Maintaining Cross Viewpoint Consistency using Z

John Derrick, Howard Bowman and Maarten Steen †
University of Kent at Canterbury, U.K. {jd1,hb5,mwas}@ukc.ac.uk.

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

One of the cornerstones of the ODP framework is a model of multiple viewpoints. During the development process it is important to maintain the consistency of different viewpoints of the same ODP specification. In addition, there must be some way to combine specifications from different viewpoints into a single implementation specification. The process of combining two specifications is known as unification. Unification can be used as a method by which to check consistency. This paper describes a mechanism to unify two Z specifications, and hence provide a consistency checking strategy for viewpoints written in Z.

Keyword Codes: C.2.4, D.2.1, D.2.2.
Keywords: Distributed Systems, Specification, Tools and Techniques.

1 INTRODUCTION

This paper discusses the implications and integration of formal techniques, in particular Z, 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 [9] 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 to observe a system from a suitable perspective and at a suitable level of abstraction [11, 14]. 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, and we aim to provide a mechanism by which specific techniques can be used within ODP. The suitability of a wide spectrum of FDTs is currently being assessed. Amongst these Z is likely to be used for at least the information, and possibly the enterprise and computational, viewpoint. The first compliant ODP specification, the Trader, is being written using Z for the information and computational viewpoint.

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

Whilst it has been accepted that the viewpoint model greatly simplifies the development of system specifications and offers a powerful mechanism for handling diversity within ODP, the practicalities of how to make the approach work are only beginning to be explored. In particular, one of the consequences of adopting a multiple viewpoint approach to development 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, [5]. However, the actual mechanism by which consistency can be checked and maintained is only just being addressed [8, 7, 6]. In particular, although Z is being used as a viewpoint specification language in ODP, there is as yet no mechanism to describe the combination of different Z viewpoint specifications, or the consistency of them.

In Section 2 we develop a unification mechanism for Z specifications. In Section 3 we present an example of the technique by specifying the dining philosophers problem using viewpoints. Section 4 discusses consistency checking of viewpoint specifications, and we make some concluding remarks in Section 5.

2 UNIFICATION IN Z

One of the cornerstones of the ODP framework is a model of multiple viewpoints. Clearly the different viewpoints of the same ODP specification must be consistent, i.e. the properties of one viewpoint specification do not contradict those of another. In addition, during the development process there must be some way to combine specifications from different viewpoints into a single implementation specification. This process of combining two specifications is known as *unification*. Furthermore, the unification of two specifications must be a refinement of both, see [3]. Unification can also be used, because of this common refinement, as a method by which to check consistency. To check the consistency of two specifications, we check for contradictions within the unified specification.

The mechanism we describe is a general strategy for unifying two Z specifications. As such it is not specific to any particular ODP viewpoint, nor is it tied to any particular instantiation of the architectural semantics. However, this generality does not reduce its applicability, indeed it is possible that unification can be used to describe an interaction mechanism between descriptions in Z of objects in such a way that is currently not supported by Part 4 of the reference model.

Given a refinement relation, $\sqsubseteq$, defined in a formal specification techniques, we can characterize the unification of two specifications as the least refinement of both, ie:

$$U(T_1, T_2) = \{T : T_1, T_2 \sqsubseteq T \text{ and if } T_1, T_2 \sqsubseteq S \text{ then } T \sqsubseteq S\}$$

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, [12]. Two specifications will thus be consistent if their unification can be implemented [1]. The ability for the unification to be implemented is known as *internal validity*, and for Z specifications this holds when the specification is free from contradictions.

Z is a state based FDT, and a Z specification 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 specification's state and operation schemas. We assume the reader is familiar with the language and refinement relation, introductionary texts include [12, 13, 16].

The unification algorithm we describe is divided into three stages: normalization, common refinement (which we usually term unification itself), and re-structuring. Normalization identifies commonality between two specifications, and re-writes the specifications into normal forms suitable for unification. 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.

2.1 Normalization

Given two different viewpoint specifications of the same (ODP) system, the commonality between the specifications needs to be identified. Clearly, the two specifications that are to be unified have to represent the world in the same way within them (eg 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 co-viewpoint mappings that describe the naming, and other, conventions in force. Once the commonality has been identified, the appropriate elements of the specifications are re-named.

