Retrenchment and the Mondex Electronic Purse (Extended Abstract)

Richard Banach[1], Michael Poppleton[2] Czeslaw Jeske[1] and Susan Stepney[3]

[1] Department of Computer Science, University of Manchester,
Manchester M13 9PL, UK,
{banach,cj}@cs.man.ac.uk
[2] Department of Electronics and Computer Science,
University of Southampton, Highfield,
Southampton SO17 1BJ, UK,
mrp@ecs.soton.ac.uk
[3] Department of Computer Science, University of York,
Heslington, York YO10 5DD, UK,
susan.stepney@cs.york.ac.uk

Abstract. Some of the success stories of model based refinement are recalled, as well as some of the annoyances that arise when refinement is deployed in the engineering of large systems. The way that retrenchment attempts to alleviate such inconveniences is reviewed. The Mondex Electronic Purse formal development provides a highly credible testbed for examining how real world refinement difficulties can be treated via retrenchment. The contributions of retrenchment to integrating the real implementation with the formal development are surveyed, and the extraction of commonly occurring 'retrenchment patterns'is suggested.

1 Refinement: Pros and Cons

Model based refinement is well known as the standard technique for progressing abstract system designs towards implementations. The abstract designs are typically expressed in a modelling language permitting the maximum of expressivity, abstraction, mathematical rigour, and succinctness, without concern for executability. The lower level models lean increasingly towards the actual capabilities of real computing devices, and the algorithms that they must utilise. There are a number of specific formulations of model based refinement, which can differ as regards particular technical details, but which share the same overall strategy for establishing the correctness of an implementation: namely that for every run of the concrete system, there must be a run of the abstract system which maintains the desired notion of correct correspondence between them. Among the more well known techniques we can mention Z [24, 31, 16], B [1, 23, 19], VDM [12, 18, 7], RAISE [14, 30] and ASM [10, 8, 21, 22].

Besides being well established in the academic sphere, refinement has had notable successes on the industrial front in recent years. We can cite the Mondex Purse [28, 29] and Multos Operating System [27, 26] for Z, the MÉTÉOR project [6] and numerous other railway system projects in France and elsewhere for B, and a number of language definitions and language abstract machine definitions for ASM [25, 9, 15].

Despite these undoubted successes, practitioners have known for some time that when refinement is used as the sole means of progressing from an abstract model to a concrete one, then certain difficulties can plague the development process due to the exacting nature of typical refinement proof obligations. This is not a technical difficulty with refinement, rather it is a manifestation of human inclination to view certain things as abstractions/concretisations of the same phenomenon, that some given refinement formalism does not permit to be so viewed. Since the human notion of abstraction is inevitably imprecise, and the mathematical notion of abstraction pertaining to any specific refinement formalism is *de facto* extremely precise, some dislocation between the two is bound to occur sometimes.

Usually, if the scale of the problem is small, this dislocation can be overcome easily enough. Frequently it is sufficient to make some small adjustment to one or other of an abstract/concrete pair of models to bring them into line. However, when the problem size is large, such manipulations can become prohibitive; this may be on grounds of sheer cost, or on more prosaic grounds. For large problems, there are usually stakeholders other than the refinement specialists involved, who 'own' the models in question, and they may simply not agree to changes in the models as suggested by the refinement specialists, regardless of the latters' protestations. Thus the human aspects of the development milieu become paramount. This is nothing more than a corollary of the fact that the construction of large systems is an engineering problem, and not purely a problem in formal system construction. The key desiderata in the two domains are just different.

A simple and commonly occurring example arises with natural number arithmetic. Implementable whole numbers are invariably finite. So arithmetic always generates within-bounds and out-of-bounds cases. If there are n different quantities in the model, then for a typical operation there will be one all-within-bounds case, and easily of the order of $2^n - 1$ out-of-bounds cases[1] of various flavours to consider in the model. In the system specification, the syntactic descriptions of the latter can often swamp that of the single all-within-bounds case, which is the one of most interest. For such reasons it is often desirable to idealise the arithmetic and use unbounded naturals at the abstract level; these cause no exceptions. Unfortunately in the overwhelming majority of refinement formalisms there is no refinement from unbounded naturals to bounded ones that handles a sensible selection of the operations that are normally needed.

2 Retrenchment

Retrenchment [2, 3, 4] was introduced in order to be able to address the difficulties caused by the situation discussed above, and many others in which the humans' idea of 'abstraction' is in conflict with what is permitted by some notion of model based refinement. It proceeds by inverting the usual trajectory from broad principles to proof obligations found in refinement. In refinement, the starting point is a notion of correct correspondence between abstract and concrete models, from which proof obligations are derived. In retrenchment by contrast, the proof obligations are manipulated so that they can encompass situations like the example above, and whatever broad principles

[1] Or more, depending on the details of the required operation.

can be derived therefrom, are sought. Certainly the typical guarantees offered by refinement are forfeit.

