DOI 10.1007/s00165-007-0060-5
BCS © 2007
Formal Aspects of Computing (2008) 20: 5–19

The certification of the Mondex electronic purse to ITSEC Level E6

Jim Woodcock, Susan Stepney, David Cooper, John Clark and Jeremy Jacob

Department of Computer Science, University of York, Heslington, YO10 5DD York, UK.
E-mail: jim@cs.york.ac.uk

Abstract. Ten years ago the Mondex electronic purse was certified to ITSEC Level E6, the highest level of assurance for secure systems. This involved building formal models in the Z notation, linking them with refinement, and proving that they correctly implement the required security properties. The work has been revived recently as a pilot project for the international Grand Challenge in Verified Software. This paper records the history of the original project and gives an overview of the formal models and proofs used.

Keywords: Certification; Correctness; Electronic finance; Grand challenges;
Grand Challenge in Verified Software; ITSEC Level E6; Mondex; Refinement;
Security; Smart cards; Theorem proving; Verification; Verified Software Repository; Z notation

1. Introduction

The Mondex electronic purse was certified to ITSEC Level E6 in 1997. The modelling, refinement, and proof of correctness were carried out by Stepney and Cooper, with consultancy from Woodcock; Clark and Jacob carried out the evaluation.

1.1. Structure of the paper

The first part of the paper gives an historical account of the certification on the Mondex electronic purse. Section 2 describes the specifiers' tale and Sect. 3 describes the evaluators' tale. The following two Sects. (4 and 5) draw some lessons from certification as a commercial activity and describe the impact the work has had on research in computer science. Section 6 introduces the formal description of Mondex in more detail; it describes two key models and their role in the proof of security properties. Section 7 explains in more detail the structure of the models and the proof. Section 8 describes the abstract model security properties, and Sect. 9 describes the concrete model security properties. The final section in this part of the paper, Sect. 10, describes the abstract model and its security policy. In Sect. 11, we describe Mondex as a pilot project in the international Grand Challenge in Verifying Software. We conclude the paper in Sect. 12. A slightly different version of the material in Sect. 2 has previously appeared as part of [Ste01]. The overview of Mondex is based on that given in [SCW00].

Correspondence and offprint requests to: J. Woodcock, E-mail: jim@cs.york.ac.uk

2. The specifiers' tale

In the early 1990s, *platform seven* (at that time part of the National Westminster Bank) started developing a smartcard-based electronic cash system, *Mondex*. A smartcard is a piece of plastic resembling a credit card, but with an embedded programmable microprocessor. Mondex is a smartcard application designed to work like electronic cash, suitable for low-value cash-like transactions, with no third-party involvement, and no cost per transaction. Once the issuing bank has loaded electronic cash into the system, it has very little further control over it: just like real cash.

Because of this lack of third-party control, it is crucial that the security of the card cannot be broken; otherwise criminals could electronically "print" money with ease. So *platform seven* decided to develop the full Mondex product to one of the very highest standards available at the time: ITSEC Level E6 [ITS91].[1] This mandates stringent requirements on software design, development, testing, and documentation procedures, and also mandates the use of formal methods to specify the high-level abstract security policy model, to specify the lower-level concrete architectural design, and to provide a formal proof of correspondence between the two levels, in order to show that the concrete design enjoys the abstract security properties.

The target platform smartcard (state-of-the-art in the early 1990s) had an 8-bit processor, a low clock speed, limited memory (256 bytes of dynamic RAM, and a few kilobytes of slower EEPROM), and no built-in operating system support for tasks such as memory management (it was likened by some to "a BBC micro on plastic").[2] Also, power could be withdrawn at any point during the processing of a transaction, if say the card was withdrawn from the reader. Engineers at *platform seven* designed and implemented the secure cash-transfer protocol under these severe space and speed constraints. Early in 1994, they contacted Logica to deliver the specification and proof requirements of the E6 development. Logica's formal methods team (Stepney and Cooper) chose the Z notation [Spi92a, WoD96] in which to write the two specifications and to perform the correspondence proof.

We had little difficulty formalising the concrete architectural design from the existing semi-formal design documents, but the task of producing an abstract security policy model that both captured the desired security properties (in particular, that "no value is created" and that "all value is accounted for") and provably corresponded to the lower-level specification, was much harder. A very small change in the design (very small from the perspective of the formal methods team, that is) would have made the abstraction much easier, but was deemed by *platform seven* to be too expensive, as the parallel implementation work was already beyond that point. Eventually, after much experimentation with different approaches, an abstraction of the original design was successfully constructed.

At that stage, the correspondence proof hit an unexpected snag. Although the Logica team were confident that the proof obligation implied by the Mondex specifications was true, they were unable to prove it using the standard proof rules for Z available in the literature at the time [Spi92a]. Consultancy with Woodcock (then at Oxford University Computing Laboratory) explained why: those particular proof rules are suitable for only some classes of specification, and Mondex appeared to be of a different class.