Normalization will also expand data-type and schema definitions into a normal form. The purpose of normalization is to hide the structuring of schemas (which needs to be hidden in order to provide automatic unification techniques) and expand declarations into maximal type plus predicate declarations. For example, normalization of a declaration part of a schema involves replacing every set X which occurs in a declaration $x : X$, with its corresponding maximal type and adding predicates to the predicate part of the schema involved to constrain the variable appropriately.

Normalization also expands schemas defined via the schema calculus into their full form. All schema expressions involving operations from the schema calculus can be expanded to a single equivalent vertical schema. Examples of normalization appear in [12].

2.2 State Unification

The purpose of state unification is to find a common state to represent both viewpoints. The state of the unification must be a data refinement of the state of both viewpoints, since viewpoints represent partial views of an overall system description. Furthermore, it should be the least refinement whenever possible. This is needed to ensure we do not add too much detail during unification because additional detail might add inconsistencies that were not due to inconsistencies in the original viewpoint specifications. Clearly, unification as a consistency checking strategy is more useful if it is also true that an inconsistent unification implies inconsistent viewpoint specifications, rather than just consistent unifications implying consistent viewpoints.

The essence of all constructions will be as follows. If an element x is declared in both viewpoints as $x : T_1$ and $x : T_2$ respectively, then the unification will include a declaration $x : T$ where T is the least refinement of T_1 and T_2. The type T will be the smallest type which contains a copy of both T_1 and T_2. For example, if T_1 and T_2 can be embedded in some maximal type then T is just the union of $T_1 \cup T_2$. The proof of correctness of this unification is given in [2]. If T_1 and T_2 cannot be embedded in a single type then the unification will declare x to be a member of the disjoint union of T_1 and T_2. In these circumstances we again achieve the least refinement of both viewpoints. Lack of space precludes a discussion of this construction here.

Given two viewpoint specifications both containing the following fragment of state description given by a schemas D_1 and D_2, then D represents the unification of the two:

$$\begin{array}{|l} \hline D_1 \\ x : S \\ \hline pred_S \\ \hline \end{array} \qquad \begin{array}{|l} \hline D_2 \\ x : T \\ \hline pred_T \\ \hline \end{array} \qquad \begin{array}{|l} \hline D \\ x : S \cup T \\ \hline x \in S \implies pred_S \\ x \in T \implies pred_T \\ \hline \end{array}$$

whenever $S \cup T$ is well founded. (Axiomatic descriptions are unified in exactly the same manner.) This representation is needed in order to preserve the widest range of possible behaviours.

2.3 Operation Unification

Once the data descriptions have been unified, the operations from each viewpoint need to be defined in the unified specification. Unification of schemas then depends upon whether there are duplicate names. For operations defined in just one of the viewpoint specifications, these are 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, wrt 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, [10]. Given two schemas A and B representing operations, both applicable on some unified state, then

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

represents the unification of A and B, 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.

2.3.1 Example

As an illustrative example we perform state and operation unification on a simple specification of a classroom. The example consists of the state represented by the schema *Class*, and operation *Leave*. The two viewpoint specifications to be unified are:

$$Max : \mathbb{N}$$

$$\begin{array}{|l}\textit{Class}\\ \hline d : \mathbb{P}\{1,2\}\\ \hline \# d \leq Max\\ \hline \end{array}$$

$$\begin{array}{|l}\textit{Leave}\\ \hline \Delta Class\\ p? : \{1,2\}\\ \hline p? \in d\\ d' = d \setminus \{p?\}\\ \hline \end{array}$$

$$Min : \mathbb{N}$$

$$\begin{array}{|l}\textit{Class}\\ \hline d : \mathbb{P}\{2,3,4\}\\ \hline \# d \geq Min\\ \hline \end{array}$$

$$\begin{array}{|l}\textit{Leave}\\ \hline \Delta Class\\ p? : \{2,3,4\}\\ \hline \# d > Min + 1\\ p? \in d\\ d' = d \setminus \{p?, 2\}\\ \hline \end{array}$$

As described above, we first unify the state model, i.e. the schema *Class* in this example, which becomes:

$$\begin{array}{|l}\textit{Class}\\ \hline d : \mathbb{P}\{1,2\} \cup \mathbb{P}\{2,3,4\}\\ \hline d \in \mathbb{P}\{1,2\} \implies \# d \leq Max\\ d \in \mathbb{P}\{2,3,4\} \implies \# d \geq Min\\ \hline \end{array}$$

