewpoint consistency using Z. In *IFIP International Conference on Open Distributed Processing*. Chapman Hall, 1995. To appear.

[12] J. Derrick, H. Bowman, and M. Steen. Viewpoints and objects. In *Ninth Annual Z User Workshop*, Limerick, September 1995. Springer-Verlag. To appear.

[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] A. Fantechi, S. Gnesi, and G. Ristori. Compositional logic semantics and LOTOS. In L. Logrippo, R.L. Probert, and H. Ural, editors, *Protocol Specification, Testing and Verification, X*, Ottawa, Canada, June 1990. North-Holland.

[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] ISO 8807. *LOTOS: A Formal Description Technique based on the Temporal Ordering of Observational Behaviour*, July 1987.

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

[21] ISO/IEC JTC1/SC21/WG7. *Basic reference model of Open Distributed Processing - Parts 1-4*, July 1993.

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

[23] H. Kilov. Precise specification of behaviour in object-oriented standardization activities. *Computer Standards & Interfaces*, 15:275–285, 1993.

[24] S. King. Z and the refinement calculus. In D. Bjorner, C.A.R. Hoare, and H. Langmaack, editors, *VDM '90 VDM and Z - Formal Methods in Software Development*, LNCS 428, pages 164–188, Kiel, FRG, April 1990. Springer-Verlag.

[25] 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.

[26] G. Leduc. A framework based on implementation relations for implementing LOTOS specifications. *Computer Networks and ISDN Systems*, 25:23–41, 1992.

[27] P. F. Linington. Introduction to the Open Distributed Processing Basic Reference Model. In J. de Meer, V. Heymer, and R. Roth, editors, *IFIP TC6 International Workshop on Open Distributed Processing*, pages 3–13, Berlin, Germany, September 1991. North-Holland.

[28] C. Miguel, A. Fernandez, and L. Vidaller. Extending LOTOS twoards performance evaluation. In M. Diaz and R. Groz, editors, *Formal Description Techniques, V*, Lannion, France, October 1992. North-Holland.

[29] E. Najm and J-B. Stefani. Dynamic configuration in LOTOS. In K.R Parker and G.A. Rose, editors, *Formal Description Techniques, IV*, Sydney, Australia, November 1991. North-Holland.

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

[31] 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.

[32] B. Potter, J. Sinclair, and D. Till. *An introduction to formal specification and Z*. Prentice Hall, 1991.

[33] B. Ratcliff. *Introducing specification using Z*. McGraw-Hill, 1994.

[34] R. Reed, W. Bouma, J.D. Evans, M. Dauphin, and M. Michel. *Specification and Programming Environment for Communication Software*. North Holland, 1993.

[35] S. Rudkin. Inheritance in LOTOS. In K.R Parker and G.A. Rose, editors, *Formal Description Techniques, IV*, Sydney, Australia, November 1991. North-Holland.

[36] 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.

[37] 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.

[38] R. Sinnott. *An Initial Architectural Semantics in Z of the Information Viewpoint Language of Part 3 of the ODP-RM*, 1994. Input to ISO/JTC1/WG7 Southampton Meeting.

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

[40] C.A. Vissers. FDTs for open distributed systems, a retrospective and a prospective view. In R.L. Probert L. Logrippo and H. Ural, editors, *Protocol Specification Testing and Verification X*. North-Holland, 1990.

[41] A. Vogel. On ODP's architectural semantics using LOTOS. In J. de Meer, B. Mahr, and O. Spaniol, editors, *2nd International IFIP TC6 Conference on Open Distributed Processing*, pages 340–345, Berlin, Germany, September 1993.

[42] C. Wezeman and A. J. Judge. Z for managed objects. In J. Bowen and J. Hall, editors, *Eighth Annual Z User Workshop*, pages 108–119, Cambridge, July 1994. Springer-Verlag.

[43] J.C.P. Woodcock and C.C. Morgan. Refinement of state-based concurrent systems. In D. Bjorner, C.A.R. Hoare, and H. Langmaack, editors, *VDM '90 VDM and Z - Formal Methods in Software Development*, LNCS 428, pages 340–351, Kiel, FRG, April 1990. Springer-Verlag.

[44] P. Zave and M. Jackson. Conjunction as composition. *ACM Trans. on Soft. Eng. and Method.*, 2:379–411, 1993.

Biographies

Dr H. Bowman has been a lecturer at the University of Kent since October 1993. Prior to this he received his Ph.D. from the University of Lancaster in 1991. He then completed a two year SERC postdoctoral research associateship at the University of Lancaster developing a new FDT (based on LOTOS) for formally describing distributed multimedia systems. His current research interests include the application of formal description techniques in ODP standardisation activities and the formal validation and verification of real-time systems.

Dr J. Derrick gained a D.Phil in 1987 from Oxford for work on functional analysis, he then joined STC Technology Ltd to work on the ESPRIT funded RAISE project. Since 1990 he has been a Lecturer in Computer Science at the University of Kent. He has worked on the theoretical foundations of formal methods and non-interleaving models of concurrency, as well as applications of formal methods to ODP and distributed computing. His current interests include developing techniques for the use of FDTs within ODP and formal definitions of consistency and conformance.

Professor P.F. Linington has been involved in computer communication since 1975; he worked on the UK Coloured Book protocols and participated in the ISO work on OSI from its inception, being Rapporteur for FTAM until 1984. From 1979 he was a member of the UK Dep. of Industry's Data Communication Protocols Unit. In 1983, he became Head of the Joint Network Team and the Network Executive, responsible for network coordination and operation of the JANET network.

In 1987 he became the Professor of Computer Communication at the University of Kent. His research interests span networks and distributed systems, currently concentrating on distributed multimedia systems exploiting audio and video information. In ISO, he is currently involved in the standardization of Open Distributed Processing. He chairs the BSI panel on ODP and leads the UK delegation. He also chairs the internal technical review committee for the Esprit ISA project (previously ANSA).

Ir. M.W.A. Steen obtained an MSc(Eng) in Computer Science from the University of Twente, The Netherlands, in 1993. He is currently supported by an E.B. Spratt bursary from the University of Kent at Canterbury to do a Ph.D. in Computer Science. His research focusses on the application of formal methods, in particular LOTOS, to Open Distributed Processing.