Retrenchment and the Atomicity Pattern

Richard Banach,
Czeslaw Jeske
School of Computer Science,
University of Manchester,
Manchester M13 9PL, UK
{banach,cj}@cs.man.ac.uk

Anthony Hall
Independent Consultant,
United Kingdom
anthony@anthonyhall.org

Susan Stepney
Dept. of Computer Science,
University of York,
Heslington,
York YO10 5DD, UK
susan.stepney@cs.york.ac.uk

Abstract

The issues surrounding the question of atomicity, both in the past and nowadays, are briefly reviewed, and a picture of an ACID (atomic, consistent, isolated, durable) transaction as a refinement problem is presented. An example of a simple air traffic control system is introduced, and the discrepancies that can arise when read-only operations examine the state at atomic and finegrained levels are handled by retrenchment. Non-ACID timing aspects of the ATC example are also handled by retrenchment, and the treatment is generalised as the retrenchment Atomicity Pattern. The utility of the pattern is confirmed against a different case study, the Mondex Electronic Purse.

1 Introduction

Atomicity is by no means a new issue in the design of computer systems: insights about mutual exclusion primitives and their consequences have developed since the earliest days of the subject [12, 33, 30, 29]. The deepening understanding of atomicity mechanisms led to the development of efficient distributed operating systems [38, 24, 18], and bolstered with maturing knowledge about data representation, has led to the flowering of a database industry offering products which are sufficiently reliable, and sufficiently easily usable, that they now occupy mission critical positions in many organisations [13, 22, 20, 16, 28, 41]. The vast information resources available on the web provide ever increasing opportunities for applications in all spheres to benefit from a distributed approach.

The watchword of the implementation of an atomic action in these pardigms is the ACID (atomic, consistent, isolated, durable) transaction [25]. This provides the default goal to which implementations now aspire, providing the maximum possible conceptual clarity for the transaction concept, convenient for higher levels of applications.

At the heart of the atomicity question is some notion of refinement. One has a picture of a task, performed atomically at the abstract level, but broken up into fragments, usually co-operating via a protocol, at a more concrete level. The two are supposed to achieve the same ends, so the concrete ought to be some sort of refinement of the abstract. We do not debate here the optimal formulations of refinement for this: in fact many adequate possibilities exist.

In the ideal ACID-refinement-based formulation, the protocol always either runs to a successful conclusion, or the whole attempt gets wiped, leaving no trace. Nowadays however, although the ACID ideal is still highly prized, the necessity to relax some or all of its precepts to avoid excessive performance penalties is widely recognised. This is prompted by scenarios such as web services, long-lived workflows, highly concurrent and highly distributed environments.

Our aim in this paper is to illustrate the capabilities of retrenchment [8, 9, 31, 10, 32] in dealing with the various kinds of circumstance which arise that can spoil the ideal ACID based refinement view of atomicity. On the one hand, the 'atomic action refined to distributed algorithm' perspective generates a strong pull to find common structure across many such situations. On the other, the necessity of departure from the ideal can arise in a myriad ways, leading to a proliferation of incompatible special cases if a common account is pitched at an inappropriate level. It turns out that retrenchment can provide a vehicle for capturing a useful degree of commonality across such situations, while leaving room for incompatibilities regarding specific details.

The commonality arises via the retrenchment *Atomicity Pattern*, which we introduce in this paper. This is an arrangement of refinements and retrenchments that we show is common to atomicity situations. The idea is developed in the remainder of this paper as follows. In Section 2 we introduce our main example, a pidgin air traffic control (ATC) display application, abstracted from the CDIS development [23], in which critical pieces of information must reach a

family of displays in the correct order (and ultimately in a timely fashion). An 'ideal specification' of this is atomic; a 'more realistic specification' captures some of the realities of asynchrony and of constraints on timing delays. The gap between them is bridged by a series of small retrenchments. (The small size of the model evolution steps is rather reminiscent of the Event-B approach [1, 2], and of many practical ASM refinements [15, 14].) In Section 3 we indicate how one might proceed towards an implementation from such a starting point. In Section 4 we abstract from the phenomena introduced thus far, into a general structure for capturing a whole class of similar situations involving atomicity, its refinement and its possible breakdown: the *Atomicity Pattern* itself. In Section 5 we confront the *Atomicity Pattern* with a different scenario, namely the atomicity issues arising in the Mondex Purse [39], and we see that the proposed structure can account adequately for the phenomena in Mondex. Section 6 recapitulates and concludes.

N.B. We express all our models using the Z notation, for its relative conciseness (which we amplify by taking occasional notational liberties). An Appendix summarises the refinement and retrenchment rules we use.

2 The Abstract Pidgin ATC System

In an air traffic control system, there is (among other things) a family of workstations *WS*, at which the air traffic controllers sit and do their work. The workstations display a variety of items of information to the controllers, some of them more critical than others. For the most critical items, correct ordering and timeliness are important issues: the controllers must be made aware of changes in the critical items as they occur, within a tightly controlled *LATENCY*, so that safety in the aerodrome is not compromised.

For simplicity, we assume that there is just one critical item, the *QNH* value (modelled as a natural number say), and that this is the only item on the workstation display. For an air traffic control system this is, admittedly, a rather drastic simplification.

The system ideal is atomic update of all the displays, so we can build an abstract A model consisting of a single *QNH* value, updated by an *ANewQnh* operation, and observed by an *AShowWs* operation which outputs the *QNH* value displayed on each workstation:[1]

$$\begin{array}{|l} \textit{Aworld} \\ \hline \textit{Aqnh} : \textit{QNH} \\ \hline \end{array}$$

$$\begin{array}{|l} \textit{ANewQnh} \\ \hline \Delta \textit{Aworld} \\ \textit{Aqnh}? : \textit{QNH} \\ \hline \textit{Aqnh}' = \textit{Aqnh}? \\ \hline \end{array} \qquad \begin{array}{|l} \textit{AShowWs} \\ \hline \textit{Aworld} \\ \textit{Adisp}! : \textit{WS} \to \textit{QNH} \\ \hline \textit{Adisp}! = \textit{WS} \times \{\textit{Aqnh}\} \\ \hline \end{array}$$

The atomicity of the ideal model is reflected in the fact that *AShowWs* always outputs a constant function. However, the reality is that the updates to the system *QNH* value are broadcast to the individual workstations over a network. This generates transmission delays, and the various tolerances in the system cause these to be observable, within limits. These aspects require a more detailed model than the ideal A model, and we approach the construction of the appropriate 'realistic specification' in a number of steps.

