Weak Refinement in Z

John Derrick, Eerke Boiten, Howard Bowman and Maarten Steen*

Computing Laboratory, University of Kent, Canterbury, CT2 7NF, UK.
(Phone: + 44 1227 764000, Email: J.Derrick@ukc.ac.uk.)

Abstract. An important aspect in the specification of distributed systems is the role of the internal (or unobservable) operation. Such operations are not part of the user interface (i.e. the user cannot invoke them), however, they are essential to our understanding and correct modelling of the system. Various conventions have been employed to model internal operations when specifying distributed systems in Z. If internal operations are distinguished in the specification notation, then refinement needs to deal with internal operations in appropriate ways. However, in the presence of internal operations, standard Z refinement leads to undesirable implementations.

In this paper we present a generalization of Z refinement, called weak refinement, which treats internal operations differently from observable operations when refining a system. We illustrate some of the properties of weak refinement through a specification of a telecommunications protocol.

Keywords: Refinement; Distributed Systems; Internal Operations; Process Algebras; Concurrency.

1 Introduction

Now the use of Z for the specification of sequential systems is gaining acceptance, attention is being turned to new domains of applicability - one such example is the use of Z for the specification of concurrent and distributed systems [5, 17, 14, 13, 19]. One aspect that is important in the specification of distributed systems is the role of the internal (or unobservable) operation. Such operations are not part of the user interface (i.e. the user cannot invoke them), however, they are essential to our understanding and correct modelling of the system. Internal operations (or actions) arise naturally in distributed systems, either as a result of modelling concurrency or the non-determinism that is inherent in a model of such a system. For example, internal operations can be used to model communication (e.g. as in the language CCS [15]), non-determinism arises as a by-product of this interpretation. Internal operations are also central to abstraction specification through hiding, a particularly important example of

* This work was partially funded by British Telecom Research Labs., and the EPSRC under grant number GR/K13035.

this is to enable communication to be internalised - a central facet in the design of distributed systems.

Languages specifically targeted at concurrent systems typically have a notion of internal action or operation built into the language. For example, internal operations form a vital part of the theory of process algebras, and a special symbol is reserved for the occurrence of such an internal event (e.g. i or τ). If an internal operation is distinguished in the specification notation, then refinement and equivalence relations defined over the language need to deal with internal operations in appropriate ways. One way is to treat an internal event no differently from observable events, an example of such a relation is strong bisimulation in a process algebra [15]. However, it is well recognised that this is inappropriate as a refinement relation, and that internal events should typically have a different role within refinement and equivalence relations. Examples of relations in which the observable is differentiated from the internal are weak bisimulation [15], testing equivalence [3], reduction and extension [4], failures refinement [11] and Hennessy's testing pre-orders [10]. Central to these relations is the understanding that internal events are unobservable, and that refinement relations must refine the observable behaviour of a specification differently from its internal behaviour.

A number of authors have adopted conventions for specifying internal operations when modelling systems in Z. In each case the internal operation is specified as normal and either has a distinguished name or informal commentary telling us that it is not part of the user interface. If internal operations appear explicitly in a Z specification, we need to consider the possibility of refining these specifications. How should we treat the refinement of internal operations in Z? We seek here to contribute to the debate by making a proposal called *weak refinement*. This has a similar relation to ordinary Z refinement as weak bisimulation does to strong bisimulation in a process algebra. In particular, we define weak refinement by considering the stand point of an external observer of the system, who manipulates operations in the user interface.

Such an external observer will require that a retrieve relation is still defined between the state spaces of the abstract and concrete specifications and that each abstract observable operation AOp is recast as a concrete observable operation COp. The weak refinement relation is defined to ensure that the observable behaviour of the concrete specification is a refinement of the observable behaviour of the abstract specification.

Throughout the paper we assume the state plus operations style of Z specification, and our discussion takes place within that context.

The structure of the paper is as follows. In Section 2 we review the need for internal operations in Z specifications. Section 3 presents an example of a specification and refinement involving internal operations, the example illustrates that standard Z refinement is too liberal in the presence of internal operations. Section 4 formulates the generalization that we call weak refinement, which is motivated by the treatment of internal events in process algebras. Section 5 revisits the protocol example to show that weak refinement has the required properties of a refinement where internal operations have been specified. Section 6 discusses some properties of this refinement, and we conclude in Section 7.

2 Internal Operations

When modelling sequential systems in Z, the operations represent the user interface. That is, a state change occurs in the system if and only if the user invokes one of the operations. However, when modelling concurrent and distributed systems it is convenient to model *internal* operations. These internal operations represent operations over which the user has no control (hence the name internal). Since they are not part of the user interface they can be invoked by the system (potentially non-deterministically) whenever their pre-conditions hold. They can arise either due to the natural non-determinism of a distributed system [11], or due to communication within the system [15] or due to some aspect of the system being hidden at this level of abstraction [2]. The necessity for the specification of internal events in process algebras is well recognised [15], and a number of researchers have found it convenient or necessary to specify internal operations in Z when specifying distributed systems [7, 16, 19, 21, 9]. In each case the internal operation is specified as normal and either has a distinguished name or informal commentary telling us that it is not part of the user interface. We will see examples of both below. Used in this way, Z is clearly sufficient as a notation for the specification of internal operations or actions.

Is this necessary however, why not leave internal operations to process algebras? Well, Z is particularly suited to the specification of parts of a distributed system which contain large amounts of state information. Typical to this class are managed objects [20] or the information viewpoint of the Open Distributed Processing reference model [12], where the specifications contain a lot of state but there is also a need to model internal operations such as alarms.

Although Z is adequate as a notation for the specification of internal actions/operations, the usual Z refinement rules for operations are inappropriate for specifications containing internal operations. As we shall see (at the end of Section 3.2) they are inappropriate because they allow a refinement to contain more non-determinism than is acceptable. This situation is clearly undesirable, and we must re-formulate refinement for internal operations if they are to be used in Z specifications. This is what we seek to do here.

3 Refinement

A Z specification describes the state space together with a collection of operations. The Z refinement relation [18, 21], defined between two Z specifications, allows both the state space and the individual operations to be refined in a uniform manner.

