RAL-93-100

Science and Engineering Research Council
Rutherford Appleton Laboratory
Chilton DIDCOT Oxon OX11 0QX

RAL-93-100

Invariants, Frames and Postconditions: a Comparison of Two Formal Specification Notations

J Bicarregui and B Ritchie

December 1993

Invariants, Frames and Postconditions: a Comparison of Two Formal Specification Notations†

Juan Bicarregui
Brian Ritchie
Systems Engineering Division
SERC Rutherford Appleton Laboratory
Oxon OX11 0QZ, UK

Abstract

VDM and B are two "model-oriented" formal methods. Each gives a notation for the specification of systems as state machines in terms of a set of states with operations defined as relations on that set. Each has a notion of refinement of data and operations based on the principles of reduction of non-determinism and increase in definedness.

This paper makes a comparison of the two notations through an example of a communications protocol previously formalised in [BA91]. Two abstractions and two reifications of the original specification are given.

Particular attention is paid to three areas where the notations differ: to the use of postconditions that assume the invariant as opposed to postconditions that enforce it; to the explicit "framing" of operations as opposed to the "minimal frame" approach; and to the use of relational postconditions as opposed to generalised substitutions.

1 Introduction

In [BA92], Bruns and Anderson describe a communications protocol in CCS with value-passing. A data model for the values is given which is, in effect, a model of the state of the device. This model is defined in terms of the usual data constructors of model-oriented specification, but without the use of invariants.

The part of the protocol described is a mechanism for manipulating a series of flags that indicate the status of some shared-memory buffers. These flags are used to ensure that there is no "data-tearing" as multiple processors simultaneously read and write to the buffers. For the operations

†An abriged version of this paper was presented at the FME'93 Symposium [BR93].

that update these flags, semaphores are used to ensure that each operation has uninterrupted access to the flags. Thus this part of the behaviour can be described as a purely sequential system.

This paper considers some alternative data-models for the specification (and reification) of these status flags. In particular, attention is paid to the use of invariants in the data model and frames of reference in the operations definitions, neither of which are available in the data modelling language of [BA92]. It is argued that these features can play a key role in describing the system in a "natural" fashion and can thus help to deepen our understanding of the model.

VDM [Jones90] and B [Abrial92b] are used for the analysis, and particular attention is paid to some areas where the notation differ: to the use of postconditions that assume the invariant as opposed to postconditions that enforce it; to the explicit "framing" of operations as opposed to the "minimal frame" approach; to the use of relational postconditions as opposed to generalised substitutions. In this small example, there is little scope for the effective use of structuring of specifications that is one of the major features of the B method. Familiarity with the basic concepts and notation of VDM and B is assumed.

The remainder of this first section is an informal description of the application and desired protocol. The second through fifth sections present the development in VDM. Section two presents a formal specification of the system at a level of abstraction similar to the "abstract" description of [BA92]. Motivated by an analysis of the invariant of that specification, section three describes two further abstractions that can be made. Section four provides an alternative model of the system that makes it possible to write more useful framing information about the operations, the fifth section extends this model to the "improved" protocol of [BA92]. The sixth section considers the development again using B, presenting those elements of the development that highlight the differences in the notations. The last section is a discussion of some of the points arising from the example and their treatment in the two notations.

1.1 The Multiprocessor Shared-Memory Information Exchange (MSMIE)

MSMIE, Multiprocessor Shared-Memory Information Exchange, is a protocol that addresses intra-subsystem communications with "several features which make it ideally suited to inter-processor communications in distributed, microprocessor-based nuclear safety systems" [MSMIE]. It has been used in the embedded software of Westinghouse nuclear systems designs.

The protocol uses multiple buffering to ensure that no "data-tearing" occurs, that is, it ensures that data is never overwritten by one process whilst it is being read by another. One important requirement is that neither writing nor reading processes should have to wait for a buffer to become available; another is that "recent" information should be passed, via the buffers, from writers to readers. In the simplification considered in [BA92] it is assumed that information is being passed from a single writing "slave" processor, to several reading "master" processors.

The information exchange is realised by a system with three buffers. Very roughly, at any time, one buffer is available for writing, one for reading and the third is either in between a write and

a read and hence contains the most recently written information, or between a read and a write and so is idle.

The status of each buffer is recorded by a flag which can take one of four values:

- s -"assigned to slave." This buffer is reserved for writing, it may actually be being written at the moment or just marked as available for writing.
- n -"newest." This buffer has just been written and contains the latest information. It is not being read at the moment.
- m -"assigned to master." This buffer is being read by one or more processors.
- i -"idle." This buffer is idle, not being read or written and not containing the latest data.

The names of the master processors that are currently reading are also stored in the state.

As mentioned earlier, neither the slave and master processors that access the buffers in parallel nor the actual transference of data are modelled here. This analysis concerns only the operations that modify the buffer status flags. These operations are protected by a system of semaphores which allow each operation uninterrupted access to the state and thus their behaviour is purely sequential.

There are three of these operations:

- *slave* This operation is executed when a write finishes. *slave* sets the status of the buffer that was being written to "newest" thus replacing any other buffer with this status.
- *acquire* This is executed when a read begins. The new reader name (passed as a parameter) is added to the set of readers and status flags are updated as appropriate.
- *release* Executed when a read ends, this removes a reader from the set and updates flags as appropriate.

The details of the behaviour of these operations are quite intricate and their precise description is left to the formal specification in the following section.

It should be noted however that, as it stands, the protocol could have the undesirable property that information flow from slave to master can be held up indefinitely. This possibility is ruled out in the original system [MSMIE] via timing constraints whereas [BA92] suggests an improvement to the protocol (using a fourth buffer) that eliminates the possibility without recourse to timing arguments. This improved protocol is also examined in later sections.

2 A VDM specification of MSMIE

The state in [BA92] is defined as

> "a set of three pairs (a, l) where a is the buffer status, drawn from $\{\mathsf{i}, \mathsf{s}, \mathsf{n}, \mathsf{m}\}$, and l is the buffer identification, drawn from $\{1, 2, 3\}$. The buffers are given as a set rather than a tuple to enable pattern matching rules in the description of the protocol."

The pattern matching rules do indeed give a concise description of the transitions of the system, in particular, the associative and commutative properties of sets are used to good effect in order to avoid much repetitive case analysis. However, the present authors found that considerable effort was required to check that the patterns given were exhaustive and that the effects of overlaps between patterns were sensible. This difficulty is exacerbated by the fact that many of the states in the model are unreachable but no invariant on the state type is given to exclude them.

