Viewpoints and consistency: translating LOTOS to Object-Z

John Derrick, Eerke Boiten, Howard Bowman and Maarten Steen
Computing Laboratory, University of Kent, Canterbury, CT2 7NF, UK.
(Phone: + 44 1227 764000, Email: J.Derrick@ukc.ac.uk.)

Abstract

This paper presents a translation between the formal description technique LOTOS and the object-oriented specification language Object-Z. The need for such a translation lies in the use of formal methods in viewpoint specification, and in particular in the Open Distributed Processing standard. The use of viewpoints as a set of partial interlocking specifications brings an obligation to check the consistency of these partial specifications, and to do so we need to relate specifications written in differing languages. The work presented here aims to support the consistency checking of viewpoints written using formal methods by defining a translation from LOTOS to Object-Z. A LOTOS specification describes both an ADT component and a behavioural model, the former is translated into the Z type system, and the behavioural specification is translated into a collection of Object-Z classes where we relate LOTOS actions to operations in the Object-Z specification. A case study is presented which illustrates the translation and consistency checking techniques discussed in the paper.

Key words: Distributed Systems; Open Distributed Processing; Formal methods (Object-Z, LOTOS); Viewpoints; Consistency; Partial Specification.

1 Introduction

In this paper we define a translation between the formal description technique LOTOS [6] and the object-oriented specification language Object-Z [15]. The motivation for deriving such a translation arises from the use of formal methods in viewpoint specification and distributed systems design.

Specification by viewpoints [17] is advocated as a structuring method for the description of complex systems. Each viewpoint represents one perspective of the envisaged system, and thus viewpoints provide a true separation of concerns. In addition, each viewpoint can use a specification language which is dedicated to its particular perspective - acknowledging the generally held belief that no (formal) method applies equally well to all domains of application.

Our motivation for studying viewpoint specification derives from its use in distributed systems design, and in particular in the Open Distributed Processing (ODP) standard [22, 21, 1]. There are five viewpoints, with fixed pre-determined roles, in ODP: *enterprise*, *information*, *computational*, *engineering* and *technology*. Requirements and specifications of an ODP system can be made from any of these viewpoints. For example, the *computational* viewpoint is concerned with the algorithms and data flow of the distributed system function. It represents the system and its environment in terms of objects which interact by transfer of information via interfaces. The *engineering* viewpoint, on the other hand, is more concerned with the distribution mechanisms and the provision of the various transparencies needed to support distribution.

Inherent in any viewpoint approach is the need to check or manage the *consistency* of viewpoints and to show that the different specifications do not impose contradictory requirements [18]. The mechanisms needed to do this depend on the viewpoint languages used, and we have a particular interest in the use of formal techniques because the ODP reference model places an emphasis on the use of formalism. The reference model includes an architectural semantics which describes the application of formal methods to the specification of ODP systems. Of the available notations, state-based languages such as Z are likely to be used for at least the information, and possibly other, viewpoints. Because ODP has adopted an object-based approach to specifying distributed systems, the object-oriented variant of Z, Object-Z, has been advocated as a language that will meet many of the requirements of ODP viewpoint specification [15, 8]. For the computational and engineering viewpoints, LOTOS is a strong candidate in addition to other, less formal, notations.

Because viewpoints overlap in the parts of the system that they describe, in order to check consistency the relationship between the viewpoints needs to be documented. In simple examples these parts will be linked implicitly by having the same name and type in both viewpoints. In general, however, we may need more complicated descriptions for relating common aspects of the viewpoints, such descriptions are called *correspondences* in ODP [22]. A collection of viewpoints can then be defined to be consistent if and only if a common refinement can be found (i.e. a specification that refines all the original viewpoints) with respect to the correspondences between the viewpoints.

The strategy we envisage to check the consistency of one ODP viewpoint written in Object-Z with another written in LOTOS is as follows. First translate the LOTOS specification to an observationally equivalent one in Object-Z, then use the mechanisms defined in [4, 8] to check the consistency of the two viewpoints now both expressed in Object-Z. These mechanisms attempt to find a common refinement of the two viewpoints - if one exists the viewpoints are consistent [1].

The aim of the work described here is to support such a consistency checking mechanism by providing a translation of LOTOS into Object-Z. The background to the problems of consistency checking in ODP, and the motivation for considering these two particular languages is discussed in Section 2. In Section 3 we provide a brief introduction to the languages Object-Z and LOTOS. Section 4 then defines and illustrates the translation between the languages. The ADT component of a LOTOS specification is translated directly into the Z type system. To translate the behavioural aspect of a LOTOS specification, we map each LOTOS process to an Object-Z class. Adopting this approach allows a natural mapping to be identified between many of the behavioural constructs in the two languages, for example, we find that process instantiation in LOTOS corresponds naturally to object instantiation in Object-Z. Section 5 discusses the consistency checking techniques as applied to the case study. Finally, we conclude in Section 6.

2 Background

The objective of ODP 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. 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. A simple interconnection model is not powerful enough for the construction of complex distributed applications. 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.

[1] In fact these mechanisms are defined for Z as opposed to Object-Z. Section 5 discusses why these techniques are also relevant to Object-Z.

The ODP standardization initiative is a response to these issues, and provides a framework for the specification and standardisation of distributed systems.

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. The use of multiple views of a system is one method of achieving a suitable decomposition of a complex design into a manageable form. The ODP Reference Model (RM-ODP) has adopted such a mechanism, and has identified a number of *viewpoints*. The viewpoints enable different participants to observe a system from a suitable perspective and at a suitable level of abstraction [25, 1]. Requirements and specifications of an ODP system can be made from any of these viewpoints.

The set of viewpoints has been chosen so that the resultant specifications together address the complete set of concerns involved in providing a specification of the system. However, as with other viewpoint models [17, 18], the ODP 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 the 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.

Although ODP is a framework for standardization, rather than a design methodology, implementation of standards requires precise and unambiguous interpretations of specifications and standards. For this reason formal methods play an important role within ODP, indeed the reference model states that *The work of the RM-ODP is based on the use, as far as possible, of formal description techniques to give it a clear and unambiguous interpretation* [22]. In support of this, the architectural semantics (given in Part 4 of the reference model) provides an interpretation of ODP modelling concepts which enables viewpoints to be written in a number of formal description techniques (FDTs).

The diversity inherent in a complex framework such as ODP means that a number of different (formal) techniques are applicable to differing aspects of the model, and the choice of which language(s) to use in which viewpoint is a central issue. The available FDTs also offer significant diversity. For example, LOTOS, Estelle [20] and SDL [9] are targeted at issues of explicit concurrency and interaction (specifying ordering and synchronisation of abstract events). On this basis LOTOS is a strong contender for use in the computational viewpoint. In contrast, model based techniques such as Object-Z, Z [30] and VDM [23] describe specifications in terms of data state change, and are particularly suited to use in the enterprise and information viewpoints. In addition, the approach taken in the reference model 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 has led to interest in the use of object-based specification languages for use within ODP viewpoints, and Object-Z is a leading candidate for use in the information viewpoint. However, it should be noted that none of these FDTs fully address the specification requirements of modern distributed processing and Open Distributed Processing in particular [8]. Therefore to use FDTs effectively within ODP, specific languages are used within particular viewpoints. For the potential of ODP to be fully exploited it is therefore necessary to provide a mechanism to support consistency checking across viewpoints written in those FDTs.

