From ODP Viewpoint Consistency to Integrated Formal Methods

Eerke A. Boiten[a,*], John Derrick[b]

[a] *School of Computing, University of Kent*
Canterbury, Kent, CT2 7NF, UK
[b] *Department of Computer Science, University of Sheffield,*
Sheffield, S1 4DP, UK

Abstract

Questions asked by research into ODP Viewpoint Consistency led to fundamental questions in refinement and contributed greatly to insights and interest in Integrated Formal Methods; research in those areas is still ongoing, while the answers provided remain largely unincorporated into model driven development.

In this paper we survey some of the work done on consistency checking for multiple viewpoints, and subsequent work on generalised notions of refinement, which in turn led to work on integrations of state-based and behavioural formal methods.

Keywords: ODP viewpoints, consistency, behavioural and state based specification, refinement

1. Introduction

In 1994 at the Computing Laboratory of the University of Kent at Canterbury, a new research team was formed looking at formal methods for distributed systems, initially consisting of Howard Bowman and John Derrick with the support of Peter Linington. Distributed systems standards were of particular interest to this group, and thus initial research concentrated on the issues raised by the emerging ODP model. A central research project of the group was "Cross Viewpoint Consistency in ODP" (1995-1997) [21], funded by EPSRC with additional support from BT Research; Eerke Boiten joined as a research associate and Maarten Steen as a PhD student. This work continued in EPSRC funded projects "ODP Viewpoints in a Development Framework" (1998-2001) [54] and "A Constructive Framework for Partial Specification" (2000-2003) [19].

In this paper, we describe how the issues explored initially in the context of consistency checking of ODP viewpoint specifications have set off research in

*Corresponding author
Email addresses: E.A.Boiten@kent.ac.uk (Eerke A. Boiten), J.Derrick@dcs.shef.ac.uk (John Derrick)

three directions:

- general theories of development and consistency, with implications for model driven development (Section 2);
- sophisticated and novel notions of *refinement*, and thus also of abstraction (Section 3);
- integration of state-based and behavioural formalisms (Section 4).

The latter two topics are still subject of active research by the authors. In this paper we very briefly sketch some of the work in this area, and the subsequent sections detail in turn some work in each of the three directions mentioned above.

2. General Theories of Development and Consistency

A framework like the ODP reference model lends itself to the use of formal description techniques. Indeed, formal description has been extensively employed in Open Distributed Processing [41, 22, 57, 55, 58, 53]. Within ODP, formal description was viewed as enabling precise and unambiguous interpretation of ODP standards, which is after all the familiar motivation for employing FDTs in standardization activities. However, the spectrum of FDT usage in ODP is both extensive and diverse, as is the range of available FDTs. For example, LOTOS [12, 44], Estelle [45] and SDL [18] are targeted at issues of explicit concurrency and interaction (specifying ordering and synchronisation of abstract events). In contrast, approaches such as Z [62] and VDM [49] address specification of software systems in terms of data state change.

Importantly, none of these FDTs fully addressed the needs of all the viewpoints appearing in the Open Distributed Processing standard, since the remit of systems covered by the standard is extremely broad, encompassing, for example, both information modelling and description of engineering infrastructures. Initial explorations of how to best specify the particular viewpoints [39, 16] suggested that in particular the *information* viewpoint could be represented in a data-rich formalism like Z [62]. The *computational* viewpoint, on the other hand, fits more naturally with a behavioural formalism – for example LOTOS [12] which was developed and applied in the context of telecommunications standardisation.

Thus, in exploring cross viewpoint consistency for ODP, we were faced with the issue of reconciling multiple specifications, potentially in widely varying formal models, e.g., languages such as Z and LOTOS. Cross viewpoint consistency needs to be distinguished from "internal" consistency, i.e., whether any requirements in a single specification contradict each other – this may occur in some (e.g., logic based) formalisms, but not in others (e.g., process algebras) – which is a largely independent issue.

2.1. A small example

A simple example illustrates some of the issues, and is taken from [6], where we specify a communications protocol (an abstraction of the Signalling System No. 7 protocol described in [42]) from two viewpoints - loosely based on the computational and engineering viewpoints. The former is specified in LOTOS, and the latter, because it is heavily state dependent, in Z. We assume readers are broadly familiar with the notations used.

Computational Viewpoint in LOTOS: The protocol is specified in terms of two sequences *in* and *out* (which represent messages that have arrived in the protocol (*in*), and those that have been forwarded (*out*)). Incoming messages are added to the left of *in*, and the messages contained in *in* but not in *out* represent those currently inside the protocol.

The protocol delivers without corrupting or re-ordering, i.e., the *out* sequence is a suffix of the *in* sequence. Messages are of type *element*, which contains a distinguished value *null*.