The specification given here makes the choice of a sequence of three buffers for the state description. In addition, an invariant is used to exclude unwanted values from the state type.

2.1 The state

Possible values of the status flags are given via an enumerated type; the type of the names of master (reading) processors is deferred.

types

$Status = \{\mathsf{s}, \mathsf{m}, \mathsf{n}, \mathsf{i}\}$

$MName = \mathsf{token}$

The state is composed of three buffer status flags and a set of the names of the currently reading masters. The invariant captures the fact that only certain states are reachable by the operations. It gives restrictions as to the possible combinations of status flags, namely that there is always exactly one buffer assigned to the writing slave; there is at most one currently being read and at most one with newest data that is not being read; and the set of reader names is empty precisely when there is no buffer being read. The initial state assigns one buffer to the slave and records that the other two buffers are idle.

state Σ of
b : $Status^*$
ms : $MName$-set
inv $mk\text{-}\Sigma(b, ms)$ $\triangleq$ len $b = 3 \wedge$
$count(\mathsf{s}, b) = 1 \wedge$
$count(\mathsf{m}, b) \in \{0, 1\} \wedge$
$count(\mathsf{n}, b) \in \{0, 1\} \wedge$
$(count(\mathsf{m}, b) = 0 \Leftrightarrow ms = \{\})$
init $mk\text{-}\Sigma(b, ms)$ $\triangleq$ $b = [\mathsf{s}, \mathsf{i}, \mathsf{i}] \wedge ms = \{\}$
end

where[1]

$count : Status \times Status^* \to \mathbb{N}$

$count(status, l)$ $\triangleq$ len $(l \rhd status)$

A validation condition on the state

We observe that only four combinations of buffers are allowed by the invariant:

$$\forall mk\text{-}\Sigma(b, ms): \Sigma \cdot \{b(1), b(2), b(3)\}_m \in \{\{\mathsf{s},\mathsf{i},\mathsf{i}\}_m, \{\mathsf{s},\mathsf{i},\mathsf{n}\}_m, \{\mathsf{s},\mathsf{i},\mathsf{m}\}_m, \{\mathsf{s},\mathsf{n},\mathsf{m}\}_m\}$$

where we have used $\{\ldots\}_m$ as a notation for bags (multisets), for example $\{\mathsf{s},\mathsf{i},\mathsf{i}\}_m$ is the bag containing one 's' and two 'i's.

Thus the invariant has captured, and brought to the fore, properties that would otherwise have to be deduced by looking in detail at the definitions of the operations. It makes it possible to build quickly our intuition of the workings of the specified machine. We know immediately that there is always one buffer reserved for writing, at most one being read, and at most one with newest data not being read.

2.2 The Operations

Slave

The first operation, *slave*, is executed when a write completes. It reassigns the status of the buffer just written, previously s, to n, thus replacing any other n buffer. It also non-deterministically

[1]Here, range restriction is used on sequences, viewing them as maps from natural numbers to elements.

chooses another available buffer which is to be the new buffer reserved for writing and assigns to it status s.

slave ()
ext wr b : *Status**
pre true
post $\forall i \in \{1,2,3\} \cdot$
$(\overleftarrow{b}(i) = \mathsf{s} \Rightarrow b(i) = \mathsf{n}) \wedge$
$(\overleftarrow{b}(i) = \mathsf{m} \Rightarrow b(i) = \mathsf{m})$

The postcondition may, at first sight, seem to be to liberal: what should happen to any buffer that had status n or i?. However, in conjunction with the invariant and the frame, it ensures that no other n buffer remains, that exactly one new s buffer is chosen, and that no new m buffers are added. Thus for example we can write the following validation property for *slave* which can be proved in order to increase confidence in the correctness of the postcondition:

$$\overleftarrow{b}(i) \in \{\mathsf{n},\mathsf{i}\} \Rightarrow b(i) \in \{\mathsf{i},\mathsf{s}\}$$

Note that all three implications could have been equivalences without changing the operation.

Acquire

The second operation, acquire, is executed when a read is about to start. It adds the new reader's name, passed as a parameter, to the record of active readers and reassigns status flags as necessary.

If there is a buffer currently being read then the new read also begins to read that same buffer and no status change is required. Otherwise the new read starts on the buffer with newest data, status n, and reassigns the status of that buffer to m.

The operation can only be executed in these two situations and this information is recorded in the precondition which requires that there is either a status m or status n buffer. The precondition also records the fact that the operation is only required to function when the new reader is not already in the set of readers.

Note that, in selecting which buffer is to be read, it is not always possible to choose the buffer with newest data. This situation occurs when there are currently buffers with both status m and n, which arises when the data in the n buffer has become available since the start of an ongoing read, that is, when there has been a *slave* since an *acquire* for which there has not yet been a corresponding *release*. In this situation, were the new master to begin reading the n buffer, there would then be two buffers reserved for reading. Consequently, should another *slave* now occur, attempting to preserve this new data would leave no buffer being available for another write to start, thus contradicting one of the fundamental requirements of the protocol: that processors

should never have to wait to gain access to buffers. The invariant is designed to prevent this possibility, by insisting that there is always one (and precisely one) buffer with status s.

$acq\ (l\colon MName)$
ext wr b : $Status^*$
wr ms : $MName$-set
pre $l \notin ms \wedge$
$\exists i \in \{1,2,3\} \cdot b(i) = \mathsf{n} \vee b(i) = \mathsf{m}$
post $ms = \overleftarrow{ms} \cup \{l\} \wedge$
$\forall i \in \{1,2,3\} \cdot$
if $\overleftarrow{b}(i) = \mathsf{n} \wedge \overleftarrow{ms} = \{\}$ then $b(i) = \mathsf{m}$ else $b(i) = \overleftarrow{b}(i)$

It is worth observing that the last line of the postcondition could have been written as

if $\overleftarrow{b}(i) = \mathsf{n}$ then $b(i) \in \{\mathsf{n}, \mathsf{m}\}$ else $b(i) = \overleftarrow{b}(i)$

or simply as

$\overleftarrow{b}(i) \neq \mathsf{n} \wedge ms \neq \{\} \Rightarrow b(i) = \overleftarrow{b}(i).$

The apparent non-determinism in the alternatives is illusory as the invariant will ensure that there is no real choice as to what status to assign to any buffer that previously had status n. However, the longer and apparently stronger postcondition is preferred as the shorter versions seem to be more cryptic.

