Retrenching the Purse: Finite Exception Logs, and Validating the Small

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

Michael Poppleton
School of Electronics
and Computer Science
University of Southampton
Southampton SO17 1BJ, UK
mrp@ecs.soton.ac.uk

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

Abstract

The Mondex Electronic Purse is an outstanding example of industrial scale formal refinement, and was the first verification to achieve ITSEC level E6 certification. A formal abstract model and a formal concrete model were developed, and a formal refinement was hand-proved between them. Nevertheless, certain requirements issues were set beyond the scope of the formal development, or handled in an unnatural manner. The retrenchment Tower Pattern is used to address one such issue in detail: the finiteness of the purse log (which records unsuccessful transactions). A retrenchment is constructed from the lowest level model of the purse system to a model in which logs are finite, and is then lifted to create two refinement developments of the purse, working at different levels of detail, and connected via retrenchments, forming the tower. The tower development is appropriately validated, vindicating the design used.

1 Introduction

The Mondex Electronic Purse [18], produced by the NatWest Development Team in the mid 1990s, is a system of Smartcard-based electronic purses, carrying currency for use in every normal kind of transaction — the trial in Swindon UK in the late 1990s enabled the purses to be used in everyday shopping as well as conventional e-commerce applications. Clearly, any electronic purse is a security-critical application. For this reason, the developers of Mondex (formerly a part of NatWest Bank), employed state of the art methods to ensure the implementation was as robust as possible in the face of the most inviting of attacks — ones attempting to forge nonexistent money on the purses.

When it was created, the Mondex Purse achieved an ITSEC [16] rating of E6 (nowadays equivalent to a Common Criteria EAL7 rating). ITSEC E6 requires a formal abstract model, a formal concrete model and a proof of correspondence between them. In the case of Mondex this proof of correspondence was a refinement, formally proved to be correct by hand. (Since Mondex, the JavaCard [9] has enjoyed an even more exacting development, in which the the formal refinement proof was checked by machine.) The Mondex Purse remains an impressive achievement, and its development was a trailblazer for showing that fully formal techniques could be applied within realistic time and cost limitations on industrial scale applications.

The abstract model of the Mondex Purse system describes a world of purses which exchange monetary value through atomic transactions, and specifies the security properties demanded of this world: firstly, and above all, the impossibility of creating nonexistent value; and secondly, the traceable accounting of all value in the system, whether correctly transacted or lost in transit.

The concrete design model describes a system of purses which is distributed, transferring value via an insecure and potentially lossy medium using a three-step protocol. Security features are implemented locally on each purse. In the field the purses must be self-sufficient, defending the integrity of their monetary contents in the face of the most pessimistic assumptions regarding their environment that can still lead to the maintenance of the security properties. Transactions that appear not to be proceeding as required (from an individual purse's point of view) are aborted, and their details logged locally (the precise formal modelling of the logs is a central issue of this paper). Central reconciliation of purses' local logs can lead to the retrieval of money genuinely lost in transit, while still defending against fraudulent attempts to create nonexistent value, provided the most fundamental security assumptions (those asserting the non-forgeability of the critical protocol messages) remain unblemished.

Since the purses regard their environment as hostile, every atomic operation that they can perform, must in and of itself, preserve the security invariants — a purse cannot make any assumptions about its environment's intentions, or the environment's inclination (or otherwise) to conform to the way that the purse operations are intended to be used in the normal playout of the protocol. Given these facts, and the intention to use refinement as the means of achieving correctness, the most straightforward way of assuring the robustness required, is to have each concrete atomic operation be the refinement of some abstract atomic operation. If this is true, then since the security properties mentioned above are functional properties of the individual abstract steps,[1] and each concrete step maps to an abstract step via the refinement's retrieve relation (see below), it is clear that the concrete steps inherit the security properties of the abstract steps that they map to, and that the environment's intentions truly become irrelevant.

The separation between the abstract and concrete levels in Mondex is significant, in a logical as well as a functional sense, and it is this separation that contributes in large part to the validation obtainable from the formal proof. Nevertheless, the necessity of having a refinement, taking into account that refinement's proof obligations can be quite demanding in how abstract and concrete models are permitted to be related, meant that a number of requirements issues, legitimately the concern of the formal development, had to be passed over in silence —i.e. set beyond the scope of the formal development, or handled in an unnatural manner— since they would strictly speaking have broken the validity of the refinement had they been incorporated in the models that were used. One can say that curtailing the scope of a refinement *always* happens to some extent, since for example, it is never practical to prove refinement all the way down to the physical hardware, with the latter's conformance to its specification assured only within manufacturing limitations rather than absolutely.

Retrenchment [3, 4] was introduced as a framework that weakens and generalizes refinement, essentially in allowing the main refinement operation proof obligation (PO) to be weakened in the postcondition by a *concession*. (Normally this PO just asserts that the retrieve relation, classically an invariant, is re-established.) By interfering in this way with the structure of the PO, which is conventionally derived from stipulated correctness principles, the connection with those principles is severed, but the applicability of the resulting PO is widened. Things that come within the scope of retrenchment as a result, include the impossibility of refining infinite to finite types, or the continuous variables of real-world physical models (so commonly occurring in the safety-critical applications for which rigorous software techniques are utilised) to discrete ones. As well as such originally envisaged applications [17], retrenchment has also proved useful as a vehicle for the flexible layering in of contrasting requirements, even conflicting ones, in a formal development [5].

If refinement offers strong guarantees of correctness, but is limited as regards the ideal remit of its applicability, and retrenchment forfeits correctness guarantees but is much more widely applicable, then the most profitable strategy would be to employ a judicious combination of the two. One way of doing this is via the *Tower Pattern*, a systematic arrangement of refinements and retrenchments, which allows refinement developments incorporating different but incompatible levels of real-world detail, to be related via suitable retrenchments. In this paper we focus on one of the issues imperfectly covered by refinement in the Mondex development —the finiteness of the local purse exception log— and illustrate how the *Tower Pattern* allows for a less unnatural treatment.

The rest of the paper is structured as follows. In section 2 we give an overview of the Mondex refinement development, and identify the requirements issues that motivate the application of retrenchment. In view of the fact that the refinement treatment of these issues leaves something to be desired, we regard these issues as 'retrenchment opportunities', i.e. opportunities for the relatively novel retrenchment technique to make a worthwhile contribution to their formal treatment via refinement. Section 3 introduces the *Tower Pattern* and the theoretical results on the algebraic interaction of retrenchments and refinements that assist in its application [14]. The structural components of the tower (the constituent notions of refinement and retrenchment) are also made precise. Section 4 focuses on the finiteness of the Mondex purse's log, and defines a retrenchment between the original Mondex concrete purse (in which logs are unbounded) and a new purse model, identical to the former in all but the purse log's boundedness. This retrenchment is then extended to the world of purses by a suitable adaptation of the Z promotion used in [18]. Given these ingredients, Section 5 overviews how the buiding blocks assembled thus far can be lifted using the previously mentioned algebraic results, and thus constitute the buiding blocks of the tower as a whole. Section 6 gives a validation of the lifted retrenchment, and contrasts the retrenchment approach to dealing with this scenario, with other ways of dealing with the issue of infinite ideal domain vs. finite actual domain in a refinement context. Section 7 concludes.