First we build the AH model, which just includes history information:

$$\begin{array}{|l} \textit{AHworld} \\ \hline \textit{AHhist} : \mathrm{seq}_1\ \textit{QNH} \\ \hline \end{array}$$

$$\begin{array}{|l} \textit{AHNewQnh} \\ \hline \Delta \textit{AHworld} \\ \textit{AHqnh}? : \textit{QNH} \\ \hline \textit{AHhist}' = \\ \quad \textit{AHhist} \frown \langle \textit{AHqnh}? \rangle \\ \hline \end{array} \qquad \begin{array}{|l} \textit{AHShowWs} \\ \hline \textit{AHworld} \\ \textit{AHdisp}! : \textit{WS} \to \textit{QNH} \\ \hline \textit{AHdisp}! = \\ \quad \textit{WS} \times \{\textit{last}\ \textit{AHhist}\} \\ \hline \end{array}$$

With suitable initialisations, it is not hard to see that the A and AH models are interrefinable under (equality output relations and) the retrieve relation:

$$\begin{array}{|l} R_{A,AH} \\ \hline \textit{Aworld} \\ \textit{AHworld} \\ \hline \textit{Aqnh} = \textit{last}\ \textit{AHhist} \\ \hline \end{array}$$

Next, we build the AA model, which introduces asynchrony by allowing the *AAShowWs* operation to output a selection of values from the history; the only restriction being that displaying *QNH* values out of order is forbidden. The latter is achieved by keeping a record of the current *QNH* value for each workstation, and allowing it to advance using the operation *AAWsUpdate*:[2]

$$\begin{array}{|l} \textit{AAworld} \\ \hline \textit{AAhist} : \mathrm{seq}_1\ \textit{QNH} \\ \textit{AAseq} : \textit{WS} \to \mathbb{N} \\ \hline \mathrm{ran}\ \textit{AAseq} \subseteq \\ \quad 1 \mathinner{..} \#\textit{AAhist} \\ \hline \end{array} \qquad \begin{array}{|l} \textit{AAShowWs} \\ \hline \textit{AAworld} \\ \textit{AAdisp}! : \textit{WS} \to \textit{QNH} \\ \hline \textit{AAdisp}! = \\ \quad \textit{AAseq} \fatsemi \textit{AAhist} \\ \hline \end{array}$$

[1] N.B. Each model-specific schema and variable is prefixed by a letter or two indicating the relevant model: A for abstract, AH for A with history, etc.

[2] Henceforth, the phrase RestSame..... means that any other variables in scope but not explicitly assigned to are to remain unchanged, something handled less tersely in legal Z.

AANewQnh

$\Delta AAWorld$
$AAqnh? : QNH$

$AAhist' =$
$\quad AAhist \frown \langle AAqnh? \rangle$
RestSame.....

AAWsUpdate

$\Delta AAworld$
$AAws? : WS$

$(AAseq'\ AAws?) \geq$
$\quad (AAseq\ AAws?)$
RestSame.....

This time conventional refinement is too demanding a notion to describe the relationship between AH and AA, since the outputs of the *ShowWs* operations do not match up.[3] We need the greater flexibility of retrenchment to capture what is going on. The retrenchment needed is given by the retrieve relation $R_{AH,AA}$ and the output relation for *ShowWs*, with all other retrenchment data trivial:[4]

$O_{AH,AA,ShowWs}$

$AHworld'$
$AAworld'$
$AHdisp! : WS \rightarrow QNH$
$AAdisp! : WS \rightarrow QNH$

$AAdisp! = AAseq' \fatsemi AAhist'$
$AHdisp! = WS \times \{last\ AAhist'\}$

$R_{AH,AA}$

$AHworld$
$AAworld$

$AHhist = AAhist$

A retrenchment output relation generalises a refinement one, in the sense that it can refer to the state variables in relating the observed outputs, whereas a refinement output relation can't. This extra flexibility is needed here: the AH output is the constant value resulting from the latest atomic update, whereas the AA output is a selection of potentially older values from the system history, since there may be some workstations which are not completely up to date.

The retrenchment just introduced is of a special kind. We call it a retrenchment-enhanced refinement (RE-Ref), since it falls short of being a refinement by the smallest of margins. Thus, if we removed the outputs from the systems, we would have a perfectly good refinement, and the observed relationship between the outputs would be derivable from it. Putting it another way, if it is only asynchrony that a change of model is introducing, then the true information contained in the state histories is not being lost, so any disagreement in the outputs of read-only operations on the states, should be explicable from the state histories; i.e. the change of model indeed ought to be capable of being described by an RE-Ref. Of course, similar observations apply when it is the inputs at stake rather than outputs. In that case the asynchrony considerations mean that corresponding inputs arrive at different times in the two models. This can be handled in the within relation of the later occurring operation; a case in point is to be found in Section 5. The utility of RE-Refs in handling issues of asynchrony is the first contribution that retrenchment makes to the atomicity arena.

The next step is to introduce the time aspect, so that we can bring the asynchrony under control. This leads to the AT model, containing an *ATtimenow* variable with values in *TIME* (which is the naturals say), and which is updated by an *ATTick* operation. Also each update of the *QNH* value is timestamped in the *AThisttime* variable:

ATworld

$AThist : \mathrm{seq}_1\ QNH$
$ATseq : WS \rightarrow \mathbb{N}$
$ATtimenow : TIME$
$AThisttime : \mathrm{seq}_1\ TIME$

$\#AThist = \#AThisttime$
$\mathrm{ran}\ ATseq \subseteq 1\ ..\ \#AAhist$
$\forall i, j : \mathrm{dom}\ AThisttime \bullet i \leq j \mid$
$\quad AThisttime(i) \leq AThisttime(j)$
$last\ AThisttime \leq ATtimenow$

ATTick

$\Delta ATworld$

$ATtimenow' =$
$\quad ATtimenow + 1$
RestSame.....

ATNewQnh

$\Delta ATworld$
$ATqnh? : QNH$

$AThist' = AThist \frown \langle ATqnh? \rangle$
$AThisttime' = AThisttime \frown \langle ATtimenow \rangle$
RestSame.....