The use of multiple viewpoints is not unique to ODP, and different approaches use different mechanisms by which to assess consistency. Here we take a constructive view of consistency that is oriented towards system development and define a collection of viewpoints to be consistent if and only if a common refinement can be found (i.e. a specification that refines all the original viewpoints) with respect to the correspondences between the viewpoints. The *least* such common refinement of two viewpoints is known as their *unification*. Such a unification of two viewpoints has all the requirements imposed by both viewpoints, however, it imposes no extra requirements besides those contained in the first two viewpoints (or else consistency checking with yet another viewpoint might unnecessarily fail). Because of this property, finding unifications for pairs of viewpoints is a constructive way of establishing consistency.

Elsewhere (e.g. [4, 5, 8]) we have described how a unification of two viewpoints can be constructed when they have been specified in Z. A particularly important aspect of this work was to locate two conditions which were sufficient to ensure consistency of the two Z viewpoints, and therefore to allow automation of as much as possible of the unification process. For that reason we have developed a range of Z unification tools together with theorem proving support [2]. The consistency conditions can be automatically generated from a Z unification tool and fed into a theorem prover. Given that the complexity and structure of the consistency conditions are almost exclusively determined by the predicates that occur in the viewpoint specifications [5], the existing methods for automated theorem proving in Z (e.g. [24, 7]) can be used to discharge these consistency conditions. The extension of this work to object based languages has been considered in [14], which discusses consistency checking in object oriented variants of Z.

The work reported in this paper provides a translation between LOTOS and Object-Z, and by combining this translation mechanism with the above Z unification techniques we aim to support the consistency checking of one ODP viewpoint written in Object-Z with another written in LOTOS as follows. First translate the LOTOS specification to an observationally equivalent one in Object-Z, then check the consistency of the two viewpoints now both expressed in the same language. These mechanisms attempt to find a common refinement of the two viewpoints - if one exists the original viewpoints were consistent. In Section 4 we will illustrate this approach through a simple case study.

Although the motivation for this work arises from the use of viewpoints in the ODP reference model, it should be noted that both the translation algorithm and consistency checking techniques are generic and that they can be applied in an arbitrary viewpoint framework [3]. Indeed it should be stressed that our motivation in this paper is to present our work on translation, and therefore the partial specifications used in our running example are not ODP viewpoints but small fragments of behaviour which serve to illustrate the translation process. Elsewhere we have considered issues specifically arising from ODP [4, 5].

3 The Languages Object-Z and LOTOS

3.1 Object-Z

Object-Z is an object-oriented extension of the specification language Z, which has been developed over a number of years and is perhaps the most mature of all the proposals to extend Z in an object-oriented fashion. It has been advocated as one of the languages suitable for use in the ODP viewpoints, particularly in respect of the information viewpoint of the reference model.

Object-Z uses a class schema to encapsulate a state schema together with the operations acting upon that state. It is represented as a named box with zero or more generic parameters. The class schema may include local type or constant definitions, at most one state schema and initial state schema together with zero or more operation schemas. A class may also inherit a number of other classes. The local type and constant definitions of an inherited class are available in the inheriting class. The schemas of an inherited class are either implicitly available or implicitly conjoined with identical named schemas of the inheriting class.

A simple example of an Object-Z class is given by the following:

M

$$count, value1, value2 : \mathbb{N}$$

$\textit{INIT}$

$$\Delta(count)$$

$$count' = 0$$

a

$$\Delta(count, value1)$$
$$input? : \mathbb{N}$$

$$count = 0 \wedge count' = 1$$
$$value1' = input?$$

b

$$\Delta(count, value2)$$
$$input? : \mathbb{N}$$

$$count = 1 \wedge count' = 2$$
$$value2' = input?$$

c

$$output! : \mathbb{N}$$

$$count = 2$$
$$output! = max(value1, value2)$$

The variables $count, value1$ and $value2$ declared in the (unnamed) state schema are local to the class. The initial state schema $\textit{INIT}$ defines the initial values of the variables in the state schema. The class specified above has three operations: a, b and c. The operations in this class allow two integers to be inserted (using a and b), and the operation c will output the maximum of those values. Names ending in a ? denote input, and those ending in a ! denote output. Primes ($'$) are used to denote the value of a state variable after an operation has occurred.

Each operation has a Δ-list which contains those state variables which may change when the operation is applied to an object of that class. An operation does not change the state variables that are not listed in its Δ-list. Hence the operation a implicitly contains a predicate $value2 = value2'$. The preconditions of the operations force them to be invoked in a particular order. The interpretation of operations in an Object-Z class differs from that in Z, in that an Object-Z operation cannot occur outside its precondition[2]. This interpretation of operation preconditions is crucial for the correctness of the translation defined in this paper.

The behaviour of the class M is best illustrated by looking at a simple transition system representation of it, see figure 1.

A class can also include instances, i.e. objects, of other classes as state variables. This allows the concise specification of the interaction between components of a system. For example,

[2] In Z, an operation is undefined (but enabled) outside its precondition.

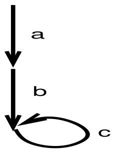

Figure 1: The behaviour of the class M

OZMax3

$A, B : M$

INIT

$A.\mathit{INIT}, B.\mathit{INIT}$

$in1 \mathrel{\widehat=} A.a$
$in2 \mathrel{\widehat=} A.b$
$in3 \mathrel{\widehat=} B.b$
$out \mathrel{\widehat=} B.c$
$i \mathrel{\widehat=} (A.c \mathbin{||} B.a[output?/input?])$

specifies a class with two state variables, A and B, which are objects of the class M. Initially the objects are in their initial state. The objects have operations applied to them using the dot notation, this notation is made precise in the semantics defined in [27]. Informally we can view references to objects as follows. If C is a class the the declaration $c : C$ declares c to be a variable whose value is a reference to an object of class C. Then $c.\mathit{INIT}$ is a predicate which denotes whether the object c conforms to C's initial state schema. The operation $c.Op$ transforms the object referenced by c according to the definition of the operation Op defined in the class C.

The operation i represents an *internal* operation, i.e. one which can be invoked by the object whenever the precondition of i holds, but which cannot be controlled externally[3]. The semantics of internal operations is identical to observable operations, however, weak bisimulation equivalences [26] defined over the semantics will treat internal operations differently to observable operations.

Object-Z provides additional schema operators to those defined in Z. The parallel operator $||$ enables communication between objects to be specified, it behaves like conjunction but also equates inputs and outputs with the same basename [15]. To define the operator, meta-functions $\beta_?$ and $\beta_!$ are used which return the basenames (i.e. apart from the ? and !) of the inputs and outputs respectively. The operation $Op_1 \| Op_2$ is then defined as[4]

$$[Op_1][y_1!/y_1?, \ldots, y_m!/y_m?] \wedge [Op_2][x_1!/x_1?, \ldots, x_n!/x_n?]$$

where $\beta_!(Op_1) \cap \beta_?(Op_2) = \{x_1, \ldots, x_n\}$ and $\beta_?(Op_1) \cap \beta_!(Op_2) = \{y_1, \ldots, y_m\}$.

For example, in the synchronisation $(A.c \mathbin{||} B.a[output?/input?])$ we have relabelled *input*? to *output*? in $B.a$. The effect of the parallel composition then specifies that communication takes place between the two objects A and B.

The notation $Op_1 \bullet Op_2$ denotes enrichment, in that the schema text of Op_1 enriches the environment in which Op_2 is interpreted. That is $Op_1 \bullet Op_2 = [Op_1;\ d \mid p]$ when $Op_2 = [d \mid p]$ and the

[3] Not all versions of Object-Z define and use internal operations in the same way, here we use a distinguishing name to denote such operations.

[4] In this paper we extend the definition of $||$ with the convention that, in the presence of any type clashes of common variables, $Op_1 \| Op_2$ is defined to have predicate *false*.

free variables of d do not include any variables declared in Op_1.

