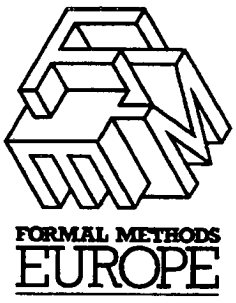
Tim Denvir



FACS Europe

The Newsletter of the BCS Formal Aspects of Computing Science Special Interest Group and Formal Methods Europe.

Series I Vol. 1, No. 1 Autumn 1993

Contents

Editorial	2
Invitation to FACS members	2
FME '93	3
A Special Welcome to FME Members	4
Raise Column	5-6
Understanding the differences between VDM and Z	7-30
A Response to Florence, Dougal and Zebedee	31-32
Current German Activities in Security and Correctness	33-38
Formal Specification of Neuorsis	39-40
Formal Methods Survey	41-43
Anyone for Fractal Programming	44-45
Wittgenstien — The Film	46-47
Notices and Calls for Papers	48-51
Forthcomming Events	52-54
Contributions and Guidelines	55
BCS FACS Committee 1992/93	56

Editorial

Welcome to FACS Europe, the first joint Newsletter of FACS and FME.

FACS - Formal Aspects of Computing - is a special interest group of the British Computer Society (BCS). John Cooke has written a short article about FACS elsewhere in this newsletter, describing the benefits of FACS membership. Members of Formal Methods Europe who are not members of FACS are particularly invited to read this, and FACS members too may find there are advantages to their membership which they had temporarily forgotten about!

FME - Formal Methods Europe - is an association based in the European Community and supported by the European Commission (DG XIII). The mission of FME is to stimulate the use of formal methods by European industry and to provide a meeting place for the European formal methods community, with a major emphasis on industrial providers and users, current or potential.

Producing newsletters for organisations like FME and FACS requires a lot of voluntary editorial effort. The objectives of the two newsletters are almost indistinguishable. Given the limited available resources, the committees of FACS and FME have decided to collaborate and produce a combined FACS-FME newsletter, starting with this current issue. Combining the two will avoid duplication of a scarce effort and, we hope, provide an opportunity for an improvement in quality and scope. FACS news will be available to a wider audience and we hope to receive contributions from a greater variety of sources.

Henceforth members of both FACS and FME will receive the joint newsletter.

Tim Denvir chairman FACS

Invitation to FACS Members

FME organises a conference every eighteen months. The next, FME'94, will be held in Barcelona from 24 to 28 October 1994. The previous and first FME Symposium was held in Odense in April 1993.

FME organises industrial seminars in various locations in the EC. Other activities are planned, including providing a panel of speakers on formal methods for introductory seminars and the setting up of a Formal Methods Tools Database.

Membership of FME is open to current and potential industrial users of formal methods, and any other interested persons. Membership is by application to the chairman, see backpage.

There is currently no charge for membership. Members will be requested to provide brief details of their involvement (if any) in formal methods and to assist FME in its mission.

Martyn Thomas chairman FME

Acknowledgements

This edition of the newsletter was produced by:

Jawed Siddiqi	Sheffield Hallam University
Chris Roast	Sheffield Hallam University

We would like to put on record our appreciation of Brian Monahan's support while convalescing, we wish him well.

Duplication and distribution by the Department of Computing and Management Sciences, Sheffield Hallam University.

FME'93

Odense, Denmark, April 1993

A Special Welcome to Members of FME

The UK has always been active in formal computing, and it was therefore quite natural that, in 1979, a special interest group in the Formal Aspects of Computing Science (FACS or, in full, BCS-FACS) be created under the umbrella of the British Computer Society.

Although FACS was originally set up with the primary function of supporting the British Formal methods community, we now have members in America, Asia and Australasia as well as in mainland Europe. Apart from organising workshops in the UK, we also co-sponsor meetings further afield; and some five years ago we launched the Formal Aspects of Computing journal, the international scope of which is truly reflected in the composition of its editorial board.

In the true spirit of collaboration we have for many years offered the BCS discount rate to members of other 'sister' national computing societies with which the BCS has reciprocal arrangements; all members of FACS are treated equally regardless of being members of BCS (etc.) or not, or whether they reside in the UK or not. They receive this newsletter, they get direct mailings about future meetings - which they can attend at discounted rates, they can subscribe to EATCS (the European Association for Theoretical Computer Science) at the same time as renewing their annual FACS subscription. They can also subscribe to "Formal Aspects of Computing" at over 70% discount, and obtain the proceedings of the FACS workshops at discounted prices (typically 20% below normal).

The rates for 1993 were:

FACS membership subscription (full price)	25
FACS membership subscription (for Members of BCS, ACM, GI, AFCET, ACIA, etc.)	10
FAC journal vol 4, 1992 (6 regular issues + one special)	33
FAC journal vol 5, 1993 (6 issues)	33
EATCS subscription	10

To be sent membership forms, to ascertain rates for 1994, or to obtain a sample copy of our journal, please contact me see backpage.

John Cooke

Sheffield Hallam University
School of Computing & Management Sciences
100 Napier Street
Sheffield, S11 8HD

November 1993

Dear Reader,

Welcome to the first edition of FACS Europe. I hope you find it an interesting read.

We are sorry it's late, this is due to the print machine breaking down. We would like you to make a seasonal mind shift, that is assume this is the winter edition and that the next edition will be in spring.

Wishing you a happy winter solstice, and hoping that reading it will inspire you to contribute to the next edition.

Jawed Siddiqi

RAISE Column

A Language-Independent Definition of Refinement

Maurice Naftalin, Lloyd's Register
tcsmpn@aie.lreg.co.uk

Some time back I wrote a paper([1]) which proposed a model of the refinement process as a series of operations building and modifying a structure which summarises an entire program development. This structure, which has a graphical interpretation, is intended to provide a means of visualising the complete refinement process. The starting-point of its definition was a part which I called a "refinement tree". This is a formalisation of the structures that arise during use of a refinement calculus. The idea is this: a specification (or program; for this discussion there is no need to distinguish) can be represented as a tree in the syntax of your favourite language[1]. Viewed abstractly, each node in this tree (called an *Abstract Syntax Tree* or *AST*) is either an atom or else a composite consisting of an operator and a tuple of operand subtrees, each of appropriate type. If you decide that some node in such a tree is in need of improvement, you make a *refinement*, which is a structural link between the part being improved (the *source*) and a better version (the *target*). The target is another abstract syntax tree, any node of which may again be refined.

This idea was expressible quite concisely in VDM-SL syntax:

Reftree = *AST* | *Refmt*

AST = *Atom* | *Oprnode*

Oprnode :: *opr* : token
subtrees : *Reftree**

Refmt :: *source* : *AST*
target : *Reftree*

and this specification served well when extended to describe how refinement trees were linked together in the refinement process as a whole, what operations were appropriate to the structure representing the process, and so on. There is an unsatisfactory aspect to it, however: the definition of *Oprnode* provides no way of referring to properties of the language (other than the constant and rather unhelpful observation that a syntactic construct consists of an operator and some operands). So it is impossible to specify any properties of refinement trees other than those which are completely independent of the language in use. For example, you cannot specify when a refinement tree is well-typed, what its meaning is, and so on. What I really wanted was to define a structure parameterised by language. In this situation an unstructured specification language could only provide a choice between omitting the language entirely (as I did) or choosing a language arbitrarily and cluttering the specification with its details.

The structuring facilities of the RAISE Specification Language (RSL) provide an escape from this dilemma. A simple-minded language definition in RSL is:

```
scheme
  LANG =
    class
      type
        Atom,
        Opr_token,
        Syntactic_Type,
        Atomtypes = Atom ~m~> Syntactic_Type,
        Oprtypes = Opr_token ~m~> Oprtype,
        Oprtype :: argts : Syntactic_Type* result : Syntactic_Type
```

[1] or languages: although this note describes refinement in a wide-spectrum language, that is not central to the argument.

```
    value
      at : Atomtypes,
      ot : Oprtypes
  end
```

This specification defines the types of *Atom* and *Opr_token* (corresponding to the types *Atom* and token in the VDM specification) as RSL sorts (*i.e.* abstract types). (The additional sort *Syntactic_Type* is added to allow type-checking of refinement trees). Subsequently, extensions of this specification can be defined to populate these types. At the same time, the values (constants) *at* and *ot* will be defined to associate each atom and operator with appropriate syntactic types.

Now a language-independent specification of refinement trees can be given. In fact, it looks very similar to the original VDM one: the only difference is that the types of *Atom* and *Opr_token* are parameters from the language definition.

```
scheme
  REFTREE(lng : LANG) =
    class
      type
        Reftree == ASTtoReftree(AST) | RefmttoReftree(Refmt),
        AST ==
          AtomtoAST(lng.Atom) | mkOprnode(opr : lng.Opr_token, subtrees : Reftree*),
        Refmt :: source : AST  target : Reftree
    end
```

Something new has in fact been achieved, however: it is now possible to define properties of the structure which are partly dependent on the language. For example, the specification of a typechecker for refinement trees is:

```
scheme
  REFTREE_TYPES(lng : LANG) =
    extend REFTREE(lng) with
      class
        value
          welltyped : Reftree × lng.Syntactic_Type → Bool
          welltyped(r, s) ≡
            case r of
              ASTtoReftree(ast) →
                case ast of
                  AtomtoAST(a) → lng.at(a) ≡ s,
                  mkOprnode(o, st) →
                    (∀ i : Nat • i ∈ inds (st) ⇒ welltyped(st(i), lng.argts(lng.ot(o))(i))) ∧
                    (lng.result(lng.ot(o)) ≡ s)
                end,
              RefmttoReftree(rfmt) → welltyped(target(rfmt), s)
            end
      end
```

Of course, this is still highly schematic and oversimplified. These three specifications would all require considerable enhancement before they could serve as adequate abstractions from real languages or refinement structures. But the example as it stands serves to show how two overlapping concerns, the properties of a refinement structure and the properties of the language composing it, can be cleanly separated — given the right structuring facilities.

Reference

Naftalin M. P. *A Model of the Refinement Process*, in Jones C. B., Shaw R. C., and Denvir T. (eds), Proceedings of the Fifth BCS-FACS Refinement Workshop, 211–229, Springer-Verlag, 1993.

Understanding the differences between VDM and Z

I. J. Hayes*

Department of Computer Science
University of Queensland
e-mail: Ian.Hayes@uqcspe.cs.uq.oz.au

C. B. Jones

Department of Computer Science
University of Manchester
e-mail: cbj@cs.man.ac.uk

J. E. Nicholls

Programming Research Group
Oxford University

August 20, 1993

Abstract

This paper attempts to provide an understanding of the interesting differences between two well-known specification languages.

The main ideas are presented in the form of a discussion. This was partly prompted by Lakatos' book 'Proof and Refutations' but, since this paper is less profound, characters from the childrens' television series 'The Magic Roundabout' are the speakers: Zebedee speaks for Z, Dougal puts the VDM position, and Florence acts as the user.

The specifications which are presented have been made similar so as to afford comparison – in neither the VDM nor the Z case would they be considered to be ideal presentations. Some technical details are relegated to footnotes.

Discussion

Florence: I know that some people are confused by the existence of two specification languages, Z and VDM, which have a lot in common.

Dougal: Yes, there is certainly a common objective in Z and VDM and in many places one can see that the same solution has been adopted.

The use of both VDM and Z has been concentrated on the specification of *abstract machines*, and they both take the same so called 'model-oriented' approach. Jointly they differ from the so called 'algebraic' specification languages (these might be better called 'property-oriented' to contrast with 'model-oriented') which concentrate on specifying abstract data types.

Zebedee: The primary difference between these approaches is that VDM and Z both give an explicit model of the state of an abstract machine – the operations of the abstract machine are

defined in terms of this state – whereas 'algebraic' approaches give no explicit model of the type – an abstract data type is specified by axioms giving relationships between its operations. For (a much-overused) example, a stack in VDM and Z would typically be modelled as a sequence, while in the 'algebraic' approaches axioms such as

$$pop(push(x, s)) = s$$

would be given.

Dougal: Before going further, let's be clear what we mean by 'VDM'. Strictly, VDM was always seen as a development method in the sense that rules are given to verify steps of development (data reification and operation decomposition). I guess that our discussion today will be confined to the specification language used in VDM.

Florence: Does that have a name?

Dougal: I wish I could say 'no'! But I have to confess that it is sometimes known as 'Meta-IV' – the draft VDM standard talks about 'VDM-SL',[1] but let's talk about the significant differences between the two specification languages.

Florence: Probably what hit me first about the difference between VDM and Z specifications is the appearance of the page. VDM specifications are full of keywords and Z specifications are all 'boxes'. Is this a significant difference?

Dougal: VDM uses keywords in order to distinguish the roles of different parts of a specification. For example, a module corresponds to an abstract machine with state and operations; a state has components and an invariant; and operations have separate pre- and post-conditions. These different components are distinguished as they have different purposes in the specification. To do this VDM makes use of keywords to introduce each component. All this structure is part of the VDM language.

Zebedee: In Z almost none of the structure Dougal mentioned is explicit in the language. A typical specification consists of lots of definitions, many of which are *schemas* – the boxes Florence was referring to. Schemas are used to describe not only states but also operations. The status of any particular schema is really only determined by the text introducing it, although it isn't hard to guess the purpose of a schema by looking at its definition. Schemas are also used as building blocks to construct descriptions of machine states or facets of operations. Such component building blocks may not correspond to any of the VDM categories.[2]

Florence: But don't you need the extra structure that VDM gives if you want to do formal refinements?

Zebedee: Yes, but to perform refinements you also need to consider a target programming language. For example, if you wanted to produce Modula code then you could give a definition module for an abstract machine in which the state and the operations are defined by Z schemas. Such a definition module would be very close to a VDM module in terms of structure.

Florence: How about an example – something other than stacks please!

Zebedee: A simple relational database, known as NDB, has been presented in both notations. Let's use that.[3] After you Dougal.

[1] The notation used here is close to that of the 'Committee Draft' of the VDM-SL Standard but there may be minor syntactic differences.

[2] It is perhaps worth explaining that Z has a number of levels: a basic mathematical notation (similar to that of VDM); the schema notation; and conventions for describing the state and operations of an abstract machine using Z schemas. Little new notation is introduced in the third level, only conventions for making use of the other notation to describe abstract machines. It is worth noting that Z notation – the first two levels – has been used with a different set of conventions for other purposes, such as specifying real-time systems.

[3] Originally formally specified in [Wel82] and revised in [Wal90], this was used as a challenge problem in [FJ90]

Dougal: Someone writing a VDM specification of a system normally sketches a state before going into the details of the operations which use and modify that state.[4]

For the NDB description, the overall state will have information about entities and relations. In order to build up such a state, a number of sets are taken as basic.

Eid Each entity in the database has a unique identifier taken from the set *Eid*.

Value Entities have values taken from the set *Value*.

Esetnm An entity can belong to one or more entity sets (or types). The names of these sets are taken from the set *Esetnm*.

Rnm The database consists of a number of relations with names taken from the set *Rnm*.

All but the first of these can conveniently be thought of as parameters to the specification of the NDB database system; *Eid* is an internal set about which we need to know little – we just regard it as a set of 'tokens'.