Release

The release operation is executed when a reading, master processor finishes its read. The name of the processor is removed from the set of readers and again, status flags reassigned as required.

If this master is not the last one currently reading, then no change is required to the status flags. However, if this is the last master currently reading the m buffer, then this buffer must have its flag reassigned. There are two possibilities. On the one hand, should there be another buffer with status n available at this time, that is if a write has been completed since the current "chain of reads" began on this buffer, then the m buffer no longer contains the most recent data and so should now be set to i. On the other hand, if there has been no write since the chain of reads began, and hence there is no n buffer available, the m buffer contains the most recent data and its status should be reset to n.

$rel\ (l\colon MName)$
ext wr b : $Status^*$
wr ms : $MName$-set
pre $l \in ms$
post $ms = \overleftarrow{ms} - \{l\} \wedge$
$\forall i \in \{1,2,3\} \cdot$
if $ms = \{\} \wedge \overleftarrow{b}(i) = \mathsf{m}$
then $b(i) \in \{\mathsf{n},\mathsf{i}\} \wedge count(\mathsf{n}, b) = 1$
else $b(i) = \overleftarrow{b}(i)$

Again there is some choice as to how much of the information that is deducible from the invariant should be made explicit in the postcondition. For example the first conjunct of the 'then' clause $b(i) \in \{\mathsf{n},\mathsf{i}\}$ could have been omitted as no other possibilities are permitted by the invariant, or alternatively, the whole 'then' clause could be replaced by a more explicit form

if $\exists \mathsf{j} \in \{1,2,3\} \cdot \overleftarrow{b}(j) = \mathsf{n}$ then $b(i) = \mathsf{i}$ else $b(i) = \mathsf{n}$

It is debatable which gives the clearer specification.

This specification has given a fairly algorithmic description of which buffers are assigned to what status by each operation. This is a good level of abstraction at which to reason about whole system safety properties such as the freshness of the data transferred from slave to masters which is the focus of [BA92]. Much of the detail of this specification, however, is undesirable clutter for other purposes and it is interesting to give more "external" views of the system, as is done in the next section.

3 Two more-abstract specifications

In this section we give two formal abstractions of the above specification. The new specifications maintain the same external behaviour, however the abstract states are progressively simpler than the one just given. The abstractions arise by ignoring detail in the state model that is unnecessary to capture the external behaviour. Retrieve functions from concrete to abstract states are also given which are many-to-one thus demonstrating "implementation bias" in the concrete specification.

As it is usual to give more concrete specifications successively higher numbers, from now on we will use Σ_2 to refer to the state of the specification given earlier.

3.1 A first abstraction: ignoring the identity of buffers

Taking inspiration from the validation condition on the state of the above specification, we can give a more abstract specification where, rather than explicitly giving the status of each individual buffer, the state only records which of the four possible *combinations* of buffer the machine is in.

types

$Status_1 = \{\mathsf{sii}, \mathsf{sin}, \mathsf{sim}, \mathsf{snm}\}$

state Σ_1 of
  bs : $Status_1$
  ms : $MName$-set
inv $mk\text{-}\Sigma_1(bs, ms) \triangleq ms = \{\} \Leftrightarrow bs \in \{\mathsf{sii}, \mathsf{sin}\}$
init $mk\text{-}\Sigma_1(bs, ms) \triangleq bs = \mathsf{sii} \wedge ms = \{\}$
end

operations

$slave\ ()$
ext wr bs : $Status_1$
pre true
post $(\overleftarrow{bs} \in \{\mathsf{sii}, \mathsf{sin}\} \Rightarrow bs = \mathsf{sin}) \wedge$
  $(\overleftarrow{bs} \in \{\mathsf{sim}, \mathsf{snm}\} \Rightarrow bs = \mathsf{snm})$

As in the earlier specification of *slave*, there is no change to the readers of the m buffer, thus there is no need to access ms.

$acq\ (l: MName)$
ext wr bs : $Status_1$
  wr ms : $MName$-set
pre $l \notin ms \wedge bs \neq \mathsf{sii}$
post $ms = \overleftarrow{ms} \cup \{l\} \wedge$
  if $\overleftarrow{ms} = \{\}$ then $bs = \mathsf{sim}$ else $bs = \overleftarrow{bs}$

$rel\ (l: MName)$
ext wr bs : $Status_1$
  wr ms : $MName$-set
pre $l \in ms$
post $ms = \overleftarrow{ms} - \{l\} \wedge$
  if $ms = \{\}$ then $bs = \mathsf{sin}$ else $bs = \overleftarrow{bs}$

The retrieve function from the first, more concrete, specification to this one is simple to define by cases.

$retr_{2\text{-}1} : \Sigma_2 \to \Sigma_1$

$retr_{2\text{-}1}(mk\text{-}\Sigma(b_1, b_2, b_3, ms)) \triangleq$
  cases $(count(\mathsf{n}, [b_1, b_2, b_3]), count(\mathsf{m}, [b_1, b_2, b_3]))$ of
  $(0,0) \to mk\text{-}\Sigma_1(\mathsf{sii}, ms)$
  $(1,0) \to mk\text{-}\Sigma_1(\mathsf{sin}, ms)$
  $(0,1) \to mk\text{-}\Sigma_1(\mathsf{sim}, ms)$
  $(1,1) \to mk\text{-}\Sigma_1(\mathsf{snm}, ms)$
  end

This specification abstracts away from the behaviour of the individual buffers and so it does not help us to reason about the algorithm for updating them. However, it does exhibit a useful congruence on the original state space and makes the property of not returning to the sii states very clear. This observation motivates the following further abstraction.

3.2 A further abstraction

In this specification we abstract away from the buffers entirely: their place being taken by a single boolean flag that records whether a write has ever occurred. Although this specification is consequently extremely simple, it still exhibits the same external behaviour as the original.

state Σ_0 of
  b : $\mathbb{B}$
  ms : $MName$-set
inv $mk\text{-}\Sigma_0(b, ms) \triangleq b = \mathsf{false} \Rightarrow ms = \{\}$
init $mk\text{-}\Sigma_0(b, ms) \triangleq b = \mathsf{false} \wedge ms = \{\}$
end

The operations specifications are now very simple:

operations

$slave$ ()
ext wr b : $\mathbb{B}$
pre true
post b = true

$acq\ (l\colon MName)$
ext rd b : B
wr ms : $MName$-set
pre $b = \mathsf{true} \wedge l \notin ms$
post $ms = \overleftarrow{ms} \cup \{l\}$

$rel\ (l\colon MName)$
ext wr ms : $MName$-set
pre $l \in ms$
post $ms = \overleftarrow{ms} - \{l\}$

The retrieve function is straightforward.