ATWsUpdate

$\Delta ATworld$
$ATws? : WS$

$(ATseq'\ ATws?) \geq$
$\quad (ATseq\ ATws?)$
RestSame.....

ATShowWs

$ATworld$
$ATdisp! : WS \rightarrow QNH$

$ATdisp! = ATseq \fatsemi AThist$

Thus far we have a straightforward superposition refinement [4, 21, 27] of the AA model, since we have just added some new data and operations (and no new observations of the new data), old ones remaining unchanged. A retrieve relation that simply forgets the time in abstracting from the AT model easily proves it. However this is not enough. We need to distinguish well behaved workstations from badly behaved ones. The former get their updates done within *LATENCY* timesteps, the others don't. A well behaved workstation satisfies:

ATWellBhWs

$ATworld$
$ws? : WS$

$\forall sq : (ATseq\ ws?) + 1\ ..\ \#AThist \bullet$
$\quad ATtimenow - (AThisttime\ sq) \leq LATENCY$

[3] Also, the AA operation *AAWsUpdate*, does not correspond to any AH model operation. However, we can allow this in a refinement if the new operation refines an (unstated) AH operation whose body is skip. Since *AAWsUpdate* only manipulates the *AA* variable *AAseq*, which is invisible (via the retrieve relation $R_{AH,AA}$) to the *AH* model, it does indeed refine skip. See the Appendix.

[4] I.e. given by identities on inputs and outputs, false for concessions.

i.e. all its unprocessed updates were introduced less than *LATENCY* ago. We want the refinement part of the eventual relationship between the AA and AT models to insist that all workstations are well behaved:[5]

$$
\begin{array}{|l}
\hline
R_{AA,AT} \\
AAworld \\
ATworld \\
\hline
AAworld \text{ “ = ” } ATworld \\
\forall\, ws? : WS \bullet ATWellBhWs \\
\hline
\end{array}
$$

To deal with the (small but nonzero) possibility that network delays turn out to be greater than desirable, leading to the failure of the retrieve relation, we need the full power of retrenchment. It is actually the innocuous *ATtick* operation we need to focus on, since it is the passage of time which causes workstations to become badly behaved. At this point we stub our toe on a small retrenchment pebble.

Since there is no *Tick* operation in the AA model, normal retrenchment policy dictates that there will be no retrenchment data (i.e. within, output or concedes relations) associated with *Ttick*. The normal policy is justified by observing that genuinely new operations introduced during a model evolution step, will concern aspects absent from the prior model, and thus any attempt to relate them to the prior model are likely to appear artificial. However, the passage of time may reasonably be taken as a universal (if usually unstated) feature of models, so that viewing the present case as a retrenchment of an unstated skip is entirely justified. This understood, the retrenchment's within and output relations can be trivial, the concession being where the interest lies:

$$
\begin{array}{|l}
\hline
C_{AA,AT,Tick} \\
\Delta AAworld \\
\Delta ATworld \\
\hline
\forall\, ws? : WS \bullet \exists\, sq : (ATseqws?) + 1 \mathinner{..} \#AThist \bullet \\
\quad ATtimenow - (AThisttime\ sq) = LATENCY \\
\qquad \Rightarrow \neg\, ATWellBhWs' \\
\hline
\end{array}
$$

This shows that any workstation with a *LATENCY*-old update outstanding, will become badly behaved at the next tick unless it is updated beforehand.

We now have a route from the utterly atomic A model, to model AT, which abstracts the inevitable asynchrony of an implementation, but which allows the quality of that asynchrony to be quantified via a retrenchment. The A model specifies an unattainable perfection, while the AT model represents a more complex but more realistic specification.

[5] We use “ = ” between schemas to abbreviate a set of equalities between corresponding variables that differ only in the model-identifying prefix.

Refinement alone can never reconcile these two widely separated viewpoints, but retrenchment can.

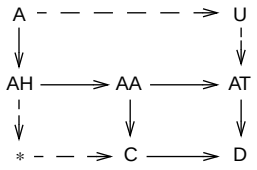

Figure 1. Models, refinements (vertical arrows), retrenchments (horizontal arrows), making up a commuting diagram of the ATC specification development.

The retrenchment from A to AT itself is the composition of the A-AH refinement with the AH-AA and AA-AT retrenchments [6]. We omit the details of the calculation, save to say that the situation is sufficiently straightforward, that the result is obtained by simply translating the variables occurring in the non-trivial bits of the earlier retrenchments to those of the A and AT models, in the obvious way.

The retrenchment utilises the *LATENCY* parameter and permits a stochastic analysis of the circumstances under which the relevant concession becomes valid. Such an analysis can provide a useful negotiating pivot between customer and supplier — the customer would be interested in a precise statement of what constituted timing failure and how often it occurred, but the details of what happened subsequently would be more a matter for the supplier, taking into account the higher level invariants demanded of the system. This scenario illustrates in minature the second contribution that retrenchment makes to the atomicity issue, namely the straightforward incorporation into a formal account, of matters that make implementations of atomic actions insufficiently ACIDic.

What we have so far is the solid arrows of the upper layer of Fig. 1, which is a commuting diagram of verical refinements and horizontal retrenchments. These connect a family of models involved in our ATC development, abstracted from CDIS [23]. Customarily, the refinement component of the A-AT retrenchment (i.e. A-AH) would enable the A-AT retrenchment to be lifted to generate a more abstract model U using results in [26]. However the fact that A and AH are interrefinable, means that nothing useful would be gained by doing this.

3 Towards an Implementation of the Pidgin ATC System

Considering the move towards an implementation of the ATC System, we refine our preceding models. We start with model AA, since that is the first along the A-AT path which

incorporates asynchrony, which is unavoidable in any implementation. For lack of space our remarks will be merely indicative rather than comprehensive. We sketch model C, a refinement of AA. Here is its state schema:

Cworld

$$
\begin{array}{l}
Cmaxseq : SQNO \\
Cwsseq : WS \rightarrow SQNO \\
Cwsqnh : WS \rightarrow QNH \\
Cethqnh : SQNO \nrightarrow QNH \\
\hline
\mathrm{dom}\, Cethqnh = 1 \mathinner{..} Cmaxseq \\
Cwsqnh = Cwsseq \fatsemi Cethqnh
\end{array}
$$

Model C's 'more realistic' description of the system contains a family of workstations, each containing its own portion of the system state; hence the map *Cwsqnh*.[6] The network is modelled as an 'ether' of messages containing *QNH* updates, to which individual workstations help themselves. To disambiguate and preserve order, we have a sequence number type *SQNO* (modelled as a positive natural number say), and the ether thus becomes a map *Cethqnh* from sequence numbers to *QNH* values. Each workstation keeps track of where it is up to with a local copy of the latest sequence number it has processed (via the map *Cwsseq*).