With this unified state model we can unify the operation *Leave* on this state. To do so we calculate the pre and post-conditions in the usual manner, and for this we need to expand the schema *Leave* into normal form in each viewpoint. This will involve, for example, declaring $p? : \mathbb{N}$ and adding $p? \in \{1,2\}$ as part of the predicate for the description of *Leave* in the first viewpoint. The pre-condition of *Leave* in the first viewpoint is then $p? \in d \cap \{1,2\}$ (in fact this is the part of the pre-condition which is distinct from the pre-condition in the second viewpoint, the rest acting as a state invariant). Hence, the unified *Leave* becomes:

$$\begin{array}{|l}\textit{Leave}\\ \hline \Delta Class\\ p? : \mathbb{N}\\ \hline (p? \in d \cap \{1,2\}) \vee (p? \in d \cap \{2,3,4\} \wedge \# d > Min + 1)\\ (p? \in d \cap \{1,2\}) \implies d' = d \setminus \{p?\}\\ (p? \in d \cap \{2,3,4\} \wedge \# d > Min + 1) \implies d' = d \setminus \{p?, 2\}\\ \hline \end{array}$$

To show that the unified *Leave* is indeed a refinement of *Leave* in viewpoint one we will decorate elements in viewpoint one with a subscript one. We use the retrieve relation

$$
\begin{array}{|l}
\multicolumn{1}{l}{R_1} \\ \hline
Class \\
Class_1 \\ \hline
d_1 \in \{d\} \cap \mathbb{P}\{1,2\} \\ \hline
\end{array}
$$

to describe the refinement between the unified state and the state in the first viewpoint. To demonstrate the refinement is correct, we make the following deductions. Suppose $pre\ Leave_1 \wedge \Delta R_1 \wedge Leave$, we have to show the result of this schema is compatible with $post\ Leave_1$. Now if $pre\ Leave_1$, then $p? \in d_1 \in \{d\} \cap \mathbb{P}\{1,2\}$, and hence $d' = d \setminus \{p?\}$. Then $d_1' \in \{d'\} \cap \mathbb{P}\{1,2\} = \{d \setminus \{p?\}\} \cap \mathbb{P}\{1,2\}$. So $d_1' = d' \cap \{1,2\} = (d \setminus \{p?\}) \cap \{1,2\} = d_1 \setminus \{p?\}$, since by $pre\ Leave_1$, $p? \in \{1,2\}$. The deduction that $pre\ Leave_1 \wedge R_1 \implies pre\ Leave$ is similar. These two deductions complete the proof that the unification is a refinement of viewpoint one. The case for viewpoint two is symmetrical.

3 EXAMPLE - DINING PHILOSOPHERS

To illustrate unification with Z, we shall consider the following viewpoint specifications of the dining philosophers problem. In the dining philosophers problem, [4], a group of N philosophers sit round a table, laid with N forks. There is one fork between each adjacent pair of philosophers. Each philosopher alternates between thinking and eating. To eat, a philosopher must pick up its right-hand fork and then the left-hand fork. A philosopher cannot pick up a fork if its neighbour already holds it. To resume thinking, the philosopher returns both forks to the table.

The three viewpoint specifications we define are the philosophers, forks and tables viewpoints. The philosophers and forks describe individual philosopher and fork objects and the operations available on those objects. The table viewpoint describes a system constructed from those objects and the synchronisation mechanism between operations upon them. We shall then describe the unification of the three viewpoints.

Although this example is not one of an ODP system, it provides a suitable illustration of the issues involved in viewpoint specification and consistency checking.

3.1 The Philosophers Viewpoint

This viewpoint considers the specification from the point of view of a philosopher. A philosopher either thinks, eats or holds her right fork. Note that since the latter is just a state of mind there is no need to describe the operations from a forks point of view at all in this viewpoint. A philosopher object is just defined by the state of the philosopher, and initially a philosopher is thinking.

$$PhilStatus ::= Thinking \mid HasRightFork \mid Eating$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{PHIL} \\ \hline
status : PhilStatus \\ \hline
\end{array}
\qquad
\begin{array}{|l}
\multicolumn{1}{l}{InitPHIL} \\ \hline
PHIL' \\ \hline
status' = Thinking \\ \hline
\end{array}
$$

We can now describe the operations available. A thinking philosopher can pick up its right-hand fork. Philosophers who hold their right fork can begin eating upon picking up their left-hand fork. Finally to resume thinking, a philosopher releases both forks.