Operation refinement is the process of recasting each abstract operation AOp into a concrete operation COp, such that (informally) the following holds. The pre-condition of COp may be weaker than the pre-condition of AOp, and COp may have a stronger post-condition than AOp. That is, COp must be applicable whenever AOp is, and if AOp is applicable, then every state which COp might produce must be one of those which AOp might produce. Data refinement extends operation refinement by allowing the state space of the concrete operations to be different from the state space of the abstract operations. Refinement for sequential systems specified in Z is well documented and understood. How does refinement behave in the presence of internal operations?

As an illustration of refinement involving internal operations we consider the specification and refinement of a telecoms protocol (the Signalling System No. 7 standard) adapted from [21]. The first specification defines the external view of the protocol, subsequently we develop a sectional view which specifies the route that messages take through the protocol.

3.1 Specification 1: the external view

Let M be the set of messages that the protocol handles. The state of the system comprises two sequences which represent messages that have arrived in the protocol (*in*), and those that have been forwarded (*out*).

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Ext}} \\
\hline
\textit{in}, \textit{out} : \operatorname{seq} M \\
\hline
\exists\, s : \operatorname{seq} M \bullet \textit{in} = s \frown \textit{out} \\
\hline
\end{array}
$$

Incoming messages are added to the left of *in*, and the messages contained in *in* but not in *out* represent those currently inside the protocol. The state invariant specifies that the protocol must not corrupt or re-order. Initially, no messages have been sent:

$$\textit{ExtInit} \mathrel{\widehat=} [\, \textit{Ext}' \mid \textit{in}' = \langle\,\rangle \,]$$

Two operations then model the transmission (*Transmit*) and reception (*Receive*) of messages into and out of the protocol. Their specification is straightforward. In the *Receive* operation, either no message is available (e.g. they are all on route in the protocol) or the next one is output, this choice is made non-deterministically at this level of abstraction.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Transmit}} \\
\hline
\Delta \textit{Ext} \\
m? : M \\
\hline
\textit{in}' = \langle m? \rangle \frown \textit{in} \\
\textit{out}' = \textit{out} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{Receive} \\
\Delta Ext \\
\hline
in' = in \\
\#out' = \#out + 1 \lor out' = out \\
\hline
\end{array}
$$

3.2 Specification 2: the sectional view

The sectional view specifies the route the messages take through a number of *sections*. Let N be the number of sections. Each section in the route may receive and send messages, and those which have been received but not yet sent on are in the section. The messages pass through the sections in order. In the state schema, $ins\ i$ represents the messages currently inside section i, $rec\ i$ the messages that have been received by section i, and $sent\ i$ the messages that have been sent onwards from section i. The state and initialization schemas are then given by

$$
\begin{array}{|l}
\hline
\textit{Section} \\
rec, ins, sent : \mathrm{seq}(\mathrm{seq}\, M) \\
\hline
N = \#rec = \#ins = \#sent \\
rec = ins \frown\frown sent \\
front\ sent = tail\ rec \\
\hline
\end{array}
\qquad
\begin{array}{|l}
\hline
\textit{SectionInit} \\
Section' \\
\hline
\forall\, i : 1..N \bullet \\
\quad rec\ i = ins\ i = sent\ i = \langle\,\rangle \\
\hline
\end{array}
$$

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

$$
\begin{array}{|l}
\hline
\textit{STransmit} \\
\Delta Section \\
m? : M \\
\hline
head\ rec' = \langle m? \rangle \frown (head\ rec) \\
tail\ rec' = tail\ rec \\
sent' = sent \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{SReceive} \\
\Delta Section \\
\hline
rec' = rec \\
front\ ins' = front\ ins \\
last\ ins' = front(last\ ins) \\
front\ sent' = front\ sent \\
last\ sent' = \langle last(last\ ins) \rangle \frown (last\ sent) \\
\hline
\end{array}
$$

Here, the new message received is added to the first section in the route in *STransmit*, and *SReceive* will deliver from the last section in the route. In the first specification, messages arrive non-deterministically, in the sectional view this is represented by the progress of the messages through the sections. Therefore in this more detailed design, we need to specify how the messages progress through the sections, and we do so by defining an operation *Daemon* which non-deterministically selects a section to make progress. The oldest message is then transfered to the following section, and nothing else changes. The important part of this operation is then:

$$
\begin{array}{|l}
\textit{Daemon} \\
\hline
\Delta \textit{Section} \\
\hline
\exists\, i : 1..N-1 \mid \\
\quad \textit{ins}\ i \neq \langle\,\rangle \bullet \\
\quad \textit{ins}'i = \textit{front}(\textit{ins}\ i) \\
\quad \textit{ins}'(i+1) = \langle \textit{last}(\textit{ins}\ i)\rangle \frown \textit{ins}(i+1) \\
\quad \forall\, j : 1..N \mid j \neq i \wedge j \neq i+1 \bullet \textit{ins}'j = \textit{ins}\ j \\
\hline
\end{array}
$$

Daemon is an internal operation (the informal commentary accompanying the specification tells us this), and so can be invoked by the system whenever its pre-condition holds. As noted in [21]: *This operation is not part of the user interface. The user cannot invoke Daemon, but it is essential to our understanding of the system and to its correctness.*

The sectional view is a refinement of the original, where the retrieve relation (which is a total function, i.e. $\forall\, \textit{Section} \bullet \exists_1\, \textit{Ext} \bullet \textit{Retrieve}$) is given by:

$$
\begin{array}{|l}
\textit{Retrieve} \\
\hline
\textit{Ext} \\
\textit{Section} \\
\hline
\textit{head rec} = \textit{in} \\
\textit{last sent} = \textit{out} \\
\hline
\end{array}
$$