In the data typing part of the LOTOS specification one needs to specify an algebraic type *seq* of sequences, most of which is omitted below. The main part is the process specification, where two actions model the behaviour of the protocol, i.e., the transmission and reception of messages. The *transmit* action accepts a new message and adds it to the *in* sequence. The *receive* action either delivers the latest value as an output (which is then also added to the output sequence), or a null value is output, modelling the environment's "busy waiting" (in which case *out* is unaltered). Initially, no messages have been sent. The specification can be written:

specification
type seq **is** element, bool, nat **with**
  sorts *seq*
  opns $empty\_seq :\rightarrow seq$
    $add : element, seq \rightarrow seq$
    $\neq : seq, seq \rightarrow bool$
    $first : seq \rightarrow element$
    $last : seq \rightarrow element$
    $front : seq \rightarrow seq$
    $- : seq, seq \rightarrow seq$
    $cat : seq, seq \rightarrow seq$
    $\# : seq \rightarrow nat$
  eqns
    (* most characterising equations for operations omitted *)
    forall $x, y : element, q : seq$
    ofsort *element*
      $last(add(x, empty\_seq)) = x$
      $last(add(x, add(y, q))) = last(add(y, q))$
endtype
behaviour
  $Protocol[transmit, receive](empty\_seq, empty\_seq)$

where

process $Protocol[transmit, receive](in, out : seq) : \textbf{noexit} :=$
$transmit?x : element;\ Protocol[transmit, receive](add(x, in), out)$
$[]$
$receive!null;\ Protocol[transmit, receive](in, out)$
$[]$
$[in \neq out] \rightarrow receive!last(in - out);$
$Protocol[transmit, receive](in, add(last(in - out), out))$
endproc

endspec

Engineering Viewpoint in Z: This viewpoint describes the route the messages take through the medium in terms of a number of sections represented by a non-empty sequence of signalling point codes (SPC). Each section may send and receive messages of type M, and those that have been received but not yet sent on are said to be in the section. The messages pass through the sections in order. In the state schema, *ins* i represents the messages currently inside section i, *rec* i the messages that have been received by section i, and *sent* i the messages that have been sent onwards from section i. The state and initialization schemas can be described by

$[M, SPC]$

Section

$route : \text{iseq } SPC$
$rec, ins, sent : \text{seq}(\text{seq } M)$

$route \neq \langle\,\rangle$
$\#route = \#rec = \#ins = \#sent$
$rec = ins \frown\frown sent$
$front\ sent = tail\ rec$

InitSection

$Section'$

$\forall i : \text{dom } route \bullet$
$rec\ i = ins\ i = sent\ i = \langle\,\rangle$

where $\frown\frown$ denotes pairwise concatenation of the two sequences (so for every i we have $rec\ i = ins\ i \frown sent\ i$). The predicate $front\ sent = tail\ rec$ ensures that messages that are sent from one section are those that have been received by the next. This specification also has operations to transmit and receive messages:

Transmit

$\Delta Section$
$m? : M$

$route' = route$
$head\ rec' = \langle m? \rangle \frown (head\ rec)$
$tail\ rec' = tail\ rec$
$sent' = sent$

Receive

$\Delta Section$
$m! : M$

$route' = route \land rec' = rec$
$front\ ins' = front\ ins$
$last\ ins' = front(last\ ins)$
$front\ sent' = front\ sent$
$m! = last(last\ ins)$
$last\ sent' = \langle m! \rangle \frown (last\ sent)$

In this viewpoint, the new message received is added to the first section in the route in *Transmit*, and *Receive* will deliver from the last section in the route. In the computational viewpoint, messages arrive non-deterministically, but in this viewpoint the progress of the messages through the sections is modelled explicitly. To do this we use an internal action *Daemon* which chooses which section will make progress in terms of message transmission. The oldest message is then transfered to the following section, and nothing else changes. The important part of this operation is:

Daemon

$$\Delta Section$$

$$\exists\, i : 1..\#route - 1 \mid ins\ i \neq \langle\,\rangle \bullet$$
$$ins' i = front(ins\ i)$$
$$ins'(i+1) = \langle last(ins\ i)\rangle \frown ins(i+1)$$
$$\forall\, j : \mathrm{dom}\, route \mid j \neq i \wedge j \neq i+1 \bullet ins' j = ins\ j$$

Even with a small example one can see that viewpoints might be written in very different languages, they might describe data types (and even types) in different ways, they might have operations and/or events in one viewpoint that are not in the other, etc The central question our work addressed was thus: how to reconcile two viewpoints written, as in the above example, in possibly different languages, with these types of differing attributes? The reconciliation will depend on whether the viewpoints are *consistent*.

2.2. Viewpoint consistency

Viewpoint methods were already common in requirements engineering [36, 35] but the problem of checking consistency between viewpoint specifications given in formal notations [3] did not have any ready made solutions beyond translating all viewpoints into a low level common language like predicate logic [68]. Inspired by Leduc's work on development relations for LOTOS [51] we developed a general theory of *development relations* [17, 9] encompassing:

- *refinement* relations within a formal notation;
- *conformance* and *implementation* relations, between formal notations and implementation notations;
- *translations* between different formal notations.

There were two crucial realisations in working this out in detail. The first was that a viewpoint specification – particularly in a language with multiple development relations – does not necessarily speak for itself: we also need to know how it is expected to relate to any final implementation, i.e., a viewpoint specification consists of a specification *and* a development relation. For example, in the above scenario one would need to know which LOTOS conformance relation was to be used as the implementation check before one could meaningfully

perform a consistency check between the two viewpoints. The second is that in order to combine, compare, and generally relate two viewpoint specifications, we need to know how their components and named elements relate to each other. For example, how do the components in the LOTOS data type and process specification relate to the state and operations given in the second Z viewpoint specification - that is, how do the types, operations and other aspects (compare *in*, *out* with *rec* and *sent*). This is, of course, the reflection of the ODP concept of a *correspondence*. Taking these into account,

- we would consider collections of pairs (*spec*, *dev*) where *spec* is a specification, and *dev* is an associated development relations, with correspondence relations between the different specifications in the collection;
- *consistency* of such a collection is then defined as the existence of a common image under the respective development relations, respecting and taking into account the various correspondences.