$$\begin{array}{l} \textit{GetRightFork} \\ \hline \Delta PHIL \\ \hline status = Thinking \\ status' = HasRightFork \\ \hline \end{array}$$

$$\begin{array}{l} \textit{GetLeftFork} \\ \hline \Delta PHIL \\ \hline status = HasRightFork \\ status' = Eating \\ \hline \end{array}$$

$$\begin{array}{l} \textit{DropForks} \\ \hline \Delta PHIL \\ \hline status = Eating \\ status' = Thinking \\ \hline \end{array}$$

3.2 The Forks Viewpoint

This viewpoint specifies a fork object. Each fork is either free or busy. The fact that the philosopher might change state when a fork is picked up or dropped does not concern forks. The state of the fork is given by a $FORK$ schema, and initially a fork is free.

$$ForkStatus ::= Free \mid Busy$$

$$\begin{array}{l} \textit{FORK} \\ \hline fstatus : ForkStatus \\ \hline \end{array}$$

$$\begin{array}{l} \textit{InitFORK} \\ \hline FORK' \\ \hline fstatus' = Free \\ \hline \end{array}$$

The operations available allow a free fork can be picked up, and both forks can be released.

$$\begin{array}{l} \textit{Acquire} \\ \hline \Delta FORK \\ \hline fstatus = Free \\ fstatus' = Busy \\ \hline \end{array}$$

$$\begin{array}{l} \textit{Release} \\ \hline \Delta FORK \\ \hline fstatus = Busy \\ fstatus' = Free \\ \hline \end{array}$$

3.3 The Tables Viewpoint

This viewpoint has a number of schemas from the other viewpoints as parameters, these are given as empty schema definitions. Upon unification the non-determinism 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:

$$\begin{array}{l} \textit{PHIL} \\ \hline \hline \end{array}$$

$$\begin{array}{l} \textit{InitPHIL} \\ \hline \hline \end{array}$$

$$\begin{array}{l} \textit{GetRightFork} \\ \hline \Delta PHIL \\ \hline \end{array}$$

$$\begin{array}{l} \textit{GetLeftFork} \\ \hline \Delta PHIL \\ \hline \end{array}$$

$$\begin{array}{l} \textit{DropForks} \\ \hline \Delta PHIL \\ \hline \end{array}$$

$$\begin{array}{l} \textit{FORK} \\ \hline \hline \end{array}$$

$$\begin{array}{l} \textit{InitFORK} \\ \hline \hline \end{array}$$

$$\begin{array}{l} \textit{Acquire} \\ \hline \Delta FORK \\ \hline \end{array}$$

$$\begin{array}{l} \textit{Release} \\ \hline \Delta FORK \\ \hline \end{array}$$

The system from the table viewpoint is defined by a collection of fork and philosopher objects:

$$N : \mathbb{N}$$

$$
\begin{array}{|l}
\textit{Table} \\
\hline
\textit{forks} : 1..N \rightarrow \textit{FORK} \\
\textit{phils} : 1..N \rightarrow \textit{PHIL} \\
\hline
\end{array}
$$

Initially the table consists of forks and philosophers all in their respective initial states.

$$
\begin{array}{|l}
\textit{InitTable} \\
\hline
\textit{Table}' \\
\hline
\exists\, \textit{InitFORK}, \textit{InitPHIL} \bullet \operatorname{ran} \textit{forks}' = \{\theta \textit{InitFORK}\} \wedge \operatorname{ran} \textit{phils}' = \{\theta \textit{InitPHIL}\} \\
\hline
\end{array}
$$

Here we use promotion (ie 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 operations on the table, we define a schema $\Phi\, \textit{Table}$ which will allow individual object operations to be defined in this viewpoint. See [13] for a discussion of the use of promotion.

$$
\begin{array}{|l}
\Phi\, \textit{Table} \\
\hline
\Delta\, \textit{Table} \\
\Delta \textit{PHIL} \\
\Delta \textit{FORK} \\
m? : 1..N \\
n? : 1..N \\
\hline
\textit{phils}(n?) = \theta \textit{PHIL} \wedge \textit{phils}' = \textit{phils} \oplus \{\textit{phils}(n?) = \theta \textit{PHIL}'\} \\
\textit{forks}(m?) = \theta \textit{FORK} \wedge \textit{forks}' = \textit{forks} \oplus \{\textit{forks}(m?) = \theta \textit{FORK}'\} \\
\hline
\end{array}
$$

