ELECTRONIC WORKSHOPS IN COMPUTING

Series edited by Professor C.J. van Rijsbergen

**D. J. Duke , University of York, UK and A.S. Evans, University of Bradford,
UK (Eds)**

2nd BCS-FACS Northern Formal Methods Workshop

Proceedings of the 2nd BCS-FACS Northern Formal Methods
Workshop, Ilkley, 14-15 July 1997

Coupling Schemas: Data Refinement and View(point) Composition

E.A. Boiten, J. Derrick, H. Bowman and M.W.A. Steen

Springer

©Copyright in this paper belongs to the author(s)

Published in collaboration with the
British Computer Society

Coupling Schemas: Data Refinement and View(point) Composition

Eerke Boiten, John Derrick, Howard Bowman, Maarten Steen

Computing Laboratory, University of Kent

Canterbury, CT2 7NF, U.K.

Phone: +44 1227 764000

Email: E.A.Boiten@ukc.ac.uk*

Abstract

We define the notion of a *coupling schema* in Z , and describe the role it plays in data refinement, view composition, and viewpoint unification. In each case coupling schemas relate several state schemas. In data refinement they occur as retrieve relations (abstraction schemas). In specification by views, coupling schemas provide a link between the various views. For viewpoint specification, coupling schemas are closely related to correspondence relations between state schemas in the viewpoints. Simple properties of coupling schemas (e.g. totality, functionality, and consistency) are shown to have important consequences in the techniques listed, and to be very useful for exhibiting the relations between these techniques. It turns out that views and viewpoints can both be seen as variations on, or even as generalisations of, data refinement.

1 Introduction

More or less independently, three different techniques in Z have been described for developing and verifying specifications by creating operations of one abstract data type from the operations of another abstract data type. The oldest and most widely known of these is data refinement. More recent are view composition [13], and viewpoint unification [4], both of these are used in specification styles which seek to reduce complexity by decomposing along aspects rather than along components. It has long been clear that all these techniques are similar and yet different, but a coherent explanation of their similarities and differences has not been given so far. This paper addresses this issue by explaining all these techniques in terms of (what we call) *coupling schemas*. Informally, a coupling schema is a schema which relates two state spaces – in section 2 we give an exact definition. In the next section we define a number of properties of coupling schemas, like totality, functionality, and consistency. Then we discuss all three techniques individually. In section 4 it is shown how coupling schemas can be viewed as abstraction schemas (retrieve relations) and as such specify data refinements. In section 5 we show how coupling schemas are used as linking invariants between views. In section 6 it is demonstrated how coupling schemas are generated from correspondence relations in the process of unifying viewpoint specifications. Finally, in the last section we relate the different techniques, showing how they differ and under which conditions they coincide.

2 Coupling Schemas

We assume the traditional states-with-operations style of specification in Z [15]. In this approach, an “abstract data type” is described by a collection of Z schemas. One of these is called the *state schema*, and all its component names are (by convention) given by standard identifiers, i.e. in particular they do *not* end in $!$, $?$, or $'$.

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

Example 1 ¹

<i>File</i>
$left, right: seq Char$

is a valid state schema (*left* and *right* standing for the text left and right of the cursor in an editor), and so is

<i>Grid</i>
$lines: seq seq Char$
$x, y: \mathbb{N}$
$lines \in wrapped$
$(x, y) \in cursors\ lines$

For definitions of *wrapped* and *cursors* and other details of the editor specification, see the appendix. In brief, *Grid* requires that *lines* is a sequence of lines of limited length, with (x, y) denoting a valid cursor position in it.

□

Any other schema of the abstract data type is a so-called operation schema². For simplicity, we make the (usual) assumption that any operation schema includes the following components. Assuming the state schema is called *D*, there is a declaration ΔD which stands for both *D* (i.e. all its components – interpreted as the “before”-state of the operation) and *D'* (all *D*’s components primed) which represents the “after”-state. Additionally there may be declarations of variables whose names end in !, denoting outputs of the operation, and variables with names ending in ?, denoting inputs. Finally there must be a predicate defining the relation between all these components (which may be just true).

Example 2 An operation on the *File* state defined above could be

<i>csrRight</i>
$\Delta File$
$right \neq \langle \rangle$
$left' = left \cap \langle head(right) \rangle$
$right' = tail(right)$

with no inputs or outputs, or

<i>insertChar</i>
$\Delta File$
$c?: Char$
$left' = left \cap \langle c? \rangle$
$right' = right$

which does have an input.

□

Given the style of specification described above, we can define what a coupling schema is.

¹Much of the example material will be drawn from the editor example as presented in [13], based on [16].

²We regard initialisation schemas as operation schemas, acknowledging however that they get a special treatment in the conditions for data refinement. We will only refer to them in the rest of this paper when strictly necessary.

Definition 3 (Coupling schema) A coupling³ schema is a schema of the form

S	
D_1	
D_2	
$pred_S$	

such that D_1 and D_2 are state schemas ($pred_S$ may be any predicate). In this case we also say that S is a coupling between D_1 and D_2 (or between D_2 and D_1).

□

D_1 and D_2 may have component names in common – this is no problem if the types of these components are compatible⁴. One might observe that, since all its components are taken from state schemas, any coupling schema may also play the role of a state schema. This will turn out to be important later.

Example 4 A possible coupling between *File* and *Grid* is:

<i>Editor</i>	
<i>File, Grid</i>	
$left \cap right = \cap / lines$	
$\#left = x + \Sigma(i : 1 \dots y \Leftarrow 1 \bullet \#lines[i])$	

which links the states by requiring that both views represent the same text, and that the cursor positions match: *left* should be as long as all of the lines before line *y* together, plus all of line *y* up to column *x*.