For ODP, such collections could directly correspond to the predefined viewpoints, or they could be of even smaller granularity. *Constructive* consistency checking does not actually require the construction of such a shared image (implementation?) – it may be possible to construct instead a higher level specification which correctly reflects the requirements of all viewpoints, called a *unification*. In the context of the example above this may well be a single specification written in either LOTOS or Z which encapsulates all of the requirements given by the two viewpoints in a single description.

This is what is refered to in the many category theory based methods for partial specification as a "push-out"; however, those methods tend to stop at proving their existence, whereas in practice they need to be constructed syntactically. For a language like LOTOS this may be hard [17], but for the special case where all viewpoints are written in Z we have shown how this can be done [5]. An overview, with detailed case study, of our general methods applied to ODP viewpoint consistency is given in [6]. In the next subsection we briefly explain how some of these ideas apply to the example introduced above.

2.3. Interlude - example revisited

The correspondences in the above example are fairly straightforward - the protocol transmits one type of message, so M and *element* should be identified. Moreover, the operations and actions described in the two viewpoints are different perspectives of the same function, so we should link *Transmit* to *transmit* and *Receive* to *receive* (and implicitly the inputs and outputs of the operations are identified). Finally, it is clear that *in* and *out* in the LOTOS viewpoint in some way represent information that is also represented by *rec*, *ins* and *sent* in the Z viewpoint. However, this is not a matter of simply identifying these components, rather we note that they are related via the following predicate: $head\ rec = in \wedge\ last\ sent = out$. So these correspondences can then be docu-

mented as a relation

$$\{(M, element), (Transmit, transmit), (Receive, receive), (head\ rec, in), (last\ sent, out)\}$$

With these correspondences in place, one can now check for consistency. Here the viewpoints are written in Z and LOTOS, and we return to the problem in Section 4 when we discuss integrating state-based and behavioural specification languages.

2.4. Viewpoints in UML and MDD

The most popular incarnation of multiple-paradigm, multiple-viewpoint specification has of course been for many years UML and Model Driven Development. Curiously, this community is interested in correct implementation (automated, through transformations, where possible), to some extent in refinement, and in consistency between different models and model types, but not in underpinning this with a fixed semantics for the notations. We have shown at various UML workshops on consistency how our ODP inspired techniques also apply in the UML context [23, 7] – however, these papers also point out the fundamental issue, evident from our general theory if not already self-evident, that there is virtually an equivalence between a specification language having a semantics, having a notion of consistency, and having a notion of refinement. Thus, for as long as the MDD community agrees to disagree on a common semantics for their notations, their notions of consistency and refinement (and correct implementation) must necessarily be poor approximations.

3. Liberalised Refinement

Our general theory of consistency checking as described above concentrates on finding common refinements of different viewpoint specifications. However, looking at concrete examples, it became clear that the standard refinement relations for most specification languages are too restrictive for this usage. In particular, looking at the different ODP viewpoints and their correspondences, we noticed that

- viewpoints may omit any mention of operations and data which are irrelevant to their particular concerns;
- viewpoints may abstract from detailed control mechanisms, inputs, and outputs, that will appear for the same operation in a different viewpoint;
- viewpoints may view at different levels of granularity: what appears as a simple action in one, may be a sequence of actions in another one; some of these actions may be "internal", outside the environment's control.

Even with a very simple example like the one above some of these issues are already present, and none of these issues is accounted for in standard Z refinement, and not all of them are addressed for process algebras like LOTOS.

Thus, initially driven by the desire to allow a wider variety of abstraction in viewpoint specifications written in Z, we explored more general notions of refinement for Z, including weak refinement (introducing internal actions), action refinement, and I/O refinement. An overview of these is given in [8], and full details of all of them are in a research monograph [27]. Here we briefly recount the main details.

The standard method of verifying refinements in Z (and other state-based languages such as B, event-B etc) is to use *simulations*, of which there are two varieties - downward and upward (confusingly also sometimes called forward and backward resp.). Each of these are sound (i.e, the use of a downward simulation is a valid refinement etc), and together they are jointly complete.

The theory of data refinement this is based on is that described by He Jifeng, Hoare and Sanders [43] which provides a useful model of program development in terms of abstract data types for which the set of operations is known already. However, there were restrictions, e.g., the abstract data types (ADTs) were assumed to have identical sets of operations, i.e., they were conformal. Moreover, the application of this theory in languages like Z [62] further restricted this context. There the emphasis in the early work on refinement in Z was on the incomplete forward simulation rule, forbidding postponement of non-determinism; and it was implied that input and output were immutable (i.e., in a refinement step, inputs and outputs could not change). Recent work [66, 63, 4] has relaxed these unnecessary restrictions by more fully exploiting the theory of [43] as well as generalising it at key points.