3.2 LOTOS

A LOTOS [6] specification of a system defines the temporal relationships among the interactions that constitute the externally observable behaviour of the system. A specification consists of two parts: the *behaviour expression* describes the process behaviour and its interaction with the environment whilst the *abstract data type* (ADT) describes the data structures and value expressions used within the behaviour expression. Basic LOTOS refers to the subset of (full) LOTOS that considers only the temporal aspects of behaviour without value passing or the ADT component.

A simple example of a LOTOS specification is given by the following:

Specification Max3 [in1, in2, in3, out] : **noexit**
type natural **is**
  sorts *nat*
  opns $0 :\rightarrow nat$
    $succ : nat \rightarrow nat$
    $largest : nat, nat \rightarrow nat$
  eqns
    forall $x, y : nat$
    ofsort *nat*
      $largest(0, x) = x;$
      $largest(x, y) = largest(y, x);$
      $largest(succ(x), succ(y)) = succ(largest(x, y));$
endtype

  behaviour
    hide *mid* **in** (Max2[$in1, in2, mid$] $|[mid]|$ Max2[$mid, in3, out$])
  where
    process Max2[a, b, c] : **noexit** :=
      $a?x : nat;\ b?y : nat;\ c!largest(x, y);\ stop$
      []
      $b?y : nat;\ a?x : nat;\ c!largest(x, y);\ stop$
    endproc
  endspec

This specification defines a four gate process that accepts three natural numbers at three input gates (in any order), and then offers the largest of them at an output gate. A specification or process behaviour expression is built by applying operators to other behaviour expressions. A behaviour expression may also include instantiations of other processes (e.g. Max2), whose definitions are provided in the where clause of the process definition. The *terminals* of a behaviour expression are the occurrences of the processes *stop*, *exit* or process instantiations (including recursion) within that expression.

The (atomic) observable interactions that a process may engage in are called the *events* or *actions* of that process. An event is thought of as occurring at an interaction point, or *gate*, and in the absence of data passing, the event and gate names coincide. In the above specification the system may interact with its environment via gates *in1, in2, in3, out.* Since *mid* is hidden it does not appear in the gate list. Hidden events give rise to unobservable actions (denoted i). Furthermore, the unobservable (or internal) action i is also user-definable, in that it can appear directly in a specification, and is used to model the potential non-determinism of a given system. There also exists a special action δ, which is not user-definable, but whose occurrence indicates the successful termination of a process (and can be used to enable a subsequent process).

The basic data type specification, as illustrated above, consists of a signature and, possibly, a list of equations. The equations used here define the natural numbers and the function *largest* is used to return the maximum of two integer values. A labelled transition system can be used to illustrate the behaviour (see figure 2) of *Max2*.

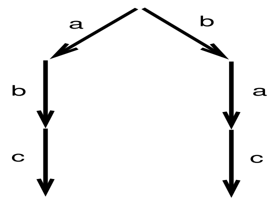

Figure 2: The behaviour of the process $Max2$

The translation we define in this paper is verified against a common semantic model of the two languages. This model is based upon the semantics for Object-Z described in [27], which effectively defines a state transition system for each Object-Z specification. By embedding the standard labelled transition system semantics for LOTOS into it in an obvious manner we can use it as a common semantic basis for the two languages. With such a common semantic model the translation can be verified correct by showing that a LOTOS specification and its Object-Z translation are bisimular as labelled transition systems. The details of this verification are provided in [11].

4 The Translation from LOTOS to Object-Z

In this section we define the translation from LOTOS to Object-Z. The ADT component of a LOTOS specification is translated directly into the Z type system (see Section 4.2 below). To translate the behavioural aspect of a LOTOS specification, we note that there is a strong correlation between classes in object-oriented languages and processes in concurrent systems [32, 19, 27]. We use this correlation as the basis for the translation of the behaviour (which is described in Sections 4.1 and 4.3 below), and map a LOTOS process to an Object-Z class. Adopting this approach allows a natural mapping to be identified between many of the behavioural constructs in the two languages, for example, we find that process instantiation in LOTOS corresponds naturally to object instantiation in Object-Z.

To map a LOTOS process to an Object-Z class we will relate their observable atomic actions, i.e. events in LOTOS and operations in Object-Z. Therefore the translation will map each LOTOS action into an equivalent Object-Z operation schema. For example, a LOTOS specification containing the behaviour $in?x : nat;\ out!(x+2);\ stop$ will be translated into an Object-Z class which contains operation schemas with names *in* and *out*. The Object-Z operation schemas have appropriate inputs and outputs to perform the value passing defined in the LOTOS specification. In addition, each operation schema includes a predicate to ensure that it is applicable in accordance with the temporal behaviour of the LOTOS specification.

We begin the translation by illustrating how specifications are turned into a number of Object-Z classes, each one representing a behaviour expression of the LOTOS specification. Section 4.3 contains the heart of the translation where the translation of a LOTOS behaviour expression is defined. We illustrate the translation algorithm by translating the LOTOS specification *Max3* given in Section 3.2. This will then be checked for consistency with the Object-Z specification described in Section 3.1.

4.1 Specifications

Consider the LOTOS specification *Max3* which contains a type definition and a behaviour:

Specification Max3 [in1, in2, in3, out] : **noexit**
type
    type definition
endtype

behaviour
    hide mid **in** (Max2[$in1, in2, mid$] $\mid [mid] \mid$ Max2[$mid, in3, out$])
where
    process Max2[a, b, c] : **noexit** :=
        $a?x : nat;\ b?y : nat;\ c!largest(x, y);\ stop$
        []
        $b?y : nat;\ a?x : nat;\ c!largest(x, y);\ stop$
    endproc
endspec

This is translated to an Object-Z specification consisting of a translation of the type definition together with a number of Object-Z classes representing the behaviour and process definitions.

One class will represent the behaviour of *Max2*, and another class will represent the overall behaviour **hide** mid **in** (Max2[$in1, in2, mid$] $\mid [mid] \mid$ Max2[$mid, in3, out$]). This latter class will contain objects of type *Max2* which will correspond to the process instantiations in the behaviour. We begin, however, by translating the data types.

4.2 Translation of Data Types

The type definition in the specification is translated directly into the Object-Z type system. LOTOS data types are specified using the language for abstract data types ACT ONE [16]. ACT ONE is an algebraic specification method to write parameterized as well as unparameterized ADT specifications.

The type specification in *Max3* illustrates the most basic form of data type specification in LOTOS consists of a signature and, optionally, a list of equations. Its translation will introduce a given set to represent the sorts (here nat), together with an axiomatic definition which introduces the operations constrained by the behaviour of the equations[5]. Thus we translate the above to:

$$[nat]$$

[5]Strictly speaking we would also have to include predicates ensuring that we restrict the models of the Z axiomatic definition to the initial ones only. An alternative approach would be to code the ADT definitions directly as Z free types, which by definition are restricted to the initial models. So in this example the first two definitions would be replaced by the free type declaration

$$nat ::= zero \mid succ(nat)$$

together with an axiomatic definition for $+$.

$$
\begin{array}{|l}
0 : nat \rightarrow nat \\
succ : nat \rightarrow nat \\
largest : nat \times nat \rightarrow nat \\
\hline
\forall x, y : nat \bullet \\
\quad largest(0, x) = x \\
\quad largest(x, y) = largest(y, x) \\
\quad largest(succ(x), succ(y)) = succ(largest(x, y))
\end{array}
$$

Notice that in the translation of nullary operations (ie constants), we remove the arrow, as in $\rightarrow nat$. The commas in an n-ary operation are replaced by $\times$ in the Z translation. The **ofsort** *nat* is superfluous in the Z specification.