□

A slightly more liberal notion is that of a proto-coupling defined below, which instead of including the complete state schemas only requires their declaration parts.

Definition 5 (Proto-coupling) Any schema of the form

T	
$Decls_1$	
$Decls_2$	
$pred_T$	

is a *proto-coupling schema* for schemas $D_1 \hat{=} [Decls_1 \mid pred_1]$ and $D_2 \hat{=} [Decls_2 \mid pred_2]$

□

Theorem 6 For proto-coupling T , D_1 and D_2 as in Definition 5, T is a coupling between D_1 and D_2 iff

$$pred_T \Rightarrow pred_1 \wedge pred_2$$

Proof If the condition holds, T is equivalent to $[D_1; D_2 \mid pred_T]$ and thus a coupling between D_1 and D_2 according to Definition 3. Conversely, the condition is also the weakest guarantee for the predicates $pred_i$ to hold.

□

³This term was introduced in [9] to avoid loaded terms like “abstraction” and “representation”.

⁴We will assume that this is the case, i.e. we only discuss type-correct Z specifications.

So far we have only defined the notion of a coupling schema, but we have not explained what its purpose is, or why it is a useful notion. Generally speaking, given a coupling schema between two states, and operations defined on one of them, we want to use the coupling schema to generate operations on one of the other two state spaces involved (the other state and the coupling) which are in some sense comparable to the original operations. In the rest of this paper we will describe three variants on this theme. In order to be able to characterise these more clearly, and to describe their relations, we first define some useful properties of coupling schemas.

3 Properties of Coupling Schemas

The notion of consistency derives from viewpoint unification, but it gives a clear indication why coupling schemas are more useful than proto-coupling schemas. Informally, consistency forbids the proto-coupling to link two states of which exactly one is excluded by its predicate.

Definition 7 (Consistency) A proto-coupling schema T between D_1 and D_2 , all defined as in Definition 5, is *consistent* iff

$$pred_T \Rightarrow (pred_1 \Leftrightarrow pred_2)$$

□

Example 8 Let $D \hat{=} [x:\mathbb{N}]$ and $E \hat{=} [y:\mathbb{Z} \mid y > 0]$, then $S \hat{=} [x:\mathbb{N}; y:\mathbb{Z} \mid x=y]$ is an inconsistent proto-coupling between D and E . Informally, D 's legal value 0 is linked to E 's illegal 0.

□

Theorem 9 Every coupling schema is consistent.

Proof From Theorem 6 it follows that $pred_1$ and $pred_2$ should both hold when $pred_T$ does, and both being true they are also equivalent.

□

Definition 10 (Functionality) A proto-coupling S between D_1 and D_2 is *p-functional* from D_1 to D_2 iff it can be interpreted as a partial function from D_1 to D_2 , i.e.

$$\forall D_1 \bullet (\exists D_2 \bullet S \Rightarrow \exists_1 D_2 \bullet S)$$

□

Example 11 The schema *Editor* is p-functional from *Grid* to *File* – for every *Grid* a unique pair of (*left*, *right*) can be found. However, it is *not* p-functional from *File* to *Grid* – this represents the fact that in general a *File* can be formatted as a *Grid* in many ways.

□

Definition 12 (Totality) A proto-coupling S between D_1 and D_2 is *total* on D_1 iff

$$\forall D_1 \bullet \exists D_2 \bullet S$$

□

Example 13 *Editor* is total on *Grid*, and it takes a little reasoning to show that it is *not* total on *File*. Lines can only be formatted by breaking them *between* words, and the lines in a *Grid* are of a limited length. Thus, if a *File* contains a word that is longer than the maximum line length, there is no corresponding *Grid*.

□

Definition 14 (Derived schema) A state schema D_1 is *derived* from a state schema D_2 by a proto-coupling S between D_1 and D_2 iff

$$\forall D_2 \bullet \exists_1 D_1 \bullet S$$

□

There is a better known name for this notion. A schema is derived iff it is the image of a function (i.e. a total and p-functional coupling schema):

Theorem 15 For a proto-coupling S between D_1 and D_2 , D_1 is derived from D_2 by S iff S is total on D_2 and p-functional from D_2 to D_1 .

□

We can say that *File* is derived from *Grid* (leaving the coupling schema involved implicit). A special case of a derivation is *subsumption*:

Definition 16 (Subsumption) A state schema D is subsumed by a state schema E in a coupling schema $S \triangleq [D; E \mid pred_S]$ iff $S \Leftrightarrow [E \mid pred_S]$. When no coupling schema is mentioned, D is subsumed by E implies it is subsumed for all possible coupling schemas (i.e. in $[D; E \mid \text{true}]$).

□

This implies that every component of D also occurs with a compatible type in E , and that the predicate of E together with $pred_S$ imply the predicate of D . Hiding the extra components of a subsumed schema returns a restriction of the subsumed schema, the resulting schema's predicate may be stronger. Clearly any coupling schema subsumes the state schemas it is defined between.

Another special case of a derivation is isomorphy between schemas.

Definition 17 (Isomorphy) Two state schemas D_1 and D_2 are *isomorphic* iff there exists a coupling S between D_1 and D_2 by which D_1 is derived from D_2 and D_2 is derived from D_1 .

In that case, the abstract data types (D_1, Ops_1) and (D_2, Ops_2) are isomorphic if Ops_1 and Ops_2 contain the same set of operation names, and if for each pair of corresponding operations Op_1 and Op_2 it is the case that $S \wedge Op_1 \wedge S' \Leftrightarrow S \wedge Op_2 \wedge S'$. (However, abstract data types may be isomorphic even when their state schemas are not.)

□

S being a total function in both directions implies that it is a total bijection.