This was necessary in the context of viewpoints because such partial specifications may mention only aspects of the system of relevance to their particular viewpoint. For example, only part of the external interface for an operation may appear in a given viewpoint, and we require that the common refinement (constructed in a consistency check) can add to, or alter, this interface (i.e., the IO of an operation). This required a more liberal notion of refinement. Consider the following small example, taken from [8]:

A two-dimensional world, in which an object moves about and its movement and position can be observed, is represented with state space $2D$, initialisation *Init* and two operations *Move* and *Where*:

2D

$$x, y : \mathbb{Z}$$

Move

$$\Delta 2D$$

Init

$$2D$$

$$x = y = 0$$

Where

$$\Xi 2D$$

$$x!, y! : \mathbb{Z}$$

$$x! = x' \land y! = y'$$

As *Where* is total and deterministic, it cannot be refined any further (without change of state space). Possible operation refinements for *Move* in the standard theory include the following:

DontMove

$$\Xi 2D$$

Swap

$$\Delta 2D$$

$$x' = y \land y' = x$$

StepLeft

$$\Delta 2D$$

$$x' = x - 1 \land y' = y$$

StepRight

$$\Delta 2D$$

$$x' = x + 1 \land y' = y$$

However, the following, seemingly valid developments are not valid refinements:

Adding inputs and outputs a system where *Move* does not simply have a non-deterministic result, but there is external control that we were previously unaware of, i.e. *Move* is replaced by

Translate

$$\Delta 2D$$

$$x?, y? : \mathbb{Z}$$

$$x' = x + x? \land y' = y + y?$$

This is not a valid refinement, because *Translate* has inputs that *Move* did not have.

Changing the type of inputs and outputs a system where the output of *Where* is in polar coordinates – also not allowed because conformity enforces identical output types.

Replicating operations a system where various kinds of *Move* are possible that we did not distinguish above, e.g. *Move* is replaced by *both StepLeft* and *StepRight*.

Adding internal operations a system which introduces an internal clock, which is left unchanged by *Move* and *Where* but incremented by an in-

ternal operation *Tick*, e.g.

$$\begin{array}{|l}\hline Timed2D \\ x, y, clock : \mathbb{Z} \\ \hline\end{array}$$

$$\begin{array}{|l}\hline TInit \\ Timed2D \\ \hline x = y = clock = 0 \\ \hline\end{array}$$

$$\begin{array}{|l}\hline TMove \\ \Delta Timed2D \\ \hline clock' = clock \\ \hline\end{array}$$

$$\begin{array}{|l}\hline TWhere \\ \Xi Timed2D \\ x!, y! : \mathbb{Z} \\ \hline x! = x' \land y! = y' \\ \hline\end{array}$$

$$\begin{array}{|l}\hline Tick \\ \Delta Timed2D \\ \hline clock' = clock + 1 \\ x' = x \land y' = y \\ \hline\end{array}$$

See also the example above, where the internal operation *Daemon* was introduced in the second viewpoint.

Adding an external operation a system where an additional (external) operation is available, e.g.

$$\begin{array}{|l}\hline Reset \\ \Delta 2D \\ \hline x' = y' = 0 \\ \hline\end{array}$$

which is *not* viewed as a concrete occurrence of *Move* (although as it happens, *Reset* does refine *Move*).

To deal with such developments one can generalise the standard definition of refinement in a number of ways:

IO refinement. In the standard model operations and their more concrete versions are required to have identical inputs and outputs. This requirement can be traced back to the conformity condition on the ADTs when interpreting the original theory in Z. In fact this condition can be relaxed to produce IO refinement rules (i.e., simulation rules) that are (still) a consequence of the data refinement theorems of Jifeng He, Hoare and Sanders.

These simulation rules use input and output transformers to change the representation of the inputs and outputs in a manner similar to the change of state space in a data refinement as achieved by the retrieve relation. There are some simple conditions on these transformers: input transformers must be total on the abstract input: every abstract input must still be allowable. Similarly, output transformers should be injective from abstract output to concrete output: different abstract outputs should be distinguishable by leading to different concrete outputs.

The most obvious instances of simple input refinement are:

- addition of an input whose value is not used in the new operation;

- replacement of an input by an isomorphic one (total bijective input transformer)

The replacement of *Move* by *Translate* can now be justified in two steps: the first is a simple input refinement step introducing inputs $x?$ and $y?$ which are not used. The corresponding input transformer takes the input of *Move* – there was none, so it is the empty tuple – and relates it to any pair $x?, y? : \mathbb{Z}$. The second step is then an operation refinement, reducing the non-determinism by using $x?$ and $y?$.

Similarly one can define simple output refinements, the most obvious instances include:

- addition of an output whose value does not depend on the state;
- replacement of an output by an isomorphic one (total bijective output transformer)

So the version of *Where* which returns polar coordinates as its output is of course an instance of a total bijective output transformer.

Adding internal operations. To deal with the addition of internal or unobservable operations in a state-based context of refinement one can either view them as stuttering steps or generalise the existing standard theory further. The full generality is given by the latter course of action, and that draws inspiration from the use of internal events in a process algebra. This results in a set of *weak refinement* rules [26] that ensure the observable behaviour of the refined ADT is a valid refinement of the observable behaviour of the original ADT.

To do so we still assume the sets of observable operations in a refinement are conformal (i.e, the same in each), but we extend the notion of a data type to additionally include a set of internal operations, and the rules allow the introduction or removal of internal operations during a refinement. We then move from the application of a single observable operation *Op* to a situation where a finite number of internal operations are allowed before and after the occurrence of that operation, as is typical in a process algebraic definition. Specifically, this corresponds to the change from $P \xrightarrow{a} Q$ to $P \stackrel{a}{\Longrightarrow} Q$ in a process algebra when internal (i.e. unobservable) actions are taken into account. Using this definition of refinement, one can derive simulation rules for verifying particular examples such as the one above (e.g., the refinement into *Timed*2*D*).