Under this refinement *STransmit* and *SReceive* correspond to *Transmit* and *Receive* respectively, and the internal operation *Daemon* corresponds to the external operation $\Xi \textit{Ext}$ (i.e. the identity operation on *Ext*), and we can prove (with appropriate quantification over the states) the refinement by showing that:

$$
\begin{array}{l}
\textit{SectionInit} \wedge \textit{Retrieve} \Rightarrow \textit{ExtInit} \\
\mathrm{pre}\ \textit{Transmit} \wedge \textit{Retrieve} \Rightarrow \mathrm{pre}\ \textit{STransmit} \\
\mathrm{pre}\ \textit{Transmit} \wedge \textit{Retrieve} \wedge \textit{STransmit} \wedge \textit{Retrieve}' \Rightarrow \textit{Transmit} \\
\mathrm{pre}\ \textit{Receive} \wedge \textit{Retrieve} \Rightarrow \mathrm{pre}\ \textit{SReceive} \\
\mathrm{pre}\ \textit{Receive} \wedge \textit{Retrieve} \wedge \textit{SReceive} \wedge \textit{Retrieve}' \Rightarrow \textit{Receive} \\
\mathrm{pre}\ \Xi \textit{Ext} \wedge \textit{Retrieve} \Rightarrow \mathrm{pre}\ \textit{Daemon} \\
\mathrm{pre}\ \Xi \textit{Ext} \wedge \textit{Retrieve} \wedge \textit{Daemon} \wedge \textit{Retrieve}' \Rightarrow \Xi \textit{Ext}
\end{array}
$$

So far so good. We can specify a system that contains non-determinism in some of the operations in its user interface (e.g. *Receive*), but which doesn't contain any internal operations. We can then refine this specification to one that contains internal operations that correctly model (in the sense of a refinement existing) the abstract specification. Here we have used the standard Z refinement relations, which have been perfectly adequate at this level.

However, we can refine this sectional view further. Consider the *Daemon* operation. This operation is partial (as it does not specify what happens if $ins\ i = \langle\rangle$ for every i), and using standard Z refinement we can weaken its pre-condition, and refine it to the following:

$$
\begin{array}{|l}
\hline
\textit{NDaemon} \\
\Delta \textit{Section} \\
\hline
(\forall i : 1..N-1 \bullet ins\ i = \langle\rangle \Rightarrow ins'1 = \langle m \rangle \land m \in M) \lor \\
(\exists i : 1..N-1 \mid \\
\quad ins\ i \neq \langle\rangle \bullet \\
\quad ins'i = front(ins\ i) \\
\quad ins'(i+1) = \langle last(ins\ i)\rangle \frown ins(i+1) \\
\quad \forall j : 1..N \mid j \neq i \land j \neq i+1 \bullet ins'j = ins\ j) \\
\hline
\end{array}
$$

This operation has the same functionality as before, except that in addition the system can invoke it non-deterministically (since it is an internal operation) initially to insert an arbitrary message into the first section. Thus initially there are two possible behaviours of the system: as before the user could invoke *Transmit* to insert a message into the protocol, or now the system could non-deterministically invoke *NDaemon* which corrupts the input stream of the protocol before the user has inserted any messages.

The specification which contains the sectional view operations together with this new *NDaemon* is a refinement of the sectional view. Yet clearly implementations which introduce arbitrary amounts of noise into a stream of protocol messages are unacceptable. But in this situation, using standard Z refinement this has been allowed to happen, what has gone wrong?

We have used standard Z refinement here, and at issue is the refinement of internal operations. Internal operations have behaviour which isn't subject to the normal interpretation of operations (that are in the user interface), so it is not surprising then that using normal refinement brings about unexpected (and undesirable) consequences.

3.3 The firing condition interpretation

One possible solution is described by Strulo in [19], which has the merit of simplicity, but, as we shall see, perhaps constrains refinement too far. Strulo calls internal operations *active*, and operations in the user interface *passive*. The firing condition interpretation is the idea that the pre-condition states the only times the operation can happen at all instead of saying an operation is undefined (but possible) outside its pre-condition.

To define refinement, [19] identifies three regions for an operation (unconstrained, empty and interesting) and the applicability and correctness refinement rules are then re-interpreted for internal operations as:

$$\vdash COp \Rightarrow AOp$$
$$\vdash (\exists State' \bullet AOp) \land (\exists State' \bullet \neg AOp) \Rightarrow$$
$$(\exists State' \bullet COp) \land (\exists State' \bullet \neg COp)$$

The three regions of an operation represent: (1) states where the operation is divergent because no constraints are made on the after state (the unconstrained region), (2) states outside the usual pre-condition but which aren't divergent (the empty region), and (3) the remaining states where some but not all after states are allowed (the interesting region). For a full discussion the reader should consult [19].

In terms of these interpretations and the regions of definition of an operation, the first condition prevents an operation becoming possible where it was impossible, and the second condition ensures that the concrete operation doesn't become impossible where it was defined and possible.

Application of these ideas to the above example shows that with the firing condition interpretation, *NDaemon* is not a refinement of *Daemon*. Thus we successfully stop the pre-condition of an internal operation from being weakened in an unacceptable manner. However, to achieve this a barrier has been placed between observable and unobservable operation refinements. In particular, for hybrid specifications (ones involving both internal and observable operations), the refinement rules used depend on the type of operation - standard refinement for observable operations, and the firing condition interpretation for internal operations.

But the division is not always as simple as that, on occasion we may wish to *introduce* internal operations during a refinement, or we may wish to *remove* internal operations in a refinement. The refinement of the external view to the sectional view is an example of the introduction of internal operations, and we will give an example of their removal shortly.

However, we find that under the firing condition interpretation, the sectional view is not a refinement of the external view of the protocol, because now *Daemon* does not correspond to ΞExt under the firing condition interpretation refinement rules. To overcome this, can we restrict the use of the firing condition interpretation refinement rules to when the abstract operation is internal? The following very simple example will illustrate that we cannot.

Consider an abstract specification with an operation AOp in the user interface, and an internal operation IOp. The concrete specification consists of a single operation COp. Both have state space *State* consisting of a $mode : \{0, 1\}$. Initially *mode* is set to 0. The only operations in the specifications are given by:

$$
\begin{array}{|l}
\hline
\textit{AOp} \\
\Delta State \\
\hline
mode = 0 \land mode' = 1 \\
\hline
\end{array}
\qquad
\begin{array}{|l}
\hline
\textit{IOp} \\
\Delta State, \\
error! : yes \mid no \\
\hline
mode = 1 \land mode' = 0 \\
error! = yes \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{COp} \\
\Delta State, \\
error! : yes \mid no \\
\hline
mode = mode' = 0 \land error! = yes \\
\hline
\end{array}
$$

It is natural to view the concrete specification as a refinement of the abstract. In the abstract, after invoking AOp an error message will occur (triggered by the internal operation IOp happening non-deterministically, which it eventually always will), in the concrete, after invoking COp an error message will occur. This type of removal of internal events lies at the heart of all treatments of internal operations in process algebras. However, under the firing condition interpretation, the concrete operation is not a refinement of the abstract, because no operation that was possible can become impossible - even if the internal behaviour has moved elsewhere. The fact that IOp has an output here is immaterial to the essence of the example - the aspect of internal operations with output is discussed in Section 5.2.