Example 18 $[x:\mathbb{N}]$ and $[y:\mathbb{Z}]$ are isomorphic, as witnessed by the coupling schema

$x:\mathbb{N}; y:\mathbb{Z}$
$(y \in \mathbb{N} \wedge x = 2*y) \vee (y \notin \mathbb{N} \wedge x = \Leftrightarrow 2*y \Leftrightarrow 1)$

□

4 Data Refinement

Data refinement is a non-symmetric relation between abstract data types. One data type (the “concrete” one) is said to implement another (the “abstract” one) if their initialisations are compatible, and if every “abstract” operation is correctly represented by one “concrete” operation. We refer to [15, 17] for the exact details of this – in particular, in this paper we have little interest in operation refinement, and do not consider forward vs. backward simulations.

As above, data refinement is often posed as a verification problem – given the two abstract data types, prove that data refinement holds. Central in this is the so-called *abstraction schema* or *retrieve relation*, which links the “concrete” and the “abstract” state. If this is indeed represented as a schema, it is a clear instance of a coupling schema: its signature is the combination of the abstract and concrete state schemas, and it has an additional predicate denoting which concrete value represents which abstract one.

Data refinement is often presented as a generalisation of operation refinement. A folk theorem in the Z world is that the aspect of operation refinement can be factored out. Thus, a data refinement step in the traditional sense consists of a “most general” data refinement (determined only by the retrieve relation – this idea probably derives from [12, 14]) possibly followed by an operation refinement step. Woodcock and Davies [17] apply this in their section on “Calculating data refinements”; Ainsworth et al [3] specify a data refinement (indeed the most general one possible) by the retrieve relation only.

Following this folk theorem, we can interpret any coupling schema as specifying uniquely (up to some sensible equivalence) a data refinement from one of its state spaces to the other one. Generally this will only satisfy the conditions of data refinement in one of the two possible directions – an abstraction schema, apparently, is a coupling schema with a direction.

Definition 19 (Data refinement from coupling) Given an abstract data type $\mathcal{D}$ with state schema D and a coupling schema R defined by

R	
D	
E	
$pred_R$	

the (*weakest candidate*) data refinement from $\mathcal{D}$ determined by R is the abstract data type $\mathcal{E}$ whose state schema is E , and which contains for each operation in $\mathcal{D}$ given by

$D\text{Op}$	
ΔD	
$Decls$	
op	

an operation in $\mathcal{E}$ given by

$E\text{Op}$	
ΔE	
$Decls$	
$\exists D \setminus E, D' \setminus E' \bullet pred_R \wedge op \wedge pred_{R'}$	

□

Note that this is only a *candidate* weakest data refinement: if there exists one, this is it. However, the data refinement condition for the initialisation schema, and non-totality of the coupling schema on the abstract state may prevent the existence of *any* valid data refinement for this particular coupling schema. The construction above is a straightforward generalisation of the construction used by Woodcock and Davies for calculating data refinements.

As is well known, the conditions for data refinement can be much simplified if the abstraction schema is a total function from the abstract to the concrete state. This shows the relevance of totality and p-functionality for coupling schemas in this case. Indeed, Woodcock and Davies calculate data refinements only when these conditions hold. It should be no surprise that the refinements can be “calculated” when the concrete state is “derived” (our terminology) from the abstract state, i.e. when the implicit relational compositions in EOp are functional compositions. (Though the occurrence of an inverse function may still cause calculational problems.)

5 View Composition

The method of specification by *views* is presented by D. Jackson in [13]:

A view is a partial specification of a program, consisting of a state space and a set of operations. A full specification is obtained by composing several views, linking them through their states (by asserting invariants across views) ...

Specification by views is proposed as a method to overcome problems of complexity, to allow separation of concerns, and as such it is comparable to viewpoint specification as discussed in the next section. One of the contributions of the current paper is to pinpoint the difference between these two techniques, see section 7.

The above description of views and how to link them bears a close resemblance to data refinement, and to our description of the general use of coupling schemas. Clearly “a state space and a set of operations” can be viewed to be an abstract data type – making explicit that the operations are indeed assumed to be on *that* state space. (Note that Jackson lists additional methods of composing views, viz. through their operations, which are closer to promotion-based techniques and which we will not discuss here.) The focus is on a different state space, however: view composition uses the fact that a coupling schema is a state schema itself.

Definition 20 (View composition) The view composition of abstract data types $\mathcal{D}$ and $\mathcal{E}$ with state schemas D and E with respect to a coupling schema R between D and E is an abstract data type which includes only the following:

- R as its state schema
- For each operation Op in $\mathcal{D}$, an operation $ROp \triangleq [\Delta R \mid Op]$
- For each operation Op in $\mathcal{E}$, an operation $ROp \triangleq [\Delta R \mid Op]$

If $\mathcal{D}$ and $\mathcal{E}$ have operation(-name)s in common, the new abstract data type includes the conjunction of the two modified operations for each of those instead of the modified operations.

□

Example 21 By composing the *File* and *Grid* views, using the *Editor* coupling, the following operations are added to the newly generated abstract data type (whose state schema is *Editor*):

$$csrRightE \triangleq [\Delta Editor \mid csrRight]$$

$$insertCharE \triangleq [\Delta Editor \mid insertChar]$$

□

One could almost say that view composition is a symmetrical variant of data refinement. In the final section we will be more specific.