2 The Mondex Purse: Refinements, Retrenchment Opportunities

As mentioned above, the Mondex Electronic Purse was developed by the NatWest Development Team, and achieved its ITSEC E6 classification via a hand proved formal refinement from abstract to concrete. The hand proved refinement

[1] It is (easily) determined that for each individual abstract step, the pair of before and after states preserves the security properties. So the Boolean function '*before state satisfies security property* $\Rightarrow$ *after state satisfies security property*' always evaluates to true, and expresses the safety invariants as functional properties of the abstract system.

demanded a readable and independently verifiable suite of documentation, and from this commercially sensitive version, a public version was generated [18]. This retains the essential elements, while removing some of the detail which was either confidential or simply not very interesting. The refinement consists of three models: A(abstract), B(between), and C(concrete). The A model is a highly abstract expression of atomic value transfer between purses, allowing for an atomic notion of loss in transit. The atomicity makes the security invariants, 'No value created' and 'All value accounted' trivial to prove. It is important to note that the A model is targetted purely at these security properties and no others. So it does not address all the many other system requirements. Model B captures the elements of the value transfer protocol, and is thus nonatomic. It is also enhanced with extra structure and constraints needed to achieve a backward refinement from model A; a backward refinement was the strategy used in the Mondex development to get from model A to model B, though recent work has shown that, strictly speaking, it is not necessary to do this [7]. Model C is model B without the extra structure and constraints. These can be established by an induction on the length of the execution, leading to a forward refinement between models B and C. It is thus shown that model C is a refinement of model A.

Accepting that the development described in [18] is a development of the security properties alone of the Mondex Purse, it is no surprise that many important aspects of the Mondex system do not get a proper treatment within [18]. Perhaps more surprising is the fact that even taking this into account, some requirements aspects, in principle deserving to be included within the formal development, since they potentially impact on the security properties if mishandled, were nevertheless omitted or handled unnaturally in the modelling, in order to establish the refinement. One of the aims of our work on Mondex, is to show how such rather brittle aspects of formal development via refinement, can be mollified by making use of retrenchment. Not only does this improve the overall quality of the formal development, but it also provides excellent vindication for the retrenchment technique itself, especially when it is used appropriately in tandem with refinement. Here is a brief summary of the Mondex 'retrenchment opportunities'; in this paper we focus in detail on the full log issue. The other retrenchment opportunities are treated elsewhere:

- Sequence Number: The integrity of the protocol depends partly on the sequence number of the transaction in progress. Sequence numbers occur in the B, C models where they are naturals; in reality they are bounded numbers, but potentially large.
- Log Full: Transfers completing abnormally are aborted and logged locally by purses. The relationship between local purse logs and the 'All value accounted' security property is rather complicated, and is outlined in footnote 8. Suffice it to say here that purses' log contents are vital. Logs occur in the B, C models where they are unbounded; in reality they are finite, and decidedly small.
- Hash Function: The concrete models implement the abstract 'lost value' component in terms of an off-card archive into which purses' log contents are saved (see footnote 8 again). A purse needs to be assured that the data is safely in the archive before it can clear it from its own, highly constrained, log memory. Safe archival is signalled to the purse using a 'clear' code. The purse log contents are assumed to be in total injective correspondence with the clear codes, as that property is required in the proof of the maintenance of the security properties. In reality of course a cryptographic hash function is used, which is neither total, nor injective, but is informally argued to be 'sufficiently injective'.
- Balance Enquiry: Each purse has a balance enquiry operation. If this is invoked in the middle of a B (or C) model transaction, a discrepancy can occur between the model A and model B balances since the model A transaction is atomic and the model B one isn't. This is handled formally by a modelling trick, using finalisation instead of the enquiry operation to observe the state: the resulting treatment of balance enquiries can appear so counterintuitive that the operation was removed from [18].

3 The Tower Pattern, Refinements and Retrenchments

In this section we outline the technical strategy for our work, and give the necessary technical definitions.

Model based refinement is a formal development technique which comes in a number of specific flavours; we will focus on the version for the Z language since that was used in [18]. Retrenchment is a different formal development technique, possessing different properties to refinement, but intended to be compatible with it. One way of making them interact productively is in the *Tower Pattern* illustrated in Fig. 1. This shows a collection of models connected by refinements in the vertical direction, and by retrenchments in the horizontal one. The diagram commutes, so there are many ways of navigating it. There are also many ways of building it, relying on various 'square completion' constructions explored in depth in [14]. The technical details of the latter are taxing, to say the least, so we do not go into them here. Sufficient to say that one can start with any path in Fig. 1 that goes from one corner to the diagonally opposite one, and use the technical results to build the rest. Moreover, the specific requirements issue (or issues) dealt with via the retrenchment is (or are) largely decoupled from the overall tower structure, underlining its generality and wide utility. This flexibility and wide applicability makes the tower richly deserve the 'pattern' epithet.

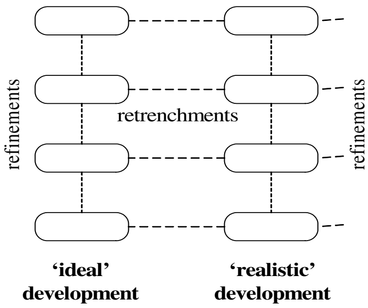

Figure 1. The *Tower Pattern*.

The tower is applied in a bottom up manner in the case of Mondex. This is shown in Fig. 2, where we see the development of [18] down the left hand side. Emerging from model C, is a retrenchment taking us to model D. In this paper the retrenchment deals with the move from unbounded logs to bounded ones. Once model D is in place, we compose the refinement from B to C with the retrenchment from C to D to give a retrenchment from B to D. Using results of [14], we can now factorise this retrenchment the other way, yielding model E, which raises the level of abstraction of model D to that of model B.

It turns out that there is a backward refinement from model A to model E. This enables us to complete the tower by making model F a copy of model A.

$$\begin{tikzcd}
\textbf{A–bstract} \arrow[r, "\text{id}"] \arrow[d, "\text{ref}"'] & \textbf{F–iltered} \arrow[d, "\text{ref}"] \\
\textbf{B–etween} \arrow[r, "\text{ret}"] \arrow[d, "\text{ref}"'] \arrow[dr, dotted, "\text{ref ; ret}"'] & \textbf{E–levated} \arrow[d, "\text{ref}"] \\
\textbf{C–oncrete} \arrow[r, "\text{ret}"'] & \textbf{D–iscrete}
\end{tikzcd}$$

Figure 2. The *Tower Pattern* applied to Mondex.

We now briefly review the notions of refinement and retrenchment we need for the sequel. The nomenclature in our definitions will be in line with that in the various models in our tower structure for Mondex. We just give the forward rules for both refinement and retrenchment, since these are sufficient for the BCDE part of the tower, all that is needed.