Any realistic consistency checking toolbox will also contain direct translations from axiomatic descriptions of standard structured types (e.g. sets and sequences) into their Z mathematical toolbox (cf. [30]) equivalents. We will assume that this translation has indeed been made in this example (and hence identify *nat* and $\mathbb{N}$).

LOTOS also allows extensions and combinations of type specification by importing a reference to a type definition and possibly enriching it with additional sorts, operations and equations. The translation of such a type definition in Z consists of the translation of the imported definition together with a translation of the enriching sorts, operations and equations as given sets together with an axiomatic definition. In LOTOS, parameterized data type specifications represent generic specifications which can be instantiated later. Such a parameterized type is translated to a generic data type in Z. It is not possible to model type renaming within the Z type system at this level of abstraction. However, a translation of a LOTOS specification using type renaming can be found by first re-writing the LOTOS specification into one where the renamings have already been carried out and then translating into Z.

4.3 Process definitions and behaviour expressions

To translate a process definition we first translate its behaviour expression into an Object-Z class by successively applying the rules given below, working bottom up beginning with the LOTOS terminals, until each operator/terminal has been translated. The variables introduced in a class' state schemas are assumed to be unique with respect to other state variables introduced during the translation of a process. We also assume the existence of a boolean type *bool*.

4.3.1 Translating the process *Max2*

To translate the behaviour of *Max2* we first note that the terminals are the two instances of *stop*. After translation of the terminals, the rules for action prefix will be applied, and finally the rule for choice is used. We begin with the branch $a?x : nat;\ b?y : nat;\ c!largest(x, y);\ stop$. The translation rule for *stop* is the following:

1. Inaction. $B = stop$ translates to the Object-Z class

$$
\begin{array}{|l}
\hline
\textbf{B} \\
\quad \begin{array}{|l} \hline s : bool \\ \hline \end{array} \\
\quad \begin{array}{|l} \hline \textit{INIT} \\ \Delta(s) \\ \hline \neg\, s' \\ \hline \end{array} \\
\hline
\end{array}
$$

That is the translation maps a LOTOS process that cannot engage in any action to an Object-Z class with no operations. Both will therefore deadlock. □

Continuing with the behaviour under consideration we have to translate $c!largest(x, y);\ stop$ using the rule for action prefix together with the translation of $stop$ as the simple class given above. The action prefix rule is the following:

2. Action prefix. Let $B[a, Op_1, \ldots, Op_n] = a\ ?x : T\ !E\ [pred];\ P$. The occurrence of $pred$ here is to act as a selection predicate, i.e., the action is offered precisely when $pred$ evaluates to true. Then (assuming P has already been translated into an Object-Z class) B translates to the class

$$
\begin{array}{|l}
\hline
\textbf{B} \\
\quad \begin{array}{|l} \hline P.\textit{STATE} \\ t : bool \\ x : T \\ \hline \end{array} \\
\quad \begin{array}{|l} \hline \textit{INIT} \\ \Delta(t) \\ \hline t' = true \wedge P.\textit{STOP} \\ \hline \end{array} \\
\quad a \mathrel{\widehat=} ([\Delta(t, x), ch_1? : T, ch_2! : U \mid t \wedge \neg\, t' \wedge ch_2! = E \wedge x' = ch_1? \wedge pred[x'/x]]\ ⨟\ P.\textit{INIT}) \vee P.a \\
\quad Op_1 \mathrel{\widehat=} P.Op_1 \\
\quad \vdots \\
\quad Op_n \mathrel{\widehat=} P.Op_n \\
\hline
\end{array}
$$

where $P.\textit{STATE}$ denotes the state schema of the class P, and $U = type(E)$, and $B.\textit{STOP}$ is a schema that ensures no operation in B will be enabled.

The temporal ordering defined in the LOTOS behaviour offers action a followed by the ordering defined by P. The translation simulates the same behaviour by using a boolean state variable, t say, and the Object-Z translation of P. Initially, t is true (so the precondition of a holds) but every operation in P is disabled (through $P.\textit{STOP}$). After a occurs that portion of behaviour is disabled ($\neg\, t'$), but operations in P are now enabled ($P.\textit{INIT}$ holds). All operations in the class P are promoted to the class B ($Op_i \mathrel{\widehat=} P.Op_i$) to make them available. In addition, P may contain further occurrences of the operation a, these should be available once the initial a is performed, hence we disjoin $P.a$ to the definition of the operation a.

The LOTOS value and variable declarations are simulated by the input, output and state variables in the Object-Z class. The rule presented here generalises to an arbitrary number of variable and value declarations in the obvious manner. □

In our example the action is $c!largest(x, y)$, with this action prefix and using the current translation of $stop$ the rule produces the following Object-Z fragment.

$$
\begin{array}{|l}
\hline
\begin{array}{|l}
\hline
s_2, s_3 : \mathit{bool} \\
\hline
\end{array} \\
\\
\textit{INIT} \\
\begin{array}{|l}
\hline
\Delta(s_2, s_3) \\
\hline
s_2' \land \neg\, s_3' \\
\hline
\end{array} \\
\\
c \\
\begin{array}{|l}
\hline
ch! : \mathbb{N} \\
\Delta(s_2, s_3) \\
\hline
s_2 \land \neg\, s_2' \land \neg\, s_3' \\
ch! = \mathit{largest}(x, y) \\
\hline
\end{array} \\
\hline
\end{array}
$$

It is easy to see that this Object-Z class has the same behaviour as $c!\mathit{largest}(x, y);\ \mathit{stop}$. Notice that the specification contains undeclared variables (x, y here) until the complete behaviour has been translated, eventually the LOTOS variable declarations will introduce state variables into the class.

The subsequent two applications of action prefix are similar (but this time with inputs) and the result is the following:

$$
\begin{array}{|l}
\hline
\begin{array}{|l}
\hline
x, y : \mathbb{N} \\
s_0, s_1, s_2, s_3 : \mathit{bool} \\
\hline
\end{array} \\
\\
\textit{INIT} \\
\begin{array}{|l}
\hline
\Delta(s_0, s_1, s_2, s_3) \\
\hline
s_0' \land \neg(s_1' \lor s_2' \lor s_3') \\
\hline
\end{array} \\
\\
a \mathrel{\widehat=} [\Delta(x, s_0, s_1, s_2, s_3), ch? : \mathbb{N} \mid \\
\qquad s_0 \land s_1' \land \neg(s_2' \lor s_3' \lor s_0') \land x' = ch?] \\
b \mathrel{\widehat=} [\Delta(y, s_0, s_1, s_2, s_3), ch? : \mathbb{N} \mid \\
\qquad s_1 \land s_2' \land \neg(s_0' \lor s_1' \lor s_3') \land y' = ch?] \\
c \mathrel{\widehat=} [ch! : \mathbb{N}, \Delta(s_2, s_3) \mid s_2 \land \neg\, s_3' \land \neg\, s_2' \land ch! = \mathit{largest}(x, y)] \\
\hline
\end{array}
$$

The translation of the other branch $b?y : nat;\ a?x : nat;\ c!\mathit{largest}(x, y);\ \mathit{stop}$ is similar and produces another class with three operations a, b, c, but this time b will be enabled initially.