Now we can begin to think about the types which are constructed from these basic sets. NDB is a binary relational database consisting of relations containing tuples which each have two elements: a *from* value and a *to* value. A tuple contains a pair of *Eid*s; in VDM the type *Tuple* is defined as follows

$$\begin{array}{l} \mathit{Tuple} :: \mathit{fv} : \mathit{Eid} \\ \qquad\quad \mathit{tv} : \mathit{Eid} \end{array}$$

Then a relation can be defined as a set of such pairs:

$$\mathit{Relation} = \mathit{Tuple}\text{-set}$$

To define whether a relation is to be constrained to be one-to-one – or whatever – four distinct constants are used

$$\mathit{Maptp} = \textsc{OneOne} \mid \textsc{OneMany} \mid \textsc{ManyOne} \mid \textsc{ManyMany}$$

Relation information (*Rinf*) contains information stored for a relation: apart from the *Tuple* set, an element of *Maptp* provides the constraint on the form of relation allowed. The consistency of the *tp* and *r* fields of *Rinf* is expressed as an invariant.

$$\begin{array}{l} \mathit{Rinf} :: \mathit{tp} : \mathit{Maptp} \\ \qquad\quad \mathit{r} : \mathit{Relation} \\ \mathsf{inv}\ (\mathit{mk\text{-}Rinf}(\mathit{tp}, r)) \triangleq \\ \quad (\mathit{tp} = \textsc{OneMany} \Rightarrow \forall t_1, t_2 \in r \cdot t_1.\mathit{tv} = t_2.\mathit{tv} \Rightarrow t_1.\mathit{fv} = t_2.\mathit{fv}) \wedge \\ \quad (\mathit{tp} = \textsc{ManyOne} \Rightarrow \forall t_1, t_2 \in r \cdot t_1.\mathit{fv} = t_2.\mathit{fv} \Rightarrow t_1.\mathit{tv} = t_2.\mathit{tv}) \wedge \\ \quad (\mathit{tp} = \textsc{OneOne} \Rightarrow \forall t_1, t_2 \in r \cdot t_1.\mathit{fv} = t_2.\mathit{fv} \Leftrightarrow t_1.\mathit{tv} = t_2.\mathit{tv}) \end{array}$$

It's worth noting that the set of values defined by a definition with an invariant only contains values which satisfy the invariant. So we can only say that $\mathit{rel} \in \mathit{Rinf}$ if the relation, $\mathit{rel}.r$ is consistent with the map type $\mathit{rel}.\mathit{tp}$. Because invariants can be arbitrary predicates, type membership is only partially decidable.

Florence: Is Z's notion of type the same as what Dougal just described?

Zebedee: No, but this is a difference in the use of the word 'type' rather than a real difference between Z and VDM. In Z the term 'type' is used to refer to what can be statically type checked. This is more liberal than what Z calls a 'declared set' which is what VDM calls a 'type'.

Dougal: Well, let me get through the rest of the state; then we can make more comparisons.

---

to which a Z response is given in [Hay92].

[4]The description of the NDB state presented here is given *postfacto* rather than attempting to emulate the process by which specifications are produced.

In NDB, relations are identified not only by their names but also by the entity sets of the values they relate (so a database can contain two relations called OWNS: one between PEOPLE and CARS and – at the same time – another between PEOPLE and HOUSES). So a key for a relation contains three things

$$\begin{aligned} Rkey :: \; & nm : Rnm \\ & fs : Esetnm \\ & ts : Esetnm \end{aligned}$$

The overall state which we are aiming for has three components: an entity set map (esm) which defines which entities are in each valid entity set; a map (em) which contains the value of each identified entity; and a third map (rm) which stores the relevant $Rinf$ for each $Rkey$. The invariant records the consistency conditions between the components.

$$\begin{aligned} Ndb :: \; & esm : Esetnm \xrightarrow{m} Eid\text{-set} \\ & em : Eid \xrightarrow{m} Value \\ & rm : Rkey \xrightarrow{m} Rinf \end{aligned}$$

$$\begin{aligned} & \mathsf{inv}\ (mk\text{-}Ndb(esm, em, rm)) \triangleq \\ & \quad \mathsf{dom}\ em = \bigcup \mathsf{rng}\ esm \wedge \\ & \quad \forall rk \in \mathsf{dom}\ rm \cdot \\ & \qquad \{rk.fs, rk.ts\} \subseteq \mathsf{dom}\ esm \wedge \\ & \qquad \forall mk\text{-}Tuple(fv, tv) \in rm(rk).r \cdot fv \in esm(fs) \wedge tv \in esm(ts) \end{aligned}$$

Later, we'll look at how this (together with the initial state and the operations) gets grouped into a module.

Florence: How would the above state be presented in Z?

Zebedee: All the details would be almost identical, but the specification would be structured differently. The specification would consist of a sequence of sections and each section would present a small set of the state components along with operations on just those components.

The Z approach to structuring specifications is to try to build the specification from near orthogonal components. We look for ways of splitting the state of the system so that we can specify operations on just that part of the state that they require.

For NDB we have chosen to split the specification into three parts:

1. entities and their types or entity sets,
2. a single relation, and
3. multiple relations.

Finally, we put these specifications together to give the final specification.

Rather than follow the normal Z approach here, I'll give all of the state components from the different sections together, so that we can compare the state with that used for the VDM specification. As for the VDM, our basic sets are the following:

$$[Eid, Esetnm, Value]$$

As with the VDM the sets $Esetnm$ and $Value$ can be thought of as parameters, and the set Eid is used locally within the specification.

For a database we keep track of the entities that are in an entity set (of that type). Every entity must be of one or more known types. The following schema, *Entities*, groups the components of the state together with a invariant linking them.

Entities

$esm : Esetnm \twoheadrightarrow (\mathbf{F}\ Eid)$
$em : Eid \twoheadrightarrow Value$

$\mathrm{dom}\ em = \bigcup \mathrm{ran}\ esm$

Florence: How does this differ from the VDM so far?

Zebedee: It's virtually identical – bar the concrete syntax. The notation $\mathbf{F}\ Esetnm$ is equivalent to the VDM notation Esetnm-set.

Another approach to defining the state in Z would be to define *esm* as a binary relation between *Esetnm* and *Eid*. This leads to simpler predicates in the specifications because the Z operators on binary relations are closer to the operations required for NDB's binary relations. *Entities* is defined using Z's binary relations in [Hay92], but for our comparison here it is simpler to use the same state as the VDM version. That way we can concentrate on more fundamental differences.

Dougal: Yes, these modelling differences are interesting but not the point of today's discussion; but it is worth saying that binary relation notation could also be added to VDM and the same alternative state used.

Florence: So far that only covers the components *esm* and *em* of the VDM *Ndb* state.

Zebedee: Right, and at this point we would give a set of operations on the above state, but in order to provide a more straightforward comparison with the VDM state, we shall skip to the remainder of the state. The relations used in NDB are binary relations between entity identifiers (rather than entity values). A *Tuple* is just a pair of entity identifiers and a *Relation* is modeled as a Z binary relation between entity identifiers.

$$Tuple == Eid \times Eid$$
$$Relation == Eid \leftrightarrow Eid$$

Florence: Is that really different from VDM?

Zebedee: Well, yes and no. Both the VDM and Z versions use a set of pairs for a relation – note that $Eid \leftrightarrow Eid$ is a shorthand for $\mathbf{P}(Eid \times Eid)$. The difference is that, in Z, relations are predefined and have a rich set of operators defined on them.

Dougal: There are a couple of points I'd like to pick up from what Zebedee has said. When we make a selection of basic building blocks for specifications, we are clearly influenced by experience. There is nothing deep in, say, the omission of relations from VDM (or – say of optional objects from Z). Once again, the remarkable thing is just how similar the selection in Z and VDM is. As I said above, if I were writing a large specification which needed relations, I would just extend VDM appropriately.

Florence: What about the *Maptp* in Z?

Zebedee: *Maptp* is virtually identical:

$$Maptp ::= OneOne \mid OneMany \mid ManyOne \mid ManyMany$$

When a relation is created its type is specified as being one of the following four possibilities: it is a one-to-one relation (i.e., an injective partial function), a one-to-many relation (i.e., its inverse is a partial function), a many-to-one relation (i.e., a partial function), or a many-to-many relation. In Z, the set of binary relations between X and Y is written $X \leftrightarrow Y$, the set of partial functions is written $X \nrightarrow Y$, the set of one-to-one partial functions is written $X \rightarrowtail Y$, and the inverse of a relation r is written $r^{\sim}$.

A relation is created to be of a particular type and no operation on the relation may violate the type constraint.

Rinf

$$tp : Maptp$$
$$r : Relation$$

$$(tp = OneOne \Rightarrow r \in Eid \rightarrowtail Eid) \land$$
$$(tp = ManyOne \Rightarrow r \in Eid \nrightarrow Eid) \land$$
$$(tp = OneMany \Rightarrow r^{\sim} \in Eid \nrightarrow Eid)$$

Dougal: If you expanded the definitions you would get the same constraint as in the VDM version.

Zebedee: Yes, they are exactly the same.

Florence: We still don't have NDB's named relations in Z.

Zebedee: That's next, but again at this point in the normal flow of a Z specification operations would be defined on the state *Rinf*. The database consists of a number of relations with names taken from the set *Rnm* (essentially a parameter set).

$$[Rnm]$$

A relation is identified by its name and the 'from' and 'to' entity sets that it relates. This allows a number of relations to have the same *Rnm* provided they have different combinations of 'from' and 'to' entity sets.

Rkey

$$nm : Rnm$$
$$fs, ts : Esetnm$$

The entities related by each relation must belong to the entity sets specified by the relation key.

Ndb

$$Entities$$
$$rm : Rkey \nrightarrow Rinf$$

$$\forall rk : \mathrm{dom}\, rm \bullet$$
$$\quad \{rk.fs, rk.ts\} \subseteq \mathrm{dom}\, esm \land$$
$$\quad (\forall t : (rm\ rk).r \bullet$$
$$\quad\quad first\ t \in esm(rk.fs) \land second\ t \in esm(rk.ts))$$

Florence: Because you have included *Entities* you now have all the state components, so that this is equivalent to the VDM *Ndb* state.

Zebedee: Yes.

Florence: Why don't we look at initialisation?

Dougal: The initial state (in this case it is unique) is defined in VDM as

$$\mathrm{init}(ndb) \triangleq ndb = mk\text{-}Ndb(\{\}, \{\}, \{\})$$

Zebedee: In Z the initial state is defined by the following schema (given using the horizontal form of presentation):

$$Ndb\_Init \mathrel{\widehat=} [Ndb \mid esm = \{\} \land em = \{\} \land rm = \{\}]$$

Again, if we followed the more structured presentation of NDB, we would probably define the set of allowable initial *Entities* and then define the allowable initial Ndb states in terms of it.

Florence: Why don't we look at operations?

Dougal: The simplest operation adds a new entity set name to the set of known entity sets, *esm*. The set of entities associated with this new name is initially empty. In VDM this can be defined.

ADDES (*es*: *Esetnm*)
ext wr $esm : Esetnm \xrightarrow{m} Eid\text{-set}$
pre $es \notin \text{dom } esm$
post $esm = \overleftarrow{esm} \cup \{es \mapsto \{\}\}$

Zebedee: In Z that's

ADDES0

$\Delta Entities$
$es?: Esetnm$

$es? \notin \text{dom } esm \land$
$esm' = esm \cup \{es \mapsto \{\}\} \land$
$em' = em$

where $\Delta Entities$ introduces the *before* and *after* (primed) states:

$$\Delta Entities \mathrel{\widehat=} Entities \land Entities'$$

Florence: One obvious syntactic difference is that VDM uses hooked variables for the *before* state and unhooked variables for the *after* state, whilst Z uses undecorated variables for the *before* state and primed variables for the *after* state. In addition, Z uses variable names ending in '?' for inputs (and names ending in '!' for outputs). Apart from the differences in syntax, I notice that VDM uses an externals clause and distinguishes the pre-condition.

Zebedee: Yes, Z does not have any equivalent of an externals clause. The predicate must define the final values of all variables, even if the variable is unchanged, such as *em*. This is why in Z one divides up the state into small groups of components and defines sub-operations on each group, before combining the sub-operations in order to define the full operation. For a large specification with many state components, if one had to define the operations on the whole state, then there would be many boring predicates stating that many of the variables are unaffected. Dividing up the state avoids this problem, although it is still necessary to promote the operations on the substates to the full state at some stage.

Dougal: In VDM, an operation is always written in a module. This provides the appropriate state but one can use the external clause of an operation specification to make it self-contained and to restrict the frame.

Zebedee: In Z, the state is explicitly included – via a 'Δ' or 'Ξ' schema usually – within the operation, so the operation schema can stand on its own.

With regard to the pre-condition, although the same logical expression appears in the Z schema *ADDES0* as in the VDM operation *ADDES*, it is not separated out. For this operation that doesn't make a large difference between the Z and VDM versions, but for other operations it can.

Dougal: OK, let's look at deleting an entity set in VDM.

DELES (*es*: *Esetnm*)
ext wr *esm* : *Esetnm* $\xrightarrow{m}$ *Eid*-set
rd *rm* : *Rkey* $\xrightarrow{m}$ *Rinf*
pre $es \in \mathrm{dom}\, esm \wedge esm(es) = \{\} \wedge$
$\forall rk \in \mathrm{dom}\, rm \cdot es \neq rk.fs \wedge es \neq rk.ts$
post $esm = \{es\} \lhd \overleftarrow{esm}$

Zebedee: In Z this would be written

DELES0

$\Delta Entities$
$es?: Esetnm$

$es? \in \mathrm{dom}\, esm \wedge esm(es) = \{\} \wedge$
$esm' = \{es?\} \lhd esm \wedge$
$em' = em$

Florence: But isn't it missing part of the pre-condition in the VDM version?

Zebedee: Yes, *DELES0* is only defined on the state *Entities*, so it is impossible to talk about the state component *rm*. To define the equivalent of the VDM operation, we need to promote *DELES0* to the full *Ndb* state. We do this by defining the schema ΞRM which introduces the full *Ndb* state and constrains *rm* to be unchanged. This is then conjoined with *DELES0*.

$\Xi RM \mathrel{\widehat{=}} [\Delta Ndb \mid rm' = rm]$
$DELES \mathrel{\widehat{=}} DELES0 \wedge \Xi RM$

Florence: But I still can't see the missing bit of the pre-condition!

Zebedee: That's because it's not visible! But the Z *DELES* has the same pre-condition as the VDM operation.

Florence: How do I get to see it?

Zebedee: In Z, the pre-condition of an operation characterises exactly those inputs and initial states such that there exists a least one possible combination of outputs and final state that satisfies the operation specification. For *DELES* the pre-condition is

pre *DELES*

Ndb
$es?: Esetnm$

$\exists Ndb' \bullet$
$es? \in \mathrm{dom}\, esm \wedge esm(es) = \{\} \wedge$
$esm' = \{es?\} \lhd esm \wedge$
$rm' = rm$

The predicate can be expanded to

pre *DELES*

$$
\begin{array}{l}
Ndb \\
es?: Esetnm \\
\hline
\exists\, Entities';\ rm': Rkey \nrightarrow\!\!\!\!\!+\!\!\!\!\!\rightarrow\ Rinf \bullet \\
\quad (\forall\, rk: \mathrm{dom}\, rm' \bullet \\
\qquad \{rk.fs, rk.ts\} \subseteq \mathrm{dom}\, esm' \wedge \\
\qquad (\forall\, t: (rm'\ rk).r \bullet \\
\qquad\quad first\ t \in esm'(rk.fs) \wedge second\ t \in esm'(rk.ts))) \wedge \\
\quad es? \in \mathrm{dom}\, esm \wedge esm(es) = \{\} \wedge \\
\quad esm' = \{es?\} \lhd esm \wedge \\
\quad rm' = rm
\end{array}
$$

which can be simplified to

pre *DELES*

$$
\begin{array}{l}
Ndb \\
es?: Esetnm \\
\hline
es? \in \mathrm{dom}\, esm \wedge esm(es) = \{\} \wedge \\
(\forall\, rk: \mathrm{dom}\, rm \bullet es? \neq rk.fs \wedge es? \neq rk.ts)
\end{array}
$$

Dougal: That's now the same as the VDM pre-condition.

Florence: Wasn't all that a bit complicated compared to the VDM version?