Let model B be given by the ADT $(B, BInit, \{BOp, BI_{Op}, BO_{Op} \mid Op \in \mathsf{Ops}\})$, and let model C be given by the ADT $(C, CInit, \{COp, CI_{Op}, CO_{Op} \mid Op \in \mathsf{Ops}\})$. So schemas B, C give the abstract and concrete state spaces for the forward refinement from B to C, and the corresponding per-operation I/O spaces are given by schemas BI_{Op}, BO_{Op} and CI_{Op}, CO_{Op} respectively. We assume a retrieve relation $R_{BC} : [\,B\,;\,C\,]$ between the two state spaces, and for each operation Op, input and output mapping relations $RI_{BC,Op} : [\,BI_{Op}\,;\,CI_{Op}\,]$ and $RO_{BC,Op} : [\,BO_{Op}\,;\,CO_{Op}\,]$. Forward refinement is given by three proof obligations (POs), *initialization, applicability* and *correctness*:

$$\forall\, C' \bullet CInit \Rightarrow \exists\, B' \bullet BInit \wedge R'_{BC} \tag{1}$$

$$\forall\, B;\ BI_{Op};\ C;\ CI_{Op} \bullet R_{BC} \wedge RI_{BC,Op} \wedge \operatorname{pre} BOp \\ \Rightarrow \operatorname{pre} COp \tag{2}$$

$$\forall\, B;\ BI_{Op};\ C;\ CI_{Op};\ C';\ CO_{Op} \bullet \\ R_{BC} \wedge RI_{BC,Op} \wedge \operatorname{pre} BOp \ \wedge COp \\ \Rightarrow \exists\, B';\ BO_{Op} \bullet BOp \wedge R'_{BC} \wedge RO_{BC,Op} \tag{3}$$

Note that (1)-(3) do not mention finalisation. We deal with the issue of observation, and specifically of relating the outputs of the abstract and concrete models (normally handled via finalisation) 'on the fly', in line with the tack taken in retrenchment. Moreover, applicability turns out to be a trivial matter in Mondex: all operations are wrapped with an *Ignore* option (which, at the levels of abstraction of these models, just skips). This means that all operations are always enabled, even if (when a purse is not in the intended state) nothing of interest happens.

The C to D development is a forward retrenchment. For this, the abstract model is the C ADT, and the concrete model is given by ADT $(D, DInit, \{DOp, DI_{Op}, DO_{Op} \mid Op \in \mathsf{Ops}\})$. Similar notational conventions apply as before. The retrenchment is given by firstly a *retrieve* relation $R_{CD} : [\,C\,;\,D\,]$ between the state spaces; and secondly by the within, output and concedes relations on a per-operation basis. The *within* relation is between the input-state spaces $W_{CD,Op} : [\,CI_{Op}\,;\,C\,;\,DI_{Op}\,;\,D\,]$. The *output* and *concedes* relations are formally defined over both of the full input-state-output frames with types $O_{CD,Op}\,;\,C_{CD,Op} : [\,CI_{Op}\,;\,C\,;\,C'\,;\,CO_{Op}\,;\,DI_{Op}\,;\,D\,;\,D'\,;\,DO_{Op}\,]$, though we often omit such parts of these signatures as are not needed. We call these three relations the *retrenchment data*.

Two POs define a retrenchment between two models: *initialisation* as for refinement (1), and *correctness* which is analogous to refinement correctness (3); note that applicability issues are understood to be subsumed in (5) via the within relation:

$$\forall\, D' \bullet DInit \Rightarrow \exists\, C' \bullet CInit \wedge R'_{CD} \tag{4}$$

$$\forall\, C;\ CI_{Op};\ D;\ DI_{Op};\ D';\ DO_{Op} \bullet R_{CD} \wedge W_{CD,Op} \wedge DOp \\ \Rightarrow \exists\, C';\ CO_{Op} \bullet COp \wedge ((R'_{CD} \wedge O_{CD,Op}) \vee C_{CD,Op}) \tag{5}$$

4 The Finite Log Retrenchment

The place where the properties of the purse log come to the fore is the operation for aborting a transaction. It turns out that the business of aborting a transaction is by far the most subtle aspect of the Mondex purse from a theoretical point of view; see footnote 8 and [7]. Fortunately none of this impacts on the way the finite size of the log is handled via retrenchment, so our task is tractable in a paper of this size.

We start our treatment by focusing on the simplest operation of [18] that is nontrivially affected, namely the abort operation for a single purse in the C model, given in the schema *CAbortPurseOkay*. This does the following.[2] If the purse's state is either of *Cepv*, *Cepa* (i.e. expecting value or expecting ack), then the purse is in a critical part of the protocol (either as receiver or as sender), and must log the transaction it is aborting. So the authenticated payment details *CpdAuth* get added to the exception log, *CexLog*. The sequence number *CnextSeqNo* is incremented, the purse status, *Cstatus*, is reset, and the purse's state components aside from these, gathered in the schema *CConPurseAbort* remain unchanged, i.e. $\Xi CConPurseAbort$.

$$
\begin{aligned}
&CConPurseAbort == \\
&\quad CConPurse \setminus (CexLog, Cstatus, CnextSeqNo)
\end{aligned}
$$

CAbortPurseOkay

$$
\begin{array}{|l}
\Delta CConPurse \\
Cm?, Cm! : CMESSAGE \\
\hline
\Xi CConPurseAbort \\
CLogIfNecessary \\
CnextSeqNo' \geq CnextSeqNo \\
Cstatus' = CeaFrom \\
Cm! = \perp
\end{array}
$$

CLogIfNecessary

$$
\begin{array}{|l}
\Delta CConPurse \\
\hline
CexLog' = CexLog \cup (\text{if } Cstatus \in \{Cepv, Cepa\} \\
\quad \text{then } \{CpdAuth\} \text{ else } \varnothing)
\end{array}
$$

In the above the log is a set, so can grow in cardinality without restriction. Also, the sequence number *CnextSeqNo* is a natural, thus unbounded. For expository clarity, we will focus on the log's cardinality and condone the unboundedness of the sequence number, recognising that a proper treatment would need to take both into account.

In the D model, we make the purse log a set of maximum size *LOGMAX*. If the log still has plenty of room, things go as before. However if the current abort fills it, we must prevent further transactions, since if they were to go wrong, there would be no place to log *their* details. We therefore take a purse into a new status *DexLogFull* as the last empty slot of the log gets filled. We then have to guard every other purse operation which does not stipulate a specific before-state in its precondition, with a clause that ensures that the purse is not in *DexLogFull* in the before-state. In fact however, aside from the abort operation itself, and some harmless skip-like operations (like the *Ignore* wrappers mentioned earlier), there are no such operations. The only operation that will be enabled (nontrivially) in the *DexLogFull* state is the operation to clear the log, which is itself the seat of a separate 'retrenchment opportunity'.[3]

In the following, *DConPurse* 'is as' *CConPurse*, with the exception of *DexLog* which is a finite set instead of an unbounded set, (and apart from the extra possible status value just introduced). Also *DConPurseAbort* 'is as' *CConPurseAbort*. $\perp$ is a general purpose message used in Mondex which is of-no-concern. Note that the following design for *DAbortPurseOkay* is conservative, in that the operation is always enabled (i.e. it is enabled even in the *DexLogFull* state, though under normal operation it should never be called there). This is in keeping with the Mondex philosophy of placing the minimum of assumptions on the expected behaviour of the environment.