We now apply the rule for choice which, in general, is given by

3. Choice. $B[Op_1, \ldots, Op_n] = P[\,]Q$ translates to the Object-Z class

B

$P.\mathit{STATE}$
$Q.\mathit{STATE}$

INIT
$P.\mathit{INIT}$
$Q.\mathit{INIT}$

$Op_1 \mathrel{\widehat=} (P.Op_1 \wedge Q.\mathit{STOP}) \vee (Q.Op_1 \wedge P.\mathit{STOP})$
$\vdots$
$Op_n \mathrel{\widehat=} (P.Op_n \wedge Q.\mathit{STOP}) \vee (Q.Op_n \wedge P.\mathit{STOP})$

The translation of choice makes a copy of both P and Q available in the Object-Z class. Initially, all operations from P and Q are available since both $P.\mathit{INIT}$ and $Q.\mathit{INIT}$ hold. However, once an operation in one branch of the choice is invoked ($P.Op_1$ say), operations from the other branch will be disabled ($\ldots \wedge Q.\mathit{STOP}$). This ensures that initially a choice is available between operations from P and Q, but that once that choice is resolved operations from only one class are available. (We have adopted the obvious convention in this paper that if Op is not in the class Q then $Q.Op$ is taken to be false.) This successfully mimics the choice specified in the LOTOS behaviour. □

Our example contains a choice between $a?x : nat;\ b?y : nat;\ c!largest(x,y);\ stop$ and $b?y : nat;\ a?x : nat;\ c!largest(x,y);\ stop$. Each branch has been translated into an Object-Z class, the first is given above, the second is similar (except uses a different boolean variable, t, say). Applying the choice rule will combine the two classes, so that initially the first operation from each class is enabled, but subsequently only operations from one of the branches will be enabled. The actual translation is mechanical, and after some simplification it results in the following

$Max2$

$x, y : \mathbb{N}$
$s : s_0 \mid s_1 \mid s_2 \mid s_3 \mid t_1 \mid t_2 \mid t_3$

INIT
$\Delta(s)$

$s' = s_0$

$a \mathrel{\widehat=} [\Delta(x,s), ch? : \mathbb{N} \mid (s = s_0 \wedge s' = s_1 \vee s = t_1 \wedge s' = t_2) \wedge x' = ch?]$
$b \mathrel{\widehat=} [\Delta(y,s), ch? : \mathbb{N} \mid (s = s_1 \wedge s' = s_2 \vee s = s_0 \wedge s' = t_1) \wedge y' = ch?]$
$c \mathrel{\widehat=} [\Delta(s), ch! : \mathbb{N} \mid (s = s_2 \wedge s' = s_3 \vee s = t_2 \wedge s' = t_3) \wedge ch! = largest(x,y)]$

This completes the translation of the process *Max2*.

4.3.2 Translating the process *Max3*

To translate the *behaviour* of *Max3*, we first note that it contains two instantiations of the process *Max2*, whose definition is given in the **where** clause of *Max3*. The Object-Z translation will thus contain the definition of the class *Max2* followed by that of *Max3*. The terminals in the behaviour

hide *mid* **in** (Max2[$in1, in2, mid$] $|[mid]|$ Max2[$mid, in3, out$])

are the two process instantiations, and we use the following rule.

4. Instantiation. Let $B[a_1, \ldots, a_n] = P[a_1, \ldots, a_n](E_1, \ldots, E_m)$, where $P[g_1, \ldots, g_n](x_1 : t_1, \ldots, x_m : t_m)$ is defined elsewhere. This translates to the Object-Z class (where the identifier A is unique in B)

B

$A : P$

$A.x_1 = E_1 \wedge \ldots \wedge A.x_m = E_m$

INIT

$A.\mathit{INIT}$

$a_1 \mathrel{\widehat=} A.g_1$

$\vdots$

$a_n \mathrel{\widehat=} A.g_n$

Process instantiation therefore has a natural counterpart in Object-Z as object instantiation. The identifier used is chosen to be unique because for each process instantiation a new object is instantiated. The substitution of actual gate names for formal gate names is achieved in the translation by operation renaming and promotion ($a_1 \mathrel{\widehat=} A.g_1$). The replacement of the parameter list $x_1, \ldots, x_m$ by value expressions $E_1, \ldots, E_m$ is represented as a predicate equating the variables in the object instantiation to its value ($A.x_1 = E_1 \wedge \ldots$). □

In our example, we have two process instantiations each with a different gate set. Each process instantiation is translated to an object instantiation and gate sets become operation name relabellings upon promotion (e.g. $in1 \mathrel{\widehat=} A.a$). Thus the process instantiations become:

$A : Max2$

INIT

$A.\mathit{INIT}$

$in1 \mathrel{\widehat=} A.a$
$in2 \mathrel{\widehat=} A.b$
$mid \mathrel{\widehat=} A.c$

$B : Max2$

INIT

$B.\mathit{INIT}$

$mid \mathrel{\widehat=} B.a$
$in3 \mathrel{\widehat=} B.b$
$out \mathrel{\widehat=} B.c$

Subsequently we need to translate the parallel composition (Max2$[in1, in2, mid]$ $|[mid]|$ Max2$[mid, in3, out]$) using the following.

5. Parallel composition. $B[Op_1, \ldots, Op_n] = P \mid [G] \mid Q$ translates to the Object-Z class

B

$P.\mathit{STATE}$
$Q.\mathit{STATE}$

INIT
$P.\mathit{INIT}$
$Q.\mathit{INIT}$

$Op_1 \mathrel{\widehat=} (P.Op_1 \lor Q.Op_1)$
$\vdots$
$Op_2 \mathrel{\widehat=} (P.Op_2 \parallel Q.Op_2)$
$\vdots$

where an operation schema definition appears for each operation Op in the gate list of B, and takes the form of that of Op_1 if $Op \notin G \cup \{\delta\}$, and takes the form of that of Op_2 if $Op \in G \cup \{\delta\}$.

The translation of the parallel composition $B[\ldots] = P \mid [G] \mid Q$ defines an Object-Z class with operations whose behaviour depends on whether the associated action is in G. If it is not, no synchronisation occurs, and therefore the translation offers a straight choice between, say, $P.Op_1$ and $Q.Op_1$. If it is in G, then the operation can occur precisely when it occurs in both P and Q. This is achieved by using the Object-Z parallel operator between the two operations, e.g., $P.Op_2 \parallel Q.Op_2$. The full LOTOS value passing synchronisation aspects are also preserved with this operator. □

In our example the only synchronisation is on gate *mid*. All other operations are simply included in the translated Object-Z class because they occur in just one of the original classes. The synchronisation of *mid* is defined as $mid \mathrel{\widehat=} (A.c \parallel B.a)$. Finally, *mid* is hidden, and we use the rule.

6. Hiding. $B[Op_1, \ldots, Op_m] = \text{hide } g_1, \ldots, g_n \text{ in } P$ translates to the Object-Z class

B

$P.\mathit{STATE}$

INIT
$P.\mathit{INIT}$

$i \mathrel{\widehat=} (P.g_1) \setminus (\text{inouts } g_1) \lor \ldots \lor (P.g_n) \setminus (\text{inouts } g_n)$
$Op \mathrel{\widehat=} P.Op$

where an operation schema definition of the form $Op \mathrel{\widehat=} P.Op$ appears for each operation $Op \in \{Op_1, \ldots, Op_m\} \setminus \{g_1, \ldots, g_n\}$, and (inouts g_i) is the list of all input and output parameters of the schema g_i.