Note that we use two inputs $m?, n?$, because we want to control later the synchronisation between operations on forks and those on philosophers. System operations to get the left and right forks, and to drop both forks can now be defined.

$$\textit{GLF} \mathrel{\widehat{=}} (\Phi\, \textit{Table} \wedge \textit{GetLeftFork} \wedge \textit{Acquire} \wedge [\, n?, m? : 1..N \mid m? = n?\,]) \setminus (\Delta \textit{FORK}, \Delta \textit{PHIL})$$
$$\textit{GRF} \mathrel{\widehat{=}} (\Phi\, \textit{Table} \wedge \textit{GetRightFork} \wedge \textit{Acquire} \wedge [\, n?, m? : 1..N \mid m? = (n? \bmod N + 1)\,]) \setminus (\Delta \textit{FORK}, \Delta \textit{PHIL})$$
$$\textit{DF} \mathrel{\widehat{=}} (\Phi\, \textit{Table} \wedge \textit{DropForks} \wedge \textit{Release} \wedge [\, n?, m? : 1..N \mid m? = n?\,]) \setminus (\Delta \textit{FORK}, \Delta \textit{PHIL})$$

The last schema in each conjunction performs the correct synchronisation between the individual object operations.

3.4 Unifying the Viewpoints

Since the fork and philosopher object descriptions are independent, ie there are no state or operation schemas in common, the unification of these two viewpoints is just the concatenation of the two specifications. We do not re-write that concatenation here.

The Table specification does have commonality with the other two viewpoints. For each state or operation schema defined in two viewpoints (ie the Table and one other), we build one schema in the unification. In fact, the separation and object-based nature (in a loose sense) of this example means that we will not make extensive use of unification by pre- and post-conditions. This is desirable, since it reduces the search for contradictions in the consistency checking phase. In fact, our experiences with viewpoint specifications confirms that such a viewpoint methodology is really only feasible if one adopts this object-based approach.

For example, the schema *FORK* defined in the Table viewpoint is just a parameter from the fork viewpoint, and consequently its unification will just be:

```
┌─ FORK ──────────────────────────
│ fstatus : ForkStatus
└─────────────────────────────────
```

Similarly the unification of *GetLeftFork* from the Table and Philosophers viewpoint is

```
┌─ GetLeftFork ───────────────────
│ ΔPHIL
├──────────
│ status = HasRightFork
│ status' = Eating
└─────────────────────────────────
```

since the pre-condition of *GetLeftFork* in Table is just false. Notice that this provides a mechanism in Z by which to achieve functionality extension across viewpoints in a manner previously not supported.

4 CONSISTENCY CHECKING OF VIEWPOINT SPECIFICATIONS

The unification mechanism can be applied to yield 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.

In addition, we require operation consistency. This is because a conformance statement in Z corresponds to an operation schema(s), [15]. Thus a given behaviour (ie 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 : From the general form of state unification given in Section 2.2, it follows that the 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:

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

So the unification is consistent as long as $(pre\ A \land pre\ B) \Rightarrow (post\ A = post\ B)$.

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.

Example 1 : The classroom

State Consistency : The unified state in this example was given by

$$\begin{array}{|l}
\textit{Class} \\
\hline
d : \mathbb{P}\{1,2\} \cup \mathbb{P}\{2,3,4\} \\
\hline
d \in \mathbb{P}\{1,2\} \implies \#d \leq Max \\
d \in \mathbb{P}\{2,3,4\} \implies \#d \geq Min \\
\hline
\end{array}$$

To show consistency, we need to show that if $d \in \mathbb{P}\{1,2\} \cap \mathbb{P}\{2,3,4\}$, then both $\#d \leq Max$ and $\#d \geq Min$ hold. Suppose the class consisted of just the element 2, i.e. $d = \{2\}$. Both pre-conditions in the unified state, $d \in \mathbb{P}\{1,2\}$ and $d \in \mathbb{P}\{2,3,4\}$, now hold giving the state invariant $Min \leq \#d \leq Max$. Thus the consistency of the viewpoint specifications of the classroom requires that $Min \leq Max$. This type of consistency condition should probably fall under the heading of a *correspondence rule* in ODP, [9], that is a condition which is necessary but not necessarily sufficient to guarantee consistency.