Adding visible operations. To cope with the example above, where we wished to replicate an existing visible operation, one needs a further generalisation where we would give an explicit mapping from the abstract index set to the concrete index set. This mapping needs to be total (every abstract operation has at least one concrete counterpart) and injective (every concrete operation reflects at most one abstract operation). This generalisation is compatible with others defined above, and allows the addition of visible operations without adding ones that fundamentally alter the behaviour in a specification.

Action, or non-atomic, refinement. A key aspect of viewpoints in general is that they are not necessarily at the same level of abstraction. In terms of

behavioural specifications this means that an event or operation in one viewpoint might well be implemented by not one, but a sequence of concrete operations in the eventual system. Such action, or non-atomic, refinements arise in a number of settings quite naturally in addition to viewpoints or partial specification (see [1]) and they allow initial specifications to be described independently of the structure of the eventual implementation. The desired final structure can then be introduced by non-atomic refinements which support a change of operation granularity absent if we require conformity.

[24] discusses how action refinement can be supported in Z. In essence one allows the implementing sequence to appear in the simulation rules, so that they can be used to make step-by-step comparisons as before. We omit the details here, see [27] for the technicalities, sufficient to note that they allow verification of the examples above, and also naturally use the ideas of IO transformations in order to come up with quite general transformations between abstract and concrete.

The above is but a brief overview of some work on generalising refinement that came out of the requirements of ODP. As can be seen, the main outcome of this work is that the abstraction methods listed above can all be incorporated in generalisations of refinement, and that most of these generalisations are orthogonal.

4. Integration of State-Based and Behavioural Specification

The ODP consistency checking problem (for example, between the viewpoints in the example above) brings with it the issue of reconciling state-based specifications (such as Z, OCL, or state charts) with behaviourally based ones (such as LOTOS and other process algebras, or sequence diagrams). In our work on ODP we considered initially two approaches to this: the separated use of each formalism in carefully constructed layers and templates [28], and a translation approach, typically from behavioural to state-based formalisms [31].

4.1. Relating LOTOS and Z - and subsequent consistency checking

Comparing viewpoints written in LOTOS and Z requires that we bridge a gap between completely different specification paradigms. Although both languages can be viewed as dealing with states and behaviour, the emphasis differs between them. One solution is to adopt a more behavioural interpretation of Z, and in [31] a translation between LOTOS and Z is described by defining a common semantics for LOTOS and a subset of Z in an extended transition system, which is used to validate the translation from full LOTOS into Z [25]. The essential idea behind the translation is to turn LOTOS processes into objects in an OO version of Z, and hence if necessary into Z. We illustrate with a few parts of the above example. The translation turns the data type definitions in LOTOS into equivalent axiomatic definitions in Z. E.g., the definition of *element* would be translated to a definition of *element* in Z:

$$[element]$$

$$null : element$$

And the remaining part can be translated directly to an axiomatic declaration in Z, viz:

$$[seq]$$

$$\begin{array}{l} empty\_seq : seq \\ add : element \times seq \rightarrow seq \\ last : seq \rightarrow element \\ \hline \forall\, x, y : element, q : seq \bullet\ last(add(x, empty\_seq)) = x \\ \qquad\qquad \wedge\, last(add(x, add(y, q))) = last(add(y, q)) \end{array}$$

For the LOTOS behaviour expression each transition (which represents a LOTOS action) becomes a Z operation schema with explicit pre- and post-conditions to preserve the temporal ordering. The pre- and post-conditions are derived from the start and end state of each transition together with the guard of the transition. The data-typing content of a transition is incorporated into the operation schema's declaration.

For example, the above LOTOS viewpoint will be translated into a Z specification which contains operation schemas with names *transmit* and *receive*. The operation schemas have appropriate inputs and outputs (controlled by channels $ch?$ and $ch!$) to perform the value passing defined in the LOTOS process. Each operation schema includes a predicate (defined over the state variable s) to ensure that it is applicable in accordance with the temporal behaviour of the LOTOS specification. Thus the behaviour expression in the above viewpoint is translated to the following Z schemas.

State

$$\begin{array}{l} s : \{s_0, s_1, s_2, s_3\} \\ in, out : seq \\ x : element \end{array}$$

Init_State

$$\begin{array}{l} \Delta State \\ \hline s' = s_0 \\ in' = empty\_seq \\ out' = empty\_seq \end{array}$$

transmit

$$\begin{array}{l} \Delta State \\ ch? : element \\ \hline s = s_0 \wedge s' = s_1 \wedge x' = ch? \end{array}$$

receive

$$\begin{array}{l} \Delta State \\ ch! : element \\ \hline (s = s_0 \wedge s' = s_2 \wedge ch! = null) \vee \\ (in \neq out \wedge s = s_0 \wedge s' = s_3 \\ \quad \wedge ch! = last(in - out)) \end{array}$$

i

$\Delta State$

$(s = s_1 \land s' = s_0 \land in' = add(x, in) \land out' = out) \lor$
$(s = s_2 \land s' = s_0 \land in' = in \land out' = out) \lor$
$(s = s_3 \land s' = s_0 \land in' = in \land out' = add(last(in - out), out))$