Hiding in the context of LOTOS transforms the hidden observable actions of a process into unobservable actions. In the presence of value passing the data is also hidden. In the Object-Z class the hiding of actions is represented by the change of operation name ($i \mathrel{\widehat=} (P.g_1) \ldots$), and data hiding by hiding both the inputs and outputs ($\ldots \setminus (\text{inouts } g_1)$). □

Thus the hiding rule in this context causes *mid* to be renamed with the distinguished operation name i. Hence we finally derive the class representing *Max3* as specified below.

$$
\begin{array}{|l}
\hline
\textit{Max3} \\
\quad \begin{array}{|l|}\hline A, B : Max2 \\ \hline \end{array} \\
\quad \begin{array}{|l}\hline \text{\textit{INIT}} \\ A.\text{\textit{INIT}}, B.\text{\textit{INIT}} \\ \hline \end{array} \\
\quad in1 \mathrel{\widehat=} A.a \\
\quad in2 \mathrel{\widehat=} A.b \\
\quad in3 \mathrel{\widehat=} B.b \\
\quad out \mathrel{\widehat=} B.c \\
\quad i \mathrel{\widehat=} (A.c \parallel B.a) \\
\hline
\end{array}
$$

The remaining rules for LOTOS constructs which we have not considered here are given in full in [11]. Although the general formulation of some of the rules may seem complex at first, the translation process can be automated. Indeed, heuristics can be built for commonly occuring fragments of behaviour (e.g. a sequence of action prefixes as in the example above) so that little simplification is in fact needed.

5 Checking the viewpoints for Consistency

Although the main emphasis of this paper is an illustration of the translation mechanism, we complete the picture in this section by showing how we can check viewpoints for consistency once they have been translated into the same language.

The translated LOTOS specification can be compared with the Object-Z specifications of M and *OZMax3* given in Section 3.1, and we apply the consistency checking techniques as described in [4, 5]. To show that two viewpoint specifications are consistent we need to show that there exists a common refinement of the two specifications with respect to the correspondences between the viewpoints, where the correspondences document the overlap or commonality between the viewpoints. We begin by identifying the correspondences between the viewpoints.

At an object level we can identify certain classes. The class $Max3$ and *OZMax3* will be implemented as one component in the final system, and we can identify operations with identical names (e.g., the operations $in1$ in the two viewpoints represent partial specifications of the same event). We can also identify M and $Max2$ as representing the same class. Finally, it is clear that *count* in class M represents information that is also represented by the state variable s in the class $Max2$. However, unlike the other correspondences this is not a matter of simply identifying these components, and we note that the relation $R \subseteq \mathbb{N} \times \{s_0, s_1, s_2, s_3, t_1, t_2, t_3\}$ which relates their values is given by $\{(0, s_0), (1, s_1), (2, s_2), (2, s_3)\}$.

Having identified the relationship between the two viewpoints we now construct a least refined specification of the two viewpoints, i.e., a specification which is a refinement of both original viewpoints. This unification we build will depend, therefore, on the particular refinement relation used to construct it. Here we will use the standard Z refinement relation for state-based systems as described in [30, 31].

The standard Z refinement relation allows an operations precondition to be weakened upon refinement and for the operations postcondition to be strengthened. In Object-Z the precondition of an operation represents its guard, whereas in Z an operation is enabled but undefined outside its precondition. It is natural, therefore, for refinement of a Z operation to allow weakening of its precondition, but usually this is not allowed for a refinement of an Object-Z operation, i.e. the precondition of a refined operation in Object-Z must be identical to the original precondition.

However, in the context of partial specifications an operation represents only a partial description of its full specification, therefore it is natural (and indeed desirable) to allow a weakening of preconditions upon refinement when constructing the unification of two operations. We will therefore use the standard Z refinement relation which allows weakening of preconditions in the construction of unifications in Object-Z.

Given that the structure of $Max3$ and *OZMax3* are identical, it follows that we just have to consider whether we can find a common refinement of M and *Max2*; if we can the two viewpoints overall will be consistent [6]. The unification of M and *Max2* is constructed in two phases. In the first phase, a unified state schema for the two viewpoints has to be constructed, and this relies on the correspondences between the two viewpoints. The viewpoint operations are then adapted to operate on this unified state. At this stage we have to check that a condition called *state consistency* is satisfied. In the second phase, called *operation unification*, pairs of adapted operations from the viewpoints which are linked by a correspondence have to be combined into single operations on the unified state. This also involves a consistency condition (*operation consistency*) which ensures that the unified operation is a refinement of the viewpoint operations.

We build the unified state space using a totalisation of the relation R (for details of the totalisation of a relation see [4]), we then adapt the operations of each viewpoint to make them operate on the unified state.

An algorithm described in [4, 5] calculates the adaptions of each specification. In our example the unified state space will be:

$$
\begin{array}{|l}
\hline
x, y : \mathbb{N} \\
s : s_0 \mid s_1 \mid s_2 \mid s_3 \mid t_1 \mid t_2 \mid t_3 \\
\hline
\end{array}
$$

Because this is the same state space as $Max2$, the adapted $Max2$ is unchanged from the original. However, because of how the correspondence relation links up values in the two state spaces, *count* is linked to both s_2 and s_3 when *count* has the value 2. On the adapted state space it must therefore still be possible to apply c an arbitrary number of times when it is in either of these states. The adapted M is thus given by

[6] i.e. we use a refinement which is monotonic

$$\textit{adaptedM}$$

$$value1, value2 : \mathbb{N}$$
$$s : s_0 \mid s_1 \mid s_2 \mid s_3$$

$$\textit{INIT}$$
$$\Delta(s)$$
$$\overline{}$$
$$s' = s_0$$

$$a$$
$$\Delta(s, value1)$$
$$input? : \mathbb{N}$$
$$\overline{}$$
$$s = s_0 \wedge s' = s_1$$
$$value1' = input?$$

$$b$$
$$\Delta(s, value2)$$
$$input? : \mathbb{N}$$
$$\overline{}$$
$$s = s_1 \wedge s' = s_2$$
$$value2' = input?$$

$$c$$
$$\Delta(s)$$
$$output! : \mathbb{N}$$
$$\overline{}$$
$$s, s' \in \{s_2, s_3\}$$
$$output! = max(value1, value2)$$

Figure 3 shows the state transition diagrams for the original M, its adapted version, that for *Max2* and the unification of M with *Max2*. Given the correspondence relation used it is easily seen that the adapted version of M and its original specification represent the same behaviour, i.e. after an a and a b operation the operation c can be performed any number of times.

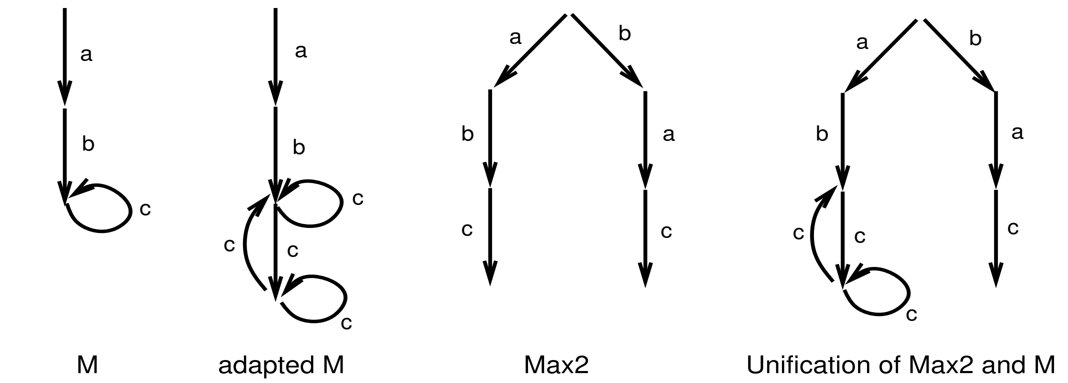

Figure 3: Unifying M and *Max2*

We can now attempt to unify the operations. The unification of two viewpoint operations should exhibit possible behaviour of each of the viewpoint operations in each situation where the viewpoint operation was applicable. This requirement can be formalised using pre- and postconditions. The unified operation will be applicable whenever one of the viewpoint operations is, i.e. its precondition is the disjunction of the viewpoint operation preconditions. Moreover, when the unified operation is applied to a state satisfying one particular precondition, a state should result that satisfies the corresponding postcondition. The unification of operations A and B is given by