Noting that $SQNO == \mathbb{N}_1$, the reader will quickly realise that this is (mathematically) little more than a slightly more verbose restatement of the state schema of the AA model, with an additional dependent variable, *Cwsqnh*. Recognising this, if we now map the operations in the obvious way (details omitted), we conclude that the given C will be interrefinable with AA. Obviously we could contemplate more dramatic refinements of the AA model, but what we have will do for purposes of illustration.

Similarly, we can build a model D, refining AT. Here is its state schema, again omitting the operations:

Dworld

$$
\begin{array}{l}
Dmaxseq : SQNO \\
Dwsseq : WS \rightarrow SQNO \\
Dwsqnh : WS \rightarrow QNH \\
Dethqnh : SQNO \nrightarrow QNH \\
Dtimenow : TIME \\
Dethtime : SQNO \nrightarrow TIME \\
\hline
\mathrm{dom}\, Dethtime = \mathrm{dom}\, Dethqnh = 1 \mathinner{..} Dmaxseq \\
Dwsqnh = Dwsseq \fatsemi Dethqnh
\end{array}
$$

The two refinements AA-C and AT-D will be related by not only the AA-AT retrenchment, but a retrenchment C-D. This latter retrenchment will be the obvious counterpart of the AA-AT retrenchment at the (supposedly) lower level of abstraction of C and D. Done properly, it all yields a commuting square of retrenchments and refinements in the lower right half of Fig. 1. In fact, not only can one build this commuting square independently, but one can generate the D model and the AT-D and C-D constructions (up to interrefinability) using the Postjoin Theorem from [26]. Obviously, one could develop the C and D models further towards implementation by making the modelling increasingly realistic.

The above takes care of the lower layer of Fig. 1, aside from the model labelled '*'. Model '*' refines AH and is retrenchable to C. It can be obtained via the lowering construction in [26] from AH, AA, C and their relationships, or independently, again yielding a commuting square. The fact that AA and C are interrefinable, means that '*' contains nothing new beyond AH, and the fact that its workstation updates must be atomic, means that it is unrealistic.[7]

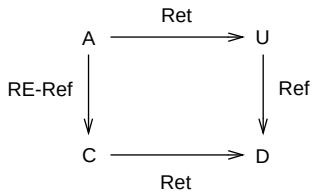

Figure 2. The Atomicity Pattern

4 The Retrenchment Atomicity Pattern

The last few remarks indicate that a protocol implementation at the most abstract level possible[8] has to be refinable from C, not from A. And yet A captures the most transparent expression of what one would like the protocol to do, so it would be regrettable to exclude it from a rigorous development. The way to reconcile these views, is to pursue the suggestion that an RE-Ref can indeed be usefully viewed as a kind of refinement, rather than as a retrenchment, which, strictly speaking, it is. This straightens out the composition along the path A-AH-AA-C, into an RE-Ref, collapsing the left hand part of Fig. 1. The resulting RE-Ref from A to C now expresses a useful change in modelling perspective, in the vertical direction, as an almost-refinement. Moreover, incorporating the ACIDity losing aspects of the retrenchment C-D via the composition A-C-D and then performing the lifting construction from [26], results in a model U entirely equivalent to the one constructed before, since the overall composition A-C-D yields the same overall composed retrenchment from A to D as before. This is a useful observation since it is often initially easier to express the

[6] This is best captured formally via Z promotion, though for brevity we will not use that here.

[7] Unrealistic because of the intepretation of the model as a distributed system, rather than any mathematical difficulty.

[8] I.e. incorporating the fewest constraints while retaining implementability.

ACIDity losing aspects in a more concrete model than in a more abstract one. The collection of models A, C, D, U, and their interrelationships constitute the retrenchment *Atomicity Pattern*, instantiated for this particular example; see Fig. 2.

Referring to Fig. 1, we see that it is an instance of the *Tower Pattern* [11]. This makes the *Atomicity Pattern* a special case of the *Tower*. However, a number of features make the *Atomicity Pattern* deserve to be singled out specially.

First and foremost is the use of RE-Refs (specifically avoiding more general kinds of retrenchment) in the left hand side of the diagram — this collapses the zig-zag that would result if the retrenchment aspects were singled out as such there. Pure loss of atomicity implies that abstract and concrete states can be adrift of the ideal part way through the protocol, but only in a very controlled way, since the concrete is still a refinement of the abstract in the absence of I/O. This close, yet non-ideal relationship between the states, means that observed inputs and outputs which closely reflect those state values will also be in a close, yet non-ideal relationship. For this reason, the very restricted RE-Refs are sufficient for this kind of situation.

Second, is the fact that the fairly large gap between the A and C models encourages us to take a broader perspective on how an atomic and a non-atomic model can be synchronised. The synchronisation mechanism is in fact captured in the retrieve relation between the two models. Considering our example, the finegrained path A-AH-AA-C strongly suggests an early synchronisation; i.e. in each protocol run, *ANewQnh* is synchronised via the retrieve relation with *CNewQnh*,[9] with the rest of the concrete protocol following behind. This gives an A-C refinement with retrieve relation $R_{A,C_{early}}$ below. However, this is but one possibility.

In general, the single step of an atomic protocol can be mapped to practically any step of a concrete protocol which implements the atomic one, provided the various (in general nondeterministic) outcomes of the two descriptions match up via the retrieve relation. Different choices merely lead to different retrieve relations between the two models.[10] As an example, consider a late synchronisation option in our CDIS example. This matches *ANewQnh* with the last *CWsUpdate* in a protocol run, identified via $CWsUpdate_{last}$ below, and implicitly requires that *CNewQnh* and all earlier occurrences of *CWsUpdate* become refinements of abstract skips. Such a synchronisation is given by retrieve relation $R_{A,C_{late}}$.