6 Viewpoint Unification

The central idea in the viewpoint specification approach [10] is that a specification consists of a collection of interlocking partial specifications, each of which describes the envisaged system from a different *viewpoint*. The main argument in favour of this is that this “horizontal decomposition” gives a more natural separation of concerns. Z is considered suitable for this approach because it naturally allows a great degree of underspecification. For collections of partial specifications to be meaningful, *consistency* between them has to be established. This is because viewpoints describe aspects of the same system, and thus, in general, overlap. Characterisations of overlaps are called *correspondences*, using the terminology of the Open Distributed Processing viewpoints framework. Our interpretation of consistency between viewpoints [5] is that common developments (refinements) of the viewpoints must exist. A constructive way of checking this is by producing (least) common refinements, so-called *unifications*. If the refinement relation involved possesses suitable theoretical properties (which refinement in Z does), consistency checking between a series of viewpoints can be done by repeatedly replacing two viewpoints by their least common refinement until only one consistent specification is left.

Viewpoint specification in Z has been extensively described in papers by Ainsworth, Wallis et al [2, 3] and ourselves [7, 4]. The explanation given here summarizes the discussion in our earlier papers. Consistency checking and unification is performed separately for each data type occurring in the two viewpoints. Thus, the simplified unification problem is between two abstract data types. A behavioural approach might be to relate traces of one data type with the possible traces of the other, but the more obvious approach is to directly relate the state spaces involved. This, in its turn, will have to consider all components of the state schemas. If both contain a declaration, say, $count:\mathbb{N}$, and these represent conceptually the same counter, then the unified state will probably have to contain this component as well. However, if one contains $valueseq: seq_1 \mathbb{N}$ and the other $valueset: \mathbb{P}\mathbb{N}$, it is not clear what the unified state should include. Maybe both represent a set of values in the state – but only the second one allows this set to be empty, how should we deal with that? Clearly more information is needed than just the two state schemas only – we need a *correspondence*, which will tell us exactly which value in one state space is “conceptually the same as” which value in the other state. Here the link to the general idea in this paper emerges:⁵

Definition 22 (Correspondence relation) A correspondence relation between state schemas D and E is a proto-coupling between D and E .

□

We will restrict the discussion in the rest of this section to state schemas which have only one component. This does not restrict expressiveness of the approach, since the type of the single component can be a product or a schema type – it only serves to reduce the amount of distracting detail. For the same reason we will only discuss cases where the different components have different names.

Example 23 Given state schemas $D \triangleq [valueseq: seq \mathbb{N} \mid valueseq \neq \langle \rangle]$ and $E \triangleq [valueset: \mathbb{P}\mathbb{N}]$, two possible proto-couplings between D and E are

$NEmp$ <hr/> $valueseq: seq \mathbb{N}$ $valueset: \mathbb{P}\mathbb{N}$ <hr/> $ran \ valueseq = valueset$	Emp <hr/> $valueseq: seq \mathbb{N}$ $valueset: \mathbb{P}\mathbb{N}$ <hr/> $ran \ valueseq = valueset$ $valueseq \neq \langle \rangle$
--	---

These proto-couplings only differ in how they treat the empty set/sequence. $NEmp$ links the empty set, a valid inhabitant of E , to the empty sequence, which is not included in D . In Emp these two are not connected. Observe (trivially from Theorem 6) that Emp is actually a coupling schema, but $NEmp$ is not.

□

Our interpretation of a value from one viewpoint state not being linked by the correspondence relation to any value

⁵If we further interpret “conceptually the same”, we might put extra restrictions on a correspondence relation, e.g. that it is a 1-1 relation between the equivalence classes it induces on both state spaces. However, such restrictions have not seemed useful in any proofs or definitions so far.

from the other viewpoint state is that it is an *allowed* value which, however, is not modelled in the other viewpoint. This allows us to model all kinds of unions, for example adding values to free types, cf. [4]. Moreover, this issue bears a direct relation to the unification being a refinement of the viewpoints. The only way to ensure that *any* possible operation of the viewpoint can be refined by a unified operation is called *state consistency* in [4], see that paper for further details.

Theorem 24 Two viewpoints (abstract data types) are *state consistent* [4] iff the correspondence relation between their state schemas is consistent.

□

Corollary 25 Two viewpoints are state consistent whenever their correspondence relation is a coupling schema. (Cf. Theorem 9.)

□

Example 26 Continuing from Example 23, *Emp* as a correspondence relation ensures state consistency, but *NEmp* does not. We could also have concluded the former from Corollary 25.

□

Thus far we have established that we need consistency, and also that using a coupling schema rather than a proto-coupling schema would guarantee consistency. So, do we really need the extra generality offered by proto-couplings if it only brings trouble? The answer to that is “Yes”. We *need* to be made aware that of two values which are linked by the correspondence, one is excluded through an additional invariant of the other viewpoint! It is much better for troublesome assumptions to show up as state inconsistencies rather than as (possibly undesirable) restrictions on operations.

For example (returning to Example 23), if *D* and *E* are indeed intended to describe the same set of values, giving *NEmp* as the correspondence relation allows us to discover that the viewpoints have inconsistent assumptions as to whether this set may be empty or not – this will be exhibited by *NEmp* being state inconsistent. On the other hand, if we use *Emp* the empty set will be considered as an intermediate state in which only operations from *E*’s viewpoint may be applicable – not an inconsistency for *D*, only additional refinement of the state space.

Now for the next difference with view composition. Like with view composition, in viewpoint unification we are going to create an abstract data type whose state schema is derived from the (proto-)coupling involved. However, our goal is now not to have syntactic simplicity like in view composition (and the Z schema calculus in general). For viewpoint unification we want *refinement* between the new abstract data type and each of the viewpoint abstract data types. Moreover, we want the refinement to be the most general one possible which respects the correspondence relation, implementing as many elements from the original state spaces as possible. This means that (for a consistent correspondence) three kinds of values will occur in the unified state space: pairs of values from the correspondence; values from one viewpoint which are not related by the correspondence; and similar for the other viewpoint. Formally, this amounts to a *totalisation*⁶ of the proto-coupling, which can be modelled using the *optional* construct [8]. This gives for any type the type of sets from that type with at most one element, which implements that a value may be either *undefined* (empty set), or a *defined* value from that type (singleton set).

$$\text{optional}[X] == \{xs : \mathbb{P} X \mid \#xs \leq 1\}$$

⁶Related to, but different from the ones used in [17].