DAbortPurseOkay

$$
\begin{array}{|l}
\Delta DConPurse \\
Dm?, Dm! : DMESSAGE \\
\hline
\Xi DConPurseAbort \\
DnextSeqNo' \geq DnextSeqNo \\
(\# DexLog < LOGMAX \Rightarrow DLogIfNecessary) \\
(\# DexLog \geq LOGMAX \Rightarrow DexLog' = DexLog) \\
(\# DexLog < LOGMAX - 1 \Rightarrow \\
\quad Dstatus' = DeaFrom \wedge Dm! = \perp) \\
(\# DexLog \geq LOGMAX - 1 \wedge \\
Dstatus \in \{Depv, Depa\} \Rightarrow \\
\quad Dstatus' = DexLogFull \wedge \\
\quad Dm! = \text{“Purse blocked. Go to bank.”})
\end{array}
$$

where *DLogIfNecessary* 'is as' *CLogIfNecessary*.

[2] In Z, declarations and relevant properties are parceled into schemas. Modularity is encouraged by including schemas in other schemas.

We employ the common convention of pre-capitalizing only the names of types (schema and other). We augment this convention by prefixing a single character $A, B, \ldots$ to a name as required, to denote the Mondex model in question. Thus *CThing* is a schema or other type in the C model, whereas *Dthing* is a variable, usually a schema component, in the D model. To save space we employ a further lexical schema convention by saying *DSchema* 'is as' *CSchema*, which indicates that the text of *DSchema* can be generated from that of *CSchema* by replacing all *Cthing*s by *Dthing*s.

[3] There are alternative designs of course. For instance we could check the capacity of the log at the beginning of each transaction, proceeding only if there is room for an exception record, should it be needed. Some such feedback is obviously given to users anyway, but our design takes the code for that out of the security-critical part of the application. In any event, it is probably better to alert the user to the full log *before* he/she needs another transaction.

The C and D models are related by a retrenchment, so we now discuss the retrenchment data for *DAbortPurseOkay*. In the following schemas, *CDConPurseAbortEquality* stands for for equalities between the C and D variables in *CConPurseAbort* and *DConPurseAbort*. Under the vacuous *within* constraint, the *CAbortPurseOkay*, *DAbortPurseOkay* pair establishes either $R'_{CD} \land O_{CD,AbortPurseOkay}$ (*retrieve* and *output*), or $C_{CD,AbortPurseOkay}$ (the *concession*). The concession makes explicit the lack of correspondence between C and D purse states and outputs.

$$\begin{array}{l}
\underline{R_{CD}} \\
CConPurse;\ DConPurse \\
CDConPurseAbortEquality \\
\hline
(Cstatus = Dstatus \lor Dstatus = DexLogFull) \\
CnextSeqNo = DnextSeqNo \\
CexLog = DexLog
\end{array}$$

$$\begin{array}{l}
\underline{W_{CD,AbortPurseOkay}} \\
CConPurse;\ DConPurse \\
Cm? : CMESSAGE \\
Dm? : DMESSAGE
\end{array}$$

$$\begin{array}{l}
\underline{O_{CD,AbortPurseOkay}} \\
Cm! : CMESSAGE \\
Dm! : DMESSAGE \\
\hline
Cm! = Dm!
\end{array}$$

$$\begin{array}{l}
\underline{C_{CD,AbortPurseOkay}} \\
CConPurse';\ DConPurse' \\
CDConPurseAbortEquality' \\
Cm! : CMESSAGE \\
Dm! : DMESSAGE \\
\hline
CexLog' \supseteq DexLog' \\
Cstatus' = CeaFrom \\
Dstatus' = DexLogFull \\
CnextSeqNo' = DnextSeqNo' \\
Cm! = \bot \\
Dm! = \text{“Purse blocked. Go to bank.”}
\end{array}$$

4.1 The Retrenchment in the World of Purses

Purses are of course not intended to exist in isolation, but to coexist and interact with other purses (even though each individual purse takes a very defensive attitude to its environment as is evident from the security properties). This means that the total system consists of a community of identically specified components (and some central facilities). The Z promotion methodology, which we briefly review, has been designed to cope smoothly with such situations.

In promotion [21, 13, 19, 20], a collection of identically defined local state (*LS*) instances are typically aggregated via an indexing function, forming the world state:

$$\begin{array}{l}
\underline{World} \\
f : Ind \rightarrow\!\!\!\!\!\!\!+\ LS
\end{array}$$

Operations on the world which are extensions of local operations (eg. *LSOp* extended to *WorldOp*), are treated systematically via the framing schema $\Phi LocOp$. This: (a) singles out which component of the world will perform *LSOp*, using the value of an input index $i?$; (b) stipulates that all *other* components skip; (c) allows the selected component to perform a completely unrestricted state change, ready for further restriction in a moment. (It achieves (c) by identifying the change in a separate copy of the local state (from θLS to $\theta LS'$) with the change from $f(i?)$ to $f'(i?)$).

$$\begin{array}{l}
\underline{\Phi LocOp} \\
\Delta World;\ \Delta LS;\ i? : Ind \\
\hline
i? \in \mathrm{dom} f \\
\theta LS = f(i?) \\
f' = f \oplus \{i? \mapsto \theta LS'\}
\end{array}$$

Now, constraining the change in ΔLS using the local definition of *LSOp*, makes the world component $i?$ do only what *LSOp* permits.

$$WorldOp == \exists\, \Delta LS \bullet \Phi LocOp \land LSOp$$

What we just discussed was the basic index-function-based form of promotion. In Mondex, we need a slightly more elaborate formulation, which has in fact inspired the definition of a number of promotion *patterns* as generalisations [19, 20].

Just as in the C world, individual D model purses are promoted to a D world of purses, via the schemas that follow. We make use of our shorthand conventions. So *DConWorld* ‘is as’ *CConWorld*; both of these contain, beyond the purse-*NAME*-indexed map of local purse states, the *Dether* of all messages ever sent between purses (and not yet lost), and the central *Darchive* of all transaction exception logs uploaded from purses. There are two additional *DConWorld* constraints: each internal purse name is equal to its corresponding index, and each archive entry identifies its originating purse.

$$\begin{array}{l}
\underline{DConWorld} \\
DconAuthPurse : NAME \rightarrowtail\!\!\!\!\!\!\!+\ DConPurse \\
Dether : \mathbb{P}\, DMESSAGE \\
Darchive : \mathbb{P}\, DLogbook \\
\hline
\forall\, n : \mathrm{dom}\, DconAuthPurse \bullet \\
\quad (DconAuthPurse\ n).Dname = n \\
\forall\, nld : Darchive \bullet \mathrm{first}\ nld \in \mathrm{dom}\, DconAuthPurse
\end{array}$$

Promotion of the D model 'is as' that of the C model of [18]: the framing schema ΦDOp 'is as' ΦCOp, where $Dm?, Dm!$ are the input and output messages to and from *DAbortPurseOkay*.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\Phi DOp} \\
\hline
\Delta DConWorld;\ \Delta DConPurse \\
Dm?, Dm! : DMESSAGE \\
Dname? : NAME \\
\hline
Dm? \in Dether \\
Dname? \in \operatorname{dom} DconAuthPurse \\
\theta DConPurse = DconAuthPurse\ Dname? \\
DconAuthPurse' = DconAuthPurse \oplus \\
\qquad\qquad\qquad \{Dname? \mapsto \theta DConPurse'\} \\
Darchive' = Darchive \\
Dether' \subseteq Dether \cup \{Dm!\} \\
\hline
\end{array}
$$