$R_{A,C_{early}}$

$$Aworld$$
$$Cworld$$
$$Aqnh = Cethqnh\ Cmaxseq$$

$R_{A,C_{late}}$

$$Aworld$$
$$Cworld$$
$$Aqnh = Cethqnh(min\,\mathrm{ran}\,Cwsseq)$$

$CWsUpdate_{last}$

$$CWsUpdate$$
$$\forall ws : WS \bullet ws \neq ws? \Rightarrow (Cwsseq\ ws) > (Cwsseq\ ws?)$$

(N.B. In our example, both early and late formulations of the refinement are *forward* simulations, since the broadcast protocol is deterministic; i.e. all the workstations always get successfully updated (assuming weak fairness). In general, early synchronisation requires *backward* simulation to handle nondeterminism after the synchronisation point.)

Third, is a fact prompted by the preceding parenthetic remark. The detailed complexities of simulations in which the concrete state is matched to the abstract state after each concrete step of the protocol, can be largely avoided if we take a more coarse grained approach to refinement, à la ASM refinement [15, 14, 34, 35]. Here, the refinement becomes insensitive to state values in the middle of a concrete protocol run, and the retrieve relation is only required to match up abstract and concrete states at the beginning and end. This *en bloc* approach can yield considerable simplifications in the description of a single run of the protocol, but makes the description of interleaved concurrent protocol runs by independent agents rather more problematic.

In our example, *ANewQnh* together with a suitable collection of *AShowW*ss would be refined *en bloc* to an entire concrete protocol run with suitable *CShowW*ss interspersed. Done properly, this would make the previously observed discrepancies between abstract and concrete outputs disappear, since the coarser grain would enable us to schedule the abstract and concrete *ShowW*ss so that they matched up, the details of the scheduling being concealed in the interior of the coarse grained refinement. We do not give the details here, due to the technical complexity of dealing with the interleavings of independent updates. This approach gives further encouragement to the view that an RE-Ref is after all a species of refinement.

5 The Mondex Purse

Having developed the *Atomicity Pattern*, in this section we confront it with a different but nevertheless realistically

[9] Many conventional refinement notions demand that abstract operations are refined by operations *with the same name*. However this is just a technical convenience, and is easily generalised to the case where for each concrete step of interest, one can identify a step of *some* abstract operation of which it is a refinement; official Z refinement is like this. Our discussion presupposes this generalisation where necessary.

[10] Observe an interesting phenomenon. When a refinement preserves the atomicity, i.e. abstract and concrete steps match up 1–1 in related runs, it is usually the case that the retrieve relation is 'obvious': there is essentially only one choice that makes sense. The situation changes dramatically when atomicity is *not* preserved. Then, the variety of synchronisations leads to a variety of retrieve relations, and rarely are any of them 'obvious'.

grounded example, the Mondex Purse, to verfy the genericity of the description of atomicity situations that it furnishes.

The Mondex Purse is a smartcard electronic purse for containing genuine money, and as such, is a security critical application. The 1990s development of Mondex was the first such development to achieve the highest possible ITSEC rating of E6, equivalent these days to a Common Criteria rating of EAL7 [19]. The ITSEC E6 rating requires there to be an abstract model, a concrete model, and a proof of correspondence between them; for Mondex, the proof was a manual refinement proof between abstract and concrete Z models. The details of the Mondex project are commercially sensitive; however a public version was produced [39]. The development in [39] remains an impressive achievement, and a trailblazer for showing that fully formal techniques could be applied within realistic time and cost limitations on industrial scale applications. More recently, the Mondex refinement proof has been re-examined using various mechanised approaches; see [37, 36] and [40]. These not only attest to the viability of doing such developments in a fully mechanised manner, but also confirm the solidity of the original manual proof.

Despite the above, the exigiencies of refinement caused a number of issues to be treated in a less than ideal manner in Mondex. In the actual project, the treatment of these was via informal arguments, but a suitably formal treatment would obviously have been better. As such, Mondex provides a superb platform for testing out the efficacy of the retrenchment approach to handling situations which for refinement turn out to be awkward.

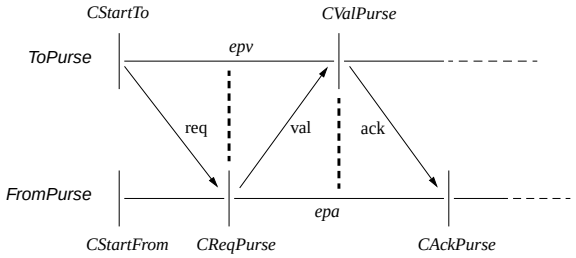

Figure 3. The Mondex Concrete Protocol

At the top level of Mondex is an abstract A model, which is a model of atomic funds transfer between purses. A transaction can: either complete successfully (lodging the funds transfered instantaneously in the destination purse); or atomically 'lose' the funds (placing them in a special 'lost' component of the state). Here are the essentials in a cut down world of only two purses, *ToPurse* and *FromPurse*, hardwired into the state, forbidding null transactions for simplicity:

Abworld

$$Afrombal, Afromlost, Atobal, Atolost : \mathbb{N}$$

AbTransferOkay

$$\Delta Abworld$$
$$Avalue? : \mathbb{N}$$

$$0 < Avalue? \leq Afrombal$$
$$Afrombal' = Afrombal - Avalue?$$
$$Atobal' = Atobal + Avalue?$$
$$Afromlost' = Afromlost$$
$$Atolost' = Atolost$$

AbTransferLost

$$\Delta Abworld$$
$$Avalue? : \mathbb{N}$$

$$0 < Avalue? \leq Afrombal$$
$$Afrombal' = Afrombal - Avalue?$$
$$Atobal' = Atobal$$
$$Afromlost' = Afromlost + Avalue?$$
$$Atolost' = Atolost$$

At the bottom level of Mondex is a concrete C model. This describes a protocol for secure funds transfer via message passing in which certain messages are assumed unforgeable. We do not have space to describe this in detail, but we can indicate how it works by reference to Fig. 3.