So, to summarise, standard Z refinement is too liberal in the presence of internal operations. An alternative approach is that suggested in [19], however, this involves a different interpretation of operations, and the refinement of internal behaviour can be too strict as the last example shows. In the next section we will seek an alternative generalization of refinement that steers a middle course by using ideas from process algebras.

4 Weak Refinement

To define weak refinement we will consider the standpoint of an *external* observer. Such an external observer will require that a retrieve relation is still defined between the state spaces of the abstract and concrete specifications and that each observable operation AOp is recast as a concrete operation COp. The refinement relation will ensure that the observable behaviour of the concrete specification is a refinement of the observable behaviour of the abstract specification.

The weak refinement rules have the same form as standard refinement, namely that:

– $\forall I_k;\ Cstate' \bullet Cinit_w \vdash \exists I_l;\ Astate' \bullet Ainit_w \land Ret'$

– $\forall I_k \bullet \mathrm{pre}_w\ AOp \land Ret \vdash \exists I_l \bullet \mathrm{pre}_w\ COp$

– $\forall I_k;\ I_p;\ I_q \bullet \mathrm{pre}_w\ AOp \wedge Ret \wedge COp_w \vdash \exists I_m;\ I_n;\ Astate' \bullet Ret' \wedge AOp_w$

except that the subscript w denotes a weak counterpart which we will define below and I_k are sequences of internal operations. The next subsection reviews the treatment of internal events in process algebras, and we use these ideas to motivate our formulation of weak refinement in the following subsection.

4.1 Internal events in Process Algebras

Refinement in a process algebra is defined in terms of the transitions a behaviour can undertake, and we write $P \xrightarrow{a} P'$ if a process (or behaviour) P can perform the action a and then evolve to the process P'. Refinements and equivalences are given in terms of such transitions. For each relation, two versions are possible - a strong relation which treats all actions identically whether observable or not, and a weak version that makes allowances for internal events and is only concerned with observable transitions.

To make allowances for internal actions, consideration is given to what is meant by an observable transition. An observable transition is taken to be any observable action preceded or succeeded by any (finite) number of internal events.

Weak or observable relations now replace transitions $P \xrightarrow{a} P'$ by their observable counterpart: $P \overset{a}{\Longrightarrow} P'$, which means that process P can evolve to process P' by undergoing an unspecified (but finite) number of internal events, followed by the action a, followed by an unspecified number of internal events. Weak bisimulation (or observational equivalence) is an example of a relation defined in such a fashion [15].

4.2 Formulating weak refinement

Throughout this subsection let the state spaces of the abstract and concrete specifications be *Astate* and *Cstate* respectively. Let *Ret* be the retrieve relation defined between the specifications. AOp and COp stand for operations on the abstract and concrete state spaces where COp implements AOp. The initial states are given by *Cinit* and *Ainit*.

Our formulation of weak refinement will be motivated by the approach taken in process algebras. Application of an operation in Z corresponds to a transition in a process algebra, and in weak refinement in place of the application of an operation Op we allow a finite number of internal operations before and after the occurrence of the operation. This corresponds to the change from $P \xrightarrow{a} P'$ to $P \overset{a}{\Longrightarrow} P'$ in a process algebra when moving from a strong to observable scenario. This can be described in the Z schema calculus by saying there exist internal operations $i_1, \ldots, i_k, j_1, \ldots, j_l$ (for $k, l \geq 0$) and the application of the composition $i_1 \fatsemi \ldots \fatsemi i_k \fatsemi Op \fatsemi j_1 \fatsemi \ldots \fatsemi j_l$. Throughout this section we abbreviate $i_1 \fatsemi \ldots \fatsemi i_k$ to i^k, and we will let I_k denote a sequence of internal actions $\langle i_1, \ldots, i_k \rangle$.

We can now re-formulate each of the three conditions for refinement for a system containing internal operations. We begin with the initialization condition.

Initialization

Without internal operations the relationship required upon initialization is that each possible initial state of the concrete specification must represent a possible initial state of the abstract specification. In the presence of internal operations an initial state might evolve internally to another state. Therefore, "each possible initial state of the concrete specification" now includes all possible evolutions of the initial state under internal operations. Likewise "a possible initial state of the abstract specification" can now include a potential evolution of the initial state due to internal operations.

To formalise this (using the abbreviation $i^k = i_1 \fatsemi \ldots \fatsemi i_k$) we require that:

$$\forall I_k;\ Cstate' \bullet Cinit \fatsemi i^k \vdash \exists I_l;\ Astate' \bullet Ainit \fatsemi i^l \land Ret$$

The quantification of the internal operations in $Cinit \fatsemi i^k$ is important. What we wish to ensure is that *every* initial concrete path (including all possible internal operations) can be matched by *some* initial abstract path (possibly involving internal operations). We abbreviate the condition to

$$\forall I_k;\ Cstate' \bullet Cinit_w \vdash \exists I_l;\ Astate' \bullet Ainit_w \land Ret'$$

Applicability

Applicability must ensure that if an abstract and concrete state are related by the retrieve relation, then the concrete operation should terminate whenever the abstract operation terminated, where termination is usually expressed in terms of satisfaction of the pre-condition of an operation. In the presence of internal operations we must allow for potential invocation of internal operations, and hence require that: if an abstract and concrete state are related by the retrieve relation, then whenever the abstract operation terminates possibly after any internal evolution then the concrete operation terminates after some internal evolution.