Because the translation is actually defined indirectly via a common semantic model, recursion is dealt with by using an internal action, which is translated as an internal Z operation with special name i. However, we can re-write it without the internal action by replacing the three operation schemas by the following two and then removing the state component s which has become redundant. In order to reason about Z specifications which contain internal actions we need to use weak Z-refinement discussed above (see also [26]), and the specification without the above internal operation is weak Z-refinement equivalent to the original.

transmit

$\Delta State$
$ch? : element$

$in' = add(ch?, in) \land out' = out$

receive

$\Delta State$
$ch! : element$

$in' = in$
$(out' = out \land ch! = null) \lor$
$(in \neq out \land ch! = last(in - out) \land$
$out' = add(ch!, out))$

The two viewpoints are now both expressed in Z, and techniques for consistency checking can be applied, e.g., see those described in [5]. Normally this involves constructing a least refined unification of the two viewpoints, in two phases. In the first phase ("state unification"), a unified state space (i.e., a state schema) for the two viewpoints has to be constructed. The essential components of this unified state space are the correspondences between the types in the viewpoint state spaces. 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 (e.g. *Transmit* and *transmit*) 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.

In the above example, the viewpoints turn out to be consistent. However, with two minor but reasonable modifications they are not. Consider an alternative computational viewpoint with the following type of impatient receive operation (here just given directly in Z):

impreceive

$\Delta State$
$m! : element$

$in' = in$
$(in = out \land out' = out \land m! = null) \lor$
$(in \neq out \land m! = last(in - out) \land out' = \langle m! \rangle \frown out)$

If we also modify the engineering viewpoint's receive operation to be total, by making it have no effect outside its precondition except for returning a *null*, i.e.

$$TotReceive \widehat{=} Receive \ \lor \ (\neg \ \mathrm{pre}\, Receive \ \land \ \Xi Section \land m! = null)$$

the resulting specifications become *inconsistent*. When the last section is empty, but there is a message in some other section, $TotReceive$ will insist that the state remain unchanged. However, in that situation *impreceive* states that this message should be added to *out*. Unsurprisingly, the only way to prevent this situation and make these operations consistent is to ensure there is no more than one section... clearly not what was intended by the viewpoint specifiers.

The translation approach induces a question for subsequent development of the viewpoint specifications, namely the following. If specifications in different formalisms are initially consistent with each other, they can clearly be developed "in opposite directions" to then become inconsistent; however, if the development relations used are incompatible in nature such an effect is almost unavoidable. Thus we also studied compatibility of refinement relations in various formalisms [32].

4.2. Integrating formal methods

Work in this area on ODP, on the emerging UML, and the general realisation that different styles of specification work best for different aspects, led to an increased interest in"integrating formal methods", with the conference series that started in 1999 still ongoing, e.g. [2, 11]. There are two strands to this work, one on integrating their semantics, and in particular understanding the relationship between the differing notions of refinement, and another on integration of two, or more, languages to derive a combined specification notation.

For example, work on the former includes that due to Josephs [50], He [47], Woodcock and Morgan [67], Bolton and Davies [13, 14], Derrick and Boiten [29] and Reeves and Streader [56]. That due to Josephs [50], He [47], Woodcock and Morgan [67] defines a basic correspondence between refinement rules in Z and the CSP failures-divergences refinement relation. The more recent work of Bolton and Davies [13, 14], Derrick and Boiten [29] and Reeves and Streader [56] investigates a direct correspondence between the relational semantic model that underpins a state-based notation such as Z and process semantics, and includes specific consideration of input and output which introduces some subtleties.

Work on integrating the notations, as opposed to the semantics, includes that which combines Z and Object-Z [61] with process algebras such as CSP and CCS.

For example, combinations of Object-Z and CSP have been investigated by a number of researchers including Smith [60], Fischer [37] and Mahony and Dong [52]. Other combinations of languages include those investigated by Galloway [40], Treharne [65] and Sühl [64]. A survey of some of these approaches is given in [38]. In this context, refinement relations that are compatible between state-based languages and process algebras are desired so that a uniform method of refinement can be presented for integrated specifications.

Our latest work in this area follows the ethos of the first strand, and is based on a more fundamental integration of behavioural and state-based specification, which we call "relational concurrent refinement" [29, 10, 30]. Behavioural formalisms have semantics that are typically given as sets of possible observations. We encode these observations explicitly in a state-based (relational) formalism, in a way which ensures that relational refinement represents the desired concurrency refinement relation. This has three very useful consequences:

- behavioural development relations can be verified in a stepwise manner using the relational method of simulations (rather than through inclusion of observation sets for entire processes);
- state-based formalisms like Z can be given non-standard semantics for concurrency based interpretations;
- the various refinement relations in the relational and behavioural world can be compared and contrasted within a single framework.

So far, relational concurrent refinement has been used to analyse many process algebra refinement relations, including internal actions and divergence, and various refinement notions for state based systems and automata.

5. Conclusions

This has been a brief survey of the work on viewpoint consistency in ODP which then influenced work on refinement and development relations, which in turn led to increased work on integrated specification formalisms. Some of these ideas have had further influence on aspects of informal software engineering such as UML and MDD.

There has been another strand to work on formal methods and ODP which we have not touched on in the above, namely the strand of work concerned with formalising the ODP standard itself. The RM-ODP defines abstract languages for the five viewpoints. Several research groups have worked on populating this abstract framework with specific formal specification notations (e.g. [53, 20, 34]). Other examples include [48], which describes an approach of using Z to represent the formal semantics of some fundamental concepts of RM-ODP (Part 2) rather than just of viewpoint-specific concepts. This is relevant because the viewpoint-specific concepts are based on and substantially use the fundamental ones. See also [33], which uses category theory as a mathematical framework for formal foundations of RM-ODP.