Two purse owners wishing to participate in a funds transfer insert their purses, the *FromPurse* and *ToPurse*, into an interface device, and type in the instructions. The device then initiates the funds transfer process by informing the two purses of the details of the required transaction. The two purses then autonomously initiate the transfer protocol. Each performs the appropriate *Start* operation, which initialises it for the transaction. As part of its initialisation the *ToPurse* sends a *req*uest message to the *FromPurse*. Having initialised, the *FromPurse* waits for the request. When it gets the request, the *FromPurse* decrements its balance and sends the money in a *val*ue message. Upon receipt of the value, the *ToPurse* increments its balance and sends an *ack*nowledgement message back to the *FromPurse*. The receipt of the acknowledgement by the *FromPurse* completes the protocol.

The above describes an unproblematic run of the protocol. Of course much can go wrong in practice. The protocol may get interrupted by accident or by design, and a purse may be subjected to deliberate attack in order to attempt to subvert its integrity (and in the ideal case, to increase the balance it contains beyond what is legitimate). The protocol must be robust against all this. Part of the protection built into the protocol is the fact that any time a purse feels like it, it has the option of doing nothing or of aborting the current transaction: this means that a purse will always respond to any request to perform any of its actions, but the response will be null or aborting if the purse does not consider the request to be appropriate in the context of its current state. This creates a large number of additional playouts of the protocol which are not illustrated in Fig. 3. Nevertheless it can be shown that all the possible playouts do indeed do the right thing, because the concrete protocol can be proved to be a refinement of the abstract A model, both for successful runs (which refine *AbTransferOkay*) and for aborting runs (which refine *AbTransferLost*). See [39] for the original account, and also [7], which discusses the properties of the

protocol in a manner compatible with the present discussion.

How does the preceding fit the *Atomicity Pattern*? Well, we have A and C models, and C refines A. Since A is atomic and C is not, any operation that reads and outputs the state values will exhibit a discrepancy if abstract and concrete versions are invoked at an inopportune moment; a balance enquiry operation is a case in point.[11] In [39] the abstract transfer is synchronised with the concrete *CReqPurse* operation, so the discrepancy shows up in a *ToPurse* enquiry if invoked while the value is in transit, i.e. between the vertical dashed lines in Fig. 3. In [7], a different refinement is given which synchronises the abstract transfer with the concrete *CValPurse* operation, which puts the discrepancy on the *FromPurse* side. Either way, there is scope for an AA model, introducing asynchrony at the most abstract level. The pros and cons of dealing with balance enquiries in various different ways are studied in depth in [7].

In [39], in between the A and C models there is a B model, but it is technically very close to the C model, so it would not play the role of the AA model that we have in mind. However it is very easy to construct a suitable AA model from scratch. We encode a transaction in progress via $AAval > 0$ for simplicity:

AAbworld

$AAfrombal : \mathbb{N}$
$AAfromlost : \mathbb{N}$
$AAtobal : \mathbb{N}$
$AAtolost : \mathbb{N}$
$AAval : \mathbb{N}$

AAbTransferStart

$\Delta AAbworld$
$AAvalue? : \mathbb{N}$

$AAval = 0$
$0 < AAvalue? \leq AAfrombal$
$AAval' = AAvalue?$
$AAfrombal' =$
$\quad AAfrombal - AAvalue?$
RestSame.....

AAbTransferOkay

$\Delta AAbworld$

$AAval > 0$
$AAval' = 0$
$AAtobal' =$
$\quad AAtobal + AAval$
RestSame.....

AAbTransferLost

$\Delta AAbworld$

$AAval > 0$
$AAval' = 0$
$AAfromlost' =$
$\quad AAfromlost + AAval$
RestSame.....

Note that this just separates out the beginning and end of a transaction, which were combined in the A model. The relationship between the A and AA models is evidently an RE-Ref, but, assuming we choose to synchronise the A model transaction late (i.e. the A model operation is synchronised with *AAbTransferOkay* or *AAbTransferLost*), the RE-Ref needs to relate discrepancies in inputs to the states via within relations, since late synchronisation implies that the AA model input occurs earlier than the A model input. In this scenario, we can use the simple retrieve relation $R_{A,AA}$ (where RestEqual..... has the obvious meaning), and within relations $W_{A,AA,TransferOkay}$ and $W_{A,AA,TransferLost}$ (whose bodies are identical):

$W_{A,AA,TransferOkay/Lost}$ ·

Abworld
AAbworld
Avalue?

$Avalue? = AAval$

$R_{A,AA}$

Abworld
AAbworld

$AAval =$
$\quad Afrombal - AAfrombal$
RestEqual.....

(N.B. If we synchronised early, though the inputs would coincide, we would need a more complex, nondeterministic, retrieve relation to intercede in what would need to be a backward simulation refinement as in [39].) Further down down the modelling hierarchy, one can relatively straightforwardly synchronise *AAbTransferStart* with the *CReqPurse* operation, *AAbTransferOkay* with *CValPurse*, and with a little further manipulation of the concrete state *AAbTransferLost* can be synchronised with a suitable *Abort*.

One can go further. With a more complex retrieve relation, one could synchronise *AAbTransferStart* with the first concrete *CStart*, *AAbTransferOkay* with *CAckPurse*, and *AAbTransferLost* with a suitable *Abort*. In the end, there are many choices. In a nutshell, the Mondex development fits the left hand side of the *Atomicity Pattern* like a glove.

What of the horizontal aspects of Fig. 2, which describe possible lack of ACIDity of the asynchronous protocol? In this regard, we note that Mondex is squarely in the financial world, where the merest whiff of 'alkalinity' in financial transactions is utterly intolerable. In fact the protocol maintains a sufficiently copious (electronic) papertrail, that aborted transactions, even though they do not achieve their original objective, can be traced, and the whereabouts of the funds they involve can ultimately be reconciled with the original intentions of the participants. In this manner, in the financial world, protocol failure is recategorised as a different kind of success.

Of course, there is nothing to stop us using the non-ACID potential of the *Atomicity Pattern* to quantify some aspects of interest of the protocol, such as the proportion of transactions that might abort under some given set of assumptions or other, but this is a case of using the possibilities of retrenchment (perfectly reasonably) as a technical convenience, rather than a pronouncement about a lack of integrity of the protocol. Another possibility would be to examine the performance of the protocol under the assumption that one or more of the security hypotheses it rests on is weakened.