This is described by saying there exists internal operations $i_1, \ldots, i_k$ such that $\mathrm{pre}(i_1 \fatsemi \ldots \fatsemi i_k \fatsemi AOp)$ where $\fatsemi$ is schema composition in the Z schema calculus. We abbreviate $\mathrm{pre}(i_1 \fatsemi \ldots \fatsemi i_k \fatsemi AOp)$ to $\mathrm{pre}(i^k \fatsemi AOp)$ or $\mathrm{pre}_w\ AOp$.

Applicability can then be expressed as

$$\forall I_k \bullet \mathrm{pre}(i^k \fatsemi AOp) \land Ret \vdash \exists I_l \bullet \mathrm{pre}(i^l \fatsemi COp)$$

Using the abbreviation $\mathrm{pre}_w\ AOp$, where we note that we have replaced $\mathrm{pre}\ AOp$ by the condition that AOp is applicable after a number of internal operations, applicability in weak refinement reduces to

$$\forall I_k \bullet \mathrm{pre}_w\ AOp \land Ret \vdash \exists I_l \bullet \mathrm{pre}_w\ COp$$

Correctness

For correctness, we require the weak analogy to the following: if an abstract state and a concrete state are related by *Ret*, and both the abstract and concrete operations are guaranteed to terminate, then every possible state after the concrete operation must be related by Ret' to a possible state after the abstract operation [18]. For the weak version pre AOp is replaced by $\text{pre}_w\, AOp$ and we ask that, every possible state after the concrete operation must be related by Ret' to a possible state after the abstract operation, except that now 'after' means an arbitrary number of internal operations may occur before and after the abstract operation. The condition thus becomes, in full,

$$\begin{aligned}\forall I_k;\ I_p;\ I_q \bullet \text{pre}(i^k \mathbin{⨾} AOp) \land Ret \land i^p \mathbin{⨾} COp \mathbin{⨾} i^q \vdash \\ \exists I_m;\ I_n;\ Astate' \bullet Ret' \land i^n \mathbin{⨾} AOp \mathbin{⨾} i^m\end{aligned}$$

which we abbreviate to

$$\forall I_k;\ I_p;\ I_q \bullet \text{pre}_w\, AOp \land Ret \land COp_w \vdash \exists I_m;\ I_n;\ Astate' \bullet Ret' \land AOp_w$$

Again the quantification of the internal operations in COp_w is important. We need to ensure is that every path involving COp and possible internal operations can be matched by some path involving AOp and (possibly) internal operations. Hence the quantification in COp_w is universal over all sequences of internal operations before and after COp.

Rules for Internal operations

We will also apply the applicability and correctness rules to internal operations. For internal operations we don't want applicability to prevent an internal operation becoming impossible where it was previously possible, indeed we want to refine out such internal operations in appropriate fashions. So for an internal operation I (defined on a state space *state*) we define its weak pre-condition (not its pre-condition) by

$$\text{pre}_w\, I = \text{pre}\, \Xi state = state$$

Although this definition of the weak pre-condition for internal operations looks strange, it does not allow us to arbitrarily weaken the pre-condition of an internal operation under weak refinement. The circumstances when we can are governed by what observable operations are present in the abstract specification, and the correctness rules for observable operations prevent the arbitrary weakening of pre-conditions of internal operations.

Applicability for internal operations will reduce to checking that the concrete state is implied by the abstract state (modulo the retrieve relation).

The final piece in the jigsaw is the meaning of correctness for internal operations. We define the weak version of an operation Op by

$$Op_w = \begin{cases} i^k \mathbin{⨾} Op \mathbin{⨾} i^l & \text{for observable } Op, \\ i^k & \text{for internal operation } Op, k \geq 0 \end{cases}$$

where $i^0 = \Xi state$ and appropriate quantification will be taken over k (and l) according to the context. This ensures that we can match up an occurrence of an internal operation in the abstract specification by zero (using $\Xi state$) or more (using i^k) internal actions in the concrete specification.

Thus to summarise, weak refinement requires that

- $\forall I_k;\ Cstate' \bullet Cinit_w \vdash \exists I_l;\ Astate' \bullet Ainit_w \land Ret'$
- $\forall I_k \bullet \mathrm{pre}_w\ AOp \land Ret \vdash \exists I_l \bullet \mathrm{pre}_w\ COp$
- $\forall I_k;\ I_p;\ I_q \bullet \mathrm{pre}_w\ AOp \land Ret \land COp_w \vdash \exists I_m;\ I_n;\ Astate' \bullet Ret' \land AOp_w$

where

$$Op_w = \begin{cases} i^k \fatsemi Op \fatsemi i^l & \text{for observable } Op, \\ i^k & \text{for internal operation } Op, k \geq 0 \end{cases}$$

and $i^0 = \Xi state$ and $\mathrm{pre}_w(Op) = \mathrm{pre}(i^k \fatsemi Op)$.

In the next section we show how these rules are applied in practice, and we shall see that although the full generality introduces complexity, in practice the overheads are not large.

5 Examples

We apply the theory we developed above to the examples presented at the start of the paper. In the protocol example, the intuitive behaviour we wish to capture is that the sectional view is a refinement of the external view, but that the third specification is not a refinement of the sectional view. This is indeed the case with weak refinement.

5.1 The Signalling Protocol

First we show the sectional view of the protocol is a weak refinement of the external view. We first prove the initialization is correct, noting that the retrieve relation is total and functional, so that we can use the usual simplification, and we show that:

$$\forall I_k;\ Ext';\ Section' \bullet SectionInit_w \land Retrieve \vdash \exists I_l \bullet ExtInit_w$$

This reduces to $\forall Ext';\ Section' \bullet SectionInit \land Retrieve \vdash ExtInit$, since there are no internal operations in the external specification, and no internal operation is applicable after $SectionInit$ in the sectional view. This can be verified as normal.