In addition, work on the ODP *architectural semantics* aims to provide interpretations of the abstract modelling and specification concepts in a number of standardised formal description techniques [46, 59, 39]. The architectural semantics aimed to provide the basis for uniform and consistent comparison between formal descriptions of the same system or standard in different FDTs. This again strengthens our motivation for developing realistic consistency checking techniques as described above.

Acknowledgements

We would like to thank our colleagues in the research projects on ODP Viewpoints: Peter Linington, Howard Bowman and Maarten Steen – the five of us were equal contributors to the ideas described in Section 2, and the success of these projects was significantly due to the excellent collaboration within the group.

References

[1] L. Aceto. *Action Refinement in Process Algebras*. CUP, London, 1992.

[2] K. Araki, A. Galloway, and K. Taguchi, editors. *International Conference on Integrated Formal Methods 1999 (IFM'99)*. Springer, York, July 1999.

[3] E. A. Boiten, H. Bowman, J. Derrick, and M. W. A. Steen. Issues in multiparadigm viewpoint specification. In Finkelstein and Spanoudakis [35], pages 162–166.

[4] E. A. Boiten and J. Derrick. IO-refinement in Z. In A. Evans, D. J. Duke, and T. Clark, editors, *3rd BCS-FACS Northern Formal Methods Workshop*. Springer-Verlag, September 1998. http://www.ewic.org.uk/.

[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*, 35(1):29–75, September 1999.

[6] E.A. Boiten, H. Bowman, J. Derrick, P.F. Linington, and M.W.A. Steen. Viewpoint consistency in ODP. *Computer Networks*, 34(3):503–537, August 2000.

[7] E.A. Boiten and M.C. Bujorianu. Exploring UML refinement through unification. In J. Jürjens, B. Rumpe, R. France, and E.B. Fernandez, editors, *Critical Systems Development with UML - Proceedings of the UML'03 workshop*, volume TUM-I0323, pages 47–62. Technische Universität München, September 2003.

[8] E.A. Boiten and J. Derrick. Liberating data refinement. In R.C. Backhouse and J.N. Oliveira, editors, *Mathematics of Program Construction, 5th International Conference, Ponte de Lima*, volume 1837 of *Lecture Notes in Computer Science*, pages 144–166. Springer, July 2000.

[9] E.A. Boiten and J. Derrick. A relational framework for the integration of specifications. *Journal of Integrated Design and Process Science*, 7(3):39–48, September 2003.

[10] E.A. Boiten, J. Derrick, and G. Schellhorn. Relational concurrent refinement II: Internal operations and outputs. *Formal Aspects of Computing*, 21(1-2):65–102, 2009.

[11] E.A. Boiten, J. Derrick, and G. Smith, editors. *Integrated Formal Methods, 4th International Conference*, volume 2999 of *Lecture Notes in Computer Science*. Springer-Verlag, April 2004.

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

[13] C. Bolton and J. Davies. Refinement in Object-Z and CSP. In M. Butler, L. Petre, and K. Sere, editors, *Integrated Formal Methods (IFM 2002)*, volume 2335 of *Lecture Notes in Computer Science*, pages 225–244. Springer-Verlag, 2002.

[14] C. Bolton and J. Davies. A singleton failures semantics for Communicating Sequential Processes. *Formal Aspects of Computing*, 18:181–210, 2006.

[15] J. P. Bowen, A. Fett, and M. G. Hinchey, editors. *ZUM'98: The Z Formal Specification Notation*, volume 1493 of *Lecture Notes in Computer Science*. Springer-Verlag, September 1998.

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

[17] H. Bowman, M.W.A. Steen, E.A. Boiten, and J. Derrick. A formal framework for viewpoint consistency. *Formal Methods in Systems Design*, 21:111–166, 2002.

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

[19] A Constructive Framework for Partial Specification, 2000-2003. http://www.cs.kent.ac.uk/archive/research/groups/tcs/framework/, last accessed 8-9-2011.

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

[21] Cross Viewpoint Consistency in ODP, 1995–1997. http://www.cs.kent.ac.uk/archive/research/groups/tcs/consistency/, last accessed 8-9-2011.

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

[23] J. Derrick, D.H. Akehurst, and E.A. Boiten. A framework for UML consistency. In L. Kuzniarz, G. Reggio, J. L. Sourrouille, and Z. Huzar, editors, *UML 2002 Workshop on Consistency Problems in UML-based Software Development*, pages 30–45, October 2002.

[24] J. Derrick and E. A. Boiten. Non-atomic refinement in Z. In J. M. Wing, J. C. P. Woodcock, and J. Davies, editors, *FM'99 World Congress on Formal Methods in the Development of Computing Systems*, volume 1708 of *Lecture Notes in Computer Science*, pages 1477–1496, Berlin, 1999. Springer-Verlag.

[25] J. Derrick, E. A. Boiten, H. Bowman, and M. W. A. 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.

[26] J. Derrick, E. A. Boiten, H. Bowman, and M. W. A. Steen. Specifying and refining internal operations in Z. *Formal Aspects of Computing*, 10:125–159, December 1998.