$[X]$
$\text{defined_} : \mathbb{P}(\text{optional}[X])$ $\text{undefined_} : \mathbb{P}(\text{optional}[X])$ $\text{the} : \text{optional}[X] \rightarrow X$
$\forall xs : \text{optional}[X] \bullet$ $\text{defined } xs \Leftrightarrow (\#xs = 1)$ $\text{undefined } xs \Leftrightarrow (\#xs = 0)$ $\text{defined } xs \Rightarrow (\text{the } xs = (\mu x : X \mid x \in xs))$

Using this we can define the totalisation of a *relation*:

$[X, Y]$
$\text{tot} : X \times Y \rightarrow \text{optional}[X] \times \text{optional}[Y]$
$\forall R : X \times Y \bullet$ $\text{tot } R = \{(x, y) : R \bullet (\{x\}, \{y\})\} \cup \{x : X \setminus \text{dom } R \bullet (\{x\}, \emptyset)\}$ $\cup \{y : Y \setminus \text{ran } R \bullet (\emptyset, \{y\})\}$

Definition 27 (Coupling totalisation) Given a proto-coupling R between $D \triangleq [x : S \mid \text{pred}_D]$ and $E \triangleq [y : T \mid \text{pred}_E]$ defined by $R \triangleq [x : S; y : T \mid \text{pred}_R]$, its coupling totalisation $\hat{R}$ with respect to D and E is given by

$\hat{R}$
$xx : \text{optional}[S]$ $yy : \text{optional}[T]$
$(xx, yy) \in \text{tot}[S, T] \{R \bullet (x, y)\}$ $\text{defined } xx \Rightarrow \text{pred}_D[\text{the } xx / x]$ $\text{defined } yy \Rightarrow \text{pred}_E[\text{the } yy / y]$

□

The first predicate totalises the relation characterised by pred_R ; the other predicates effectively turn the proto-coupling into a coupling by restriction to the state predicates.

Finally this allows a definition of viewpoint unification. The coupling totalisation of the proto-coupling will be the state schema of the generated abstract data type.

Definition 28 (Viewpoint unification) The viewpoint unification of abstract data type $\mathcal{D}$ with state schema $D \triangleq [x : S \mid \text{pred}_D]$ and abstract data type $\mathcal{E}$ with state schema $E \triangleq [y : T \mid \text{pred}_E]$ according to a proto-coupling $R \triangleq [x : S; y : T \mid \text{pred}_R]$ is the abstract data type which includes only the following:

- as its state schema $\hat{R}$, the coupling totalisation of the proto-coupling;
- for each operation $Op \triangleq [\Delta D; \text{Decls} \mid \text{pred}]$ in $\mathcal{D}$, an operation

Op
$\Delta \hat{R}$ Decls
$\text{defined } xx \wedge \text{defined } xx'$ $\text{let } x = \text{the } xx; \ x' = \text{the } xx' \bullet \text{pred}$

- similarly, for each operation $Op \triangleq [\Delta E; \text{Decls} \mid \text{pred}]$ in $\mathcal{E}$, an operation

Op $\Delta \hat{R}$ $Decls$
$\text{defined } yy \wedge \text{defined } yy'$ $\mathbf{let } y = \text{the } yy ; y' = \text{the } yy' \bullet pred$

If $\mathcal{D}$ and $\mathcal{E}$ have operation(-name)s in common, the new abstract data type includes the *operation unification* (cf. definition below) of the two modified operations for each of those instead of those modified operations.

□

Definition 29 (Operation unification) (Also called *amalgamation* [3] or *demonic join* [11].) The operation unification of two operations $Op1$ and $Op2$ on the same state schema is given by

$$(\text{pre } Op1 \Rightarrow Op1) \wedge (\text{pre } Op2 \Rightarrow Op2) \wedge (\text{pre } Op1 \vee \text{pre } Op2)$$

□

The usefulness of this is given by the following:

Theorem 30 (Operation consistency and refinement) If two operations $Op1$ and $Op2$ have a most general common operation refinement, it is their operation unification. Their operation unification is a refinement of both operations iff they are *operation consistent*, viz.

$$\text{pre } Op1 \wedge \text{pre } Op2 \Rightarrow Op1 \wedge Op2$$

holds.

Proof See [4].

□

Theorem 31 The viewpoint unification of two abstract data types is their least common data refinement whenever they have a common refinement (e.g. when the proto-coupling is consistent and all matching modified operations are operation consistent).

Proof See [4].

□

It is important to observe that not in all cases as much formal machinery is necessary in order to construct a unified state space. Most importantly, if the proto-coupling R is a total coupling on any one of the states involved, the corresponding component in $\hat{R}$ will always be **defined**, and thus does not need to be of the optional type. To put it differently, a schema, isomorphic to $\hat{R}$, will exist which does not have that component **optional**.