To verify applicability, we need to show that

$$\begin{array}{l} \forall I_k \bullet \mathrm{pre}_w\ Transmit \land Retrieve \vdash \exists I_l \bullet \mathrm{pre}_w\ STransmit \\ \forall I_k \bullet \mathrm{pre}_w\ Receive \land Retrieve \vdash \exists I_l \bullet \mathrm{pre}_w\ SReceive \\ \forall I_k \bullet \mathrm{pre}_w\ \Xi Ext \land Retrieve \vdash \exists I_l \bullet \mathrm{pre}_w\ Daemon \end{array}$$

The last equation reduces to $Ext \wedge Section \vdash Section$ since $Daemon$ is internal and for internal operations we have defined $\mathrm{pre}_w\, Daemon = \Xi Section$. In the case of $Transmit$, the weak pre-condition requirement reduces to

$$\mathrm{pre}\; Transmit \wedge Retrieve \vdash \exists\, I_l \bullet \mathrm{pre}(i^l \mathbin{\text{;}} STransmit)$$

which is true with $l = 0$. A similar argument holds for the weak pre-condition of $Receive$.

Similarly, to verify correctness, we need to show that

$$\forall\, I_p;\ I_q \bullet \mathrm{pre}\; Transmit \wedge Retrieve \wedge STransmit_w \wedge Retrieve' \vdash Transmit$$
$$\forall\, I_p;\ I_q \bullet \mathrm{pre}\; Receive \wedge Retrieve \wedge SReceive_w \wedge Retrieve' \vdash Receive$$
$$\forall\, I_p;\ I_q \bullet \mathrm{pre}\; \Xi Ext \wedge Retrieve \wedge Daemon_w \wedge Retrieve' \vdash \Xi Ext$$

For the first, we need to check that occurrences of the $Daemon$ operation before and after $STransmit$ in the concrete specification still leave us in a state that is consistent with that produced by $Transmit$ in the abstract. This is found to be true (since $Daemon \Rightarrow \Xi Ext$). The second case is similar. For the third this reduces to showing that

$$\forall\, k \bullet Ext \wedge Retrieve \wedge Daemon^k \wedge Retrieve' \vdash \Xi Ext$$

where $Daemon^k$ denotes k sequential compositions of $Daemon$. Again this is found to be true.

Therefore the sectional view is indeed a weak refinement of the external view. Moreover, the additional verification requirements imposed by the generality of weak refinement are not large in this example, being confined to the consideration of one internal operation - $Daemon$.

We shall show now that the third specification is *not* a weak refinement of the sectional view. That is, we are not at liberty to weaken the pre-condition of an internal operation arbitrarily. Consider the initialization rule that (for total functional $Retrieve$):

$$\forall\, I_k;\ Astate;\ Cstate \bullet Cinit_w \wedge Retrieve \vdash \exists\, I_l \bullet Ainit_w$$

Now in the sectional view it is not possible to apply $Daemon$ initially. However, it is possible to apply $NDaemon$ initially (where it arbitrarily inserts a new element into the protocol). Thus for the third specification to be a weak refinement of the sectional view we require that

$$SectionInit \mathbin{\text{;}} NDaemon \vdash SectionInit$$

This is clearly not true, since after $NDaemon$, ins is no longer empty.

In addition to the initialization requirement failing in this example, the requirement that

$$\forall\, I_k;\ I_p;\ I_q \bullet \mathrm{pre}_w\, STransmit \wedge Retrieve \wedge STransmit_w \wedge Retrieve' \vdash \\ \exists\, I_m;\ I_n \bullet STransmit_w$$

is also violated for the same reasons as the initial condition fails.

5.2 Internal operations with output

In the second example, presented in section 3.3, to show that the concrete is a weak refinement of the abstract, we need to prove that:

$$\forall I_k \bullet \mathrm{pre}_w\ AOp \wedge Ret \vdash \exists I_l \bullet \mathrm{pre}_w\ COp$$
$$\forall I_k;\ I_p;\ I_q \bullet \mathrm{pre}_w\ AOp \wedge Ret \wedge COp_w \vdash \exists I_m;\ I_n;\ Astate' \bullet Ret' \wedge AOp_w$$

In the refinement we will simply link the states for which $mode = 0$ as the state $mode = 1$ was purely an intermediate state for the purposes of specifying the temporal ordering of the operations. Hence the retrieve relation will be:

$$\begin{array}{|l}
\hline
\textit{Ret} \\
\textit{State} \\
\hline
mode = 0 \\
\hline
\end{array}$$

With this retrieve relation we will in fact show that the concrete operation COp implements both abstract operations AOp and IOp. Since the concrete specification does not have any internal operations we just need to show that:

$$\forall I_k \bullet \mathrm{pre}_w\ AOp \wedge Ret \vdash \mathrm{pre}\ COp$$
$$\forall I_k \bullet \mathrm{pre}_w\ AOp \wedge Ret \wedge COp \wedge Ret' \vdash \exists I_m;\ I_n \bullet AOp_w$$
$$\forall I_k \bullet \mathrm{pre}_w\ IOp \wedge Ret \vdash \mathrm{pre}\ COp$$
$$\forall I_k \bullet \mathrm{pre}_w\ IOp \wedge Ret \wedge COp \wedge Ret' \vdash \exists I_m;\ I_n \bullet IOp_w$$

We can calculate the pre-conditions needed. Note that in the case of $\mathrm{pre}_w\ AOp$ this includes states from which the system can perform an internal operation and then for AOp to successfully terminate.

$$\begin{array}{|l}
\hline
\mathrm{pre}_w\ AOp \\
\textit{State} \\
\hline
mode = 0 \vee mode = 1 \\
\hline
\end{array}
\qquad
\begin{array}{|l}
\hline
\mathrm{pre}\ COp \\
\textit{State} \\
\hline
mode = 0 \\
\hline
\end{array}$$

The applicability and correctness for the implementation of AOp as COp are then easily verified. Consideration of the internal operation amounts to showing that (because of the way the pre-condition of an internal operation is defined)

$$Ret \vdash \mathrm{pre}\ COp$$
$$Ret \wedge COp \wedge Ret' \vdash \exists k \bullet IOp^k$$

and the latter holds for $k = 0$.

So the concrete specification is indeed a weak refinement of the abstract. This illustrates an interesting aspect of specifying internal operations in Z - they can output data (in fact some interpretations of unobservableness in Z outlaw this possibility e.g. [6], but generally this is the case). This is in contrast to a process algebra where typically internal actions can have no data attributes.