Zebedee: Well, yes and no. If you want to compare it with the VDM version, then we have also done the equivalent of discharging its *satisfiability* proof obligation. In VDM this proof obligation is the main consistency check available for the specifier, and the Z pre-condition calculation can be likewise seen as a consistency check – that the calculated pre-condition agrees with the specifier's expectations.

Dougal: I believe that this *is* a significant difference between Z and VDM. There is a technical point: when development steps are undertaken, the pre-condition is required.[5] But there is also a pragmatic point: in reading many industrial (informal) specifications, I have observed that people are actually not so bad at describing what function is to be performed; what they so often forget is to record the assumptions. I therefore think that it is wise to prompt a specifier to think about the pre-condition.

Zebedee: I'd agree with that, but in both VDM and Z, provided the respective consistency checks are done, we do end up at the same point. It's only the path that is different.

With the Z approach of constructing the specification of an operation it is only when you have the final operation that the concept of a pre-condition really makes sense.[6]

Florence: What does a pre-condition mean? I'm really not clear about whose responsibility it is to avoid calls that violate the pre-condition – or are these exceptions?

Dougal: A pre-condition is essentially a warning to the user: the behaviour defined in the post-condition is only guaranteed if the starting state satisfies the pre-condition. In formal development, the user should treat the pre-condition as a proof obligation that an operation is only invoked when its pre-condition is true. It is perhaps useful to think of the pre-condition as

[5] Tony Hoare in [Hoa91] appears to argue for the use of Z to develop specifications and the use of VDM for development of the design and implementation.

[6] Although it is possible to define operators similar to schema conjunction and disjunction on pre/postcondition pairs; see [War93].

something that the *developer* of an operation can rely upon; it is permission for the developer to ignore certain situations. Exceptions are quite different – VDM does have notation (not used so far in this example) to mark error conditions which must be detected and handled by the developer. In VDM an operation specification can consist of a normal pre/postcondition pair followed by a set of exception pre/postcondition pairs.

Florence: Can we compare the treatment of exceptions? I've noticed Z doesn't have exceptions as part of the language.

Zebedee: I guess one way of viewing the Z approach is to say that it doesn't really have exceptions at all. As part of specifying an operation one specifies its behaviour both in the normal case and in the exceptional cases. Both the normal case and each exceptional case are specified in the same manner in a Z schema. Each of these schemas corresponds to a pre/post-condition pair in the VDM version.

Florence: So the difference about the use of pre/postcondition pairs in the VDM version versus a single schema in the Z version crops up here as well.

Zebedee: That's correct.

Florence: How else do they differ?

Zebedee: In terms of what they both mean, not at all. The complete operation is specified in Z by taking the disjunction of the schemas for the normal and exceptional cases. It has the same meaning as the corresponding VDM specification. For example, in both Z and VDM the preconditions of the alternatives may overlap, either between the normal case and an error case or between error alternatives, and in both Z and VDM there is a nondeterministic choice between alternatives that overlap.

Dougal: Yes, that's correct. Although VDM and Z appear to describe exceptions differently, the semantic ideas underneath the concrete syntax are virtually identical.

Florence: So why do they look so different?

Zebedee: Well mostly it is just differences in syntax but I guess there are a couple of points about building an operation specification from Z schemas using schema operators that are worth noting. Firstly, it is possible to specify more than one normal case and these further alternatives just become part of the disjunction of cases. (There is no real distinction between normal and error cases, so one can have as many of either as suits the problem in hand.) Secondly, the same *exception* schema can be used for more than one operation. This has the twin advantages of avoiding repetition and maintaining consistency between different operations in their treatment of the same exception.

Dougal: I see those advantages; it is just in the spirit of VDM to have a place for exceptions marked by keywords. As always, syntactic issues tend to be more an issue of taste than of hard scientific arguments.

Florence: How about an example?

Zebedee: Let's consider the possibility of trying to add a new entity set when the name is already in use. In Z the exception alternative is specified by

$$
\begin{array}{|l}
\hline
ESInUse \\
\hline
\Xi Entities \\
es?: Esetnm \\
\hline
es? \in \mathrm{dom}\ esm \\
\hline
\end{array}
$$

where $\Xi Entities$ introduces the *before* and *after* states and constrains them to be equal:

$$\Xi Entities \mathrel{\widehat=} [\Delta Entities \mid \theta Entities' = \theta Entities]$$

The operation augmented with the error alternative is

$$ADDESX \mathrel{\widehat=} ADDES0 \vee ESInUse$$

Dougal: In VDM *ADDES* with an exception if the entity set is already in use would be specified by adding the line

err ESINUSE $es \in$ dom esm

Zebedee: In both the VDM and Z we should really add an error report to be returned by the operation. This would be done in essentially the same way in both VDM and Z.

Dougal: In VDM, the whole specification gets put into a *module*: this is a structuring mechanism that makes it possible to build one module on top of others. One possible module syntax is illustrated in Appendix A. But I have to confess that this is still a subject of debate. In fact [FJ90] was written precisely because this debate is not yet settled.

Zebedee: Z also needs a modularisation mechanism and one proposal is developed in [HW93].

Florence: Does the issue of pre-conditions have any connection with the fact that I suspect the two notations handle partial functions differently?

Dougal: Yes, there is a loose connection and there are differences between Z and VDM here. In fact, I'll be interested to hear what Zebedee has to say on this point.

Let me set the scene.[7] Both operators like hd and recursively defined functions can be partial in that a simple type restriction does not indicate whether they will evaluate to a defined result for all arguments of the argument type: hd [] or factorial applied to minus one are examples of non-denoting terms. VDM uses non-standard logical operators which cope with possibly undefined operands: so true $\vee$ *undefined* is true as is *undefined* $\vee$ true.

Zebedee: In [Spi88] Mike Spivey uses existential equality which always delivers a truth value – where either of the operands are undefined, the whole expression is false. This enables him to stay with classical (two-valued) logic.

Dougal: Yes, I should have said that there are a couple of different approaches where one tries to trap the undefined before it becomes an operand of a logical operator. Thus, in hd [] $=_3$ 5 it is possible to define the result as false. Unfortunately, the task does not end here – any relational operators need similar special versions. Moreover, the user has to keep the distinction between the logical equality and the computational equality operator of the programming language in mind. As they say 'There ain't no such thing as a free lunch'.

Zebedee: Yes, but I did say Spivey took that approach; in the beginning, I believe that Jean-Raymond Abrial wanted to formalize the view that while 'the law of the excluded middle' held, for undefined formulae, one never knew which disjunct was true. (The work on the new Z standard is still evolving.)

Florence: This is all a bit technical – does it matter?

Zebedee: Not much – but it is an interesting difference!

Florence: What about recursively defined structures such as trees?

Zebedee: Before we start into the details of the definition of recursive structures, one approach often taken in Z specifications is to avoid recursive structures and use a flattened representation

[7] A fuller discussion can be found in [BCJ84, CJ91, JM93].

instead. For example, the specification of the Unix filing system [MS84] represents the hierarchical directory structure by a mapping from full path names to the file's contents. All prefixes of any path name in the domain of the map must also be in the domain of the map.

A representation of the filing system state[8] as a recursive structure can be defined by considering a directory to be a mapping from a single segment of a path name to a file, where a file is either a sequence of bytes or is itself a directory.

It is interesting to compare specifications of filing system operations on both representations. On the flat representation finding a file is simply application of the map representing the filing system state to the file name, whereas a recursive function needs to be provided for the recursive representation. If one considers updating operations, the flat representation provides even simpler descriptions of operations.

Dougal: The recursive approach description is used in [Jon90] and I'm not sure that I'd concede the advantages that Zebedee claims for the flat specification. But that's a modelling issue not a difference between the two specification languages.

Florence: So when do we need to use recursive structures?

Dougal: A good example is the specification of the abstract syntax of a programming language.

Florence: Are there examples outside the rather special field of programming languages?

Zebedee: Yes, consider a simple binary tree. This can be specified in Z via the following

$$T ::= nil \mid binnode\langle\langle T \times \mathbb{N} \times T \rangle\rangle$$

This introduces a new type T, a constant of that type *nil* and an (injective) constructor function *binnode*, that when given a (left sub-) tree, a natural number and a (right sub-) tree returns a tree.

Dougal: In VDM that would be

$$\begin{array}{lll} binnode :: & l & : T \\ & v & : \mathbb{N} \\ & r & : T \end{array}$$

$$T = [binnode]$$

Zebedee: There are some technical differences in the approach taken here. In Z, T is a new type, whereas for the VDM *binnode* is a new type but T is not a new type. Also, in neither Spivey [Spi92] nor the Z Base Standard are mutually recursive structures (as used in the VDM version above) allowed.

Dougal: Wouldn't that cause problems for specifying the abstract syntax of a programming language? For an Algol-like language it is common to distinguish the syntactic categories of commands and blocks, but a command may be a block and a block may contain commands.

Zebedee: If you wanted to follow the draft Z standard then you would have to merge commands and blocks into a single syntactic category and then add consistency constraints to the language to ensure that they are correctly used.

Dougal: So, it is possible to specify it, but it isn't the most natural specification.

Zebedee: True. I have to admit that I would be very tempted to extend Z to allow mutually recursive definitions.[9]

Florence: I suppose this is all linked to the semantics of the specification languages themselves.

[8] We avoid consideration of inodes and links here.

[9] The problem here is the scope of the definitions *vis a vis* constraints on the set defined by the recursive structure.

Zebedee: To the average user, the semantics of the meta-language might not be bedtime reading. The aim has always been to base the semantics of Z on set theory; Mike Spivey gives a semantics in [Spi88] but a new semantics is being developed for the language standard.

Dougal: VDM has its origins in language description and it has to cope with reflexive domains etc. The semantics in the 'Committee Draft' of the VDM-SL standard is certainly not 'bedtime reading' but for simple operations a relatively simple set theoretic semantics would suffice.

Florence: Could you each tell me a bit about the history of your chosen specification languages?

Dougal: VDM was developed at the IBM Laboratory in Vienna. The Laboratory came into existence in 1961 when Professor Heinz Zemanek of the Technical University in Vienna decided to move his whole group to an industrial home. They had previously developed a computer called Mailüfterl at the Technical University. From 1958 the group had been increasingly involved in software projects including the construction of one of the early compilers for the ALGOL 60 programming language. As time went on they found it difficult to get adequate support for their projects and eventually joined IBM. Still in the first half of the 1960s, IBM decided to develop a new programming language for which the ambition was to replace both FORTRAN and COBOL. The language, which was at first called New Programming Language (until the National Physical Laboratories in the UK objected to the acronym – the language became known as PL/I), was clearly going to be large and it was decided that it would be useful to try to apply some formal techniques to its description.

Based on their own work – and influenced by research work by Cal Elgot, Peter Landin and John McCarthy – the Vienna group developed an operational semantics definition of PL/I which they called ULD-3 (Universal Language Description; ULD-2 was the name which had been applied to the IBM Hursley contribution to this effort; the language itself was being developed mainly from Hursley along with the early compilers. ULD-1 was a term applied to the natural language description of the language.)[10] The description of PL/I in ULD-3 style ran through three versions. These are very large documents. Operational semantics is now seen as unnecessarily complicated when compared to denotational semantics. However, to make the principles of denotational semantics applicable to a language like PL/I with arbitrary transfer of control, procedures as arguments, complicated tasking, etc. required major theoretical breakthroughs and a considerable mathematical apparatus not available at the time. The effort of the formal definition uncovered many language problems early and had a substantial influence on the shape of the language.

Towards the end of the 1960s serious attempts were made to use the ULD-3 description as the basis of compiler designs. Many problems were uncovered. The over-detailed mechanistic features of an operational semantics definition considerably complicated the task of proving that compiling algorithms were correct. But again one should be clear that an achievement was made; a series of papers was published which did describe how various programming language concepts could be mapped into implementation strategies which could be proved correct from the description. A series of proposals were made which could simplify the task of developing compilers from a semantic description. One of these was an early form of an exit construct which actually led to an interesting difference between the Vienna denotational semantics and that used in Oxford. Another important idea which arose at this time was Peter Lucas' *twin machine* proof and subsequently the observation that the ghost variable type treatment in the twin machine could be replaced by retrieve functions as a simpler way of proving that this sort of data development was correct. It is worth noting that Lucas' *twin machine* idea has been re-invented several times since: the generalization of retrieve functions to relations can be seen as equivalent to twin machines with invariants.

[10] VDL stands for Vienna Description Language and was a term coined by JAN Lee for the notation used in ULD-3.

The person who initiated the move that pushed the Vienna group in the direction of denotational semantics was Hans Bekič's; he spent some time in England with Peter Landin at Queen Mary College.

During the period from 1971 to 1973, the Vienna group was diverted into other activities not really related to formal description. Cliff Jones at this time went back to the Hursley Laboratory and worked on a *functional* language description and other aspects of what has become known as VDM. In particular he published a development of Earley's recogniser which is one of the first reports to use an idea of data refinement. In late 1972 and throughout '73 and '74 the Vienna group (Cliff Jones returned and Dines Bjørner was recruited) had the opportunity to work on a PL/I compiler for what was then a very novel machine. They of course decided to base their development for the compiler on a formal description of the programming language. PL/I at that time was undergoing ECMA/ANSI standardisation. The Vienna group chose to write a denotational semantics for PL/I. This is the origin of the VDM work. VDM stands for Vienna Development Method. When they decided not to go ahead with the machine, IBM decided to divert the Vienna Laboratory to other activities in 1976. This led to a diaspora of the serious scientists. Dines Bjørner went to Copenhagen University then on to the Technical University of Denmark. Peter Lucas left to join IBM Research in the States. Wolfgang Henhapl left to take up a chair in Germany and Cliff Jones left to move to IBM's European System Research Institute (ESRI) in Brussels.

Cliff Jones and Dines Bjørner took upon themselves the task of making sure that something other than technical reports existed to describe the work that had gone on on the language aspects of VDM. LNCS 61 ([BJ78]) is a description of that work. At ESRI, Cliff Jones also developed the work on those aspects of VDM not specifically related to compiler development and the first book on what is now generally thought of as VDM is [Jon80]. Both of these books have now been supplanted. The language description work is best accessed in [BJ82] and the non-language work is best seen in – second edition – [Jon90].

Dines Bjørner's group at the Technical University of Denmark strenuously pursued the use of VDM for language description and he and his colleagues were responsible for descriptions of the CHILL programming language and a major effort to document the semantics of the Ada programming language. Cliff Jones spent 1979 to 81 at Oxford University (collecting a somewhat belated doctorate). This was an interesting period because Cliff and Jean-Raymond Abrial arrived within a few days of each other in Oxford and had some interesting interchanges about the evolving description technique which through many generations has been known as Z.

The non-language aspects of VDM were taken up by the STL laboratory in Harlow and, partly because of their industrial push, BSI were persuaded to establish a standardisation activity. This activity has not been easy to get going because of the differences between the pressures of the language description aspects of VDM and those who are only interested in pre/post-conditions, data reification and operation decomposition. It is to the credit of the standards committee that they have managed to bear in mind the requirements of both sorts of user and come up with a standard which embraces such a wide scope of technical ideas. There are now many books on VDM and more papers than even Dines Bjørner's energy could keep in a bibliography although Peter Gorm-Larsen has made an attempt to continue the work of keeping the key references in a single bibliography [Lar93].

The ideas in VDM have influenced several other specification languages including RAISE, COLD-K and VVSL.

Florence: Can you tell me how Z started?

Zebedee: Well, although there was a Z notation before 1979, many people associate the early development of Z with the period spent by Jean-Raymond Abrial in Oxford from 1979 to 1981. Abrial had used a paper he had written with Steve Schuman and Bertrand Meyer as lecture

notes for a course given in Belfast. He was invited by Tony Hoare to Oxford, and presented similar material to the Programming Research Group (PRG) where it generated considerable interest and resulting activity. The notation described in this paper includes a basis in set theory and predicate calculus, although at this time the schema notation had not been fully developed.