$retr_{1\text{-}0} : \Sigma_1 \to \Sigma_0$
$retr_{1\text{-}0}(mk\text{-}\Sigma_1(bs, ms)) \triangleq mk\text{-}\Sigma_0(bs \neq \mathsf{sii}, ms)$

4 An alternative view of MSMIE

The above specifications are based on the state recording the status of each buffer. Effectively, the state is a map from each buffer to its status. Returning to the original specification, we observe that there is always exactly one buffer with status s and at most one with status m or n. This makes it possible to invert the map and think of the state as mapping each status to a buffer.

This leads to a specification that is equivalent to the first one, but might yield a more efficient basis for an implementation. This change also makes it possible to specify the access constraints more closely.

4.1 The state

types

$BName = \{1, 2, 3\}$

$MName =$ token

state Σ_3 of
s : $BName$
n : $[BName]$
m : $[BName]$
ms : $MName$-set
inv $mk\text{-}\Sigma_3(s,n,m,ms)$ $\triangleq$ (m = nil $\Leftrightarrow$ $ms = \{\}$) $\wedge$
$nil\text{-}or\text{-}different([s,m,n])$
init $mk\text{-}\Sigma_3(s,n,m,ms)$ $\triangleq$ $mk\text{-}\Sigma(1,\mathsf{nil},\mathsf{nil},\{\})$
end

where $nil\text{-}or\text{-}different([s,n,m])$ is true if and only if each of s, n and m are each mapped to distinct $BNAME$s or nil:

$nil\text{-}or\text{-}different : [BNAME]^* \to \mathbb{B}$

$nil\text{-}or\text{-}different(l) \triangleq \forall i \in \mathsf{inds}\, l \cdot l(i) = nil \vee l(i) \notin \mathsf{elems}\,(i \lhd l)$

It is perhaps worth noting that an alternative data model would consist of a single map $\{Status\}BNAME$. However, we have chosen the above because this allows us to narrow the read and write frames of some of the operations.

The retrieve function

In this case we give the retrieve function implicitly, noting however that it is fully determined (and implementable):

$retr_{3\text{-}2}\,(mk\text{-}\Sigma_3(s,n,m,ms)\text{:}\,\Sigma_3)\,\sigma_2\text{:}\,\Sigma_2$
pre true
post let $mk\text{-}\Sigma_2(bs,ms_2) = \Sigma_2$ in
$\mathsf{len}\, bs = 3 \wedge$
$\forall i \in \{1,2,3\} \cdot (s = i \Rightarrow b_i = \mathsf{s}) \wedge$
$(n = i \Rightarrow b_i = \mathsf{n}) \wedge$
$(m = i \Rightarrow b_i = \mathsf{m}) \wedge$
$(i \notin \{s,n,m\} \Rightarrow b_i = \mathsf{i})$
$\wedge\, ms_2 = ms$

4.2 The Operations

slave

operations

$slave\ ()$
ext rd m : $[BName]$
wr n : $[BName]$
wr s : $BName$
pre true
post $n = \overleftarrow{s}$

The interaction between invariant and externals is interesting. Here, read access to m is required although m is not referred to in the specification. This is because m is linked to s via the invariant and the value of s which is not fully determined by the post-condition: any implementation will need to read m in order to ascertain what value it is valid to assign to s.

Thus rather than think of the externals clauses as giving information about the variables mentioned in the *specification*, we see them as giving details of what access to state variables an *implementation* of that operation can be allowed to make. This distinction separates their semantic role giving information about access to state variables from the syntactic role they play in binding the free variables of the pre- and post-condition.

acquire

$acq\ (l\text{: } MName)$
ext wr ms : $MName$-set
wr n, m : $[BName]$
pre $l \notin ms \wedge \neg (n = \mathsf{nil} \wedge m = \mathsf{nil})$
post $ms = \overleftarrow{ms} \cup \{l\} \wedge$
$(\overleftarrow{ms} \neq \{\} \Rightarrow m = \overleftarrow{m} \wedge n = \overleftarrow{n}) \wedge$
$(\overleftarrow{ms} = \{\} \Rightarrow m = \overleftarrow{n} \wedge n = \mathsf{nil})$

Interestingly, the last conjunct of this postcondition could be considered to be redundant. When $\overleftarrow{ms} = \{\}$ and thus $\overleftarrow{m}$ is nil, then $ms = \{l\}$ and so m must be assigned a non nil value. Now, as read access to s is prohibited, the only buffer that we can be sure is not already in use is that previously assigned to n. So any implementation that respects the frames of reference must assign this buffer to m. Then the only remaining possible value for n is nil. However, to hide so much information in the externals clause seems to be counter-productive.

release

$rel\ (l\colon MName)$
ext wr ms : $MName$-set
wr n, m : $[BName]$
pre $l \in ms$
post $ms = \overleftarrow{ms} - \{l\} \wedge$
$(ms \neq \{\} \Rightarrow m = \overleftarrow{m} \wedge n = \overleftarrow{n}) \wedge$
$(ms = \{\} \wedge n \neq \mathsf{nil} \Rightarrow n = \overleftarrow{n} \wedge m = \mathsf{nil}) \wedge$
$(ms = \{\} \wedge n = \mathsf{nil} \Rightarrow n = \overleftarrow{m} \wedge m = \mathsf{nil})$

5 The improved MSMIE

As mentioned earlier, Bruns and Anderson observe that, as it stands, the three buffer MSMIE can exhibit an undesirable behaviour. That is, it is possible for a series of overlapping reads, each beginning before the last ends, to lock-out indefinitely the latest data. They suggest an improved protocol that uses a fourth buffer to eliminate this possibility.

Surprisingly, although this new protocol exhibits the same external behaviour as the earlier one, there is no formal refinement relationship between them. To understand why this is, we recall that the part of the system modelled only concerns itself with the assignment of processors to buffers and so does not model the actual transfer of information from slave to masters. Thus, the values assigned to the status flags have no externally visible effect and all the machinations of the state can be seen as purely an implementation bias in the model.