Example 32 Using the *Editor* schema as a (proto-)coupling between *File* and *Grid*, and a straightforward extension of the above definitions for multiple component schemas (where the components from one viewpoint state are either all defined, or all undefined), the viewpoint unification of these has as its state schema

<i>Editor</i>
$left, right: \text{optional}[\text{seq } Char]$ $lines: \text{optional}[\text{seq seq } Char]$ $x, y: \text{optional}[\mathbb{N}]$
$\text{defined } left \Leftrightarrow \text{defined } right$ $\text{defined } x \Leftrightarrow \text{defined } y \Leftrightarrow \text{defined } lines$ $\text{defined } x \vee \text{defined } left$ $\text{defined } lines \Rightarrow thelines \in wrapped$ $\text{defined } lines \Rightarrow (the x, the y) \in cursors(thelines)$ $\text{defined } left \wedge \text{defined } lines \Rightarrow theleft \wedge theright = \wedge / thelines$ $\text{defined } left \wedge \text{defined } lines \Rightarrow \#theleft = thex + \Sigma(i : 1 \dots they \Leftrightarrow 1 \bullet \#thelines[i])$ $\text{undefined } lines \Rightarrow \neg \exists Grid \bullet Editor[theFile / File]$ $\text{undefined } left \Rightarrow \neg \exists File \bullet Editor[theGrid / Grid]$

(using the abbreviated substitution $theGrid / Grid$ for “the” applied to each component). However, because *Editor* is total on *Grid*, this is isomorphic to

<i>EditorTot</i>
$File$ $lines: \text{optional}[\text{seq seq } Char]$ $x, y: \text{optional}[\mathbb{N}]$
$\text{defined } x \Leftrightarrow \text{defined } y \Leftrightarrow \text{defined } lines$ $\text{defined } lines \Rightarrow thelines \in wrapped \wedge (the x, the y) \in cursors(thelines)$ $\text{defined } lines \Rightarrow left \wedge right = \wedge / thelines$ $\text{defined } lines \Rightarrow \#left = thex + \Sigma(i : 1 \dots they \Leftrightarrow 1 \bullet \#thelines[i])$ $\text{undefined } lines \Rightarrow \neg \exists Grid \bullet Editor[theFile / File]$

Its operations might include an adaptation of *csrRight* which due to totality of *Editor* on *Grid* could be written as

$$csrRightE \hat{=} [\Delta EditorTot \mid csrRight]$$

and an adaptation of *csrUp* (cf. the appendix) which could be simplified to:

<i>csrUpE</i>
$\Delta EditorTot$
$(\text{undefined } y \wedge \text{undefined } y') \vee$ $the y \geq 1 \wedge the y' = the y \Leftrightarrow 1 \wedge the x' = \min(the x, \#thelines[the y']) \wedge thelines' = thelines$

which is essentially *csrUp* decorated with lots of occurrences of “the”, plus an additional disjunct which represents the fact that a *File* which cannot be formatted as a *Grid* has no sensible interpretation of moving the cursor up.

□

7 Comparisons and Conclusions

Figure 1 illustrates the differences between the constructions in the three techniques. In all three, new operations are generated from old ones according to a (proto-)coupling. In the last two techniques a new state is constructed as well – in view composition this state is the coupling schema, in viewpoint unification it is constructed from the coupling

Figure 1: The various techniques depicted graphically. The line at the top in each denotes a coupling R between D and E . A stack of circles denotes an abstract data type (state above operations). The parts generated by the technique are indicated by double circles, with dotted arrows showing what they are generated from.

schema. In the rest of this section we will show the semantic relationships between these techniques: when do their results coincide, and which specifications are refinements of each other? In each case we will assume a coupling schema $R \triangleq [D; E \mid pred_R]$ between state schemas D and E as in the figure.

7.1 Data Refinement and View Composition

Clearly in data refinement (according to Definition 19) the resulting abstract data type is a data refinement, *provided* the initialisation condition is met and the state space is not unduly restricted. If this is not the case, there is no data refinement with the given coupling schema, as discussed before.

Data refinement from D to E can only coincide with view composition if the operations generated in both cases are on the same state space. Thus, D needs to be subsumed by E in R , making E (or R) the resulting state space in both cases. The fact that there is a restriction like that implies that neither technique is a special case of the other. When E subsumes D (and additionally, E 's datatype has no operations defined on it initially) the result of view composition and most general data refinement do indeed coincide. Assume that $D \circ p \triangleq [\Delta D; Decls \mid op]$. Then the result of data refinement for this operation is given by

$$[\Delta E; Decls \mid \exists D \setminus E, D' \setminus E' \bullet pred_R \wedge op \wedge pred_{R'}]$$

$$\begin{aligned}
 &\equiv \{ E \text{ subsumes } D, \text{ so } D \setminus E \text{ is empty} \} \\
 &\quad [\Delta E; \text{Decls} \mid \text{pred}_R \wedge \text{op} \wedge \text{pred}_R'] \\
 &\equiv \{ \text{from subsumption it follows that } E \wedge \text{pred}_R \Leftrightarrow R \} \\
 &\quad [\Delta R; \text{Decls} \mid \text{op}] \\
 &\equiv \{ \text{schema calculus} \} \\
 &\quad [\Delta R \mid \text{Dop}]
 \end{aligned}$$

which is indeed what would have resulted in view composition. In summary:

Theorem 33 Data refinement from D to E according to R coincides with view composition between D and E when E subsumes D in R .

□

This may create the impression that view composition always results in refinements. However, this is not the case in general (for more exceptions see the next subsection).

Theorem 34 The data type resulting from view composition is, in general, not a data refinement of its operand data types.

Proof (By counterexample:) the datatype consisting of *File* and *insertChar* is *not* data refined by the datatype consisting of *Editor* and *insertCharE*. The cause of this is related to *Editor* not being total on *File* (cf. Example 13): the situation where a character is added to a word which was of maximum line length is included in (the precondition of) *insertChar* but not in *insertCharE* (since in that case there is no *Editor'* which satisfies the predicate).

Additionally, if the data types involved have an operation in common, this might also prohibit the view composition being a refinement. It is a well-known fact that the operation $S \wedge T$ is not in general an operation refinement of S or of T .