[11] The fact that accounting for outputs which are incompatible for atomicity reasons cannot be done convincingly using refinement alone, led, amongst other things, to the omission of balance enquiry operations from [39].

6 Conclusions

In the preceding sections, we took a particularly simple example, based on an ATC application [23], and via a series of simple models and small model evolution steps (reminiscent of the Event-B approach [1, 2], and of many practical ASM refinements [15, 14]), teased out how issues arising from non-atomicity of the real system interacted with the remainder of the development. The step from an atomic to a non-atomic model meant that inevitably, if one examined the states of the abstract and concrete models at an inopportune moment, some discrepancy would be observed, which went beyond what traditional substitutivity based notions of refinement could cope with. Quite where the discrepancy might be observed, depended on how one chose to synchronise the atomic abstract action with one of the constituent non-atomic concrete actions which implemented it. (The very non-atomicity of the concrete model guarantees that there will be more than one such choice.) We showed that the greater flexibility of retrenchment could account for what was going on in a rather straightforward manner, one moreover, that readily lends itself to generalisation as a retrenchment *Atomicity Pattern*. We tested the pattern against a different, and if anything more challenging example, based on the Mondex Purse [39, 7], and found that it coped with flying colours.

One particularly useful aspect of the *Atomicity Pattern* was its potential for coping with situations that fell short of perfect ACIDity in the concrete protocol. We showed this in the relatively simple context of the timing aspects of the ATC application. Admittedly this is an extremely simple scenario, and, of course, lack of space precluded us from examining more demanding examples, such as the ACIDity impaired situations that arise in web services, long-lived workflows and their compensated transactions, and highly concurrent and highly distributed environments.

However we can be confident that the retrenchment based *Atomicity Pattern* approach will be able to handle these much larger situations too. Why? Well, all models of the kind we are considering are defined using a collection of events or operations — a large complex model merely has more of them, they may be more complex, and may be organised into more layers. Provided the model is sufficiently comprehensive, and captures enough of the environment if the environment is implicated in ACIDity losing behaviour, loss of ACIDity will arise through specific events. Provided we model these events appropriately, the changing properties of interest can be captured in suitable retrenchment data (i.e. the within, output, and concedes relations) attached to relevant event or operation descriptions, as we pass from an idealised model to a more realistic and imperfect one.

We will examine more extensive case studies of this elsewhere, but let us close by briefly sketching how such an example might go. Consider a long-lived workflow containing online purchasing transactions. In an ideal world, items are only sold if they are in stock. However, in the real world, it may occasionally happen that the system could sell an item that did not exist due to poor stocktaking. Thus the real world could be described using two stock variables: the 'nominal stock level' used by the system, and the 'true stock level', accurate, but unknown to the system. While the latter remained positive all would be well. But as soon as the system (unknowingly) made a sale that pushed it below zero, the ACID properties would be compromised, since the sale would commit before it became known that it was invalid. In due course, further events would ensue, that (say) resulted in the purchaser being refunded. Although the *system* would not know it had transacted a rogue sale at the moment it happened, there is nothing to stop a *model of the system* having such knowledge, and thus being able to identify loss of ACIDity through appropriate retrenchment data.

References

[1] J.-R. Abrial. Event based sequential program development: Application to constructing a pointer program. In Araki et al. [3], pages 51–74.

[2] J.-R. Abrial, D. Cansell, and D. Méry. Refinement and reachability in Event-B. In *Proc. ZB 2005*, volume 3455 of *LNCS*, pages 222–241.

[3] K. Araki, S. Gnesi, and D. Mandrioli, editors. *International Symposium of Formal Methods Europe*, volume 2805 of *LNCS*, Pisa, Italy, September 2003. Springer.

[4] R. Back and R. Kurki-Suonio. Decentralisation of process nets with centralised control. In *2nd ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pages 131–142, 1983.

[5] Baeten and Klop, editors. *Proc. CONCUR 1990*, volume 458 of *LNCS*. Springer, 1990.

[6] R. Banach, C. Jeske, and M. Poppleton. Composition mechanisms for retrenchment. 2004. submitted, http://www.cs.man.ac.uk/~banach/some.pubs/Retrench.Composition.pdf.

[7] R. Banach, C. Jeske, M. Poppleton, and S. Stepney. Retrenching the purse: The balance enquiry quandary, and generalised and (1,1) forward refinements. *Fund. Inf.*, 77:29–69, 2007.

[8] R. Banach and M. Poppleton. Retrenchment: An engineering variation on refinement. In D. Bert, editor, *2nd International B Conference*, volume 1393 of *LNCS*, pages 129–147, Montpellier, France, April 1998. Springer.

[9] R. Banach and M. Poppleton. Fragmented retrenchment, concurrency and fairness. In *Proc. IEEE ICFEM2000*, pages 143–151, York, August 2000. IEEE Computer Society Press.

[10] R. Banach and M. Poppleton. Retrenching partial requirements into system definitions: A simple feature interaction case study. *Requirements Engineering Journal*, 8:266–288, 2003.