DAbort, the promoted and wrapped operation, 'is as' *CAbort*. N.B. *DIgnore* 'is as' *CIgnore*, and just skips at world level.

$$
\begin{array}{l}
DAbort == DIgnore \lor \\
\quad (\exists\, \Delta DConPurse \bullet \Phi DOp \land DAbortPurseOkay)
\end{array}
$$

Having defined the D world and the promotion of the purse level abort operation to world level, it is time to consider the promotion of the retrenchment of individual purse operations such as *CAbortPurseOkay* into the *CConWorld* to *DConWorld* level. This is the retrenchment from *CAbort* to *DAbort*, (where *DAbort* 'is as' *CAbort*, as just noted). Any theory of promotion of retrenchments should start from the promotion of refinements, especially when (as here) we are expecting refinement and retrenchment to complement one another. A clear treatment of the promotion of refinements is given in [13], especially as regards the construction of a simple world-level retrieve relation from the distribution of the local retrieve relation through the indexing. We base our approach on this form as it is close to the form of promotion used in Mondex. Thus, given a retrieve relation R between local states *Abs* and *Conc*, the promoted retrieve relation R^P between *AbsWorld* and *ConcWorld* (with index functions $Absf, Conf$ respectively) simply says that R has to hold for each indexed component:

$$
\begin{array}{|l}
\multicolumn{1}{l}{R^P} \\
\hline
AbsWorld;\ ConcWorld \\
\hline
\operatorname{dom} Concf = \operatorname{dom} Absf \\
\forall\, n : \operatorname{dom} Concf \bullet \exists\, R \bullet \\
\quad \theta Abs = Absf(n) \land \theta Conc = Concf(n) \\
\hline
\end{array}
$$

At this point we hit a technical snag entailed by the fact that retrenchment is not the same as refinement. The essential point is this. Let us imagine that the concrete system has been running for some time and that some or many elements have already engaged in operations. With reference to an abstract system that we can imagine has been keeping up with the concrete one (insofar as it has been trying to simulate it as well as it can), we can expect that some elements will be in the local component retrieve relation R, while others may have already conceded (and so may no longer be in R). Assuming all elements are in R (as for refinement) would thus give an unduly restricted syntactic picture of the correspondence between the dynamics of the abstract and concrete worlds. Specifically, as soon as a single component has taken a step that makes the relevant concession valid, R^P has most likely ceased to hold.[4]

The snag just outlined offers a number of possible approaches to the promotion of retrenchments, depending on what one wishes to emphasise. In [6] we explore this in some detail, but space limitations here do not permit us to show the full variety of possibilities in the current context. Instead, we give a brief overview of the approaches, and develop one in detail.

Retaining the retrieve relation R^P for the retrenchment promotion leads to so-called *strong promotion* of retrenchments, since it sets the most demanding conditions. A strong promotion (between a concrete system and an abstract system trying to stay in simulation with it) remains applicable just so long as all the components of the concrete system continue to stay simulable by suitable abstract components. As soon as any concrete component makes a transition that invalidates its local retrieve relation, the world level retrieve relation R^P stops being valid, and the predicates of the strongly promoted retrenchment cease to be able to say anything about the subsequent state of affairs.

An alternative to strong promotion is *weak promotion* of retrenchments. This arises simply by replacing the universal quantification in R^P by an existential quantification. The resulting world level retrenchment is capable of making a valid pronouncement about the relationship between a concrete system and an abstract system trying to stay in simulation with it, provided now that at least one of the components is still in the local retrieve relation. With the existential quantification, it is clear that doubt can arise regarding *which* of the components is still in simulation, but given the capacity for expressing useful things regarding the relationship between abstract and concrete systems afforded by retrenchment, there is enough room in the retrenchment data to accommodate the technical details needed to handle this.

In between strong and weak promotions lies *precise promotion* of retrenchments. Rather than putting up with the imprecision latent in the existential quantification, as just alluded to, precise promotion introduces a new world level

[4] We merely say 'most likely', since the 'or' in (5) is not 'exclusive', so the retrieve relation might yet be valid too.

variable (let us call it ‘good’), whose task is to keep track of the identities of those components which still remain in the local retrieve relation (and which do not). In order to be able to do this, a separation axiom has to hold regarding the individual component level retrenchments.[5] Fortunately the majority of cases arising in practice do satisfy the necessary axiom. The presence of the ‘good’ world level variable, by its presence, not only modifies the concrete world definition, but also entails minor modifications to the various operations, so that ‘good’ can be accurately updated as individual components do or do not stay simulable. However all that is required is read-only access to the body of any operation, so this is completely benign.

Having illustrated strong and precise promotion in other papers, in this one, we choose to exemplify *weak promotion* for our treatment of the *CAbort* and *DAbort* operations. Note that this choice is independent of any details of *CAbort* and *DAbort* themselves.

The weak promotion of the *DAbortPurseOkay* retrenchment is given by the retrenchment data below. Note how the uncertainty concerning which component is claimed to be retrieving in the retrieve relation R^{PW}_{CD} is reconciled with the identification of the named component of interest in the promotion via the within relation $W^{PW}_{CD,Abort}$. Essentially the same information, $RBody^{PW}_{CD}$, is repeated in both; quantified in R^{PW}_{CD}, and free in $W^{PW}_{CD,Abort}$. Also, we have used some shorthand to avoid verbosity. So “ CDnamedConPurseAbortEquality *name* ”, as before, stands for equalities of named purses’ of-no-concern data. Similarly “ CDnamedConPurseAbortRetrieve *name* ” stands for the syntax needed to transplant the body of the single purse retrieve relation from its original context, and make it refer to the named purse of the promotion; “ CDnamedConPurseAbortConcession *name* ” does the same for the concession. Note that because the world contains data other than just replicated local data (i.e. the archive and ether), this must be dealt with explicitly in $C^{PW}_{CD,Abort}$.

The retrenchment below employs a *focused* pattern of weak promotion, in that the within, output, concedes relations only refer to the named local component *Dname*?. Since the promoted operation acts on only one element, implicitly all other elements maintain their state, whether retrieving or not. An *inclusive* variant is also available which covers all these other components, explicitly claiming R'_{CD} in the output relation and concession *provided* it holds in the before state. Since archive entries are tagged with the originating purse’s name, we can identify those C/D archive subsets corresponding to *Dname*?, and we assume for simplicity that all messages in the ether are tagged with originator’s and addressee’s names as the first two fields of the message.[6]