However, the four-buffer version is a refinement of the most abstract specification given earlier, which gives another important reason for considering those abstractions. In particular, validations of the abstract model will carry over to both the three and four buffer versions.

Of course, in this case, it is the internal properties of the model itself that are of interest, as it is these properties that influence the "freshness" of the data read by the masters. In this respect, the four buffer protocol is indeed better behaved as it would lead to a system where the delay in information transfer is at worst equal to that of the three buffer version.

In the four buffer version of MSMIE, there is also an extra status possible for buffers. o is used to denote a buffer that is still being read but no longer contains most up-to-date information.

Thus:

- s as before, is a buffer that is reserved for writing
- n as before, is a buffer that contains the latest data but is not being read (waiting for read)

m is a buffer being read, (and the newest such)

o is a buffer being read (but there is also a newer one being read)

ms is the set of masters reading m

os is the set of masters reading o.

New masters are always assigned to the n or the m buffer. m buffers are "demoted" to o status in a way that ensures that the o buffer will periodically become idle. In this way the protocol avoids the "refresh" problems of the three-buffer version. Again detailed descriptions of the mechanisms used to achieve this is given accompanying the formal text.

It might help to think of the status transitions $i \rightarrow s \rightarrow n$ as the write phase of a buffer and the transitions $n \rightarrow m \rightarrow o \rightarrow i$ as the read phase. We will see that this variant of MSMIE always has two buffers in write phase and two buffers in read phase.

5.1 The state

types

$$BName = \{1,2,3,4\}$$

state Σ_4 of

$$\begin{aligned} s &: BName \\ n &: [BName] \\ m &: [BName] \\ o &: [BName] \\ ms &: MName\text{-set} \\ os &: MName\text{-set} \end{aligned}$$

inv $mk\text{-}\Sigma_4(s,n,m,o,ms,os) \triangleq$

$$\begin{aligned} &(m = nil \Leftrightarrow ms = \{\}) \wedge \\ &(o = nil \Leftrightarrow os = \{\}) \wedge \\ &(ms \cap os = \{\}) \wedge \\ &(nil\text{-}or\text{-}different([s,n,m,o])) \wedge \\ &(m = nil \wedge n = nil \Rightarrow o = nil) \end{aligned}$$

init $\sigma_4 \triangleq \sigma_4 = mk\text{-}\Sigma_4(1, \mathsf{nil}, \mathsf{nil}, \mathsf{nil}, \{\}, \{\})$

end

The last conjunct in the invariant, which rules out the states corresponding to $\{\mathsf{s},\mathsf{o},\mathsf{i},\mathsf{i}\}_m$, is the result of the way that readers of m are released which, as in the earlier specifications, ensures that there is always an m or an n buffer remaining.

A validation property for the state

The invariant only allows the following states corresponding to the following 7 combinations of buffer status:

$$\{\mathsf{s,i,i,i}\}_m, \{\mathsf{s,i,i,n}\}_m, \{\mathsf{s,i,i,m}\}_m, \{\mathsf{s,i,m,n}\}_m, \{\mathsf{s,i,m,o}\}_m, \{\mathsf{s,i,n,o}\}_m, \{\mathsf{s,m,n,o}\}_m$$

5.2 The Operations

slave

slave ()
ext rd m, o : $[BName]$
wr n : $[BName]$
wr s : $BName$
pre true
post $n = \overleftarrow{s}$

As before the implementation will require access to m and o in order to be able to set a valid s. That is:

$$s \in BName - \{n, m, o\}$$

This access requirement is recorded in the externals even though the predicates do not mention m and o.

The descriptions of *acquire* and *release*, given via case analysis, are rather unwieldy. As different variables change in the different cases, the operations have to have wide write access and hence require a lot of clauses saying which variables do not change in that case. Thus we introduce a notational shorthand used in postconditions to say that certain state components are unchanged[2]:

$Id : A^* \to \mathsf{B}$

$Id(l) \triangleq \forall i \in \mathsf{inds}\, l \cdot l(i) = \overleftarrow{l(i)}$

[2]Note that this should be seen as a syntactic "macro", rather than an auxiliary function.

acquire

Acquire behaves in a manner very similar to before: a reader is assigned to either the n or the m buffer as appropriate. The only extra consideration is in the case where there is an n buffer waiting, an m buffer already being read, but no o buffer. In this case, where previously the new reader would have been assigned to the m buffer, it is now possible to begin the read on the n buffer, hence the improvement to the freshness of the data exchanged. The buffer that was already being read is marked as o, and correspondingly the record of processors reading that buffer, ms, is moved to os; and the new read begins on the buffer that was n, thus making it into a new m and the new reader is recorded in ms. No more masters will be assigned to the o buffer until it has been through the write cycle again.

$acq\ (l\colon MName)$
ext wr ms, os : $MName$-set
  wr n, m, o : $[BName]$
pre $l \notin ms \cup os \wedge \neg(n = \mathsf{nil} \wedge m = \mathsf{nil})$
post $(ms \cup os = \overleftarrow{ms} \cup \overleftarrow{os} \cup \{l\}) \wedge$
  $(\overleftarrow{m} = \mathsf{nil} \Rightarrow m = \overleftarrow{n} \wedge n = \mathsf{nil} \wedge Id([o, os])) \wedge$
  $(\overleftarrow{m} \neq \mathsf{nil} \wedge (\overleftarrow{o} \neq \mathsf{nil} \vee n = \mathsf{nil}) \Rightarrow Id([m, n, o, os])) \wedge$
  $(\overleftarrow{m} \neq \mathsf{nil} \wedge \overleftarrow{o} = \mathsf{nil} \wedge n \neq \mathsf{nil}$
    $\Rightarrow o = \overleftarrow{m} \wedge m = \overleftarrow{n} \wedge n = \mathsf{nil} \wedge os = \overleftarrow{ms} \wedge ms = \{l\})$

release

Release behaves exactly as before but with the extra case that it is possible to release a buffer that is reading o. When the last o reader releases, the o buffer becomes idle.