□

7.2 View Composition and Viewpoint Unification

The remarks at the end of the previous section already indicated that one crucial issue in this comparison is totality. In fact, we have

Theorem 35 With respect to a proto-coupling which is a total coupling on both state schemas involved, the state schemas resulting from view composition and viewpoint unification are isomorphic.

Proof Let the total coupling be given by $R \triangleq [D; E \mid \text{pred}_R]$ and w.l.o.g. $D \triangleq [x:S \mid \text{pred}_D]$, $E \triangleq [y:T \mid \text{pred}_E]$. The trivial coupling schema which witnesses the isomorphy is

$ \begin{array}{l} \text{TotalIso} \\ R; \hat{R} \\ \hline x = \text{thex}x \\ y = \text{they}y \end{array} $

□

This tells us in which case the generated state schemas are (as good as) equal. Additional conditions are necessary for the operations on the generated state to be equally close. It is clear that the ways in which operations are modified to the new state are comparable, with what is added to operations in Definition 28 closely resembling *TotalIso* above and *TotalIso'*.

Theorem 36 If, in addition to the conditions of Theorem 35, the sets of operation names in both abstract data types are disjoint, the abstract data types resulting from view composition and viewpoint unification are isomorphic.

□

If the abstract data types have operations in common, for view composition and viewpoint unification to coincide, conjunction and demonic join have to coincide. For a different motivation (viz. finding cases where demonic join can be expressed in the schema calculus), this is investigated in [3]. Conjunction and demonic join turn out to coincide for operations whose preconditions are equivalent. Thus we can generalise the previous theorem to:

Theorem 37 If, in addition to the conditions of Theorem 35, for each operation common to both abstract data types, both its versions for the new abstract data type have equivalent preconditions (either in view composition or in viewpoint unification), the abstract data types resulting from view composition and viewpoint unification are isomorphic.

□

Furthermore, we can observe that the demonic join of two operations is an operation refinement of their conjunction, and totalisation is a total injection. The consequence of this is interesting but not very useful:

Theorem 38 The viewpoint unification of two abstract data types according to a particular coupling is a data refinement of their view composition according to that coupling.

□

7.3 Refinement and viewpoint unification

Theorem 31 already presented an important relation between refinement and unification: unification is the least common refinement provided one exists. Also, results from the previous subsections can be combined to deduce relationships between refinement and unification from their respective relationships to viewpoint composition. The most important relation between refinement and unification is actually a result of the general theory of development and unification as we have described it in [5]⁷: unification generalises refinement.

Given two abstract data types $\mathcal{A}$ and $\mathcal{B}$ such that $\mathcal{A}$ is a data refinement of $\mathcal{B}$ with respect to coupling schema S , the viewpoint unification of $\mathcal{A}$ and $\mathcal{B}$ with respect to S is *not* identical to $\mathcal{A}$, and their state schemas are not isomorphic in general. (The unification state schema will include all components from the state of $\mathcal{B}$ in addition to those of $\mathcal{A}$.) However, the unification of $\mathcal{A}$ and $\mathcal{B}$ is isomorphic to $\mathcal{A}$ in that situation, using the normal definition of isomorphism as mutual data refinement (which generalises the special case mentioned in Definition 17):

Theorem 39 (Unification generalises refinement) If abstract data type $\mathcal{A}$ is a refinement of abstract data type $\mathcal{B}$ with respect to coupling schema S , the abstract data type $\mathcal{C}$ resulting from viewpoint unification of $\mathcal{A}$ and $\mathcal{B}$ with respect to S is isomorphic to $\mathcal{A}$.

Proof We need to prove that $\mathcal{A}$ and $\mathcal{C}$ are data refinements of each other. Clearly $\mathcal{A}$ and $\mathcal{B}$ have a common refinement (viz. $\mathcal{A}$), thus according to Theorem 31 $\mathcal{C}$ exists and is the least common refinement of $\mathcal{A}$ and $\mathcal{B}$. Thus, $\mathcal{C}$ is a refinement of $\mathcal{A}$. Moreover, because $\mathcal{A}$ is also a common refinement of $\mathcal{A}$ and $\mathcal{B}$, it is also a refinement of $\mathcal{C}$ since $\mathcal{C}$ is the *least* of these common refinements.

□

⁷Or rather an instance of the general lattice property that $a \leq b \Rightarrow a \sqcap b = a$.

7.4 Conclusions

Three techniques for developing and verifying Z abstract data types have been explained in terms of the notion of a coupling schema. Coupling schemas turned out to provide a powerful framework in which to understand these techniques. The best illustration of that is probably the sheer simplicity of Figure 1, which makes the structural differences between the techniques clear. Complementary to that, the semantical properties and differences between the techniques have been reduced to properties of the coupling schemas involved.

Besides for these three techniques, coupling schemas could also be used to describe (bi-)simulation relations, used in behavioural interpretations of Z (for example [6]).

We have presented all these techniques for the specific language Z, but clearly the individual techniques, and thus probably also the general idea, are applicable to a much wider range of specification languages (for example B [1] or the binary relations as used by Frappier et al [11]). The presentations would be quite different, however, since much of the current paper revolves around Z schemas being interpreted and used in varying ways.

Acknowledgements

We would like to thank two ZUM'97 referees for their comments on a precursor of this paper: one for suggesting the generalisation through coupling schemas, and the other for helpful explanations concerning refinement. A referee for NFMW also made a number of helpful suggestions. Carroll Morgan is thanked for helping us trace the origin of the term "coupling" (to former colleagues of one of us ...)