Jean-Raymond Abrial was in Oxford at the same time as Cliff Jones, who had already worked on the Vienna Definition Language and the Vienna Development Method. The intention was that the two should exchange ideas and objectives and there were productive communications between the two, although in the end each pursued a distinctive path.

Florence: Is there such a thing as a Z method?

Zebedee: Z is a *notation*, and there is no official method attached to it, though there are conventions and practices that make it specially suitable for specifications written in the model-oriented style. The status of Z as a mathematical notation (rather than a method) is deliberate, and gives it flexibility and open-endedness.

Florence: How did the notation develop after the first proposals?

Zebedee: As with much of the PRG research, early development of Z centred on industrial case studies. An important early case study was CAVIAR, a visitor information system for an industrial company based on requirements from STL; other case studies carried out in the early stages included those based on the UNIX File System, the ICL Data Dictionary, and several on topics in Distributed Computing. PRG members carrying out case studies included Carroll Morgan, Ian Hayes, Bernard Sufrin, Ib Sørensen, and others. Ian Hayes, in addition to his contributions to the IBM CICS project, later collected these case studies and published them in the first book on Z [Hay93].

One of the most extensive case studies has been the use of Z for defining CICS, a transaction processing system developed by IBM. The collaboration between the PRG and the Hursley development laboratory, starting in 1982 and still continuing, has been a valuable source of information and experience for both groups.

During this early period the design of the most distinctive feature of Z, the *schema*, together with related schema operations, emerged in its present form. The Z schema notation was originally introduced as a technique for structuring large specifications and was seen as a means of naming and copying segments of mathematical text, much like a textual macro language. It was later apparent that schemas could be used more generally to define the combination of specifications, and the basic operations of schema inclusion and conjunction were extended to form the more comprehensive operations that make up what has been called the *schema calculus*.

Florence: You've talked about the PRG contribution to application case studies – what about the underlying theory?

Zebedee: In early stages of Z development, the notation was described in documents produced in the PRG and locally distributed. The complete language description, *The Z Handbook* by B. A. Sufrin [Suf88], was given only a limited circulation, and in fact the first account of the notation published in book form was in the 1987 edition of the collection of *Case Studies* (second edition [Hay93]) mentioned above. Theoretical work on the foundations of Z continued in the PRG and elsewhere, and an important contribution was provided by the D.Phil. thesis of Mike Spivey, subsequently published as a book [Spi88].

With a growing number of industrial users of Z, requests for standardisation were made at Z User Meetings in 1987 and 1988. Work was started in the Programming Research Group to establish an agreed definition of the language. Starting with the best available documentation, including [Suf88], the document produced in 1989 as a result of this work, the *Reference Manual*

by J. M. Spivey became a widely accepted description of Z and provided the main starting point for the standards work described below – it is now in its second edition [Spi92].

Florence: I believe there is now a draft standard for Z – what is the status of this?

Zebedee: Towards the end of 1989 a project to develop Standards, Methods and Tools for Z was set up, with supporting funding from the UK Department of Trade and Industry. The formation of the ZIP project marked the beginning of a further stage of development, providing a stable basis for the development of national and international standards for Z. As with other projects of this kind, members of the project included both industrial and academic partners. The project was divided into four main working groups dealing with Standards, Methods and Tools – there was also a Foundations group providing theoretical support, mainly for the standards work.

The Z Standard Group developed new material for the standard, not only providing a newly written document in the style needed for a standard, but also introducing new material for the semantics (see for example [GLW91]) and logic [WB92] defined in the standard. The first draft, Version 1.0 (reference) was presented at the Z Users Meeting in December 1992 and the standards committee is now at work, reviewing and revising the document as it becomes ready for standardisation in ISO.

Meanwhile, industry users are busy using Z on projects, writing tools for Z and considering how it can be combined with other notations and methods. A good idea of the breadth and variety of interest can be gained from the Z Bibliography [Bow92].

The standards committees for Z and VDM-SL keep in touch by exchange of documents and by the appointment of liaison members. They are both subcommittees of the same BSI standards committee.

Florence: Could you give me some useful references?

Zebedee: For Z, the standard reference for the language (until the language standard appears) is Mike Spivey's [Spi92]. However, this is a language reference manual and there are some more introductory texts such as [PST91, Wor92] and a book of case studies [Hay93].

Dougal: For the non-compiler aspects of VDM, the standard reference has been [Jon90] and a case studies book is [JS90]; but [WH93] refers to Jones' book as 'austere' and either of [AI91, LBC90] might be more approachable. A good overview of VDM-SL is contained in [Daw91]; although there are several books on the language description and compiler development aspects of VDM, they haven't really come up very much in our discussion.

Florence: You have both ignored details of concrete syntax of the mathematical notation: these differences confuse some people.

Zebedee: Yes, but they are just an accident of history.

Dougal: A list of the syntactic differences has been given in a note [ISO91] from the Japanese ISO representatives.

Florence: Well, it's time for bed.

Zebedee: Boing!

Dougal: *Chases his tail for a bit before running off to bed.*

Acknowledgements

We would like to acknowledge the input from John Fitzgerald, and for permission to reuse the NDB material from [FJ90]. Peter Gorm Larsen, Lynn Marshall, Anthony Hall, Tony Hoare and Tim Clement gave us useful comments on drafts of this paper. Cliff Jones thanks the SERC for

the financial support of his Senior Research Fellowship, and Ian Hayes both the financial support from the Special Studies Program from the University of Queensland and the hospitality of the Department of Computer Science at the University of Manchester, where he visited for the first half of 1993. We would also like to thank the BCS journal *Formal Aspects of Computing* for permission to reuse the NDB material from [Hay92].

References

[AI91] D. Andrews and D. Ince. *Practical Formal Methods with VDM.* McGraw-Hill, 1991.

[BCJ84] H. Barringer, J.H. Cheng, and C. B. Jones. A logic covering undefinedness in program proofs. *Acta Informatica*, 21:251–269, 1984.

[BJ78] D. Bjørner and C. B. Jones, editors. *The Vienna Development Method: The Meta-Language*, volume 61 of *Lecture Notes in Computer Science.* Springer-Verlag, 1978.

[BJ82] D. Bjørner and C. B. Jones. *Formal Specification and Software Development.* Prentice Hall International, 1982.

[Bow92] J.P. Bowen. Select Z bibliography. In J.E. Nicholls, editor, *Z User Workshop, York 1991*, Workshops in Computing, pages 367–397. Springer-Verlag, 1992.

[CJ91] J. H. Cheng and C. B. Jones. On the usability of logics which handle partial functions. In C. Morgan and J. C. P. Woodcock, editors, *3rd Refinement Workshop*, pages 51–69. Springer-Verlag, 1991.

[Daw91] J. Dawes. *The VDM-SL Reference Guide.* Pitman, 1991.

[FJ90] J.S. Fitzgerald and C. B. Jones. Modularizing the formal description of a database system. In D. Bjørner, 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*, pages 189–210. Springer-Verlag, 1990.

[GLW91] P.H.B. Gardiner, P.J. Lupton, and Jim C.P. Woodcock. A simpler semantics for Z. In J.E. Nicholls, editor, *Z User Workshop, Oxford 1990*, Workshops in Computing, pages 3–11. Springer-Verlag, 1991.

[Hay92] I. J. Hayes. VDM and Z: A comparative case study. *Formal Aspects of Computing*, 4(1):76–99, 1992.

[Hay93] Ian Hayes, editor. *Specification Case Studies.* Prentice Hall International, second edition, 1993.

[Hoa91] C. A. R. Hoare. Preface. In *[PT91]*, pages vii–x, 1991.

[HW93] I. J. Hayes and L. P. Wildman. Towards libraries for Z. In J. P. Bowen and J. E. Nicholls, editors, *Z User Workshop: Proceedings of the Seventh Annual Z User Meeting, London, December 1992*, Workshops in Computing. Springer-Verlag, 1993.

[ISO91] ISO. Japan's input on the VDM-SL standardization, April 1991. ISO/IEC JTC1/SC22/WG19-VDM-SL.

[JM93] C.B. Jones and C.A. Middelburg. A typed logic of partial functions reconstructed classically. Logic Group Preprint Series 89, Utrecht University, Department of Philosophy, April 1993.

[Jon80] C. B. Jones. *Software Development: A Rigorous Approach.* Prentice Hall International, 1980.

[Jon90] C. B. Jones. *Systematic Software Development using VDM.* Prentice Hall International, second edition, 1990.

[JS90] C. B. Jones and R. C. F. Shaw, editors. *Case Studies in Systematic Software Development.* Prentice Hall International, 1990.

[Lar93] Peter Gorm Larsen. VDM as a mature formal method. Technical report, Institute of Applied Computer Science, April 1993.

[LBC90] J. T. Latham, V. J. Bush, and I. D. Cottam. *The Programming Process: An Introduction Using VDM and Pascal.* Addison-Wesley, 1990.

[MS84] C.C. Morgan and B.A. Sufrin. Specification of the UNIX file system. *IEEE Trans. on Software Engineering*, SE-10(2): 128–142, March 1984.

[PST91] Ben Potter, Jane Sinclair, and David Till. *An Introduction to Formal Specification and Z.* Prentice Hall International, 1991.

[PT91] S. Prehn and W. J. Toetenel, editors. *VDM'91 - Formal Software Development Methods. Proceedings of the 4th International Symposium of VDM Europe, Noordwijkerhout, The Netherlands, October 1991, Vol.2: Tutorials*, volume 552 of *Lecture Notes in Computer Science.* Springer-Verlag, 1991.

[Spi88] J.M. Spivey. *Understanding Z—A Specification Language and its Formal Semantics.* Cambridge Tracts in Computer Science 3. Cambridge University Press, 1988.

[Spi92] J.M. Spivey. *The Z Notation: A Reference Manual.* Prentice Hall International, second edition, 1992.

[Suf88] B. A. Sufrin. Notes for a Z handbook: Part 1 - the mathematical language, 1988. Programming Research Group, Oxford University.

[Wal90] A. Walshe. NDB. In *[JS90]*, chapter 2, pages 11–46. Prentice Hall International, 1990.

[War93] N. Ward. Adding specification constructors to the refinement calculus. In J.C.P. Woodcock and P.G. Larsen, editors, *Proceedings, Formal Methods Europe'93*, volume 670 of *Lecture Notes in Computer Science*, pages 652–670. Springer Verlag, 1993.

[WB92] J.C.P. Woodcock and S.M. Brien. W: A logic for Z. In J.E. Nicholls, editor, *Z User Workshop, York 1991*, Workshops in Computing, pages 77–96. Springer-Verlag, 1992.

[Wel82] A. Welsh. The specification, design and implementation of NDB. Master's thesis, University of Manchester, October 1982.

[WH93] M. Woodman and B. Heal. *Introduction to VDM.* McGraw-Hill, 1993.

[Wor92] J. B. Wordsworth. *Software Development with Z.* Addison-Wesley, 1992.

A VDM specification

This specification has been adapted from the NDB specification in [FJ90]. In some minor respects (e.g. optional relation names), it is more restrictive than the original [Wel82] (to which the reader is referred for a description of the operations – such as *ADDTUP* – which are not discussed above).

module *NDB*

parameters

types $Value, Esetnm, Rnm$: *Triv*

exports

operations $ADDES, ADDENT, ADDREL, ADDTUP,$
$DELES, DELENT, DELREL, DELTUP$

definitions

defined types

Eid = token

$Maptp = \{\text{ONEONE}, \text{ONEMANY}, \text{MANYONE}, \text{MANYMANY}\}$

$Tuple :: fv : Eid$
$tv : Eid$

$Relation = Tuple\text{-set}$

$Rinf :: tp : Maptp$
$r : Relation$

inv $(mk\text{-}Rinf(tp, r)) \triangleq arity\text{-}match(tp, r)$

$Rkey :: nm : Rnm$
$fs : Esetnm$
$ts : Esetnm$

state

$Ndb :: esm : Esetnm \xrightarrow{m} Eid\text{-set}$
$em : Eid \xrightarrow{m} Value$
$rm : Rkey \xrightarrow{m} Rinf$

inv $(mk\text{-}Ndb(esm, em, rm)) \triangleq$
$\text{dom } em = \bigcup \text{rng } esm \wedge$
$\forall rk \in \text{dom } rm \cdot$
$\{rk.fs, rk.ts\} \subseteq \text{dom } esm \wedge$
$\forall mk\text{-}Tuple(fv, tv) \in rm(rk).r \cdot fv \in esm(fs) \wedge tv \in esm(ts)$

init $(ndb) \triangleq ndb = mk\text{-}Ndb(\{\}, \{\}, \{\})$

defined functions

$arity\text{-}match(tp, r) \triangleq$
$(tp = \text{ONEMANY} \Rightarrow \forall t_1, t_2 \in r \cdot t_1.tv = t_2.tv \Rightarrow t_1.fv = t_2.fv) \wedge$
$(tp = \text{MANYONE} \Rightarrow \forall t_1, t_2 \in r \cdot t_1.fv = t_2.fv \Rightarrow t_1.tv = t_2.tv) \wedge$
$(tp = \text{ONEONE} \Rightarrow \forall t_1, t_2 \in r \cdot t_1.fv = t_2.fv \Leftrightarrow t_1.tv = t_2.tv)$

defined operations

$ADDES\ (es\colon Esetnm)$

ext wr $esm\ :\ Esetnm \xrightarrow{m} Eid\text{-set}$

pre $es \notin \mathsf{dom}\ esm$

post $esm = \overleftarrow{esm} \cup \{es \mapsto \{\}\}$

$DELES\ (es\colon Esetnm)$

ext wr $esm\ :\ Esetnm \xrightarrow{m} Eid\text{-set}$
rd $rm\ :\ Rkey \xrightarrow{m} Rinf$

pre $es \in \mathsf{dom}\ esm \wedge esm(es) = \{\} \wedge$
$\forall rk \in \mathsf{dom}\ rm \cdot es \neq rk.fs \wedge es \neq rk.ts$

post $esm = \{es\} \lhd \overleftarrow{esm}$

$ADDENT\ (memb\colon Esetnm\text{-set}, val\colon Value)\ eid\colon Eid$

ext wr $esm\ :\ Esetnm \xrightarrow{m} Eid\text{-set}$
wr $em\ :\ Eid \xrightarrow{m} Value$

pre $memb \subseteq \mathsf{dom}\ esm$

post $eid \notin \mathsf{dom}\ \overleftarrow{em} \wedge$
$em = \overleftarrow{em} \cup \{eid \mapsto val\} \wedge$
$esm = \overleftarrow{esm} \dagger \{es \mapsto \overleftarrow{esm}(es) \cup \{eid\} \mid es \in memb\}$

$DELENT\ (eid\colon Eid)$

ext wr $esm\ :\ Esetnm \xrightarrow{m} Eid\text{-set}$
wr $em\ :\ Eid \xrightarrow{m} Value$
rd $rm\ :\ Rkey \xrightarrow{m} Rinf$

pre $eid \in \mathsf{dom}\ em \wedge$
$\forall t \in \bigcup\{ri.r \mid ri \in \mathsf{rng}\ rm\} \cdot t.fv \neq eid \wedge t.tv \neq eid$

post $esm = \{es \mapsto \overleftarrow{esm}(es) - \{eid\} \mid es \in \mathsf{dom}\ \overleftarrow{esm}\} \wedge$
$em = \{eid\} \lhd \overleftarrow{em}$

$ADDREL\ (rk\colon Rkey, tp\colon Maptp)$

ext rd $esm\ :\ Esetnm \xrightarrow{m} Eid\text{-set}$
wr $rm\ :\ Rkey \xrightarrow{m} Rinf$

pre $\{rk.fs, rk.ts\} \subseteq \mathsf{dom}\ esm \wedge$
$rk \notin \mathsf{dom}\ rm$

post $rm = \overleftarrow{rm} \cup \{rk \mapsto mk\text{-}Rinf(tp, \{\})\}$

$DELREL\ (rk\colon Rkey)$

ext wr $rm\ :\ Rkey \xrightarrow{m} Rinf$

pre $rk \in \mathsf{dom}\ rm \wedge r(rm(rk)) = \{\}$

post $rm = \{rk\} \lhd \overleftarrow{rm}$

ADDTUP $(fval, tval: Eid, rk: Rkey)$

ext wr $rm : Rkey \xrightarrow{m} Rinf$
rd $esm : Esetnm \xrightarrow{m} Eid\text{-set}$

pre $rk \in \mathrm{dom}\, rm \wedge$
let $mk\text{-}Rkey(nm, fs, ts) = rk$ in
let $mk\text{-}Rinf(tp, r) = rm(rk)$ in
$fval \in esm(fs) \wedge tval \in esm(ts) \wedge arity\text{-}match(tp, r \cup mk\text{-}Tuple(fval, tval))$

post $rm = \overleftarrow{rm} \dagger \{rk \mapsto \mu(\overleftarrow{rm}(rk), r \mapsto r(\overleftarrow{rm}(rk)) \cup \{mk\text{-}Tuple(fval, tval)\})\}$

DELTUP $(fval, tval: Eid, rk: Rkey)$

ext wr $rm : Rkey \xrightarrow{m} Rinf$

pre $rk \in \mathrm{dom}\, rm$

post let $ri = \mu(\overleftarrow{rm}(rk), r \mapsto r(\overleftarrow{rm}(rk)) - \{mk\text{-}Tuple(fval, tval)\})$ in
$rm = \overleftarrow{rm} \dagger \{rk \mapsto ri\}$

endmodule *NDB*

B Z specification

This specification has been adapted from the specification of NDB given in [Hay92].