To illustrate retrenchment, we take a forward refinement operation proof obligation and turn it into a retrenchment proof obligation. For the former we take:

$$\begin{aligned} & R(u,v) \wedge RIn_{Op}(i,j) \wedge \mathrm{pre}(Op_A)(u,i) \wedge Op_C(v,j,v',p) \Rightarrow \\ & \quad (\exists\, u',o \bullet Op_A(u,i,u',o) \wedge R(u',v') \wedge ROut_{Op}(o,p)) \end{aligned}$$

In the above u, v are (Abstract/Concrete) states (primed for after-states), i, j are (A/C) inputs, o, p are (A/C) outputs. R is the retrieve relation, while $RIn_{Op}, ROut_{Op}$ are input/output relations respectively. Especially when strengthened by suitable assumptions about $R, RIn_{Op}, ROut_{Op}$, the above is equivalent to typical operation POs in the literature. To turn it into a retrenchment PO, we modify it to:

$$\begin{aligned} & R(u,v) \wedge W_{Op}(i,j,u,v) \wedge Op_C(v,j,v',p) \Rightarrow \\ & \quad (\exists\, u',o \bullet Op_A(u,i,u',o) \wedge ((R(u',v') \wedge O_{Op}(o,p;\ u',v',i,j,u,v)) \vee \\ & \qquad\qquad\qquad\qquad\qquad C_{Op}(u',v',o,p;\ i,j,u,v))) \end{aligned}$$

Now, $RIn_{Op} \wedge \mathrm{pre}(Op_A)$ has been generalised to the within relation $W_{Op}(i,j,u,v)$ which is an arbitrary relation in the before-values; $ROut_{Op}$ has been generalised to the retrenchment output relation $O_{Op}(o,p;\ u',v',i,j,u,v)$ which now allows all the variables to occur; and, most importantly, there is a concedes relation $C_{Op}(u',v',o,p;\ i,j,u,v)$ to describe what happens if the retrieve relation cannot be reestablished by a given pair of (A/C) steps.[2] From such a starting point, one derives what broad principles one is able to.

The flexibility introduced into formal development by retrenchment lends itself to many uses. At one extreme, it can be restricted to the kind of situation outlined above, namely handling irritating restrictions forced onto the development by the finiteness or other limitations of implementable data types [2]. At the other extreme, it can be used to capture very general evolution of system definitions, as new considerations impact preliminary models on the route to the final system description [5]. How far along this scale of possibilities one happens to be, depends on one's perspective about a particular change in system description. The same change in the system might be viewed by one person as a system evolution, and thus as residing firmly in the requirements engineering arena, while for another person, it could be very much tied up with the road to an implementation, and thus be viewed as a development step; much can depend on whether the individual is focused on user needs, or technology capabilities. It is important to see that from a purely engineering vantage point, there is no hard and fast boundary between these two activities: requirements evolution can blend smoothly into development and implementation. This is in contrast to the formal refinement vantage point in which the boundary is clear: anything that cannot be captured by a refinement must be a requirements evolution step. Retrenchment can build a dialogue between these two perspectives.

[2] The semicolons in O_{Op}, C_{Op} are purely cosmetic, separating the variables of 'most interest' from others, which are permitted, but less often needed.

3 The Mondex Purse and its Retrenchment Opportunities

We focus now on the Mondex Purse [29], mentioned above. This was an industrial scale development of an electronic purse smartcard application, enabling cash-like financial transactions to be performed without the use of any physical money. Central to the certification of Mondex, which achieved ITSEC level E6 (equivalent these days to Common Criteria EAL 7, [20]) is a refinement from an atomic operation for money transfer between two purses, to a distributed algorithm that fairly closely models the code for the money transfer protocol actually used (and that had previously been written[3]). Despite the impeccable refinement, a number of 'retrenchment opportunities' pertaining to gaps between the refinement models and the reality of the code, nevertheless remained. We summarise these now:

1. Sequence Number: The integrity of the protocol depends partly on the sequence number of the transaction in progress. Sequence numbers occur in the Mondex concrete model where they are naturals; in reality they are bounded numbers.
2. Log Full: Transfers completing abnormally are logged by purses. The concrete model implements the abstract 'lost value' component in terms of an off-card exception log. The card needs to be assured that the data is stored in that log before it can clear it from its own, highly constrained, log memory. Logs occur in the Mondex concrete model where they are unbounded; in reality they are finite.
3. Hash Function: Clearing a purse's log after its contents are centrally archived is done via a message containing 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. In reality of course a cryptographic hash function is used, which is neither total, nor injective, but is informally argued to be 'sufficiently injective'.
4. Balance Enquiry: Each purse has a balance enquiry operation. If this is invoked at a particular point in the middle of a concrete model value transfer, a discrepancy can occur between the abstract and concrete balances due to differences in where non-determinism is resolved in the two models. This is handled formally by a modeling trick, using finalisation instead of the enquiry operation to observe the state.