$U(A, B)$

$$\begin{array}{l} Decls \\ \hline \mathrm{pre}\, A \lor \mathrm{pre}\, B \\ \mathrm{pre}\, A \Rightarrow \mathrm{post}\, A \\ \mathrm{pre}\, B \Rightarrow \mathrm{post}\, B \end{array}$$

where *Decls* is the declarations of A and B merged together. Performing this unification for each of the operations produces the following specification

$Unification(M, Max2)$

$$\begin{array}{l} value1, value2 : \mathbb{N} \\ s : s_0 \mid s_1 \mid s_2 \mid s_3 \mid t_1 \mid t_2 \mid t_3 \end{array}$$

$\textit{I}\small{NIT}$

$$\begin{array}{l} \Delta(s) \\ \hline s' = s_0 \end{array}$$

a

$$\begin{array}{l} \Delta(s, value1) \\ input? : \mathbb{N} \\ \hline (s = s_0 \land s' = s_1 \lor s = t_1 \land s' = t_2) \\ value1' = input? \end{array}$$

b

$$\begin{array}{l} \Delta(s, value2) \\ input? : \mathbb{N} \\ \hline (s = s_1 \land s' = s_2 \lor s = s_0 \land s' = t_1) \\ value2' = input? \end{array}$$

c

$$\begin{array}{l} \Delta(s) \\ output! : \mathbb{N} \\ \hline (s = s_2 \land s' = s_3 \lor s = s_3 \land s' = s_2 \lor s = s' = s_3 \lor s = t_2 \land s' = t_3) \\ output! = max(value1, value2) \end{array}$$

Figure 3 illustrates the behaviour of the unification. To illustrate what has happened consider the occurrence of the operation c in the left hand branch. This is a refinement of the c operation from the left hand branch of *Max2* and the c from the adapted M. To see this consider the state s_2. In M, c will transform this state to either s_3 or return to s_2, the choice is non-deterministic. However, in *Max2* there is just one possible behaviour. A refinement can't introduce any non-determinism, it can only reduce non-determinism, therefore only one behaviour is allowed in the unification, that which moves from s_2 to s_3. This conforms with *Max2* and reduces the non-determinism in M.

Now consider the state s_3. In this state the operation c in *Max2* was not enabled, whereas in M c will either return the state s_3 or transform it to s_2. Since we can weaken the precondition under refinement the most general refinement of c in this state will be that defined above.

This class satisfies the consistency requirements (as defined in [5]) because the state and the operations are consistent. Therefore it is the least common refinement of the two original viewpoint specifications M and *Max2*. The viewpoints are therefore consistent.

Of course the choice of correspondence relation is important, and choosing a different one can result in inconsistent specifications. For example, suppose that we related the values of state variables *count* and s by the relation $R = \{(0, s_0), (1, t_1), (2, t_2), (2, t_3)\}$. This will result in a different adaption to the unified state, and that for M is now given by

adaptedM

$$\begin{array}{l} value1, value2 : \mathbb{N} \\ s : s_0 \mid t_1 \mid t_2 \mid t_3 \end{array}$$

INIT

$$\begin{array}{l} \Delta(s) \\ \hline s' = s_0 \end{array}$$

a

$$\begin{array}{l} \Delta(s, value1) \\ input? : \mathbb{N} \\ \hline s = s_0 \wedge s' = t_1 \\ value1' = input? \end{array}$$

b

$$\begin{array}{l} \Delta(s, value2) \\ input? : \mathbb{N} \\ \hline s = t_1 \wedge s' = t_2 \\ value2' = input? \end{array}$$

c

$$\begin{array}{l} \Delta(s) \\ output! : \mathbb{N} \\ \hline s, s' \in \{t_2, t_3\} \\ output! = max(value1, value2) \end{array}$$

The states of this adapted M and the state of the class *Max2* are consistent, however, the operations are not. For example, consider the operation a, an attempt to build its unification would produce the following operation definition

a

$$\begin{array}{l} \Delta(s, value1) \\ input? : \mathbb{N} \\ \hline (s = s_0 \wedge s' = s_1) \\ (s = s_0 \wedge s' = t_1) \\ (s = t_1 \wedge s' = t_2) \\ value1' = input? \end{array}$$

which is clearly inconsistent since the state space of *Max2* is defined as a free type, so $s_1 \neq t_1$.

6 Conclusions

Using viewpoints written in process algebras and state-based languages requires that the gap between different specification styles is bridged. To do so we have used an object-oriented variant of Z which has a natural behavioural interpretation. It is this behavioural interpretation which makes it possible to define a state transition system for Object-Z specifications. We used this state transition system as a common semantic model for the two languages, and thereby defined and verified a translation between LOTOS and Object-Z.

Related work includes [28, 19] where methods of formally specifying concurrent systems using Object-Z together with CSP are developed. However, the motivation there is not consistency checking between viewpoints, but rather the construction of one specification using a combination of two languages. The basis of the language integration defined in [28, 19] is a semantics of Object-Z classes identical to that of CSP processes, where classes are related to processes and events to operations in a similar manner to the work described here. The treatment of input and output parameters of operations is, however, slightly different leading to a different treatment of refinement [29]. The relationship between the Z and LOTOS refinement relations in the context of consistency checking in ODP is discussed in [13, 12], where the latter develops refinement relations for Z specifications that contain internal operations.

The work described in this paper builds upon earlier work described in [10] which provided a partial translation between LOTOS and Z. However, this was defined via a complex intermediate semantic model, and without a full treatment of instantiation and recursion. The direct translation defined here has the benefit of preserving some of the syntactic structure of a LOTOS specification upon translation. For example, process instantiation can be translated directly to Object-Z object instantiation. The translation also sheds light on how behavioural specification is structured in the two languages. Consider, for example, parallel composition. In LOTOS a parallel composition is formed between two complete behaviours, as in $P \| Q$. However, in Object-Z we can't form such a parallel composition between *classes*, rather we compose *operations* together using the Object-Z parallel composition schema calculus operator, as in $P.Op \| Q.Op$. Thus the translation of $P \| Q$ has to be given as one explicit class definition, but the behaviour inherent in $P \| Q$ appears in the operation definitions as $Op \mathrel{\widehat=} P.Op \| Q.Op$. Thus the translation preserves structure, but in Object-Z that structure appears at an operation level rather than a class or behavioural level.

In a similar fashion the structure of a guarded process can be seen to be mapped to a similar structure at the operation level upon translation. That is the translation of a process $[pred] \longrightarrow P$ is a class with operations $Op \mathrel{\widehat=} P.Op \bullet [pred]$, where the *pred* now appears as a guard to every operation in the translated class.

Some structure is preserved in the translation of action prefix, although it has a less natural representation in Object-Z. Consider the translation of the behaviour $a;\ P$. This will be an Object-Z class containing a definition of an operation a given by

$$a \mathrel{\widehat=} ([\Delta(t) \mid t \wedge \neg\, t'] \fatsemi P.\mathit{I\scriptsize NIT}) \vee P.a$$

The structure represented by the action prefix ; has appeared as a sequential composition $\fatsemi$ of an event a (given by $[\Delta(t) \mid t \wedge \neg\, t']$) followed by the behaviour of P. However, because in general this behaviour may be another Object-Z class this is represented by the behaviour in P becoming enabled ($P.\mathit{INIT}$). The disjunction with $P.a$ is necessary because P may contain further instances of the operation a which need promoting to this level.