$rel\ (l\colon MName)$
ext wr ms, os : $MName$-set
  wr n, m, o : $[BName]$
pre $l \in ms \cup os$
post $ms \cup os = \overleftarrow{ms} \cup \overleftarrow{os} - \{l\} \wedge$
  $(\{l\} \subset \overleftarrow{ms} \Rightarrow Id([m, n, o, os])) \wedge$
  $(\{l\} = \overleftarrow{ms} \wedge \overleftarrow{n} = \mathsf{nil} \Rightarrow Id([o, os]) \wedge n = \overleftarrow{m} \wedge m = \mathsf{nil}) \wedge$
  $(\{l\} = \overleftarrow{ms} \wedge \overleftarrow{n} \neq \mathsf{nil} \Rightarrow Id([n, o, os]) \wedge m = \mathsf{nil}) \wedge$
  $(\{l\} \subset \overleftarrow{os} \Rightarrow Id([m, n, o, ms])) \wedge$
  $(\{l\} = \overleftarrow{os} \Rightarrow Id([m, n, ms]) \wedge o = \mathsf{nil})$

In fact, in several places, parts of acq and rel give more detail than is required. For example, the last conjunct $o = \mathsf{nil}$ is redundant because we know $os = \{\}$. Similarly, the clause $o = \overleftarrow{o}$

implicit in the use of Id in the penultimate line is also redundant. As before, the redundant clauses are left in for the sake of clarity.

Retrieve function

As stated earlier this version is a data refinement of the most abstract model. The retrieve function is straightforward:

$$retr_{4\text{-}0} : \Sigma_4 \rightarrow \Sigma_0$$

$$retr_{4\text{-}0}(mk\text{-}\Sigma_4(s, n, m, o, ms, os)) \triangleq$$
$$\quad mk\text{-}\Sigma_0(n = \mathsf{nil} \wedge m = \mathsf{nil} \wedge o = \mathsf{nil}, ms \cup os)$$

We have seen five specifications which exhibit the same external behaviour. All except for the most abstract incorporate some degree of implementation bias. However, it is this very bias that is the subject under investigation. In [BA92] validation conditions are expressed in the modal μ-calculus that describe some desirable global properties of the protocol. For the purposes of comparison of the two notations considered in this paper it is sufficient to note that neither provides a formalism to express such conditions.

6 The specification using B

A similar series of specifications and refinements was constructed in B. In preference to presenting this material in full, we present only those parts that highlight the notational and stylistic differences between VDM and B which arose in this example.

This development was carried out using the current alpha-release of the B Toolkit [Abrial92a]. Although this has meant that the specifications have been required to conform exactly to the language supported by the machine[3], it has given us the advantages of automatic consistency checking that the toolkit provides. In this paper, we present the B machines as they were entered in the toolkit, though in some places syntactic sugaring may have made them more readily digestible.

We first present the most abstract specification of MSMIE using B, as a machine $b0$. This corresponds to the VDM specification with state $_0$. The machine is parameterised by a set $MNAME$ of master names (assumed to be non-empty):

[3] In the VDM specifications, we have tried to follow the draft BSI standard as far as possible, but have allowed at least one notational extension in the Id function in the previous section.

MACHINE

$b0(MNAME)$

The state consists of two variables:

VARIABLES

$b0,$
ms

Unlike VDM, the typing information for the state variables is given in the invariant. Here, the first two clauses give "static" (decidable) typing, and the third clause gives subtyping information:

INVARIANT

$b0 \in BOOL \quad \wedge$
$ms \in \mathsf{P}(MNAME) \quad \wedge$
$b0 = FALSE \Rightarrow ms = \varnothing$

The initialisation is given as a generalised substitution:

INITIALISATION

$b0 := FALSE \quad ||$
$ms := \varnothing$

At this level, the operations are very similar to the VDM ones presented earlier.

OPERATIONS

$slave \quad \widehat{=}$
$\quad b0 := TRUE;$

$acq(l1) \quad \widehat{=}$
PRE
$\quad l1 \in MNAME \quad \wedge$
$\quad b0 = TRUE \quad \wedge$
$\quad l1 \notin ms$
THEN
$\quad ms := ms \cup \{l1\}$
END;

$rel(l1) \;\widehat{=}$

PRE

$l1 \in MNAME \;\wedge$

$l1 \in ms$

THEN

$ms := ms - \{l1\}$

END

END

The major syntactic differences from VDM are that the types of the arguments are given explicitly as predicates. Also, the read and write frames of the machine operations are implicit. The read frames are always the full state of the machine, and the variables written are determined by the generalised substitution (for example, those that appear on the left side of a simple substitution). Framing is addressed further in the closing discussion. Thus, the operation *slave* writes $b0$, and *acquire* and *release* write ms.

6.1 One buffer status

The first reification $b1$ of $b0$ is presented as a B refinement. It corresponds to the VDM specification with state $_1$. This highlights the fact that the B method makes a notational distinction between refinements and other specifications.

REFINEMENT

$b1(MNAME)$

REFINES

$b0$

SETS

$STATUS = \{SII, SIN, SIM, SNM\}$

The new state also has two variables. By repeating the ms variable name, we are saying that the variable is the same as the one in the abstract spec.

VARIABLES

$b1$,
ms

Reification is handled by giving an extension of the state and a coupling invariant that relates the new components to the old, as for example in [Mor91]. State variables include those of the abstract state and any added here. However, the variables of the abstract machine are subject to full hiding, i.e. they cannot appear in definition of operations (unless they are explicitly repeated, as is done with ms). The relationship between abstract and concrete variables is given via a coupling relation. In this example, the coupling relation appears as the last conjunct of the invariant:

INVARIANT

$$b1 \in STATUS \quad \wedge$$
$$ms = \varnothing \Leftrightarrow (b1 \in \{SII,\ SIN\}) \quad \wedge$$
$$b0 = FALSE \Leftrightarrow (b1 = SII)$$

INITIALISATION

$$b1 := SII \quad ||$$
$$ms := \varnothing$$

As in $b0$, the operations are similar to those of the corresponding VDM specification.

OPERATIONS

$$slave \ \widehat{=}$$
$$b1 \in \{SII,\ SIN\} \Longrightarrow b1 := SIN$$
$$[]$$
$$b1 \in \{SIM,\ SNM\} \Longrightarrow b1 := SNM;$$

$acq(l1) \;\widehat{=}$

PRE

$l1 \in MNAME \;\wedge$

$b1 \neq SII \;\wedge$

$l1 \notin ms$

THEN

$ms := ms \cup \{l1\}$

||

IF $ms = \emptyset$ THEN

$b1 := SIM$

END

END;

$rel(l1) \;\widehat{=}$

PRE

$l1 \in MNAME \;\wedge$

$l1 \in ms$

THEN

$ms := ms - \{l1\}$

||

IF $ms = \emptyset$ THEN

$b1 := SIN$

END

END

END

6.2 3 buffer status

Now we proceed with a refinement $b2$ of $b1$. This corresponds to the VDM specification with state $_2$, i.e. our original specification. The data model (including the invariant) is similar to that of the VDM specification, and the refinement coupling is similar to the retrieve function $retr_{2\text{-}1}$.