B.1 Entities and entity sets (or types)

$[Eid, Esetnm, Value]$

Entities

$esm: Esetnm \twoheadrightarrow (\mathbb{F}\, Eid)$
$em: Eid \twoheadrightarrow Value$

$\mathrm{dom}\, em = \bigcup(\mathrm{ran}\, esm)$

$\Delta Entities \mathrel{\widehat=} Entities \wedge Entities'$

$\Xi Entities \mathrel{\widehat=} [\Delta Entities \mid \theta Entities' = \theta Entities]$

ADDES0

$\Delta Entities$
$es?: Esetnm$

$es? \notin \mathrm{dom}\, esm \wedge$
$esm' = esm \cup \{es? \mapsto \{\}\} \wedge$
$em' = em$

DELES0

$\Delta Entities$
$es?: Esetnm$

$es? \in \mathrm{dom}\, esm \land esm(es?) = \{\} \land$
$esm' = \{es?\} \lhd esm \land$
$em' = em$

ADDENT0

$\Delta Entities$
$memb?: \mathbb{F}\, Esetnm$
$val?: Value$
$eid!: Eid$

$memb? \subseteq \mathrm{dom}\, esm \land$
$eid! \notin \mathrm{dom}\, em \land$
$em' = em \cup \{eid! \mapsto val?\} \land$
$esm' = esm \oplus \{es: memb? \bullet es \mapsto esm(es) \cup \{eid!\}\}$

DELENT0

$\Delta Entities$
$eid?: Eid$

$eid? \in \mathrm{dom}\, em \land$
$em' = \{eid?\} \lhd em \land$
$esm' = \{es: \mathrm{dom}\, esm \bullet es \mapsto esm(es) \setminus \{eid?\}\}$

B.2 A single relation

$Tuple == Eid \times Eid$
$Relation == Eid \leftrightarrow Eid$

$Maptp ::= OneOne \mid OneMany \mid ManyOne \mid ManyMany$

Rinf

$tp: Maptp$
$r: Relation$

$(tp = OneOne \Rightarrow r \in Eid \rightarrowtail Eid) \land$
$(tp = ManyOne \Rightarrow r \in Eid \nrightarrow Eid) \land$
$(tp = OneMany \Rightarrow r^{\sim} \in Eid \nrightarrow Eid)$

$\Delta Rinf \mathrel{\widehat{=}} [Rinf; Rinf' \mid tp' = tp]$

ADDTUPLE0

$\Delta Rinf$
$t?: Tuple$

$r' = r \cup \{t?\}$

DELTUPLE0

$\Delta Rinf$
$t?: Tuple$

$r' = r \setminus \{t?\}$

B.3 Multiple relations

$[Rnm]$

Rkey

$nm: Rnm$
$fs, ts: Esetnm$

Ndb

$Entities$
$rm: Rkey \pfun Rinf$

$\forall rk: \mathrm{dom}\, rm \bullet$
$\quad \{rk.fs, rk.ts\} \subseteq \mathrm{dom}\, esm \land$
$\quad (\forall t: (rm\ rk).r \bullet$
$\quad\quad first\ t \in esm(rk.fs) \land second\ t \in esm(rk.ts))$

$\Delta Ndb \mathrel{\widehat=} Ndb \land Ndb'$
$\Delta REL \mathrel{\widehat=} \Delta Ndb \land \Xi Entities$

ADDREL

ΔREL
$tp?: Maptp$
$rk?: Rkey$

$rk? \notin \mathrm{dom}\, rm \land$
$\{rk?.fs, rk?.ts\} \subseteq \mathrm{dom}\, esm \land$
$rm' = rm \cup \{rk? \mapsto (\mu Rinf \mid r = \{\} \land tp = tp?)\}$

DELREL

ΔREL
$rk?: Rkey$

$rk? \in \mathrm{dom}\, rm \land$
$(rm\ rk?).r = \{\} \land$
$rm' = \{rk?\} \ndres rm$

B.4 Promotion of operations

$\Xi RM \mathrel{\widehat=} [\Delta Ndb \mid rm' = rm]$

$$ADDES \mathrel{\widehat=} ADDES0 \land \Xi RM$$
$$DELES \mathrel{\widehat=} DELES0 \land \Xi RM$$
$$ADDENT \mathrel{\widehat=} ADDENT0 \land \Xi RM$$
$$DELENT \mathrel{\widehat=} DELENT0 \land \Xi RM$$

Promote

$$\Delta REL$$
$$rk?: Rkey$$
$$\Delta Rinf$$

$$rk? \in \mathrm{dom}\, rm \land$$
$$\theta Rinf = rm(rk?) \land$$
$$rm' = rm \oplus \{rk? \mapsto \theta Rinf'\}$$

$$ADDTUPLE \mathrel{\widehat=} (\exists \Delta Rinf \bullet ADDTUPLE0 \land Promote)$$
$$DELTUPLE \mathrel{\widehat=} (\exists \Delta Rinf \bullet DELTUPLE0 \land Promote)$$

A Response to Florence, Dougal and Zebedee

Antony Hall
Email: jah@praxis.co.uk

The article "Understanding the Differences between VDM and Z" sets out to offer an understanding of the interesting differences between two well-known specification languages. My immediate reaction is to ask "interesting to whom?". As an industrial practitioner of formal methods, and one who gives courses and consultancy, I am often asked about this very topic. Typically my questioner is someone who actually wants to use one of these languages and doesn't know which to choose. Would the article help such a person? I fear not.

This is not the fault of the authors. Rather, the problem seems to me to be that the we do not really know how to characterise, objectively, the issues that are important for the user of a language. It is possible to be pretty precise about some of the differences —for example, three valued .vs. two valued logic, domains .vs. sets, and so on— and these are interesting differences to a methodologist. But there is another class of differences, such as "How do they differ in applicability?", "How do they scale up in practice?", which are interesting to potential users but which are harder to answer objectively.

The article in fact dismisses some such differences. For example, in the discussion on modelling style Dougal explicitly dismisses as irrelevant the fact that Z has a lot of operations on binary relations. From a practitioners point of view this is very far from irrelevant. Dougal "would just extend VDM appropriately". I was a bit puzzled by this, although this may be because of my ignorance of BSI VDM. It isn't clear to me that there is any mechanism to extend VDM. Furthermore, even if there is, such an extension would presumably introduce a new type, the binary relation, quite different from any other types. In Z, however, not only are binary relations built in, but there is an explicit collection of extension capabilities built in to the language. Furthermore, the fact that, for example, sequences are functions are relations are sets is enormously helpful. We recently specified and designed a large system in VVSL, a VDM extension. In writing the specification we wanted to do things like composing functions with sequences and there seemed to be no mechanism for doing this; in Z such mechanisms come free. So a discussion of the ways you can extend the two languages, and a discussion of the advantages and disadvantages of the type-homogeneity of functions, sequences etc. in Z versus new data types in VDM would be highly relevant. Similarly the strengths and weaknesses of the Z extension mechanism (fine for the mathematical language, but with no useful way of extending the schema calculus) could be brought out.

On a similar note, the question of modularisation is very important in practice. Dougal mentions that operation specifications need to be in modules, but the module mechanism in BSI VDM has received little practical testing. We looked at it for our project and it was completely useless for the kind of specification (with a lot of shared state) that we wanted to write. (We couldn't even find a way of expressing the fact that the time was the same in all modules!). The strengths and weaknesses of the Z schema are pretty well understood — it has severe theoretical shortcomings, but in practice, with decent guidelines (as taught on Praxis Z courses) it works extremely well for a variety of specification styles.

This is related to the issue of Z preconditions only making sense when you have the whole operation. This is true, and related to Tony Hoare's suggestion in footnote 5. I think this kind of issue is worth much more than a footnote. Z does give you very convenient syntactic sugar for developing your specification in small pieces. This is ENORMOUSLY useful, BUT you pay a price — the small pieces are not really implementation-level entities. It is tremendously useful to be able to define an operation in pieces, each piece defining part of the effect, and then to AND them together. But you can only do this because the parts are not really operations at all, but only relations. And it is only by courtesy that the final relation is deemed to be an operation of the system. So the pieces are not, for example, separately refinable, and their preconditions are related to the real precondition in a pretty obscure way. So this is NOT a useful way of doing development, but it IS a useful way of writing specifications. (I know, because it's very difficult to write specifications without it!) When you move to development, you always find that more VDM-like constructs appear (look at the refinement calculus, for example) — in particular, explicit separation on pre and post, and explicit frames. These are anathema to compositional specifications (if programming languages had the AND combinator, software development would be automatable) but essential for refinement. (Having said that, I've no experience of doing large scale DATA refinement in Z, but it ought to be at least as easy as it is in VDM — and I am (albeit on no very good grounds) sceptical of the refinement calculus approach to data refinement). A further advantage (at least, I think it may be an advantage) of the VDM approach is that you can of course add more constructs to your specifications like rely/guarantee, inter conditions or whatever. There is no sensible way of doing this in the normal Z style, as far as I am aware.

A particular instance is exceptions. Again, Z doesn't really distinguish exceptions, and the conventional way of expressing them is certainly clumsier than the VDM way — this is the one area where I am glad we were using VVSL not Z on our project (though the advantage was considerable reduced by the lack of operation combinators). The article downplays the syntactic issues here — surprisingly, since in his book on VDM Cliff Jones quotes, presumably with approval, Whitehead's comment that a good notation "increases the mental power of the race".

On the other hand, the article does sometimes stray into unjustified generalisations of its own. The bland statement that "Z needs a modularisation mechanism" is highly disputable. It's got one: it MAY need another one, but it may just need some good rules for using the one it's got. It is certainly possible to use the existing one for substantial specifications without too much trouble.

In summary, I think there is another article waiting to be written — less objective, perhaps, but more relevant to the growing number of industrial practitioners who are trying to understand the choices they have to make to use formal methods effectively.

Security, Reliability and Correctness of Software:
A Description of Current German Activities in the Field

M. Broy, W. Bibel, S. Jähnichen,
H.-J. Kreowski, J. Siekmann, F. Vogt*

Increasing attention is currently being given to the quality of software in the community at large. For both technological and social reasons, there is a growing need for us to be able to evaluate systems consisting largely of software components, too, according to such quality criteria as "correct", "reliable" and "secure", thus substantially increasing confidence in such systems.

At present, there are in Germany a number of projects in progress dealing with this particular topic, and others are planned. These projects, which are concerned with providing the necessary scientific and technological foundations for the development of correct software, are currently being supported by three funding agencies:

- the German Federal Ministry for Research and Technology (Bundesministerium für Forschung und Technologie, BMFT)
- the German Information Security Agency, GISA (Bundesamt für Sicherheit in der Informationstechnologie, BSI) in Bonn
- the German Research Association (Deutsche Forschungsgmeinschaft, DFG) as part of its Keynote Programme on Deduction.

To coordinate these activities, experts from all sides met for talks at Schloß Dagstuhl on March 16, 1993. At this meeting, four projects were presented and their goals discussed. These are (together with the respective funding agencies):

* Correspondence to the authors should be addressed to: Prof. S. Jähnichen, GMD-FIRST, Rudower Chaussee 5, D-1199 Berlin, Germany. (Please note new postal/zip code as of July 1, 1993: D-12489 instead of D-1199). e-mail address: jaehn@cs.tu-berlin.de

1. The KORSO (Correct Software) Project, BMFT.

   An integrated project funded by the BMFT. Its aim is the prototypical application of tools and methods, the implementation of basic knowledge and the adaptation of tools and techniques to the needs of the respective applications.

2. The VSE (Verification Support Environment) Project, GISA.

   Initiated and funded by the German Information Security Agency. The project is concerned with the concrete development of a method and corresponding tools for checking the security of software.

3. The Keynote Programme on Deduction, DFG.

   The Keynote Programme is dedicated to basic research on deduction techniques and tools.

4. Proposal for a DFG Keynote Programme on Development Techniques.

   The proposal was rejected or rather deferred by the DFG, partly on the grounds that there was no clear demarcation between it and the above-mentioned projects.

   The aim of the proposed project was to look into innovative techniques and methods for constructing complex systems consisting largely of software components.

The projects were found to have widely differing goals, but at the same time to ideally complement each other in the sense that they investigate the essential prerequisites for the production of provably correct software. It was agreed that the topic is of paramount importance for German information technology and that the necessary development techniques have not yet been properly mastered. All those present thus shared the view that there is a continuing need for research into basic techniques for the development of software. This need relates not only to the further development of so-called "formal" methods, but also to research into and development of mechanisms for integrating methods into uniform construction techniques for systems consisting largely of software components.

There was general agreement that the strong mathematical and logical orientation of training and research in the information technology field in Germany provides an ideal basis for research and development work in this area, thus ensuring an internationally recognized standard of excellence. German information technology is an international leader in this particular field,

which means there is a good chance of exercising a considerable influence on developments here and of belonging to the vanguard with respect to the anticipated industrial applications.

To provide an overview of current work in this field, more detailed treatment is given below to the three projects now in progress and to the aims of the proposed DFG Keynote Programme mentioned above.

1. **Correct Software (KORSO)** (Coordinators: Professor Broy, Professor Jähnichen), **BMFT**

Since the autumn of 1991, 14 academic and industrial partners have been working together in the integrated BMFT project KORSO on techniques for developing correct software.

The overall aim of the project is to enhance theoretical foundations, to improve development concepts and methods for producing correct software, and to design and implement the required tools. A long-term goal is to make program development techniques that are systematically geared to correctness both practicable and economically viable. For this purpose, case studies are to be used to demonstrate the suitability of such techniques. The fundamental problems confronting attempts to ensure the correctness of software are now well-known, but have at best been solved for small programs only.