$$
\begin{array}{|l}
\multicolumn{1}{l}{RBody^{PW}_{CD}} \\
\hline
CConWorld;\ DConWorld \\
Dname? : \mathrm{dom}\, CconAuthPurse \cap \\
\qquad\qquad \mathrm{dom}\, DconAuthPurse \\
\hline
\text{“ CDnamedConPurseAbortRetrieve } Dname? \text{ ”} \\
\{Dname?\} \lhd Carchive = \{Dname?\} \lhd Darchive \\
(\{Dname?\} \times \{Dname?\}) \lhd Cether = \\
\qquad (\{Dname?\} \times \{Dname?\}) \lhd Dether \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{R^{PW}_{CD}} \\
\hline
CConWorld;\ DConWorld \\
\hline
\mathrm{dom}\, CconAuthPurse = \mathrm{dom}\, DconAuthPurse \\
\exists\, Dname? : \mathrm{dom}\, CconAuthPurse \cap \\
\qquad\qquad \mathrm{dom}\, DconAuthPurse \bullet RBody^{PW}_{CD} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{W^{PW}_{CD,Abort}} \\
\hline
CConWorld;\ DConWorld \\
Cm? : CMESSAGE \\
Dm? : DMESSAGE \\
Cname? : \mathrm{dom}\, CconAuthPurse \\
Dname? : \mathrm{dom}\, CconAuthPurse \\
\hline
Cname? = Dname? \\
RBody^{PW}_{CD} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{O^{PW}_{CD,Abort}} \\
\hline
\Delta DConWorld \\
Cm! : CMESSAGE \\
Dm! : DMESSAGE \\
Dname? : NAME \\
\hline
Cm! = Dm! \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{C^{PW}_{CD,Abort}} \\
\hline
CConWorld';\ \Delta DConWorld \\
Cm! : CMESSAGE \\
Dm! : DMESSAGE \\
Dname? : NAME \\
\hline
\text{“ CDnamedConPurseAbortEquality}'\, Dname? \text{ ”} \\
\text{“ CDnamedConPurseAbortConcession } Dname? \text{ ”} \\
\{Dname?\} \lhd Carchive' = \{Dname?\} \lhd Darchive' \\
(\{Dname?\} \times \{Dname?\}) \lhd Cether' = \\
\qquad (\{Dname?\} \times \{Dname?\}) \lhd Dether' \\
\hline
\end{array}
$$

[5]The axiom demands something stronger than just that the ‘or’ in (5) be ‘exclusive’. See [8].

[6]Note that this is a considerable simplification compared to [18]. In [18] it is the case that: (i) the models do not concern themeselves with details of physical message transmission, (ii) the relevant data can nevertheless be inferred indirectly from the message body.

5 Building the Tower

Thus far we have described in moderate detail the way that retrenchment allows the real world finiteness of the purse log to be taken account of, in way that did not disturb the preexisting development. Basically we retrenched the lowest level C model of the earlier development to the D model in which logs were bounded. This gives us the bottom of the tower anticipated in Section 3. In this section we sketch how the new D model can be related to the other models in the Mondex development, completing the tower, and clarifying the relationship between log boundedness and the concerns of these higher level models. In a nutshell, we first lift the D model to the level of abstraction of the B model (giving the E model) and then it is observed that there is a refinement from the A model to the E model, due to the nonintrusiveness of the D model. So we can make the F model just a copy of the A model as indicated in Fig. 2.

The lifting of the D model uses a generic lifting construction for lifting the concrete model of a retrenchment to the level of abstraction of the abstract system. The details for the present work are in [14], building on earlier work in [1]. A system, typically called U, is constructed out of the ingredients that specify the two original systems. The required level of abstraction is defined indirectly via a collection of properties specific to the construction, and U captures this level by being refinable to any system that also enjoys these properties, making U the most abstract such system. As regards the construction, any system interrefinable with U is just as good as U, so we have the option of replacing U with a more convenient system if we wish.

In the Mondex case we start by composing the B to C refinement with the C to D retrenchment, yielding a retrenchment from B to D; details are given in [2, 14]. The generic construction is then applied to this retrenchment.

For the sake of clarity, let us first apply this to the individual purse operation *AbortPurseOkay*. The generated system U turns out to be:

protoEAbortPurseOkay
$BAbortPurseOkay;\ \Delta DConPurse$
$Dm?, Dm! : DMESSAGE$

$(R_{BD} \wedge R'_{BD} \wedge O_{BD,AbortPurseOkay}) \vee$
$(R_{BD} \wedge C_{BD,AbortPurseOkay})$

In the preceding, *BAbortPurseOkay* 'is as' *CAbortPurseOkay*, and R_{BD} 'is as' R_{CD}. Similarly $O_{BD,AbortPurseOkay}$ 'is as' $O_{CD,AbortPurseOkay}$, and $C_{BD,AbortPurseOkay}$ 'is as' $C_{CD,AbortPurseOkay}$. In *protoEAbortPurse-Okay*, note that *BAbortPurseOkay* contributes the steps of the B model and $\Delta DConPurse$ contributes all legal D changes of state consistent with either preserving the retrieve and output relations, or establishing the concession.

Since we have a retrenchment from the B model to the D model, the retrenchment operation PO is satisfied, so that for every *DAbortPurseOkay* step there is a witnessing B model step. It is not hard to see that in *protoEAbortPurseOkay*, precisely the same witnessing B step establishes the validity of the model E to model D refinement correctness PO.

We observe that the E model operation features a considerable duplication of state and other information; the B and D parts of the state say more or less the same thing via the *BDConPurseAbortEquality* in R_{BD} and R'_{BD}; also the I/O is either irrelevant or can be inferred from the D element alone. Above, we noted that it is sufficient to have a system which is interrefinable with the generic U model. So we examine *protoEAbortPurseOkay* to see if we can achieve some simplification. In fact we can replace it with:

$$EAbortPurseOkay \text{ 'is as' } B_D AbortPurseOkay$$

The $\mathrm{B_D}$ model is a hybrid of the B model and the D model. It has an unbounded log, as do the B and C models, but the state space and I/O of the D model. Aside from the naming conventions, an idea of what $B_D AbortPurseOkay$ looks like can be obtained by replacing the last clause of *CAbortPurseOkay* with the last two clauses of *DAbortPurseOkay*.[7]

The preceding is of course just a precursor of the real action, which concerns the world level operation *DAbort* and its lifting to *EAbort*. Since the individual purse operation sits neatly inside the world level operation, we should be able to discern the purse level lifting inside the world level one. Since the lifting is a generic construction, the B world abort operation has the same shape as the D world one:

$$BAbort == BIgnore \vee$$
$$(\exists\ \Delta BConPurse \bullet \Phi BOp \wedge B_D AbortPurseOkay)$$

This is as expected, except that it conceals the fact that inside *BAbort*, ΦBOp, instead of having $\Delta BConWorld$ (as we would expect) we actually have $\Delta BetweenWorld$, where *BetweenWorld* features additional structure and constraints imposed on *BConWorld* in order to enable the A to B backward refinement to discharge. Otherwise the constituents of *BAbort* 'are as' their corresponding *CAbort* ones.