REFINEMENT

$b2(MNAME)$

REFINES

$b1$

SETS

$$STATUSII = \{S2,\ I2,\ N2,\ M2\};$$

VARIABLES

$b2$,
ms

INVARIANT

/* typing */
$b2 \in \mathsf{seq}(STATUSII) \quad \wedge$
$ms \in \mathbf{P}(MNAME) \quad \wedge$
/* subtyping */
$\mathsf{size}(b2) = 3 \quad \wedge$
$\mathsf{card}(b2 \rhd \{S2\}) = 1 \quad \wedge$
$\mathsf{card}(b2 \rhd \{M2\}) \in \{0,\ 1\} \quad \wedge$
$\mathsf{card}(b2 \rhd \{N2\}) \in \{0,\ 1\} \quad \wedge$
$\mathsf{card}(b2 \rhd \{M2\}) = 0 \Leftrightarrow (ms = \varnothing) \quad \wedge$
/* coupling */
$(\mathsf{card}(b2 \rhd \{N2\}) = 0 \ \wedge\ \mathsf{card}(b2 \rhd \{M2\}) = 0) \Leftrightarrow (b1 = SII) \quad \wedge$
$(\mathsf{card}(b2 \rhd \{N2\}) = 1 \ \wedge\ \mathsf{card}(b2 \rhd \{M2\}) = 0) \Leftrightarrow (b1 = SIN) \quad \wedge$
$(\mathsf{card}(b2 \rhd \{N2\}) = 0 \ \wedge\ \mathsf{card}(b2 \rhd \{M2\}) = 1) \Leftrightarrow (b1 = SIM) \quad \wedge$
$(\mathsf{card}(b2 \rhd \{N2\}) = 1 \ \wedge\ \mathsf{card}(b2 \rhd \{M2\}) = 1) \Leftrightarrow (b1 = SNM)$

INITIALISATION

$b2 := [S2,\ I2,\ I2] \quad ||$
$ms := \varnothing$

OPERATIONS

$slave \;\widehat{=}$

(/* Find a buffer that was N and set it to I */

$@(z1).(z1 \in \{1, 2, 3\} \;\wedge\; b2(z1) = N2 \Longrightarrow b2 := (b2 \lhd\!\!\!\!- \{z1 \mapsto I2\}))$

[]

/* but if you can't find one that was N then don't worry */

$\forall z2.(z2 \in \{1, 2, 3\} \Rightarrow b2(z2) \neq N2) \Longrightarrow skip$

);

/* Then find a buffer that was S and set it to N
(there will be exactly one) */

$@(z3).(z3 \in \{1, 2, 3\} \;\wedge\; b2(z3) = S2 \Longrightarrow b2 := (b2 \lhd\!\!\!\!- \{z3 \mapsto N2\}))$;

/* Then, find one that is I and set it to S
(there will be one or two of these) */

$@(z4).(z4 \in \{1, 2, 3\} \;\wedge\; b2(z4) = I2 \Longrightarrow b2 := (b2 \lhd\!\!\!\!- \{z4 \mapsto S2\}))$;

The notation "$@(v).S$", where v is a variable and S a generalised substitution, represents an unbounded choice substitution. In the above it is used in conjunction with a guarded substitution, in the form "$@(v).(P(v) \Longrightarrow S)$"; this can be read operationally as, "if there is any v such that $P(v)$, then apply the substitution S for one such v".

Here we see a significant difference in style encouraged partly by the different treatment of invariants in B and VDM and partly by the differences between the relational postconditions and generalised substitutions. In B, the definition of an operation has to be strong enough to show that the resultant state satisfies the invariant. In this example we have given a definition of *slave* which provides more explicit algorithmic information than its VDM counterpart, giving it a more "programmatic" flavour.

A similar change in style is reflected in the definitions of *acquire* and *release*, which for brevity are not included here.

6.3 The "improved MSMIE" version

The following B machine *snmo* corresponds to the VDM specification of the 4-buffer MSMIE solution. (The 3-buffer "inverted map" version has been omitted from this paper, because the same issues arise with this machine.)

A major difference from the VDM version is that because the write frame is implicit there is no need for the *Id* clauses that appear in the VDM version.

REFINEMENT

$snmo(MNAME)$

REFINES

$b0$

VARIABLES

$sb,\ nb,\ mb,\ ob,$
$ms4,\ os4$

INVARIANT

/* typing */
$sb \in 1\,..\,4 \quad \wedge$
$nb \in \mathsf{seq}(1\,..\,4) \quad \wedge$
$mb \in \mathsf{seq}(1\,..\,4) \quad \wedge$
$ob \in \mathsf{seq}(1\,..\,4) \quad \wedge$
$ms4 \in \mathsf{P}(MNAME) \quad \wedge$
$os4 \in \mathsf{P}(MNAME) \quad \wedge$
/* subtyping */
$\mathsf{size}(nb) \in \{0,\ 1\} \quad \wedge$
$\mathsf{size}(mb) \in \{0,\ 1\} \quad \wedge$
$\mathsf{size}(ob) \in \{0,\ 1\} \quad \wedge$
$mb = [] \Leftrightarrow (ms4 = \varnothing) \quad \wedge$
$ob = [] \Leftrightarrow (os4 = \varnothing) \quad \wedge$
$[sb] \neq nb \quad \wedge$
$[sb] \neq mb \quad \wedge$
$[sb] \neq ob \quad \wedge$
$nb = mb \Leftrightarrow (nb = [] \ \wedge \ mb = []) \quad \wedge$
$nb = ob \Leftrightarrow (nb = [] \ \wedge \ ob = []) \quad \wedge$
$mb = ob \Leftrightarrow (mb = [] \ \wedge \ ob = []) \quad \wedge$
$mb = [] \ \wedge \ nb = [] \Rightarrow ob = [] \quad \wedge$
/* coupling */
$ms = ms4 \cup os4 \quad \wedge$
$b0 = FALSE \Leftrightarrow (nb = [] \ \wedge \ mb = [] \ \wedge \ ob = [])$