In the KORSO project, work is underway to make well-known techniques for developing larger programs practically implementable. This work includes the design of a general methodological framework, the development of a generic description and modelling language, the support of correctness-preserving development steps by means of mathematically sound calculi and tools based on these, as well as a constant critical review of each partner's work by reference to numerous accompanying case studies.

The current status of the project may be outlined as follows:

A methodological framework for the development of correct software has been created which is geared to the needs of formal techniques. This allows the definition of clearly distinguishable subtasks for requirements analysis, design decision support, and for problems relating to description and to validation and verification. By way of a reference language for the KORSO project, the wideband language SPECTRUM was defined, enabling all the documents generated in the development process – from the requirements specification to the (functional) program – to be written in uniform terms. In this context, work is also being done on modelling distributed systems.

In the field of verification, new, improved techniques have been developed both for the synthesis and transformational development of individual algorithms, and for the systematic construction of large, modular systems. Innovative elements here are the formal treatment, reuse and metalogical reflection of specifications, proofs and developments. At present, a concept for integrating the different support tools is being elaborated. Numerous case studies are planned to ensure the practicability and suitability of the methods and techniques developed in the context of the project. The main ones currently being worked on are:

- *Heart Patient Data Management System (HDMS) at the Berlin Heart Centre*

Here, a requirements specification was written in SPECTRUM for the system core. A significant feature of this case study, besides its size, is its treatment of problems relating to the interfaces with existing subordinate systems such as frequently occur in practical applications, but which have been given scarcely any attention so far in research work.

- *Production Control System*

Here, various competing specification and development methods are being tested A particular feature encountered here are the problems of modelling distributedness and real time typical of this application area.

Finally, concepts were drawn up for a heterogeneous support system allowing the integration of different methods, languages and logics of formal software development. Using meta-languages, the aim is to lay the foundations for a modular system of specification and verification techniques and tools providing appropriate formalisms for different application areas.

2. **Verification Support Environment (VSE)** (Spokesman: Professor Siekmann), **GISA**

The VSE project owes its origin to an initiative by the German Information Security Agency (GISA). In 1989, this newly created agency was responsible for issuing a catalogue of quality criteria for information-processing systems designed to help in evaluating the security of such systems. In the case of the two highest quality levels, the use of formal methods and specialized development tools is prescribed for the (software) production process. To make such tools available, in early 1991 the GISA commissioned a consortium of industrial and academic partners to implement a system for strictly formal development of critical system components. VSE partners are: *Dornier GmbH*, the *Gesellschaft für Prozeßrechner-Programmierung (GPP)*

Forschungsinstitut für Künstliche Intelligenz (DFKI) (German Artificial Intelligence Research Institute) in Saarbrücken, and the University of Ulm. The first phase of the project is due to end in mid-1994 with the delivery of the first prototype.

The VSE system is based on a formal method for the modular development of software components by stepwise implementation of structured specifications. This method is first of all implemented in the form of a suitably structured management system for centralized data storage and user guidance. It is in this framework that deduction systems to support the user in fulfilling the proof obligations arising during the development process are integrated. Deduction problems reflecting the mathematical concepts behind a particular method are essential features of a formal approach (unlike nonformal techniques). The VSE frame system consists of a commercial CASE tool (EPOS developed by GPP) enhanced by an appropriate formal component. This means that tool support is also available when using nonformal development techniques and for project management purposes. By combining different proof techniques and architectures, the deduction component provides a comprehensive, flexible and effective proof support system. With VSE geared to the production process, application of the system is not confined to the area of security. On the contrary, it is suitable for all areas in which software components with the highest quality requirements ("provably correct software") are called for.

Parallel to the project, two large-scale case studies are being conducted involving the redevelopment of major components of systems built in the context of industrial projects. One of the case studies deals with a control and planning system for booking, managing and executing the exchange of programmes between different broadcasting corporations; the other is concerned with a physical access control system for nuclear power plants. Former members of the industrial projects are involved in the formal redevelopment. Besides continuing work on the individual components of the VSE system and extension of the formal basis, this tentative implementation of the system in industrial practice will constitute a major element in the second project phase, which is currently in the planning stage.

3 **Deduction** (Spokesman: Professor Bibel), **DFG**

Processing knowledge and translating it into algorithmic problem solutions are essential elements in the software production process. Manipulation of this kind of (formalized) knowledge is accomplished by means of deductive mechanisms. Deduction, then, is a crucial subtechnique for the contemporary production of correct software. Its field of application, however, is much broader. That is why the DFG has, since 1992, been funding basic research in this field as part of the Keynote Research Programme on Deduction.

This programme is concerned with investigating deductive methods and testing them prototypically in a system context. The application field in view here is software production. In individual projects, specific deductive techniques are being examined with this application in mind.The consideration of concrete applications is, however, explicitly ruled out, for capacity reasons. The programme itself is rather designed to serve as a "feed" for further research projects, particularly in the field of software development.

4. "Development Techniques" (Spokesmen: Professor Vogt, Professor Kreowski), **(DFG):**

Software development, besides pursuing the sort of language-oriented concepts it has so far, must give greater attention than in the past to model-oriented or architecture-based concepts. And it is important here, with a view to practical relevance, that weakly formalized approaches, too, such as the object-oriented methods, be examined with respect to their suitability for modelling purposes. However, if formal treatment of the model-oriented and application-related approaches is the goal striven for, methods of knowledge processing and their implementation in the form of algorithmic problem solutions are specifically required. Existing formal development techniques for software systems, be they algebraically, functionally or logically based, also form a crucial cornerstone for problem solution and description and should therefore be integrated. The configuration concept is a promising approach here, and for this reason a suitable candidate for promoting the integrated use of formal development methods. This concept is based on the assumption that building blocks developed using different methods can be combined. This means that methods belonging to different specification approaches must be semantically integrated. This sort of integration requires supplementary basic research which would acquire additional synergetic impetus by close cooperation with the Deduction Programme.

A Nice Derangement of Predicates: A Formal Specification of Neurosis

E. J. Baillie
University of Hertfordshire
Hatfield
email:comqejb@herts.ac.uk

Madness, like everything else, can be formally specified. In this short paper we offer a few axioms which, we suggest, encapsulate key aspects of commonly observed behaviour and reasoning of human beings. The axioms are both internally inconsistent and incomplete with respect to any known model. Whilst this feature is not in itself original, it is clearly consistent with its subject, encouraging us in the belief that we are on the right lines. We should be pleased to hear from others working in the area.

1 The Insanity Propositions

1. $(P \Rightarrow Q) \Leftrightarrow (Q \Rightarrow P)$ (The Peugeot Principle)

as in, 'Intellectuals never look smart. I am a scruff. I must be brilliant.'
and, 'Doors on quality cars close with a nice sound. Listen to this door. There's quality for you.'

2. $(P \Rightarrow Q) \Leftrightarrow (\neg P \Rightarrow \neg Q)$

as in ,'If you don't eat up all your crusts you'll never have curly hair. So if I eat my crusts I'll have lovely curly hair.'[1]

3. $P \vee \neg P \Leftrightarrow P \wedge \neg P$ (The Law of the Included Middle)

as in, 'If I get this promotion I'll have a lot more responsibility, which I don't want. If I don't get it, I won't be able to afford my holiday, which I do want. These are mutually exclusive. I will worry about both of them.'

2 The Anxiety Predicates

1. $\forall c : Concept \bullet can\_understand(I, c) \Rightarrow must\_be\_trivial(c)$

and its corollary

1a. $\forall c : Concept \bullet can\_understand(I, c) \Rightarrow \forall a : Anyfool \bullet can\_understand(a, c)$

2. $\forall c : Concept \bullet confuses(c, Me) \Rightarrow \exists d : Cleverdick \bullet \neg confuses(c, d) \wedge deflates(d, Me)$

[1] I ate my crusts. I have straight hair.

3 The Principle of Anti-knowledge

$\forall x, y : Anything \bullet knows(x) \wedge vaguely\_similar(x, y) \wedge learns(y) \Rightarrow \neg knows(x) \wedge \neg knows(y)$

as in, 'I knew Modula2. Then I learned Ada. Now I can't write programs at all.'

Acknowledgements

I am indebted to Peugeot Talbot Marketing for a particularly helpful demonstration of proposition 1.1 in the design of the new 405 range: to my maternal grandmother for proposition 1.2: to the letters of Screwtape for proposition 1.3: and to Andrew, Mrs Malaprop and Ben Potter for their helpful contributions throughout.

Survey of Formal Methods in Software Engineering

Graeme I Parkin
National Physical Laboratory,
Teddington, Middlesex, England.
e-mail: gip@seg.npl.co.uk

29 April 1993

1 Introduction

The use of formal methods has been much heralded as the way forward for the production of good quality software but as yet they have not been widely accepted in industry. The United Kingdom's Department of Trade and Industry asked the National Physical Laboratory to investigate the reasons for this. This we did by conducting literature and industrial surveys in 1992.

The main aim of the surveys was to learn the views of people using or considering using formal methods in the areas of the: benefits, limitations, and barriers to formal methods. A secondary aim was to learn people's views of the means of assessing the contribution that formal methods make to the software life-cycle. The industrial survey also gives information on which formal methods are being used, how they are being used and with what they are being used with.

2 Literature Survey

For the literature survey we made a list of the claims made in the literature about the benefits, limitations and barriers to formal methods. We then assessed the claims made in terms of rational argument and reported experimental evidence. We concluded that the main reasons for the lack of uptake of formal methods are the use of mathematics and perhaps the lack of tools.

It was thus possible to compare the results of the literature survey with those of the industrial survey.

3 Industrial Survey

The industrial survey was aimed to reach as many people as possible to get a wide range of opinion rather than that of a few recognised experts. The survey used a questionnaire which was designed so that it did not lead people to answer the questions in a particular way. The returned questionnaires were analysed with the help of software written in UNIX commands, in particular Bourne shell and nawk scripts.

The industrial survey attracted a lot of interest with over 400 questionnaires being returned. This survey is the largest of its type ever conducted in the area of formal methods with participants mainly from industry but with significant input from academics. It should be noted that the survey is mainly of the United Kingdom.

The survey has shown that the most widely used formal methods are VDM (55%[1]) and Z (55%). The formal methods which are used to specify concurrent systems e.g. LOTOS, CCS and CSP (18%), are less widely used probably because they have been designed specifically for concurrent systems.

Formal methods are being integrated into the software life-cycle, most widely through structured methods (SSADM and Yourdon in particular) and much less through requirements analysis tools (like CORE).

Those who are using formal methods do so because customers require it (in particular the MOD and in security standards) or they work in the area of safety critical software which warrants best practice.

The survey shows three main reasons why formal methods are not being more widely used in industry:

- There is a lack of tools for formal methods, in particular commercially supported tools. There could be several reasons for this: not large enough market, lack of standards (of the most widely used formal methods only LOTOS is standardised) or not clear what type of tools are needed. An obvious way to improve this would be to get VDM and Z standardised in such a form that effective tools can be built.
- It has not been shown conclusively that there are cost benefits to be gained from the use of formal methods in producing software.
- Many of the barriers (and limitations) to the use of formal methods are the symptoms of the process of change from the use of one technique to a completely different one. This process will take some time to work itself out.

Of the above three reasons the lack of tools was confirmed by the literature survey, but not the other two. The literature survey proposed instead that the use of mathematics was a problem but the survey of industry did not confirm this.

The survey has shown clearly that people are not aware of any good techniques for assessing objectively the contribution of formal methods to the quality of the software.

4 Way forward

To overcome some of the above problems we suggest the following:

- A programme of education to spread the understanding of formal methods which would help the process of change.
- Case studies, which may show the cost benefits to be gained by using formal methods.

[1] This is the percentage of the participants who voted with this response.

- Research on metrics and data collection to help in assessing the contribution of formal methods to the production of software.
- Efforts to get VDM and Z standardised as soon as possible.

A report on the complete details of the survey can be obtained for £25 from: DITC Office, Formal Methods Survey, National Physical Laboratory, Queens Road, Teddington, Middlesex, TW11 0LW, United Kingdom. Tel: 081-943 7002. Fax: 081-977 7091.

Anyone for Fractal Programming Semantics?

Mike Stannett
13 Ibbotson Road
Sheffield S6 5AD

The popularisation of fractal-based graphics over the last decade or so has brought these intriguing and elegant mathematical objects very much into the public domain - and especially to the attention of today's game-crazy youngsters sitting up night after night with only Def Leppard and their PCs for company. Here we have a case of a whole discipline - experimental mathematics - evolving hand-in-hand with the computers that make its problems, if not soluble, then at least *compelling*. This is obviously an excellent development from the viewpoint of sharing the sheer joy of discovery that drives so many research projects, but the point I want to address here is whether or not this cross-fertilisation doesn't, in fact, work in both directions. The processes carried out for the tired-eyed kid by his fractal kit have a lot in common with processes at the heart of many types of programming semantics.

Open any introductory text on fractals, and you'll find one of the subject's standard icons: the bifurcation diagram for the function

$$f(k,x) = x + kx(1-x)$$

which arises in studies of population dynamics. For example, we can think of x as representing the proportion of a city's population who are currently suffering from some virulent infection. The parameter k represents the infection rate, and $f(k,x)$ tells us the proportion of infected citizens in the next time period. Often in such situations, it's useful to identify the long-term behaviour of the system - how big must x be before an epidemic becomes inevitable? The experimental mathematician might approach this problem by trying out lots of examples - she simply chooses some value of k, some initial value of x, and then iterates f until a clear answer becomes apparent. As is very well-known, the long-run behaviour depends very much upon the choice of k and the initial value of x - for some choices, the series of values generated by repeatedly applying f diverges, for other choices the series converges towards a repeating finite set of values, and in yet other cases the behaviour becomes chaotic. The bifurcation diagram is simply a graph showing, for each value of k along the horizontal axis, the long-run behaviour of f.

We can easily describe this bifurcation diagram formally. Suppose the experimenter chooses initial values k and x. Notice the difference between these two variables: k is *static* because the same value of k is used in each iteration, whereas x is *dynamic* because a *new* value of x is substituted-in each time. Writing $F^0(k,x) = x$ and $F^{n+1}(k,x) = f(k,F^n(k,x))$, the cross-section of the bifurcation diagram above the value k, corresponding to the initial value x, is just the set of points which can be approximated arbitrarily accurately by the values of $F^n(k,x)$ as n ranges through the naturals. That is, if we write $F_x(k)$ to denote this cross-section, then

$$F_x(k) = \bigcap_{n\in\mathbf{N}} \mathrm{Cl}_{\mathbf{R}} \{ F^m(k,x) : m \geq n \}$$

where "$\mathrm{Cl}_{\mathbf{R}}\ \mathbf{X}$" denotes the topological closure in $\mathbf{R}$ of the set $\mathbf{X}$. In general, the choice of x is irrelevant to the definition of this cross-section, and introductory texts often disregard its value. However, more rigour can be introduced if we so desire. We can regard two choices of x as being 'as good as one another' from the viewpoint of this construction if they yield the same cross-section at k, a property which can be captured by defining a collection of equivalence relations (one for each value of k) by

$$x \sim_k y \quad \text{if and only if} \quad F_x(k) = F_y(k)$$

It's easy to see that in the 'non-chaotic' regions of the bifurcation diagram, each $\sim_k$ chops **R** into *finitely many* equivalence classes, almost all of which are themselves finite. The standard assumption that all x's are equivalent in these regions is thus a fairly good approximation to the truth, although not strictly correct.