We do not have the space to go through the arguments to convince the reader that the constraints in *BetweenWorld* do

[7] As far as the various *Abort* operations of this paper are concerned, we could have made do with the D model operation *DAbortPurseOkay* itself as being interrefinable with *protoEAbortPurseOkay*. This remains the case if we also take the operations to clear the purses' logs into account. However, prior to clearing its log, a purse sends the entries in its log, one at a time, to a central archive. If we take the requisite operation into account, the log contents become externally observable. Now the protoE model has an unbounded log via the *BAbortPurseOkay* inside *protoEAbortPurseOkay*; thus the observabilty of the log contents forces *every* model interrefinable with protoE to have an unbounded log.

not materially affect our discussion. They express the consistency between the cryptographically protected messages in the ether and the purses' states. The interested reader can refer to [18]. We now retrace the earlier lifting construction and obtain:

$$\begin{array}{|l} \text{\textit{protoEAbort}} \\ \hline BAbort;\ \Delta DConWorld \\ Dm?, Dm! : DMESSAGE \\ \hline (R^{PW}_{BD} \wedge W^{PW}_{BD,Abort}) \wedge ((R'^{PW}_{BD} \wedge O^{PW}_{BD,Abort}) \vee C^{PW}_{BD,Abort}) \end{array}$$

In fact as before, we can replace *protoEAbort* by *EAbort* where:

$EAbort$ 'is as' $B_D Abort$

with $B_D Abort$ built the same way as $B_D AbortPurseOkay$ earlier, and *DetweenWorld* (which now 'is as' *BetweenWorld*) replacing occurences of *DConWorld* in *DIgnore* and ΦDOp, inside $B_D Abort$.

All that now remains is to construct model F. As it happens, the clean way that that the log full situation is dealt with, namely that the purse is prevented from initiating any more transactions until the log is cleared, means that the log full scenario becomes invisible in the A world since the A world needs no logs. This leads to the existence of a backward refinement from the A model to the E model. It is worth emphasising however that this is *not* a further instance of the lifting construction just used to build the E model, and is in fact a rather delicate (though very pleasing) property.

6 Validation

In most situations where a finite limit is an inescapable but undesirable feature of the implementation, the normal desire is that the limit is large enough not to be encountered in practice. This is certainly the case for one of the Mondex 'retrenchment opportunities', the one concerning the transaction sequence number, which was treated in [8]. The appropriate validation in that work involved showing that with a suitable choice of bound for the sequence numbers, the stochastic process whose events were the increments of the sequence number in successive transactions, had properties such that the possibility of actually *hitting* the bound, for all practical purposes lay far beyond the lifetime of the purse itself. That this was a realistic and achievable state of affairs in the sequence number scenario, was due to the fact that a one bit increase in the storage afforded to the sequence number on the smartcard leads to a doubling in capacity, because of the binary encoding of sequence numbers to bitstrings. This of course rapidly leads to a situation in which the sequence number bound is no threat.

However the scenario we face in this paper is significantly different. Exception log entries matter for their contents, not just for the total number of them. So we are faced with an essentially unary rather than a binary encoding. Furthermore, actual Mondex log entries are data structures of significant size (we did not need to go into details in this paper) and smartcard storage is tight. So there is room for not very many log entries at all in an actual Mondex purse; in reality there is room for only about 5 of them.

Our construction of the retrenchment from an unbounded log to a bounded one in earlier sections, was parameterised by the bound (though the actual value of the bound remained unstated till the previous sentence). The course of action pursued when the system reached the bound, i.e. when the retrenchment's concession became valid, was to just output a suitable message informing the user that the purse was for the time being, unusable, and that he/she ought to go to the bank to have the purse reset. This was a metaphor for what the real implementation does, which is to exhort the user to take the purse to the bank, so that the log entries may be archived centrally, and as a result, any funds genuinely lost in transit may eventually be restored to their rightful owner, following the reconciliation of purses' logs at a global level.[8]

Our job in this section is validation. We therefore ask the question whether the course of action taken when the concession becomes valid can be justified under the circumstances. The answer is a resounding yes. In footnote 8 we explained the connection between the contents of purses' logs, and the global satisfaction of the 'All value accounted' security property. Consequently, it is vital, when a purse's log is full, that it be prevented from engaging in any further transactions, since there is no guarantee that any such transaction is bound to succeed, and there is nowhere to record the details if it does not. Since the latter could cause the failure of the security invariant, it must be disallowed, and the only permissible option is thus inaction, since that will assuredly maintain any hitherto established invariant. We conclude that the design of the system as regards the log's limited size is entirely appropriate, and the retrenchment is thus validated.[9]

[8] In a little more detail, what happens is the following. If a purse cannot complete its current transaction, for whatever reason, it aborts it. Some such aborts turn out to be harmless (eg. the sender aborting a transaction before the money is sent), and some are critical (eg. a receiver aborting a transaction while the money is still in flight). Unfortunately an individual purse has no way of knowing for sure whether any abort it performs is harmless or critical. However it is a property of the concrete purse protocol that a transaction has genuinely lost money iff both sender and receiver purses have aborted and logged the transaction while in their specific critical states. This, roughly speaking, corresponds to the concrete manifestation of the 'lost' outcome of the 'All value accounted' security property. It therefore follows that a global reconciliation of all purses' logs is needed before all lost funds can be definitively traced.

[9] We noted earlier that the clean design of the behaviour upon reaching the limit was responsible for the possibility of having a backward refine-

Looking back we see that even though the structure of the retrenchments and refinements in this paper is very much like that of their counterparts in [8], as indeed is their arrangement into the appropriate *Tower Pattern*, the validation that justifies them is entirely different. This underlines the fact that the formal models of a development alone give only a limited picture. Reinforced also, is the role of retrenchment as a middleman between the formal modelling aspects, and domain specific requirements issues.

We can contrast the preceding with how other approaches tackle the same situation (thus moving subtly from validating the specific retrenchments constructed in this paper, to validating the retrenchment approach as a whole).

There are essentially three ways that refinement approaches deal with finite limit situations. The first, the 100% honest way, is to dismiss completely the unbounded model as a member of the development path. So both the abstract model and the concrete model feature exactly the same limit, and the refinement between them is completely correct. Unfortunately this fails to address the *human* desire to view the unbounded case as a valid precursor of the bounded one, and also (and more importantly) fails to separate requirements concerns. In our case, it is far more important that the purse protocol can be proved robust than that the full log problem be handled in some particular way. Therefore there is the natural human desire to concentrate on the protocol first, and then to retrofit the handling of full logs later. If an incompatibility is found between these, then there is a strong preference to change the handling of full logs to fit the protocol, rather than the converse. Such an approach however cannot be carried through in a neat top-down manner in conventional refinement. One would need an iterative re-engineering of the refinement development which is expensive and does not scale. This can impede truly large developments. The approach via retrenchment does not suffer this drawback; moreover (and most importantly) the full rigour of the 100% honest refinement approach can be recovered by applying the *Tower Pattern* construction, and navigating the tower up the newly constructed side, as we saw clearly in earlier sections.

A second way in which refinement can deal with finite limit situations is to in effect adopt some of the flexibility espoused in retrenchment, by allowing the enabledness criteria of operations to become strengthened in the bottommost refinement step. This resembles a use of the within relation of retrenchment by another name, and this method of dealing with implementation level restrictions, usually refered to as conditional refinement, is eloquently demonstrated in the account of chapter 17 of [12], amongst others. We make a number of comments on this technique. Firstly, interpreted within the framework of Section 3, it amounts to a dereliction of the applicability condition captured in (2), confirming that it falls short of being a refinement in our sense. Secondly, it can only give conditions which restrict the applicability of refinement, and cannot give a formal account of what happens when those conditions fail, which may be of vital importance from the system engineering point of view (as it is in our case). From a retrenchment perspective, this latter point comes down to the tradeoffs between using either the within relation or the concedes relation to cater for some requirements issue. The duality between these is not precise, since a weaker within relation gathers more pairs of concrete/abstract steps into the remit of the retrenchment than does a stronger within relation and a correspondingly modified concession (cf. equation (5)). We claim it is preferable to use the within relation to express constraints that are genuinely impossible to violate (whether for physical or logical reasons), such as the impossibility to 'switch on' a physical on/off switch when it is already in the 'on' position. It is better to use the concession to deal with constraints whose policing one has to engineer inside the developed system, and for which one can envisage and define contingencies should the constraints be breached.