References

- [1] Abrial J-R. The B-Book: Assigning Programs to Meanings. Cambridge University Press, 1996
- [2] Ainsworth M, Cruickshank AH, Wallis PJL, Groves LJ. Viewpoint specification and Z. *Information and Software Technology* 1994; 36(1):43-51
- [3] Ainsworth M, Riddle S, Wallis PJL. Formal validation of viewpoint specifications. *Software Engineering Journal* 1996; 11(1):58-66
- [4] Boiten E, Derrick J, Bowman H, Steen M. Consistency and refinement for partial specification in Z. In: Gaudel M-C, Woodcock J (eds) *FME'96: Industrial Benefit of Formal Methods, Third International Symposium of Formal Methods Europe*. Springer-Verlag, 1996, pp 287-306 (Lecture Notes in Computer Science No. 1051)
- [5] Bowman H, Steen MWA, Boiten EA, Derrick J. A formal framework for viewpoint consistency. Submitted for publication, 1997 (Available from <http://alethea.ukc.ac.uk/Dept/Computing/Research/NDS/consistency>)
- [6] Derrick J, Boiten EA, Bowman H, Steen M. Translating LOTOS to Object-Z. This volume
- [7] Derrick J, Bowman H, Steen M. Viewpoints and Objects. In: Bowen JP, Hinchey MG (eds): *Ninth Annual Z User Workshop*. Springer-Verlag, 1995, pp 449-468 (Lecture Notes in Computer Science No. 967)
- [8] d'Inverno M, Hu M. A Z specification of the soft-link hypertext model. In: Bowen JP, Hinchey MG, Till D (eds): *ZUM '97: The Z Formal Specification Notation Workshop*. Springer-Verlag, 1997, pp 297-316 (Lecture Notes in Computer Science No. 1212)
- [9] Feijen WHJ, van Gasteren AJM, Gries D. In-situ inversion of a cyclic permutation. *Information Processing Letters* 1987; 24
- [10] Finkelstein A, Spanoudakis G (eds). *SIGSOFT '96 International Workshop on Multiple Perspectives in Software Development (Viewpoints '96)*. ACM, 1996

- [11] Frappier M, Mili A, Desharnais J. Program construction by parts. In: Möller B (ed) Mathematics of Program Construction: Third International Conference. Springer-Verlag, 1995, pp 257-281 (Lecture Notes in Computer Science No. 947)
- [12] He Jifeng, Hoare CAR, Sanders JW. Data refinement refined. In: Robinet B, Wilhelm R (eds) Proc. European Symposium On Programming 86. Springer-Verlag, 1986, pp 187-196 (Lecture Notes in Computer Science No. 213)
- [13] Jackson D. Structuring Z specifications with views. ACM Transactions on Software Engineering and Methodology 1995; 4(4)
- [14] He Jifeng, Hoare CAR. Prespecification and data refinement. In: Data Refinement in a Categorical Setting. Oxford University Computing Laboratory, 1990 (Technical Monograph PRG-90)
- [15] Spivey JM. The Z notation: A reference manual. Prentice Hall, 1989
- [16] Suffrin BA. Formal specification of a display-oriented text editor. Science of Computer Programming 1982; 1:157-202
- [17] Woodcock J, Davies J. Using Z: Specification, Refinement, and Proof. Prentice Hall, 1996

Appendix: Details of the Editor Specification

In the *Grid* view, the state is a sequence of sequences of limited length, each constituent sequence representing a line, with a cursor position. This allows specification of operations like moving the cursor down.

A legal line is one that has a space or a newline as its last character, with no internal newlines.

$$\begin{array}{|l}
 nl, sp : Char \\
 \hline
 nl \neq sp \\
 \\
 legallines : \mathbb{P} \text{ seq } Char \\
 \hline
 \forall l : \text{seq } Char \bullet \\
 l \in legallines \Leftrightarrow (\forall i \in \text{dom } l \bullet l[i] = nl \Rightarrow i = \#l) \\
 \wedge (l[\#l] \in \{nl, sp\})
 \end{array}$$

A sequence of lines is wrapped if it has all legal lines of limited length.

$$\begin{array}{|l}
 maxlenlength : \mathbb{N} \\
 \\
 wrapped : \mathbb{P} \text{ seq seq } Char \\
 \hline
 \forall ls : \text{seq seq } Char \bullet \\
 ls \in wrapped \Leftrightarrow (\text{ran } ls \subseteq legallines) \\
 \wedge (\forall l \in \text{ran } ls \bullet \#l \leq maxlenlength)
 \end{array}$$

The set of all valid cursor positions in a sequence of sequences is given by (this is an extension of the predicate used by Jackson, this one also assigns a cursor to the empty grid):

$$\begin{array}{|l}
 [X] \\
 \hline
 cursors : \text{seq seq } X \rightarrow \mathbb{P}(\mathbb{N} \times \mathbb{N}) \\
 \hline
 \forall (ls : \text{seq seq } X); (x, y : \mathbb{N}) \bullet \\
 (x, y) \in cursors \, ls \Leftrightarrow (x \in \text{dom}(ls[y]) \wedge y \in \text{dom } ls) \\
 \vee (\text{dom } ls = \emptyset \wedge (x, y) = (0, 1))
 \end{array}$$

The complete *Grid* view then consists of a two dimensional array with a valid cursor position.

<i>Grid</i>	
$lines : seq\ seq\ Char$	
$x, y : \mathbb{N}$	
$lines \in wrapped$	
$(x, y) \in cursors\ lines$	

As an operation on *Grid* one could imagine for example

<i>csrUp</i>	
$\Delta Grid$	
$y \geq 1$	
$y' = y \Leftrightarrow 1$	
$x' = \min(x, \#lines[y'])$	
$lines' = lines$	

This paper was typeset in L^AT_EX using the MathSpad editing tool (<http://www.win.tue.nl/win/cs/wp/mathspad/>) with a special stencil for using oz.sty.