Let us now sketch what retrenchment can contribute regarding these various topics.

As regards the finiteness of sequence numbers, this is like many simple retrenchment examples in the research literature. It is not hard to write down a model which is like the Mondex concrete model except that the sequence numbers are bounded, and then to draw up a provable retrenchment between them. The greater interest lies in integrating this development step with the rest of the Mondex development. It turns out that the detail introduced in the new model can be lifted to the level of abstraction of the higher models of the development, and moreover, if one is particularly careful about how the new model has been constructed, it can turn out to be a refinement of the abstract one, though this is by no means a robust property.[4] A byproduct of the

[3] As often happens in system engineering, the implementation preceded the abstraction.

[4] The latter hinges on the fact that in Mondex, incrementing the sequence number happens as the first step of a transaction. Since transactions are allowed to fail in both abstract and concrete models, failure to increment can be amalgamated with other kinds of failure in a carefully constructed model.

retrenchment formulation is that it allows validation of the adopted sequence number limit to focus on an ingredient of the formal models (the relevant concession), which would not be possible in a purely refinement based treatment.

As regards the finiteness of the purse logs, this has many aspects that resemble the previous case. Again it is easy enough to construct a model that has a finite log, and to pursue a strategy that lifts the detail to the higher level models of the Mondex development. The differences from the previous case centre on validation aspects. For sequence numbers, the aim is to analyse the concession in order to choose a limit that will never be encountered in a purse's lifetime. For the log, since the actual log size is fixed at around 5, the preceding aim is unrealistic, and validation focuses on ensuring that the user receives appropriate feedback in order that he might be persuaded to go into the bank to have the log contents centrally archived before further use of the purse. This amounts to a different treatment of the models, and of the incompatibility between them captured in the concessions that connect them.

Note that both of the above cases feature a 'finite limit' phenomenon. A complete lower level model will contain both phenomena (as well as other things) and the description of each operation will therefore break up into (at least) four cases depending on whether the sequence number and/or purse log are still within bounds. This is already an example of the $2^n - 1$ case proliferation noted above, even though only increment operations are required for the two data items. It becomes clear that relegating the concerns regarding these details to a lower level model, as permitted by the retrenchment approach, leaving earlier models to concentrate on core functionality, is very worthwhile.

As regards the non-injectivity of the hash function, it is evident that one can write a model in which the abstract total injective function from purses' log contents to the clear codes is replaced by a hash function which is less than total and injective. The concession of the retrenchment between these models will refer to the loss of the 'all value accounted' global security property which states that even in the face of failing transactions, sufficient records are maintained across the system that all the original funds in the system can be properly accounted for. The validation of this scenario focuses on the statistical likelihood that the concession might be made valid, i.e. that a purse receives in error a clear-log message with just the right properties to make it believable.

The preceding retrenchment opportunities were all 'localised' in that each of the discrepancies discussed could be viewed as being rooted in a single operation at a time.[5] The final one, the balance enquiry quandary, is more complicated, necessitating the consideration of sequences of operations.

As noted before, in the Mondex abstract model, the atomic transfer operation captures failing transactions as well as successful ones; this is so that failing distributed transactions are able to be refinements of something abstract. This implies *early* resolution of nondeterminism in the abstract model. In the concrete model, protocol failure is predictably detected *late*. The purses' environment is assumed hostile, and the protocol might be interrupted at any moment, so each operation must alone preserve the system's global security invariants (which concern the aggregated balance of funds in the entire

[5] For example, even though sequence numbers are incremented within several purse operations, each such operation could be individually retrenched, without needing to refer to the others.

community of purses). The most straightforward way of achieving this is to have each concrete operation refine some abstract one. Inserting a balance enquiry into the middle of a concrete transfer[6] can now reveal the temporary difference between abstract and concrete balances caused by the differing nondeterminism resolution points at the two levels. Retrenchment can contribute three perspectives to this scenario. Firstly, the flexibility of the retrenchment output relation can accommodate the discrepancy in balances that arises, in a way that makes it plain that this does not constitute a breakdown in the global system invariants. Secondly it can accomplish the preceding via either forward or backward retrenchments (the Mondex refinement in question is a backward one). Thirdly, via a more fine grained formulation, it can relate the preceding to a generalised forward refinement [8, 21, 22] account of the balance enquiry quandary, since, at heart, the balance enquiry quandary is nothing more than an inversion of the order of performing two operations in the passage between levels of abstraction, an inversion which gets camouflaged through the backward refinement.