[11] R. Banach, M. Poppleton, C. Jeske, and S. Stepney. Retrenching the purse: Finite sequence numbers and the tower pattern. In J. Fitzgerald et al, editor, *FM 2005*, volume 3582 of *LNCS*, pages 382–398. Springer.
[12] M. Ben-Ari. *Principles of Concurrent Programming*. Prentice Hall, 1982.
[13] P. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.
[14] E. Börger. The ASM refinement method. *Formal Aspects of Computing*, 15:237–275, 2003.
[15] E. Börger and R. Stärk. *Abstract State Machines. A Method for High Level System Design and Analysis*. Springer, 2003.
[16] T. Connolly and C. Begg. *Database Systems: A Practical Approach to Design, Implementation and Management*. Addison Wesley, 2004.
[17] D. Cooper, S. Stepney, and J. Woodcock. Derivation of Z refinement proof rules. Technical Report YCS-2002-347, University of York, 2002.
[18] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems: Concepts and Design*. Addison Wesley, 2005.
[19] Department of Trade and Industry. Information Technology Security Evaluation Criteria, 1991. http://www.cesg.gov.uk/site/iacs/itsec/media/formal-docs/Itsec.pdf.
[20] R. Elmasri and S. Navathe. *Fundamentals of Database Systems*. Addison Wesley, 2003.
[21] N. Francez and I. Forman. Superimposition for interactive processes. In Baeten and Klop [5], pages 230–245.
[22] H. Garcia-Molina, J. Ullman, and J. Widom. *Database Systems: The Complete Book*. Prentice Hall, 2003.
[23] A. Hall. Using formal methods to develop an ATC information system. *IEEE Software*, 13:66–76, 1996.
[24] T. Harris and J. Bacon. *Operating Systems: Concurrent and Distributed Software Design*. Addison Wesley, 2003.
[25] S. Jajodia and L. Kerschberg. *Advanced Transaction Models and Architectures*. Kluwer, 1997.
[26] C. Jeske. *Algebraic Integration of Retrenchment and Refinement*. PhD thesis, University of Manchester, 2005.
[27] S. Katz. A superimposition control construct for distributed systems. *ACM TPLAN*, 15(2):337–356, April 1993.
[28] K. Loney. *Oracle Database 10g: The Complete Reference*. McGraw-Hill, 2004.
[29] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.
[30] N. Lynch, M. Merritt, W. Weihl, and A. Fekete. *Atomic Transactions*. Morgan Kaufmann, 1994.
[31] M. Poppleton and R. Banach. Retrenchment: Extending refinement for continuous and control systems. In *Proc. IWFM'00*, Springer Electronic Workshop in Computer Science Series, NUI Maynooth, July 2000. Springer.
[32] M. Poppleton and R. Banach. Structuring retrenchments in B by decomposition. In Araki et al. [3], pages 814–833.
[33] M. Raynal. *Distributed Algorithms and Protocols*. Wiley, 1988.
[34] G. Schellhorn. Verification of ASM refinements using generalized forward simulation. *JUCS*, 7:952–979, 2001.
[35] G. Schellhorn. ASM refinement and generalisations of forward simulation in data refinement: A comparison. *Theoretical Computer Science*, 336:403–435, 2005.
[36] G. Schellhorn, H. Grandy, D. Haneberg, N. Moebius, and W. Reif. A systematic verification approach for mondex electronic purses using ASMs. In *Proc. Dagstuhl 2007*, LNCS. Springer, 2007. to appear.
[37] G. Schellhorn, H. Grandy, D. Haneberg, and W. Reif. The Mondex challenge: Machine checked proofs for an electronic purse. In Misra, Nipkow, and Sekerinski, editors, *Proc. FM 2006*, volume 4085 of *LNCS*, pages 16–31. Springer, 2006.
[38] A. Silberschatz, P. Baer, and G. Gagne. *Operating System Concepts*. Wiley, 2005.
[39] S. Stepney, D. Cooper, and J. Woodcock. An electronic purse: Specification, refinement and proof. Technical Report PRG-126, Oxford University Computing Laboratory, 2000.
[40] J. Woodcock. First steps in the verified software grand challenge. *IEEE Computer*, 39(10):57–64, 2006.
[41] P. Zikopoulos, G. Baklarz, and R. Melnyk. *Official Guide to DB2 Version 8*. Prentice Hall, 2003.

Appendix: Refinements and Retrenchments

In this Appendix we briefly review the notions of refinement and retrenchment used above. For refinement, we adapt slightly the formulation in [17] as used in the Mondex development. The retrenchment rules are adapted to fit with the refinement ones. It will suffice to quote the forward rules for refinement and retrenchment.

The context of the rules is a pair of (abstract and concrete) ADTs: $(A, AInit, \{AOp, AI_{Op}, AO_{Op} \mid Op \in \mathsf{Ops}_A\})$, and $(C, CInit, \{COp, CI_{Op}, CO_{Op} \mid Op \in \mathsf{Ops}_C\})$. Here A is the abstract state schema, $AInit$ its initialisation, and for $Op \in \mathsf{Ops}_A$, Op, AI_{Op}, AO_{Op} are the abstract operation schemas, and their input and output space schemas. Similarly for the concrete side.

For refinement, the two ADTs are related by the retrieve relation $R_{A,C}$ on states, and (per operation) the input and output relations $RI_{A,C,Op}$ and $RO_{A,C,Op}$. For refinement $\mathsf{Ops}_A \subseteq \mathsf{Ops}_C$, but such that any $Op \in \mathsf{Ops}_C - \mathsf{Ops}_A$ is a refinement of a corresponding unstated abstract operation whose definition is skip.

Forward refinement is given by three main proof obligations (POs), *initialization*, *applicability* and *correctness*:

$$\forall C' \bullet CInit \Rightarrow \exists A' \bullet AInit \wedge R'_{A,C}$$

$$\forall A;\ AI_{Op};\ C;\ CI_{Op} \bullet R_{A,C} \wedge RI_{A,C,Op} \wedge \mathrm{pre}\, AOp \Rightarrow \mathrm{pre}\, COp$$

$$\forall A;\ AI_{Op};\ C;\ CI_{Op};\ C';\ CO_{Op} \bullet R_{A,C} \wedge RI_{A,C,Op} \wedge \mathrm{pre}\, AOp \wedge COp \Rightarrow \exists A';\ AO_{Op} \bullet AOp \wedge R'_{A,C} \wedge RO_{A,C,Op}$$

For retrenchment, the two ADTs are related by the retrieve relation $R_{A,C}$ on states, and (per operation) the within, output, and concedes relations $W_{A,C,Op}$, $O_{A,C,Op}$, and $C_{A,C,Op}$. For retrenchment $\mathsf{Ops}_A \subseteq \mathsf{Ops}_C$, and there is no restriction on operations in $\mathsf{Ops}_C - \mathsf{Ops}_A$.

Two POs define a retrenchment between two models: *initialisation* as for refinement, and *correctness* which is analogous to refinement correctness. Note that applicability issues are subsumed via the within relation:

$$\forall C' \bullet CInit \Rightarrow \exists A' \bullet AInit \wedge R'_{A,C}$$

$$\forall A;\ AI_{Op};\ C;\ CI_{Op};\ C';\ CO_{Op} \bullet R_{A,C} \wedge W_{A,C,Op} \wedge COp \Rightarrow \exists A';\ AO_{Op} \bullet AOp \wedge ((R'_{A,C} \wedge O_{A,C,Op}) \vee C_{A,C,Op})$$