[27] J. Derrick and E.A. Boiten. *Refinement in Z and Object-Z: Foundations and Advanced Applications*. FACIT. Springer Verlag, May 2001.

[28] J. Derrick and E.A. Boiten. Combining component specifications in Object-Z and CSP. *Formal Aspects of Computing*, 13:111–127, May 2002. Special issue based on extended papers from [2].

[29] J. Derrick and E.A. Boiten. Relational concurrent refinement. *Formal Aspects of Computing*, 15(1):182–214, November 2003.

[30] J. Derrick and E.A. Boiten. Relational concurrent refinement part iii: Traces, partial relations and automata. *Formal Aspects of Computing*, 2011. Submitted for publication.

[31] J. Derrick, E.A. Boiten, H. Bowman, and M. Steen. Viewpoints and consistency: translating LOTOS to Object-Z. *Computer Standards and Interfaces*, 21:251–272, 1999.

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

[33] Z. Diskin. On modeling, mathematics, category theory and RM-ODP. In José A. Moinhos Cordeiro and Haim Kilov, editors, *WOODPECKER*, pages 38–54. ICEIS Press, 2001.

[34] J. Dustzadeh and E. Najm. Consistent semantics for odp information and computational models. In A. Togashi, T. Mizuno, N. Shiratori, and T. Higashino, editors, *FORTE*, volume 107 of *IFIP Conference Proceedings*, pages 107–126. Chapman & Hall, 1997.

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

[36] 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.

[37] 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.

[38] C. Fischer. How to combine Z with a process algebra. In ZUM'98 [15], pages 5–23.

[39] 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.

[40] A. Galloway and W. Stoddart. An operational semantics for ZCCS. In M. G. Hinchey and Shaoying Liu, editors, *First International Conference on Formal Engineering Methods (ICFEM'97)*, pages 272–282, Hiroshima, Japan, November 1997. IEEE Computer Society Press.

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

[42] I. Hayes, M. Mowbray, and G.A. Rose. Signalling system no. 7 – the network layer. In *PSTV IX*, pages 3–14, 1990.

[43] He Jifeng, C. A. R. Hoare, and J. W. Sanders. Data refinement refined. In B. Robinet and R. Wilhelm, editors, *Proc. ESOP'86*, volume 213 of *Lecture Notes in Computer Science*, pages 187–196. Springer-Verlag, 1986.

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

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

[46] ISO/IEC JTC1/SC21/WG7. Basic Reference Model of Open Distributed Processing. ISO 10746, 1993. Part 1 to 4.

[47] He Jifeng. Process refinement. In J. McDermid, editor, *The Theory and Practice of Refinement*. Butterworths, 1989.

[48] D.R. Johnson and H. Kilov. An approach to a Z toolkit for the Reference Model of Open Distributed Processing. *Computer Standards and Interfaces*, 21(5):393–402, 1999.

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

[50] M. B. Josephs. A state-based approach to communicating processes. *Distributed Computing*, 3:9–18, 1988.

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

[52] B. P. Mahony and J. S. Dong. Blending Object-Z and timed CSP: An introduction to TCOZ. In K. Futatsugi, R. Kemmerer, and K. Torii, editors, *20th International Conference on Software Engineering (ICSE'98)*. IEEE Press, 1998.

[53] E. Najm and J.-B. Stefani. A formal semantics for the odp computational model. *Computer Networks and ISDN Systems*, 27(8):1305–1329, 1995.

[54] OpenViews: ODP Viewpoints in a Development Framework, 1998–2001. http://www.cs.kent.ac.uk/archive/research/groups/tcs/openviews/, last accessed 8-9-2011.

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

[56] S. Reeves and D. Streader. Data refinement and singleton failures refinement are not equivalent. *Formal Aspects of Computing*, 20(3):295–301, 2008.

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

[58] R. Sinnott. *An Initial Architectural Semantics in Z of the Information Viewpoint Language of Part 3 of the ODP-RM.* ISO/IEC SC21/WG7 N915, July 1994. BSI Input document to the ODP Plenary meeting in Southampton.

[59] R.O. Sinnott and K.J. Turner. Applying formal methods to standard development: The Open Distributed Processing experience. *Computer Standard and Interfaces*, 17:615–630, 1995.

[60] 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, *FME'97: Industrial Application and Strengthened Foundations of Formal Methods*, volume 1313 of *Lecture Notes in Computer Science*, pages 62–81. Springer-Verlag, September 1997.

[61] G. Smith. *The Object-Z Specification Language.* Kluwer Academic Publishers, 2000.

[62] J. M. Spivey. *The Z notation: A reference manual.* Prentice Hall, 2nd edition, 1992.

[63] S. Stepney, D. Cooper, and J. C. P. Woodcock. More powerful data refinement in Z. In ZUM'98 [15], pages 284–307.

[64] C. Sühl. RT-Z: An Integration of Z and timed CSP. In IFM'99 [2], pages 29–48.

[65] H. Treharne and S. Schneider. Using a process algebra to control B operations. In IFM'99 [2], pages 437–456.

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

[67] J. C. P. Woodcock and C. C. Morgan. Refinement of state-based concurrent systems. In D. Bjorner, C. A. R. Hoare, and H. Langmaack, editors, *VDM'90: VDM and Z!- Formal Methods in Software Development*, volume 428 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.

[68] P. Zave and M. Jackson. Conjunction as composition. *ACM Transactions on Software Engineering and Methodology*, 2(4):379–411, October 1993.