Consider full LOTOS [2], where the internal action is written i. Internal actions in LOTOS can arise as a result of direct specification or as a result of hiding observable actions. In the first case, it is syntactically illegal to associate a data attribute with an internal action, e.g. the behaviour

$$i!7;\ B$$

is not well-formed. Here action prefix is represented by **;** and a value declaration on an action is given by a **!**. In the second case, upon hiding an observable action with data, the data is hidden as well as the action. So, for example, in the behaviour

$$\text{hide } g \text{ in } g!5;\ \mathit{stop}$$

the transition i can be performed, but no data is associated with the occurrence of the internal action i.

However, it is desirable to be able to specify an internal event which does have data associated with it. Indeed [19] contains an example of such an operation - an alarm notification in a managed object. This is a typical example of the kind of application where it is necessary to be able to specify an atomic internal operation which has output associated with it. Used in this style Z is more expressive than LOTOS in terms of internal events it can specify.

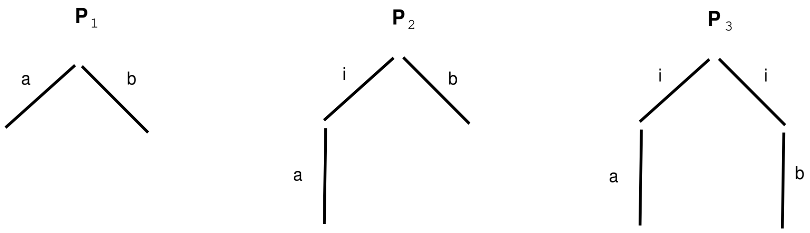
Whether or not such an internal event is unobservable is debatable, and perhaps such events mark the difference between active systems as opposed to reactive systems - the latter often modelled using a process algebra. In an active system events can be under the control of the system but not the environment (e.g. an alarm operation), such events are internal but can have observable effects (such as an alarm notification). This differs from the notion of internal in a process algebra, which equates internal with no observable transition, including output. In such an interpretation the operation IOp defined above would not be internal as we can observe its occurrence via its output, and the term *active* used in [19] could be used instead. However, the theory of weak refinement developed here is equally applicable to such a class of events.

6 Properties

6.1 Reducing non-determinism

An important aspect of refinement, in both the sequential and concurrent worlds, is the ability to strengthen an implementation by reducing the non-determinism in the abstract specification. Indeed this is a property of standard Z refinement in the absence of internal operations. Adding internal operations in a specification has introduced an additional form of non-determinism into the language. We shall see that weak-refinement allows us to reduce this type of non-determinism by removing internal operations.

Consider the behaviours described by the following transition diagrams, where a and b are observable events, and i represents an internal operation:



These specifications are not equivalent in any sense, for example in a process algebraic setting none of them are weak bisimulation equivalent. However, we would like a refinement to remove the non-determinism which is present in terms of the internal events, and for P_1 to refine P_2 which in turn refines P_3. Indeed, seen as processes they are related in the sense that, for example, P_1 **red** P_2 **red** P_3, where **red** is the LOTOS reduction relation [2].

Weak refinement, which we denote $\sqsubseteq_w$, also exhibits this property, that is $P_3 \sqsubseteq_w P_2 \sqsubseteq_w P_1$, but $P_1 \not\sqsubseteq_w P_2 \not\sqsubseteq_w P_3$. In terms of Z specifications we are giving these diagrams their obvious interpretation, for example, a Z specification of behaviour P_2 would be given by (the internal operation here is i, all the others are observable):

State

$state : \{0, 1, 2, 3\}$

Init

$\Delta State$

$state' = 0$

a

$\Delta State$

$state = 1 \land state' = 3$

b

$\Delta State$

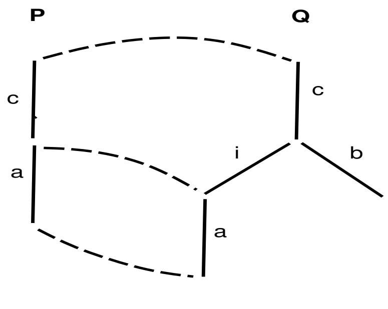
$state = 0 \land state' = 2$

i

$\Delta State$

$state = 0 \land state' = 1$

A slightly more complex example is given by the two behaviours defined by the following, where again the event i is internal and all others are observable.



Interpreted as Z specifications we find that P is a weak refinement of Q. This example is interesting because by resolving the non-determinism, the implementation never offers the operation b. The retrieve relation which shows this is a weak refinement is given by the dotted lines in the above diagram. Because pre $b \wedge Ret$ has predicate which is false, b can be implemented by any operation in the concrete specification (e.g. $\Xi State$ will do).

Notice that, as one would hope, Q is not a weak refinement of P, because we have to quantify over *all* paths of internal operations in the concrete specification in the correctness criteria for weak refinement.

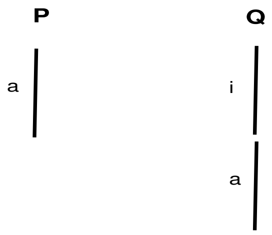
6.2 Weak refinement and refinement

In specifications without internal operations, refinement and weak refinement clearly coincide. In the presence of internal operations, neither implies the other. Since our motivation in defining weak refinement was to rule out some "refinements" of internal operations, refinement doesn't imply weak refinement (the protocol specifications provided an example of this).

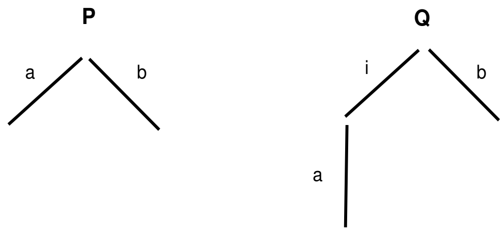
However, neither does weak refinement imply standard Z refinement. The last example given above exhibits a weak refinement (P is a weak refinement of Q), which does not have a retrieve relation which will define a standard refinement between them.

One desirable property that standard Z refinement possesses is that it is a congruence. That is, if specification S is refined by S', then in any context $C[.]$, $C[S']$ refines $C[S]$. A consequence of this is that operations can be refined individually and the whole specification is then a refinement of the original.