INITIALISATION

$sb := 1 \quad ||$
$nb := [] \quad ||$
$mb := [] \quad ||$
$ob := [] \quad ||$
$ms4 := \varnothing \quad ||$
$os4 := \varnothing$

OPERATIONS

$slave \;\widehat{=}$
$nb := [sb] \quad ||$
$sb :\in \{1, 2, 3, 4\} - \{sb\} - \mathsf{ran}(mb) - \mathsf{ran}(ob)$;

$acq(l1) \;\widehat{=}$
PRE
$l1 \in MNAME \;\wedge$
$l1 \notin ms4 \;\wedge$
$\mathsf{conc}([nb, mb]) \neq []$
THEN
SELECT $mb = []$ THEN
$mb := nb \quad ||$
$nb := [] \quad ||$
$ms4 := \{l1\}$
WHEN $mb \neq [] \wedge nb \neq [] \wedge ob = []$ THEN
$ob := mb \quad ||$
$mb := nb \quad ||$
$nb := [] \quad ||$
$os4 := ms4 \quad ||$
$ms4 := \{l1\}$
ELSE
$ms4 := ms4 \cup \{l1\}$
END
END;

$rel(l1) \;\widehat{=}$

PRE

$l1 \in MNAME \;\wedge$

$l1 \in ms4 \cup os4$

THEN

SELECT $\{l1\} \subset ms4$ THEN

$ms4 := ms4 - \{l1\}$

WHEN $\{l1\} = ms4 \wedge nb = []$ THEN

$nb := mb \;||$

$mb := [] \;||$

$ms4 := ms4 - \{l1\}$

WHEN $\{l1\} = ms4 \wedge nb \neq []$ THEN

$mb := [] \;||$

$ms4 := ms4 - \{l1\}$

WHEN $\{l1\} \subset os4$ THEN

$os4 := os4 - \{l1\}$

WHEN $\{l1\} = os4$ THEN

$ob := [] \;||$

$os4 := os4 - \{l1\}$

END

END

END

7 Discussion

As stated in the introduction, this example highlights three areas where the notations encourage different approaches. This closing section gives a brief discussion of some of the points that arose from our study of the MSMIE protocol.

Invariants

In both notations, the invariant is useful for quickly conveying an understanding of the reachable values of the state. However the use of invariants in operation definitions differs. In B, postconditions (in the form of generalised substitutions) have to be written as to ensure the maintenance of the invariant. In VDM the state invariant is effectively part of the state typing information, and as such is assumed to be maintained in addition to the postcondition.

VDM's implicit maintenance of the invariant led to the choice discussed earlier of how much of the information in the invariant is repeated in a postcondition. There was often some tension

between the most concise form that relied on properties of the invariant for its correctness, and a longer, but more explicit form, that included some redundant information. This choice can be seen as an opportunity to prove the stronger forms from the weaker. Which form is chosen may make a significant difference to the complexity of the proofs: the form that most clearly conveys the information may not be the form that will be most usable in proofs. Indeed, the stronger form is more likely to be helpful when the specification is being proved to be a reification of another, and the weaker form when it is itself being reified.

In the B notation, on the other hand, one always has to write operations that imply the preservation of the invariant. This may encourage a tendency to describe *how* the invariant is maintained, which may lead to less abstract specifications.

Operation definitions

The greater programmatic feel of the B notation is reinforced by the use of generalised substitutions, as opposed to VDM's relational post-conditions. Although the two forms have the same expressive power, in some cases (as for example in *slave* in the $b2$ machine) we found it convenient to give greater algorithmic detail in the B version. This would appear to imply that the B notation is more useful for the development of algorithms. Indeed, the process of operation decomposition has been given greater attention in the B methodology than for VDM. By contrast, perhaps VDM's relational postconditions give a greater facility for non-algorithmic specifications of complex operations.

Framing

As stated earlier, the read and write frames are given explicitly in a VDM operation, whereas in B the variable access and modification is implicit in the form of the generalised substitution.

In VDM operations, the semantic role of the read frame is often underplayed. Typically, it is interpreted as merely providing syntactic scoping for variables appearing in the precondition or postcondition. Alternatively, it could be interpreted as a constraint on implementations, restricting which state components can be read. Thus rather than think of the externals clauses as giving information about the variables mentioned in the *specification*, we see them as giving details of what access to state variables an *implementation* of that operation can be allowed to make. (See [Bic92] for further discussion of this point.)

In B, similar restrictions can be given through the hiding principles inherent in the different forms of machine structuring. For instance in this example, where we were able to narrow the read frames in the later VDM specifications, in the B counterparts there is a potential to structure the overall machine as a B "implementation" in terms of simpler machines (one for each status flag).

In the above we have emphasised three areas where our experiments have suggested that the notations of VDM and B encourage different specification styles. Each style may have its own advantages at different stages of the development process. In this example we found that the process of developing implementation code was better addressed in B's abstract machine notation. However, we also found VDM's relational postconditions more convenient for expressing wholly implicit specifications of operations, particularly when the data model involved complex interdependencies.

References

[Abrial92a] Abrial, J.R. *Introducing B-Technologies* (draft). May 1992.

[Abrial92b] Abrial, J.R. *The B Method.* Book to appear

[BA91] Bruns, G. and Anderson, S., The Formalization of a Communications Protocol. LFCS TR 91-137 (April 1991).

[BA92] Bruns, G. and Anderson, S., The Formalization of a Communications Protocol. LFCS/Adelard TR. Safety-Critical Computer Systems, April 6, 1992.

[Bic93] Bicarregui, J.C. Algorithm refinement with read and write frames, in: Woodcock, J.C.P and Larsen, P.G., *FME'93: Industrial-Strength Formal Methods,* Springer-Verlag, 1993.

[BR93] Bicarregui, J.C. and Ritchie, B. Invariants, Frames and Postconditions: a Comparison of the VDM and B Notations, in: Woodcock, J.C.P and Larsen, P.G., *FME'93: Industrial-Strength Formal Methods,* Springer-Verlag, 1993.

[Jones90] Jones, C.B. *Systematic Software Development Using VDM*, second edition. Prentice Hall, 1990.

[Mor91] Morgan, C. *Programming from Specifications*. Prentice Hall, 1990.

[MSMIE] L.L. Santoline *et al.* Multiprocessor Shared-Memory Information Exchange. IEEE Transactions on Nuclear Science. Vol.36. No.1, Feb 1989. pp. 626-633.