Now let's look at a simple exercise in semantics. I'm rather keen on English folk music, and am fairly partial to coffee, but I don't particularly care about the order in which I indulge myself with these two vices. However, listening to folk music always starts me reminiscing, and my behaviour reverts to that of several years ago (a behaviour which has, of course, evolved into that which I display today). Ignoring the more mundane activities of the day, I can be described as a process which repeatedly listens to music and drinks coffee, *i.e.*

$$Mike = (music \rightarrow YoungMike) + (coffee \rightarrow Mike)$$

where "$a \rightarrow X$" means that after performing action a, I adopt the behaviour pattern X, and where "+" represents some form of non-deterministic choice between two concurrent activities. Obviously, the behaviour defined by this specification can only be fully described by 'unfolding' the definition *infinitely* many times. On the other hand, we can obtain as good an approximation as we like simply by unfolding it any sufficiently large *finite* number of times.

This is in some ways analogous to the "fractal-kit" construction of cross-sections of the bifurcation diagram given above, except that this time we're considering the iterated values of a function

$$g(k,x) = (music \rightarrow k) + (coffee \rightarrow x)$$

As before, x (*i.e. Mike*) is used dynamically in the specification, and k is the static part. So write $G^0(k,x) = x$ and $G^{n+1}(k,x) = g(k,G^n(k,x))$, choose your favourite order topology for the class of processes (several versions are available, depending on your choice of concurrent specification language), and denote this topological space **PROCESS**. As before, the behaviour(s) of *Mike* corresponding to choosing some initial description x of the 'dynamic behaviour', together with some 'static parameter' k, are just the processes in

$$G_x(k) = \bigcap_{n \in N} \mathrm{Cl}_{\mathbf{PROCESS}} \{ G^m(k,x) : m \geq n \}$$

[*An aside*: What we might call "standard" semantics (for example in CSP) corresponds to making the initial choice "$x = STOP$", where *STOP* is some process that does absolutely nothing for all time.]

There is clearly something similar happening in these two examples, and in all situations where a function is constructed by infinite iteration - but is the resemblance simply superficial? I suggest that it is not, and, what's more, this similarity can perhaps be exploited. In one sense, programming semantics are the very antithesis of chaos - we construct a semantics for a language to ensure that behaviours described in this language have a precise realisation, *i.e.* $G_x(k)$ is required to be a singleton; we would not usually appreciate a compiler which generates periodic or chaotic code. In dynamical terminology, we'd say that programming languages are designed to be 'linear'. But in physical terms, this is surely a crazy activity, since linear languages clearly cannot generate true representations of general non-linear systems. Consequently, a multitude of physically realisable processes are presumably theoretically uncomputable, and at best approximable only with considerable computational effort - a glaring inadequacy of current theory, and one which should perhaps no longer be tolerated.

Rather than build increasingly powerful linear computational systems, would it not be more sensible to introduce non-linear languages that can represent physical systems directly? Fractals offer us beauty, but they may also offer us the hope of a new generation of semantics - and a future in which the theory of computation takes its place as a true natural science.

Wittgenstien
THE FILM

"The world is everything that is the case" are the opening words of Wittgenstein's Tractatus Logico-Philosophicus. A fleeting allusion rather than a reference is made to this in the course of a conversation in Derek Jarman's film "Wittgenstein", along with many others casually dropped during the vigorous and sometimes fraught dialogue. I cannot help feeling that these unexplained linguistic pointers must be lost on anyone who is not at least a little familiar with Wittgenstein's work and life.

Why is a review of a film appearing in the FACS/FME Newsletter? Philosophy is an attempt to answer the question "what are we talking about?" when we conduct a human discourse. The formal semantics of programming languages is an attempt to answer the question "what are we writing about?" when we write computer programs, and formal methods an attempt to determine what we are doing when we develop software. Wittgenstein's philosophy in particular concentrated on the nature of language and its relation to "the world", if indeed such a concept can be dealt with meaningfully. This stand-point has, I believe, a particular affinity for computer scientists because our principal concerns stem from the conclusions which can be legitimately drawn from language scripts. What are the consequences of this program? How certain can we be of its properties?

As films go, "Wittgenstein" is brief: 75 minutes. At present it is showing in London and Oxford. The film is minimal in its production: the backdrops are all plain colours and the props amount to a couple of beds, a blackboard and a pillar box. Many of the ideas from the Tractatus, the Blue and Brown Books and On Certainty are played out in terse dialogues between Wittgenstein and his students, Maynard Keynes, Bertrand Russell and even a Martian. Some events in Wittgenstein's life are presented by allusion, others by a more direct dramatised narrative, the latter always portrayed through words, however, rather than actions - befitting the life of a linguistic philosopher no doubt! The only exception I remember is that where L. W. insists on playing a game with two of his friends, each simulating the sun, earth and moon in their orbital paths through space. An example of the more sparse kind of reference is a silent shot of a student carefully writing down Wittgenstein's words in a blue book; Wittgenstein angrily tells him to stop, but he does not. Wittgenstein was generally reluctant to commit his thoughts to print, but he dictated the Blue Book to his class of students in Cambridge in 1933-34.

Wittgenstein, boy and man, is engagingly played by Clancy Chassay and Karl Johnson. However, for my money, the outstanding portrayal is that of Bertrand Russell played by Michael Gough. Bertrand Russell figures significantly, as indeed he should, being the one who first encouraged Wittgenstein to study philosophy. Remembering radio and television interviews of Russell, I found his somewhat crusty mannerisms mixed with an almost childlike idealism accurately reflected.

The schematic nature of the film and the licence taken with some of the chronology strongly suggest a caricature. But Russell's autobiography records that he and Lady Ottoline Morrell were indeed lovers from 1910 to 1916, which would have been about the time that Wittgenstein was starting his philosophical career at Cambridge. Lady Ottoline Morrell's clothes are extravagant, striking, flamboyant. She appears perhaps twice, briefly.

This contrast emphasises the minimalism of the remainder of the set, the props, indeed the film.

The result of this sparse approach to the production is austere and abstract: a deliberate symbolic device; the film communicates its message through words rather than pictures, although it is not without its visual imagery. This gives a neat link to the final statement of the Tractatus, again the subject of a casually dropped verbal clue, "Whereof one cannot speak, thereof one must be silent".

At first I found the burlesque nature of the film and its licence with facts rather irritating. But as time passed and the more serious side of the playful symbolism emerged I began to enjoy it considerably. I recommend it!

Tim Denvir

References

Wittgenstein, L. Tractatus Logico-Philosophicus. Routledge and Kegan Paul, 1922, reprinted 1985 (and no doubt since). ISBN 0-7100-0962-3

The Blue and Brown Books. Basil Blackwell 1967, 1987. ISBN 0-631-14660-1

On Certainty. Basil Blackwell 1969, 1984. ISBN 0-631-16940-7

Philosophical Investigations. Basil Blackwell 1953, 1986. ISBN 0-631-14670-9

Bertrand Russell. The Autobiography of Bertrand Russell, Vols. I, II, III. Allen and Unwin, 1967, 1968, 1969 respectively.

Call for Participation

Formal Aspects of Object Oriented Systems

BCS FACS (Formal Aspects of Computing Science) Group
Christmas Meeting
Imperial College London,
December 16th and 17th, 1993.

The aims of this meeting will be to review recent work on:

1. the logical basis of Object Oriented structure.
2. formal support for Object Oriented system development.
3. application of Object Oriented structuring to the development of large scale specifications.
4. formal treatment of concurrency in Object Oriented systems.

There will be invited speakers on these topics, but there will also be an opportunity for contributions from workers in the field who would like to submit papers.
Contributions, in the form of abstracts of one or two pages A4, should be sent to one of the following by 15th October, 1993.

Prof. S.J. Goldsack,
Dept of Computing,
Imperial College of Science Technology and Medicine,
London SW7 2BZ.
tel. 071 589 5111
fax. 071 589 8024
e-mail: sjg@doc.ic.uk.

Stuart Kent,
University of Brighton,
Dept of Computing Science,
Watts Building,
Moulsecoomb,
Brighton BN2 4GJ.
tel. 0273 642451
fax. 0273 642405
e-mail: sjhk@unix.brighton.ac.uk
or sjk@doc.ic.ac.uk

It is hoped that a selection of the best papers may be published in a summary proceedings. If so complete versions of the selected papers would be requested at a later date. Please indicate if you would not wish your contribution to be considered for inclusion.

Notice of Meeting and Call for Papers

8th Z User Meeting – ZUM'94

Organized by the Z User Group in association with BCS FACS

29–30th June 1994

St. John's College, University of Cambridge, UK

Programme committee:

Rosalind Barden, Logica, Cambridge
Jonathan Bowen, Oxford University
Elspeth Cusack, BT
Neville Dean, Anglia Polytechnic Univ.
David Duce, Rutherford Appleton Lab.
Anthony Hall, Praxis plc
Brian Hepworth, British Aerospace
Howard Haughton, Lloyd's Register
Mike Hinchey, University of Cambridge
Darrell Ince, Open University
Jonathan Jacky, Univ. of Washington, USA
Peter Lupton, IBM Hursley
John McDermid, York University
Sylvio Meira, Univ. of Pernambuco, Brazil
John Nicholls, Oxford University
Gordon Rose, Univ. of Queensland, Australia
Chris Sennett, DRA Malvern
Sam Valentine, University of Brighton
Jim Woodcock, Oxford University
John Wordsworth, IBM Hursley

The programme committee invites authors to submit papers on or related to the formal specification notation Z in particular and formal methods in general for presentation at the next Z User Meeting and inclusion in the published Proceedings to be distributed at the meeting.

NOTE: THE SUBMISSION DEADLINE IS 1st OCTOBER 1993

Call for Papers

The committee of the Z User Group invites the submission of papers related to the interests of Z users. Special sessions on the following themes are planned if there is enough interest, and papers on these topics are especially encouraged:

- Industrial experiences
- Application of Z to safety-critical systems
- Projects and processes for formal methods – management and organizational issues
- Z and concurrency

Papers for presentation and publication will be reviewed and selected by the programme committee. The timetable for submitted papers is as follows:

Submission of draft paper:	1st October 1993
Notification of acceptance:	30th November 1993
Final copy for Proceedings:	31st January 1994
Z User Meeting in Cambridge:	29–30th June 1994

A maximum limit of 20 pages is requested. Industrial contributors may submit extended abstracts if they prefer. Please include four copies of your submission and indicate if you wish your paper to be considered for one of the special themes. The meeting will also include:

- Tool demonstrations
- Exhibitions by publishers
- Posters or leaflets

Associated tutorials could be held immediately before or after the meeting if appropriate proposals are submitted. Please contact the tutorial chair as soon as possible about all of the above. The following invited speakers are planned (* subject to confirmation):

David Garlan, Carnegie-Mellon University, USA:	Z and education*
Mike Gordon, University of Cambridge:	Z and HOL
Leslie Lamport, DEC Systems Research Center, USA:	Z and concurrency
Jim Woodcock, Oxford University:	Z and 00-56*
Robert Worden, Chairman of Logica Cambridge:	Z and industry

Maurice V. Wilkes, Olivetti Research (Emeritus Professor, University of Cambridge):
After dinner speaker on the occasion of the 45th anniversary of the EDSAC meeting (first European computer conference) held in Cambridge, June 1949, and hosted by him.

The meeting will be sponsored by BT, Logica and Praxis and is supported by the BCS FACS special interest group and the CEC ESPRIT **ProCoS-WG** 8694 Working Group.

General enquiries about the meeting and the Z User Group may be directed to:

Jonathan Bowen (Conference Chair)
Oxford University Computing Laboratory, 11 Keble Road, Oxford OX1 3QD, UK.
Email: `Jonathan.Bowen@comlab.ox.ac.uk` Tel: +44-865-272574 Fax: +44-865-273839

Submitted papers and extended abstracts should be sent to:

Anthony Hall (Programme Chair)
Praxis Systems plc, 20 Manvers Street, Bath BA1 1PX, UK.
Email: `jah@praxis.co.uk` Tel: +44-225-444700 Fax: +44-225-465205

Proposals for tutorials, tool demonstrations, publishers' stands, and requests for information concerning local arrangements should be sent to:

Mike Hinchey (Tutorial Chair)
University of Cambridge, Computer Laboratory
New Museums Site, Pembroke Street, Cambridge CB2 3QG, UK.
Email: `Michael.Hinchey@cl.cam.ac.uk` Tel: +44-223-334419 Fax: +44-223-334678

Until beginning of October 1993:
DEC Systems Research Center, Palo Alto, CA 94301, USA
Email: `hinchey@src.dec.com`

Call for Papers

Z User Meeting 1994 – Education Half Day

A special session on Educational Issues relating to Formal Methods (Z in particular) is being organized for the Friday morning (1 July 1994) after the main Z User Meeting 1994 to be held in Cambridge (29–30th June 1994).

Submissions are now invited for papers and posters to be presented at the Education Half-Day; they should cover topics in teaching, learning and understanding formal methods (not specifically Z) both in academia and in industry.

Papers **may** be published in the Proceedings of the Z User Meeting, provided they are of sufficiently high standard and conform to the guidelines for papers to be presented at the main session. In particular they should be submitted to the main organizing committee for the Z User Meeting by 1st October 1993 (as previously announced); please mark that the paper is to be considered for the Education Session.

Alternatively, papers (or extended abstracts of no more than four pages) should be submitted by 31st December 1993 directly to:

Neville Dean,
Anglia Polytechnic University,
Applied Sciences,
Cambridge CB1 1PT, UK.
Email: cdean@va.anglia.ac.uk Tel: +44-223-352992 ext 2329 Fax: +44-223-352979

Authors will be notified by 28th February 1994 if their paper has been accepted. Final versions of accepted papers will need to be received by 31st March 1994. **Note that papers so submitted will not be included in the published proceedings.**

Proposals for posters should take the form of an abstract (no more than 500 words) and be sent to Neville Dean by 28th February 1994. Successful authors will be notified by 31 March 1994.

FORTHCOMING EVENTS

1993

September 26–October 1 OOPSLA'93
Conference on Object Oriented Programming Systems Languages and Applications,
Washington, DC., USA. *Sponsor:* SIGPLAN. *Contact:* Timlynn Babitsky, JFS Consulting, 5 Wise Ferry Ct., Lexington, SC 29072, USA; Tel: +1 (803) 957-5779.

September 27–30
Conference on Software Maintenance '93,
Montreal, Quebec, Canada. *Contact:* Marc Kellner, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15213-3890, USA. Tel: +1 (412) 268 7721; Fax: +1 (412) 268 5758; Email: mik@sei.cmu.edu

October 6–8
12th Symposium on Reliable Distributed Systems,
Princeton, NJ, USA. *Contact:* Prof. David Taylor, Department of Computer Science, University of Waterloo, Waterloo, Ontario, CANADA N2L 3G1. Tel: +1 (519) 888-4432; Email: dtaylor@grand.uwaterloo.ca

October 19–22 PNPM'93
Fifth International Workshop on Petri Nets and Performance Models,
Toulouse, France. *Contact:* G. Juanole, LAAS-CNRS, 7, Avenue du Colonel Roche, 31077 Toulouse Cedex, France; Fax: +33-61-336411; Email: juanole@laas.fr

October 25–27 SoSL
International Workshop on Semantics of Specification Languages,
Utrecht, The Netherlands. *Contact:* Annemarie Besselink, Dept of Philosophy, University of Utrecht, PO Box 80126, 3508 TC Utrecht, The Netherlands. Tel: +31 03 53 18 31; Fax:+31 30 53 28 16; Email: Annemarie.Besselink@phil.ruu.nl

October 26–29 ILPS'93
International Logic Programming Symposium,
Vancouver, Canada. *Contact:* Dale Miller, Department of Computer Science, 200 South 33rd Street, University of Pennsylvanis, Philadelphia, PA 19104-6389, USA; Fax: +1 (215) 898 0587; Email: dale@saui.cis.upenn.edu

October 26–29 Forte'93
6th International Conference on Formal Description Techniques,
Boston, MA. *Sponsor:* IFIP WG6.1. *Contact:* Richard L. Tenney, Math & Comp. Sci., Univ. of Massachusetts, Boston, MA 02125-3393. Email: rlt@cs.umb.edu

November 3–6 ISSRE 93
Fourth International Symposium on Software Reliability Engineering,
Denver. *Cosponsors:* IEEE Computer Soc. Technical Committee on Software Eng., IEEE Reliability Soc. Denver Chapter. *Contact:* Anneliese von Mayrhauser, Computer Science Dept., Colorado State Univ., Ft. Collins, CO 80523, USA; Tel: +1 (303) 491-7016; Fax: +1 (303) 491-6639. Email: avm@cs.colostate.edu. **Or** Yoshihiro Tohma, Computer Science Dept., Tokyo Inst. of Technology, 2-12-1 Oakayama Meguro-ku, Tokyo 152, Japan; Tel: +81 (3) 3726-1111, ext. 2566; Email: tohma@cs.titech.ac.jp

December 1–3
14th IEEE Real-Time Systems Symposium,
Durham, N.C. *Sponsor:* IEEE Computer Soc. TC on Real-Time Systems. *Contact:* Farnam Jahanian, IBM T.J. Watson Research Ctr., PO Box 704, Yorktown Heights, NY 10598; Tel: 784-7498; Email: farnam@watson.ibm.com

December 1–4
Fifth IEEE Symposium on Parallel and Distributed Processing,
Dallas, TX, USA. *Contact:* Prasenjit Biswas, Cyrix, 2703 N. Central Expressway, Richardson, TX 75080; Tel: +1 (214) 234-8388; Fax: +1 (214) 699-9857.