Operation Consistency : In the classroom example, this amounts to checking the operation *Leave* when

$$(p? \in d \cap \{1,2\}) \land (p? \in d \cap \{2,3,4\} \land \#d > Min + 1)$$

In these circumstances, the two post-conditions are $d' = d \setminus \{p?\}$ and $d' = d \setminus \{p?, 2\}$. These two pre-conditions apply when $p? = 2$ and $2 \in d$. A consistency check has to be applied for all possible values of d. For example, let $d = \{1, 2\}$, then $d' = d \setminus \{p?\}$. If further $\# d > Min + 1$, then in addition we have $d' = d \setminus \{p?, 2\}$. These two conditions are consistent (since $p? = 2$) regardless of Max or Min.

Let $d = \{2\}$, then both pre-conditions apply iff $Min < 0$, in which case the post-conditions are $d' = d \setminus \{2\}$ and $d' = d \setminus \{2\}$, and thus consistent.

Hence the two viewpoint specifications are consistent whenever the correspondence rule $Min \leq Max$ holds.

Example 2 : Dining Philosophers

Inspection of the unification in the Dining Philosophers example shows that both state and operation consistency is straightforward (note, however, that with non-object based viewpoint descriptions of this example, consistency checking is a non-trivial task, this points the need for further work on specification styles to support consistency checks). Hence, consistency will follow once we establish the Initialization Theorem for the unification.

The Initialization Theorem for the unification is: $\vdash \exists\, Table' \bullet InitTable$, which upon expansion and simplification becomes

$$\vdash \exists\, forks' : 1..N \rightarrow FORK, phils' : 1..N \rightarrow PHIL \bullet \operatorname{ran} forks' = \{Free\} \wedge \operatorname{ran} phils' = \{Thinking\}$$

which clearly can be satisfied. Hence the viewpoint descriptions given for the dining philosophers are indeed consistent.

5 CONCLUSIONS

The use of viewpoints to enable separation of concerns to be undertaken at the specification stage is a cornerstone of the ODP model. However, the practicalities of how to make the approach work are only beginning to be explored. Two issues of importance are unification and consistency checking. Our work attempts to provide a methodology to undertake unification and consistency checking for Z specifications.

There are still many issues to be resolved, not least the relation to the architectural semantics work. Currently the architectural semantics associates an ODP object with a complete Z specification. Thus the configuration and interactions of objects is then outside the scope of a single Z specification. The architectural semantics comments upon the lack of support for combining Z specifications; we are currently investigating the extent to which unification can provide that support and hence model interaction and communication between Z specifications which represent ODP objects.

Not withstanding this, consistency checking of two Z specifications is still important. It provides a mechanism by which to assess different descriptions of the same object, and will be needed if consistency checking of specifications written in different FDTs is to be achieved. For example, one method would involve translating a LOTOS object into a Z specification (and this type of translation is the extremely challenging part), which could then be checked for consistency via unifying the two Z specifications. Thus the solutions presented in this paper are only part of the whole consistency problem, and much work remains including application to a larger case study.

We are currently funded by the EPSRC and British Telecom to extend our approaches to unification and consistency checking to other formal languages, in particular LOTOS, and to develop tools to support the process.

References

[1] H. Bowman and J. Derrick. Towards a formal model of consistency in ODP. Technical Report 3-94, Computing Laboratory, University of Kent at Canterbury, 1994.

[2] H. Bowman and J. Derrick. Modelling distributed systems using Z. In K. M. George, editor, *ACM Symposium on Applied Computing*, pages 147–151, Nashville, February 1995. ACM Press.

[3] 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.

[4] E. W. Dijkstra. Cooperating sequential processes. In F. Genuys, editor, *Programming Languages.* Academic Press, 1968.

[5] 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.

[6] K. Farooqui and L. Logrippo. Viewpoint transformations. In J. de Meer, B. Mahr, and O. Spaniol, editors, *2nd International IFIP TC6 Conference on Open Distributed Processing*, pages 352–362, Berlin, Germany, September 1993.

[7] 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.

[8] K. Geihs and A. Mann. ODP viewpoints of IBCN service management. *Computer Communications*, 16(11):695–705, 1993.

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

[10] 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.

[11] 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.

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

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

[14] K. A. Raymond. Reference Model of Open Distributed Processing: a Tutorial. In J. de Meer, B. Mahr, and O. Spaniol, editors, *2nd International IFIP TC6 Conference on Open Distributed Processing*, pages 3–14, Berlin, Germany, September 1993.

[15] 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.

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