However, weak refinement is not a congruence, due to the presence of internal operations. To see this consider the two specifications given by the following behaviours:



Then under weak refinement these are equivalent, i.e. $P \sqsubseteq_w Q$ and $Q \sqsubseteq_w P$. However, if we add just one further operation to each specification which is applicable at the initial state, i.e. we specify the behaviour



then, as we observed earlier, Q is not a weak refinement of P. So congruence is lost with weak refinement. Incidentally, this counter-example is the same example that shows weak bisimulation is not a congruence in a process algebra, so the result here is not surprising and the ability to find observational relations which *are* congruences can be non-trivial.

7 Conclusions

The motivation for this work arose out of our interest in the use of Z for the specification of distributed systems, and in particular its use within the Open Distributed Processing (ODP) standardization initiative [12]. ODP is a joint standardisation activity of the ISO and ITU. A reference model has been defined which describes an architecture for building *open* distributed systems. Central to this architecture is a *viewpoints* model. This enables distributed systems to be described from a number of different perspectives. There are five viewpoints: enterprise, information, computational, engineering and technology. Requirements and specifications of an ODP system can be made from any of these viewpoints. Z and LOTOS are strong candidates for use in some of the ODP viewpoints, Z for the information viewpoint and LOTOS for the computational and engineering viewpoints.

The use of different viewpoints specified in different languages means we have to have mechanisms to check for the *consistency* of specifications. One aspect of this work has been the development of means to check for the consistency of two Z specifications [1], and a means to translate LOTOS specifications into Z [9]. Development of viewpoints written in different languages will be undertaken using different refinement relations, and this led to the need to develop a notion of weak-refinement in Z which is related to refinements in LOTOS. A full discussion of the relationships between the differing refinement relations is given in [8] (which incidentally assumes the firing condition interpretation discussed above).

Work related to that discussed here is that of Strulo in [19]. His proposal has much greater simplicity than that discussed here, however, it perhaps lacks full generality and involves a different interpretation of the pre-condition of an operation. Our aim here was to generalise standard Z refinement to deal with internal operations in a fully general manner, whilst maintaining the established interpretation of operations in Z.

References

1. E. Boiten, J. Derrick, H. Bowman, and M. Steen. Consistency and refinement for partial specification in Z. In M.-C. Gaudel and J. Woodcock, editors, *FME'96: Industrial Benefit of Formal Methods, Third International Symposium of Formal Methods Europe*, volume 1051 of *Lecture Notes in Computer Science*, pages 287–306. Springer-Verlag, March 1996.
2. T. Bolognesi and E. Brinksma. Introduction to the ISO Specification Language LOTOS. *Computer Networks and ISDN Systems*, 14(1):25–59, 1988.

3. E. Brinksma. A theory for the derivation of tests. In S. Aggarwal and K. Sabnani, editors, *Protocol Specification, Testing and Verification, VIII*, pages 63–74, Atlantic City, USA, June 1988. North-Holland.
4. E. Brinksma, G. Scollo, and C. Steenbergen. Process specification, their implementation and their tests. In B. Sarikaya and G. v. Bochmann, editors, *Protocol Specification, Testing and Verification, VI*, pages 349–360, Montreal, Canada, June 1986. North-Holland.
5. E. Cusack. Object oriented modelling in Z for Open Distributed Systems. In J. de Meer, V. Heymer, and R. Roth, editors, *IFIP TC6 International Workshop on Open Distributed Processing*, pages 167–178, Berlin, Germany, September 1991. North-Holland.
6. E. Cusack and G. H. B. Rafsanjani. ZEST. In S. Stepney, R. Barden, and D. Cooper, editors, *Object Orientation in Z*, Workshops in Computing, pages 113–126. Springer-Verlag, 1992.
7. E. Cusack and C. Wezeman. Deriving tests for objects specified in Z. In J. P. Bowen and J. E. Nicholls, editors, *Seventh Annual Z User Workshop*, pages 180–195, London, December 1992. Springer-Verlag.
8. J. Derrick, H. Bowman, E. Boiten, and M. Steen. Comparing LOTOS and Z refinement relations. In *FORTE/PSTV'96*, pages 501–516, Kaiserslautern, Germany, October 1996. Chapman & Hall.
9. J. Derrick, E.A.Boiten, H. Bowman, and M. Steen. Supporting ODP - translating LOTOS to Z. In *First IFIP International workshop on Formal Methods for Open Object-based Distributed Systems*, Paris, March 1996. Chapman & Hall.
10. M. Hennessy. *Algebraic Theory of Processes*. MIT Press, 1988.
11. C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.
12. ITU Recommendation X.901-904 — ISO/IEC 10746 1-4. *Open Distributed Processing - Reference Model - Parts 1-4*, July 1995.
13. L. Lamport. TLZ. In J.P. Bowen and J.A. Hall, editors, *ZUM'94, Z User Workshop*, pages 267–268, Cambridge, United Kingdom, June 1994.
14. P. Mataga and P. Zave. Formal specification of telephone features. In J. P. Bowen and J. A. Hall, editors, *Eighth Annual Z User Workshop*, pages 29–50, Cambridge, July 1994. Springer-Verlag.
15. R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.
16. G. H. B. Rafsanjani. ZEST - Z Extended with Structuring: A users's guide. Technical report, BT, June 1994.
17. S. Rudkin. Modelling information objects in Z. In J. de Meer, V. Heymer, and R. Roth, editors, *IFIP TC6 International Workshop on Open Distributed Processing*, pages 267–280, Berlin, Germany, September 1991. North-Holland.
18. J. M. Spivey. *The Z notation: A reference manual*. Prentice Hall, 1989.
19. B. Strulo. How firing conditions help inheritance. In J. P. Bowen and M. G. Hinchey, editors, *Ninth Annual Z User Workshop*, LNCS 967, pages 264–275, Limerick, September 1995. Springer-Verlag.
20. C. Wezeman and A. J. Judge. Z for managed objects. In J. P. Bowen and J. A. Hall, editors, *Eighth Annual Z User Workshop*, pages 108–119, Cambridge, July 1994. Springer-Verlag.
21. J. Woodcock and J. Davies. *Using Z: Specification, Refinement, and Proof*. Prentice Hall, 1996.

This article was processed using the LaTeX macro package with LLNCS style