December 6–7 IWSSD-7
Seventh International Workshop on Software Specification and Design,
Los Angeles area, CA, USA. *Sponsor:* IEEE Computer Society. *Contact:* Jack Wileden, Computer Science Department, University of Massachusetts, Amherst MA 01003, USA; Email: jack@cs.umass.edu

December 7–10
Symposium on the Foundations of Software Engineering,
Los Angeles, CA, USA. *Sponsor:* ACM SIGSOFT. *Contact:* Barry Boehm, Computer Science Department, University of Southern California, Los Angeles CA 90089, USA; Tel: +1 (213) 740-8163; Email: boehm@cs.usc.edu

December 15–17 FSTTCS'93
Thirteenth Conference on the Foundations of Software Technology and Theoretical Computer Science,
Bombay, India. *Contact:* Prof. R.K. Shyamasundar, FST&TCS'13, Tata Institute of Fundamental Research, Bombay 400 005, India; Fax: +91-22-215-2181. Email: fsttcs@tifrvax.bitnet

December 16–17 FACS Christmas Workshop
Formal Aspects of Object Oriented Systems,
Imperial College, University of London, UK. *Sponsor:* BCS-FACS. *Contact:* Prof. Stephen Goldsack, Imperial College; Tel: +44 (71) 589 5111; Fax: +44 (71) 589 8024; Email: sjg@cod.ic.ac.uk or Stuart Kent, University of Brighton, UK; Tel: +44 (273) 642451; Fax: +44 (273) 642405; Email: sjhk@unix.brighton.ac.uk or sjk@doc.ic.ac.uk

1994

January 5–7
The Sixth FACS Refinement Workshop on Theory and Practice of Formal Software Development,
London, UK. *Contact:* Dr Jeremy Jacob, Dept of Computer Science, University of York, Heslington, York YO1 5DD, UK; Tel: +44 (904) 432720; Email: jeremy@minster.york.ac.uk or David Till, Dept of Computer Science, City University,Northampton Square, London EC1V 0HB, UK; Tel: +44 (71) 477 8552; Email: till@cs.city.ac.uk

January 5–7 SEI CSEEE,
7th SEI Conference on Software Engineering Education,
San Antonio, Texas. *Contact:* Dr Jorge L. Diaz-Herrera, SEI, Carnegie-Mellon Univ., Pittsburgh, PA 15213-3890, USA; Tel: +1 (412) 268-7636; Fax: +1 (412) 268-5758; Email: jldh@sei.cmu.edu

January 17–219 POPL'94,
The 21st Annual Symposium on Principles Of Programming Languages,
Portland, OR, USA.*Sponsors:* ACM SIGPLAN-SIGACT *Contact:* Hans-J. Boehm, Xerox Corporation, Palo Alto Research Ctr., 3333 Coyote Hill Rd., Palo Alto, CA 94304 USA; Email: boehm@parc.xerox.com

February 24–26 STACS94,
11th Symposium on Theoretical Aspects of Computer Science,
Caen, France. *Contact:* Prof. Patrice Enjalbert, L.A.I.A.C. STACS94, Universite de Caen, F-14032 Caen Cedex, France; Tel: +33-31-45-56-16; Fax: +33-31-45-58-14. Email: stacs@univ-caen.fr

April 7–9 CC94,
International Conference on Compiler Construction,
Edinburgh, Scotland. *Contact:* Peter Fritzson, CC94, Department of Computer and Information Science, Linköping University, S-581 83 Linköping, Sweden; Tel: +46-13-281484; Fax: +46-13-282666; Email: petfr@ida.liu.se

April 11–13 CAAP94,
Colloqium on Trees in Algebra and Programming,
Edinburgh, Scotland. *Contact:* Sophie Tison, CAAP'94, University of Lille 1, LIFL, Bat. M3, F-59655 Villeneuve d'Ascq Cedex, France; Tel: +33-20434309; Fax: +33-20436566; Email: tison@lifl.fr

April 11–13 ESOP94,
European Symposium on Programming,
Edinburgh, Scotland. *Contact:* Don Sannella, ESOP'94, Laboratory for Foundations of Computer Science, Department of Computer Science, The King's Buildings, University of Edinburgh, Edinburgh EH9 3JZ, Scotland; Tel: +44 (031) 6505184; Fax: +44 (031) 6677209; Email: dts@dcs.ed.ac.uk

April 18–22
International Conference on Requirements Enginering,
Colorado Springs, CO, USA. *Sponsor:* IEEE-CS TC-SE. *Contact:* Alan Davis, Univ. of Colorado,Center for SE, 1867 Austin Bluffs Pkwy, Ste 200, PO box 7150, Colorado Springs, CO 80933-7150; Tel: +1 (719) 593-3695; Email: adavis@zeppo.uccs.edu

April 19–21 TACS'94
International ymposium on Theoretical Aspects of Computer Software, Tohoku University, Sendai, Japan. Email: tacs94@ito.ecei.tohoku.ac.jp

May 16–19 ICCL'94
5th International Conference on Computer Languages,

Toulouse, France. *Contact:* Henri E Bal, Vrije University, MCS Dept., De Boelelaan 10Sia, 1081 HV Amsterdam, The Netherlands. Tel: +31 (20) 548-5574; Fax: +31 (20) 642-7707; Email: bal@cs.vu.nl

May 16-21 ICSE
16th International Conference on Software Engineering,
Sorrento, Italy. *Sponsors:* IEEE-CS, ACM Sigsoft, AICA. *Contact:* Bruno Fadini, Dept "Informatica e Sistemistica", University of Naples "Federico II", Via Claudio 211-80125 Napoli, Italy. Tel: +39 81 768 3193; Fax: +39 81 768 3186; Email: fadini@vm.cised.unina.it

May 23–25 STOC
26th Symposium on Theory of Computing,
Montreal, Canada. *Contact:* Micheal Goodrich, Dept of Computer Science, Johns Hopkins University, Baltimore, MD 21218-2694, USA.

June 8–10
Design, Specification, Verification of Interactive Systems,
Pisa, Italy. *Sponsor:* Eurographics *Contact:* Dott. Fabio Paterno', CNUCE-C.N.R., Via S.Maria 36, 56126 Pisa, Italy; Fax: +39 50 589354; Email: paterno@vm.cnuce.cnr.it

June 13–17 PARLE'94
Parallel Architectures and Languages Europe,
Athens, Greece. *Contact:* Parle/CTI, Computer Technology Institute, 3 Kolokotroni Str., 262 21 Patras, Greece. Tel: +30 (61) 220 112; Fax: +30 (61) 222 086; Email: parle@cti.gr

June 15–17 FTCS-24
24th Annual International Symposium on Fault-Tolerant Computing, Austin, Texas, USA. *Sponsor:* IEEE cs.*Contact:* Miroslaw Malek, Dept. Electrical and Computer Engineering, University of Texas at Austin, TX 78712-1084, USA. Tel: +1 (512) 471 5704, Fax: +1 (512) 471 0954, Email: malek@emx.cc.utexas.edu

June 29–30 ZUM94
8th Z User Group Meeting
Cambridge, England. *Sponsor:* Z User Group and FACS. *Contact:* Jonathan Bowen, Oxford University Computing Laboratory, 11 Keble Road, Oxford OX1 3QD, UK. Tel: +44 (865) 272 574; Fax:+44 (865) 273 839; Email: Jonathan.Bowen@comlab.ox.ac.uk

July 4–7 LICS'94
9th IEEE Symposium on Logic in Computer Science,
Paris, France. *Sponsor:* IEEE TC-MFC, *Cosponsors:* ASL and EATCS. *Contact:* Amy Felty and Douglas Howe, AT&T Bell Labs, 600 Mountain Avenue, Murray Hill, NJ 07974, USA. Email: felty,howe@research.att.com

July 10–15 ICALP'94
21st International Colloquim on Automata, Languages, and Programming,
Jerusalem, Israel. *Contact:* E. Shamir, Department of Computer Science, Hebrew University of Jerusalem, Jerusalem 91904, Israel; Fax: +972 2630702. Email: shamir@cs.huji.ac.il

September 21–22 ISOOMS
International Symposium on Object Oiented Methodologies and Systems, Palermo, Italy. *Sponsor:* AICA *Contact:* Elisa Bertino, Dipartimento di Scienze dell'Informzione, Universita di Milano, Via Comelico, 39, Milano, Italy. Tel: +39-2-55006227; Fax +39-2-55006253; Email:bertino@disi.unige.it or Susan Urban, Computr Science Department, Arizona State University, Temple 85287-5406, USA; Tel: +1 (602) 9652784; Fax: +1 (602) 9652751; Email: urban@asuvax.eas.asu.edu

October 16–28 OOPSLA'94
Conference on Object Orientated Programming Systems, Languages and Applications.
Portland OR, USA. *Sponsor:* SIGPLAN. *Contact:* John T Richardson, IBM TJ Watson Research Center, H1-B50, PO Box 704, Yorktown Heights, NY 10598, USA; Tel: +1 (914) 784-7616; Email: jtr@watson.ibm.com

Guidelines for Newsletter Contributions

Contributions may be in the form of **single-sided** camera-ready copy, suitable for layout and sub-editing. They can also be sent to us using electronic media (i.e. by floppy disk (MS DOS or Mac)/e-mail/etc.), to be formatted in the house style. As a rule, we generally accept pure ASCII text or TEX/LATEX in order to avoid complications involving interchange between wordprocessing formats. We regret that we are unable to offer typesetting facilities for handwritten material.

If contributions are sent using proprietary wordprocessor/markup language formats (i.e. MicroSoft Word 5, FrameMaker), then these will be treated as though they were camera-ready copy. If we are unable to print them adequately or to otherwise convert to another more suitable form then the authors may be asked to provide paper copies of appropriate reproduction quality.

Artwork can be provided for appropriate inclusion, either using general formats (such as DVI files or Encapsulated PostScript) by sending camera-ready paper copy. Generally, line drawings and other high-contrast graphical diagrams will be acceptable.

Material must be of adequate quality for reproduction. Output from high quality printers with at least 300 DPI resolution is generally acceptable. Output from printers with lesser resolution (i.e. dot-matrix printers) tends not to reproduce very well and will not be of sufficiently good print quality. The Editorial Panel reserves the right to refuse publication for contributions which cannot be reproduced adequately.

Page definition information

If possible, contributions should be designed to fit standard A4 paper size, leaving a margin of at least one inch (1") on all sides. Camera ready copy should be sent in **single-sided** format, with page numbers written lightly **on the back**. Ideally, all fount sizes used should be no smaller than 10pt for clarity. Contributions should attempt to make adequate use of the space, filling at least 60% of each page, including the last one. Authors should note that all contributions may be sub-edited appropriately to make efficient use of space.

Deadlines

The production deadlines for the coming year are:

Summer	end of May, 1993	**Winter**	end of November, 1993
Autumn	end of August, 1993	**Spring**	end of February, 1994

Disclaimer

The views and opinions expressed within articles included in the FACS Europe newsletter are the responsibility of the authors concerned and do not necessarily represent the opinions or views of the editorial panel.

Addresses

Editors:

Dr. Jawed Siddiqi
Dept. of Computing and Management Sciences
Sheffield Hallam University
100 Napier Street
Sheffield, S11 8HD
United Kingdom

Tel: +44 742 533141
E-mail: J.I.Siddiqi@shu.ac.uk

Dr. Brian Monahan
Dept. of Computer Science
University of Manchester
Oxford Road
Manchester, M13 9PL
United Kingdom

Tel: +44 61 275 6137
E-mail: brianm@cs.man.ac.uk

BCS FACS Committee 1992/93

General

General enquiries about the BCS FACS group, the newsletter or its meetings can be made to:

BCS FACS
Department of Computer Studies
Loughborough University of Technology
Loughborough, Leicestershire
LE11 3TU
Tel: +44 509 222676
Fax: +44 509 211586
E-mail: FACS@lut.ac.uk

Membership fees 1993

Standard (i.e. non-BCS members) : £25
BCS members : £10

Discount subscription rates 1993

EATCS : £10
FACS Journal : £33 (6 issues, Vol. 5)

Officers

Chair	Tim Denvir
Treasurer	Roger Stone
Committee Secretary	Richard Mitchell
Membership Secretary	John Cooke
Newsletter Editors	Jawed Siddiqi & Brian Monahan
Publicity	Brian Monahan
BCS SIG representative	David Blyth
BCS SE TC representative	John Boarder (Roger Shaw)
Liaison with FACS Journal	John Cooke
Liaison with BCS FMIS group	Ann Wrightson

Committee Members

Name	Affiliation	Tel:	E-mail
R. Barden	Logica Cambridge Ltd	0223-66343	rosalind@logcam.co.uk
D. Blyth	Incord Ltd.	0202-896834	DBlyth@cix.compulink.co.uk
J. Boarder	Buckinhamshire	0494-22141	jcb@buckscol.ac.uk
D.J. Cooke	Loughborough	0509-222676	D.J.Cooke@lut.ac.uk
B.T. Denvir	Translimina Ltd.	081-882 5853	timdenvir@cix.compulink.co.uk
S.J. Goldsack	Imperial	071-589-5111x5099	sig@ic.doc.ac.uk
A.J.J. Dick	Bull Research		J.Dick@brno.uk03.bull.co.uk
R.B. Jones	ICL Winnersh	0734-693131x6536	
R.J. Mitchell	Brighton	0273-642458	rjm4@unix.brighton.ac.uk
B.Q. Monahan	Manchester	061-275-6137	brianm@cs.man.ac.uk
A. Norcliffe	Sheffield Hallam	0742-720911x2473	A.Norcliffe@scp.ac.uk
R.C.F. Shaw	Lloyd's Register	081-681-4040x4818	Roger.Shaw@aie.lreg.co.uk
J.I.A. Siddiqi	Sheffield Hallam	0742-533141	J.I.Siddiqi@shu.ac.uk
D. Simpson	Brighton	0273-600900x2450	ds33@unix.bton.ac.uk
R.G. Stone	Loughborough	0509-222686	R.G.Stone@lut.ac.uk
D.R. Till	City	071-477-8552	till@cs.city.ac.uk
A. Wrightson	Central Lancashire	0772-893242	annw@sc.uclan.ac.uk

FME Contact

Name	Affiliation	Tel:	E-mail
M. Thomas	Praxis plc	0225-444700	mct@praxis.co.uka