If P is not a process instantiation simpler translation rules can be given, for example, in translating a behaviour such as $a?x : nat;\ b?y : nat;\ c!largest(x, y);\ stop$ (part of the definition of *Max2*) intermediate variables can be used to translate the action prefixes directly to the simpler version of *Max2* given in Section 4. Similar simplifications can be given for the LOTOS choice operator when, in translating $P[]Q$, P and Q are not process instantiations. Further work to be done in this area includes development of less complex translation rules for these situations, particularly for behaviours involving action prefix and choice.

More information about the work described here (which is partially funded by British Telecom Research Labs. and the Engineering and Physical Sciences Research Council under grant number GR/K13035) can be found at:
`http://www.cs.ukc.ac.uk/research/tcs/openviews/`

References

[1] G.S. Blair and Jean-Bernard Stefani. *Open Distributed Processing and Multimedia*. Addison-Wesley, 1997.

[2] E. Boiten. Z unification tools in Generic Formaliser. Technical Report 10-97, Computing Laboratory, University of Kent at Canterbury, 1997.

[3] E. Boiten, H. Bowman, J. Derrick, and M. Steen. Managing inconsistency and promoting consistency. Submitted for publication, September 1997.

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

[5] E.A. Boiten, J. Derrick, H. Bowman, and M.W.A. Steen. Constructive consistency checking for partial specification in Z. *Science of Computer Programming*, December 1999. To appear.

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

[7] J. P. Bowen and M. Gordon. Z and HOL. In J. P. Bowen and J. A. Hall, editors, *Z User Workshop*, pages 141–167, Cambridge, July 1994. Springer-Verlag.

[8] H. Bowman, J. Derrick, P. Linington, and M. Steen. FDTs for ODP. *Computer Standards and Interfaces*, 17:457–479, September 1995.

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

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

[11] J. Derrick, E.A. Boiten, H. Bowman, and M. Steen. Translating LOTOS to Object-Z. In D.J.Duke and A.S.Evans, editors, *2nd BCS-FACS Northern Formal Methods Workshop*, Workshops in Computing. Springer-Verlag, July 1997.

[12] J. Derrick, E.A. Boiten, H. Bowman, and M. Steen. Weak refinement in Z. In J. P. Bowen, M. G. Hinchey, and D. Till, editors, *ZUM'97: The Z formal specification notation*, LNCS 1212, pages 369–388, Reading, April 1997. Springer-Verlag.

[13] J. Derrick, H. Bowman, E. Boiten, and M. Steen. Comparing LOTOS and Z refinement relations. In *FORTE/PSTV'96*, pages 501–516, Kaiserslautern, Germany, October 1996. Chapman & Hall.

[14] J. Derrick, H. Bowman, and M. Steen. Viewpoints and Objects. In J. P. Bowen and M. G. Hinchey, editors, *Ninth Annual Z User Workshop*, LNCS 967, pages 449–468, Limerick, September 1995. Springer-Verlag.

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

[16] H. Ehrig and B. Mahr. *Fundamentals of algebraic specification.* Springer-Verlag, 1985.

[17] A. Finkelstein and G. Spanoudakis, editors. *SIGSOFT '96 International Workshop on Multiple Perspectives in Software Development (Viewpoints '96).* 1996.

[18] A.C.W. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. Inconsistency handling in multiperspective specifications. *IEEE Transactions on Software Engineering*, 20(8):569–578, August 1994.

[19] C. Fischer. CSP-OZ - a combination of CSP and Object-Z. In H. Bowman and J. Derrick, editors, *Second IFIP International conference on Formal Methods for Open Object-based Distributed Systems*, pages 423–438. Chapman & Hall, July 1997.

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

[21] ISO/IEC WD 15414. *Open Distributed Processing - Reference Model - - Enterprise Viewpoint*, January 1998.

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

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

[24] I. Kraan and P. Baumann. Implementing Z in Isabelle. In J. P. Bowen and M. G. Hinchey, editors, *ZUM'95: The Z Formal Specification Notation, 9th International Conference of Z Users, Limerick, Ireland, September 7-9, 1995, Proceedings*, volume 967 of *LNCS*, pages 355–373. Springer-Verlag, 1995.

[25] 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.

[26] R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989.

[27] G. Smith. A fully abstract semantics of classes for Object-Z. *Formal Aspects of Computing*, 7(3):289–313, 1995.

[28] G. Smith. A semantic integration of Object-Z and CSP for the specification of concurrent systems. In J. Fitzgerald, C. B. Jones, and P. Lucas, editors, *Formal Methods Europe (FME '97)*, LNCS 1313, pages 62–81, Graz, Austria, September 1997. Springer-Verlag.

[29] G. Smith and J. Derrick. Refinement and verification of concurrent systems specified in Object-Z and CSP. In M. Hinchey and Shaoying Liu, editors, *First IEEE International Conference on Formal Engineering Methods (ICFEM '97)*, pages 293–302, Hiroshima, Japan, November 1997. IEEE Computer Society.

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

[31] J. Woodcock and J. Davies. *Using Z: Specification, Refinement, and Proof.* Prentice Hall, 1996.

[32] A. Yonezawa and M. Tokoro. *Object-Oriented Concurrent Programming.* MIT Press, 1987.

Biographies

Dr J. Derrick is a Senior Lecturer in Computer Science at the University of Kent. Previously he gained a D.Phil from Oxford, and then joined STC Technology Ltd to work on the ESPRIT funded RAISE project. His current interests include developing specification and design techniques for use within ODP and formal definitions of consistency and performance, and he has published extensively within this area. His work has included the following projects: PROST (DTI) *Study on Conformance Testing for ODP*; (DTI EC/4346/92) *A study into Formal Description Techniques for Object Management*; (EPSRC GR/K13035) *Cross Viewpoint Consistency in Open Distributed Processing*; FORMOSA (EPSRC/DTI) *Formalisation of the ODP Systems Architecture*; (British Telecom) *Type Management in Distributed Systems*; (EPSRC) *ODP Viewpoints in a Development Framework*, and (EPSRC) *A Specification Architecture for the Validation of Real-time and Stochastic Quality of Service*.

Dr E. Boiten is a lecturer at the Computing Laboratory at the University of Kent at Canterbury. He received his PhD in 1992 from the University of Nijmegen, The Netherlands, for work on formal program development (in particular program transformation). He has worked on various aspects of formal methods in research positions in Nijmegen and Eindhoven and Kent. His main current interests are refinement of state based systems and viewpoint specification.

Dr H. Bowman is a Senior Lecturer in the Computing Laboratory at the University of Kent at Canterbury. He received his Ph.D. from Lancaster University in 1991. His research focuses on applying formal techniques to the construction of distributed systems and he is a grant holder for a number of projects in this area. He is on the editorial board of the journal New Generation Computing and on the programme committees of a number of conferences, including, FORTE/PSTV. He was the programme co-chair of FMOODS'97, the IFIP conference on Formal Methods for Open Object Based Distributed Systems. He is co-author of a book on specifying distributed multimedia systems. He is currently on leave from Canterbury, working at VERIMAG, Grenoble and CNR-Istituto CNUCE, Pisa.

Dr M. Steen obtained an MSc(Eng) in Computer Science from the University of Twente, the Netherlands, in 1993 and a PhD from the University of Kent at Canterbury (UKC) in 1998. He is currently employed as a post-doc researcher at UKC. In the past, he has worked on techniques for partial specification in process algebra, in particular on techniques for consistency checking and composition. His current work focuses on the application of these, and other, formal methods in the area of Open Distributed Processing.