4 Patterns of Retrenchment

Contemporary trends in software construction have highlighted the usefulness of identifying common structural *patterns* for accomplishing frequently occurring tasks [13, 17, 11]. The experience with Mondex leads us in a similar direction for deploying retrenchment in dealing with commonly occurring disparities between ideal specification metaphors and the reality of the systems that they become implemented in. Perhaps the most common such situation concerns the infiniteness of convenient modeling data types versus their finite implementable counterparts. The Mondex experience shows that issues such as these do indeed easily fall into a pattern. One has first the ideal but nonimplementable refinement based development, depicted as, say, a vertical column of successive refinements, going from the most abstract model to the most concrete. By its side we have an analogous column of refinements, this time involving models using the implementable data types, making those models much more detailed as a result. The two columns are bridged by a collection of horizontal links, these being the retrenchments between corresponding ideal and realistic models. Such a *tower pattern* is easily discerned within Mondex for the sequence number and local log issues. Moreover, it is clear that this pattern is largely issue-independent: it will will work well even when the modeling disparity does not concern finiteness issues, such as for instance in the hash function scenario. Further experience with Mondex and other industrial scale formal developments is likely to throw up additional 'pattern opportunities' for retrenchment.

[6] Ordinarily it makes no sense to do this, but purses cannot afford to rely on common sense.

References

[1] J.-R. Abrial. *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, 1996.

[2] 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.

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

[4] R. Banach and M. Poppleton. Engineering and theoretical underpinnings of retrenchment. submitted, http://www.cs.man.ac.uk/˜banach/some.pubs/Retrench.Underpin.pdf, 2002.

[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] P. Behm, P. Desforges, and Meynadier J-M. Météor: An industrial success in formal development. pages 374–393.

[7] J. Bicarregui and B. Ritchie. Invariants, frames and postconditions: a comparison of the vdm and b notations. volume 670 of *LNCS*, pages 162–182. Springer, 1993.

[8] E. Börger. The asm refinement method. *Formal Aspects of Computing*, 15:237–275, 2003.

[9] E. Börger and D. Rosenzweig. The WAM – Definition and Compiler Correctness. In C. Beierle and L. Plümer, editors, *Logic Programming: Formal Methods and Practical Applications*, pages 20–90. North-Holland, 1994.

[10] E. Börger and R.F. Stärk. *Abstract State Machines. A Method for High Level System Design and Analysis*. Springer, 2003.

[11] B. Bruegge and Dutoit A. H. *Object-Oriented Software Engineering: Using UML, Patterns and Java*. Prentice Hall, 2003.

[12] J. Dawes. *The VDM-SL Reference Guide*. UCL Press/ Pitman Publishing, London, 1991.

[13] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison-Wesley Professional, 1995.

[14] RAISE Method Group. *The RAISE Method Manual*. Prentice Hall, 1995.

[15] Y. Gurevich and J. Huggins. The Semantics of the C Programming Language. In E. Börger, H. Kleine Büning, G. Jäger, S. Martini, and M. M. Richter, editors, *Computer Science Logic*, volume 702 of *LNCS*, pages 274–309. Springer, 1993.

[16] ISO/IEC 13568. *Information Technology – Z Formal Specification Notation – Syntax, Type System and Semantics: International Standard*, 2002. http://www.iso.org/iso/en/ittf/PubliclyAvailableStandards/c021573_ISO_IEC_13568_2002(E).zip.

[17] X. Jia. *Object-Oriented Software Development in Java: Principles, Patterns, and Frameworks*. Addison Wesley Longman, 1999.

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

[19] Lano K. and Haughton H. *Specification in B: An Introduction Using the B-Toolkit*. Imperial College Press, 1996.

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

[21] G. Schellhorn. Verification of ASM refinements using generalized forward simulation. *JUCS*, 7:952–979, 2001.

[22] G. Schellhorn. Asm refinement and generalisations of forward simulation in data refinement. *Theoretical Computer Science*, 2005.

[23] S. Schneider. *The B-Method*. Palgrave Press, 2001.

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

[25] R.F. Stärk, J. Schmidt, and E. Börger. *Java and the Java Virtual Machine: Definition, Verification, Validation*. Springer, 2000.

[26] S. Stepney. New horizons in formal methods. *The Computer Bulletin*, pages 24–26, 2001.

[27] S. Stepney and D. Cooper. Formal methods for industrial products. pages 374–393.

[28] S. Stepney, D. Cooper, and J. Woodcock. More powerful Z data refinement: Pushing the state of the art in industrial refinement. In J.P. Bowen, A. Fett, and M.G. Hinchey, editors, *11th International Conference of Z Users*, volume 1493 of *LNCS*, pages 284–307, Berlin, Germany, September 1998. Springer.

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

[30] H.D. Van, C. George, T. Janowski, and R. Moore. *Specification Case Studies in RAISE*. FACIT, Springer, 2002.

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