Returning to finite limits, whereas a case can be made for dealing with a genuinely unattainable limit (eg. for sequence numbers) via a within relation or by conditional refinement (since an analysis like the one in [8] can be targeted at the within relation), the same cannot be said for small limits, like the one we have in this paper. For these, a conditional refinement can only discuss what happens in a subset of the expected behaviours of the system, and so is inadequate. The refinement riposte to this is to say that violation of a small limit, leading to genuinely different system behaviour, is a top level event, needing to be included in the abstract model. This goes back to the 100% honest approach we discussed above, with its lack of separation of concerns. Of course the conditional refinement approach, by insisting that a conditional refinement is still a refinement, tends to deprive itself of the opportunity of building a tower, with the latter's advantage of being able to combine the merits of a pragmatic separation of concerns with the rigour of the 100% honest side of the tower.

A third way in which refinement can deal with finite limit situations is to approach them ... via limits (in the metric topology sense). The approach has its origins in the thesis of Neilson [15] which introduced the idea of 'acceptably inadequate refinements'. (Essentially an interchange of two quantifiers takes us from the definition of the ideal model with an infinite domain to one with a finite domain.) More recently, this idea has been extended to an approach to refinement with approximations in [11], which uses metric topology to quantify the deviation of a model with a finite

ment from model A to model E, enabling model F to just be a copy of model A. We now see that in the case of the current 'retrenchment opportunity', this was in fact forced by the security invariants.

domain from an ideal one with an infinite domain. Provided that the quantitative properties of the metric have been fully validated against system engineering imperatives (and this is an important proviso), it can yield a plausible approach in eg. the sequence number scenario, in which the bound on possible sequence numbers could be regarded as being ‘close enough to infinity’. But in eg. the purse log scenario, it is hardly likely that 5 could be regarded as being ‘close enough to infinity’ in any meaningful sense. The probable rejoinder to this might be that small limits such as this need to be treated via the 100% honest approach already discussed. Thus the metric approach is applicable in fewer scenarios than retrenchment, and does not in itself save any work at the system engineering level, because of the explicit validation of the metric that is needed. The disciplined use of retrenchment, eg. in a *Tower Pattern* framework, has the potential to organise the validation that is required anyway, without imposing the additional burden of necessarily having to find a metric to base it on.

7 Conclusions

In the preceding sections, we have introduced the Mondex development and its ‘retrenchment opportunities’, as well as the *Tower Pattern* and supporting refinement and retrenchment notions. We then selected one of the retrenchment opportunities, the finiteness of the purse log, and examined its treatment via the tower in detail. What we obtained was in effect two refinement developments, dealing with different levels of detail (and therefore implicitly focused on different spheres of concern), connected via a collection of retrenchments.

Crucial to the utility of such a treatment is validation. Retrenchment is itself an extremely permissive formal technique, not capable of offering the guarantees that refinement can, so a validation from the domain perspective, is vital in corroborating any formal development done with its help. We were able to provide clear validation for our case study, amply vindicating the approach used.

References

[1] R. Banach. Maximally abstract retrenchments. In *Proc. IEEE ICFEM2000*, pages 133–142, York, August 2000. IEEE Computer Society Press.

[2] 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.

[3] 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.

[4] R. Banach and M. Poppleton. Sharp retrenchment, modulated refinement and simulation. *Formal Aspects of Computing*, 11:498–540, 1999.

[5] R. Banach and M. Poppleton. Retrenching partial requirements into system definitions: A simple feature interaction case study. *Requirements Engineering Journal*, 8(2), 2003. 22pp.

[6] R. Banach, M. Poppleton, and C. Jeske. Retrenchment and promotion in Z. submitted for publication, 2005.

[7] R. Banach, M. Poppleton, C. Jeske, and S. Stepney. Retrenching the purse: The balance enquiry quandary, and generalised and (1,1) forward refinements. Submitted.

[8] R. Banach, M. Poppleton, C. Jeske, and S. Stepney. Retrenching the purse: Finite sequence numbers and the tower pattern. In J. Fitzgerald, I. Hayes, and T. A., editors, *Formal Methods 2005*, volume 3582 of *LNCS*, pages 382–398, Newcastle, UK, 2005. Springer.

[9] G. Barthe, P. Courtieu, P. Dufay, and M. de Sousa S. Tool-assisted specification and verification of the javacard platform. In H. Kirchner and C. Ringeissen, editors, *AMAST 2002*, volume 2422 of *LNCS*, pages 41–59. Springer, 2002.

[10] D. Bert, J. Bowen, S. King, and M. Waldén, editors. *Proc. ZB2003: Formal Specification and Development in Z and B*, volume 2651 of *LNCS*, Turku, Finland, June 2000. Springer.

[11] E. Boiten and J. Derrick. Formal program development with approximations. In H. Treharne, S. King, M. Henson, and S. Schneider, editors, *Proc. ZB 2005: Formal Specification and Development in B*, volume 3455 of *LNCS*, pages 374–392, Guildford, UK, 2005. Springer.

[12] M. Broy and K. Stølen. *Specification and Development of Interactive Systems, Focus on Streams, Interfaces, and Refinement*. Springer, 2001.

[13] J. Derrick and E. Boiten. *Refinement in Z and Object-Z*. FACIT. Springer, 2001.

[14] C. Jeske. *Algebraic Integration of Retrenchment and Refinement*. PhD thesis, University of Manchester, 2005.

[15] D. Neilson. *From Z to C: Illustration of a Rigorous Development Method*. PhD thesis, Oxford University Programming Research Group, 1990. Technical Monograph PRG-101.

[16] D. of Trade and Industry. Information Technology Security Evaluation Criteria, 1991. http://www.cesg.gov.uk/site/iacs/itsec/media/formal-docs/Itsec.pdf.

[17] M. Poppleton and R. Banach. Controlling control systems: An application of evolving retrenchment. In D. Bert, J. Bowen, M. Henson, and K. Robinson, editors, *Second International Conference of B and Z Users*, volume 2272 of *LNCS*, pages 42–61, Grenoble, France, January 2002. Springer.

[18] S. Stepney, D. Cooper, and J. Woodcock. An electronic purse: Specification, refinement and proof. Technical Report PRG-126, Oxford University Computing Laboratory, 2000.

[19] S. Stepney, F. Polack, and I. Toyn. An outline pattern language for Z. In Bert et al. [10], pages 2–19.

[20] S. Stepney, F. Polack, and I. Toyn. Patterns to guide practical refactoring. In Bert et al. [10], pages 20–39.

[21] J. Woodcock and J. Davies. *Using Z: Specification, Refinement and Proof.* Prentice-Hall, 1996.