Oxford researchers had been aware for some time of the full underlying refinement theory, and that the existing Z proof rules were incomplete, but until this project they had never come across a major, real-world example of a system that seemed to required the full generality including "backwards" refinement. Hence these backwards rules had previously (and occasionally, subsequently) been dismissed by members of the academic community as being of no practical importance. The underlying refinement theory did not apply directly to Z, and considerable work was required to establish it in Z. The development of these new proof rules exposed many previously hidden assumptions and restrictions of the classical Z refinement rules. In particular, it led to the uncovering of the importance of the finalisation proof obligation. (The impact of finalisation goes further, and has for example sparked work on classification of covert channels [CSC05].)

Because these new rules were not established in the literature, their derivation had to be presented to a standard that would convince the ITSEC evaluators who were certifying the system. So we went back to first principles, and produced an 80-page derivation of a complementary set of Z proof rules that were applicable to the Mondex specification [CSW02].

We then took these raw rules, and applied them to the specific case of Mondex, and successfully discharged the proof obligation, by hand, with more than 200 pages of mathematics. The proof discovered a small flaw in

[1] ITSEC E6 corresponds to the later-introduced Common Criteria level 7 [CoCURL].

[2] The BBC Micro was designed and built by Acorn Computers Ltd for the BBC Computer Literacy Project in the early 1980s. The original 8-bit MOS Technology 6502 processor had between 16 and 128 kbytes of memory.

one of the minor protocols; this was presented to *platform seven* in the form of a security-compromising scenario. Since this constituted a real security problem, the design was changed to rectify it.

In 1999, Mondex achieved its ITSEC Level E6 certificate: the very first product ever to do so. As a part of the ITSEC E6 process, the entire Mondex development was additionally subject to rigorous testing, which was itself evaluated. Mondex International acquired the Mondex product and its sister product Multos after the completion of the evaluation and the award of ITSEC E6. No errors were found in Mondex in any part of the system subjected to the use of formal methods.

3. The evaluators' tale

The Logica CLEF[3] was contracted by *platform seven* to perform the ITSEC Level E6 evaluation of Mondex. Since the Logica formal methods team were providing the specification and proof, it was not appropriate for them to perform the evaluation of this part of the overall development. To achieve the necessary independence, this part of the evaluation was contracted to Clark and Jacob at the University of York.

The development was to be evaluated against ITSEC Level E6. One might think that evaluating a formal development would present no technical or procedural difficulties. This was not so; the novel nature of the enterprise raised several issues.

ITSEC Level E6 requires that the formal architecture be shown formally to satisfy the formal security policy model; however, was hand-proof acceptable, or did it have to be mechanical? ITSEC Level E6 case law had still to be established on this issue. It was clear that any decisions made on this issue would have a significant effect on future developments (most immediately on an even more ambitious development by the same team). E6 was meant to be attainable by secure developments; the current development was pushing the theoretical state of the art in several respects and was very much at the forefront of commercial developments. Indeed, the fact that reduced variants of the commercial purse development form the basis of a significant amount of formal methods research in 2007 testifies to the level of ambition demonstrated by the development team (and their sponsors). Given state-of-the-art capabilities around 1997, the evaluators accepted that a hand-proof was acceptable, believing that machine proof for such developments might well be required in future years. The aim of the highest level is not to *guarantee* error-free developments (this simply cannot be done); rather, it is to provide a *very significant* reduction in risk. It was felt that hand-proof offered just such an increase over the requirements of ITSEC Level E5. But there is clearly a price to pay, not least in evaluation effort! With a fully formal development (with machine-supported proofs) the evaluators would have had to check *what* was being specified and proved, examine the soundness of the rules used, and assess the integrity of the tool support. We believe that this would have been less effort (and assessment of tool integrity would carry over to future projects), but that tool integrity *still* raises significant issues in safety-critical developments, and the formal methods community to this day has produced little in the way of rigorous arguments for tool integrity (either for theorem provers or indeed *checkers*).

Another issue requiring interpretation became known as the "formal design gap". ITSEC Level E6 does not require a fully formal development—a feature of the standard that formal methods researchers may find a little surprising. Once the architecture has been proved to satisfy the formal security policy model, the rest of the development can proceed using semi-formal or rigorous design methods. Satisfying the requirements of ITSEC Level E6 could become something of a game. If the standard were to have real "bite", then the proof elements would have to be seen to attempt something significant. Following a very minor formal refinement with a significant amount of non-formal development would defeat the spirit of the standard. But it was felt that the Mondex formal refinement was easily of sufficient ambition. The ambition of the refinements attempted was reflected in the perceived risk reduction.

The evaluators accepted compromises, such as the non-injectivity of the cryptographic hash function, as practical and sensible. Indeed, refining non-forgeable messaging remains to this day a tricky issue, since the interaction of a protocol with a specific encryption method (and encryption invariably lies at the heart of such guarantees) is a very subtle implementation affair, also bringing with it a host of derived confidentiality requirements concerning keys. The formal aspects abstract away from such implementation details; lower level assessment is required. Such issues remain as research challenges.

It is important to stress that the Mondex development was not merely an exercise in formal methods: it was first and foremost a *systems and software development for a real client*. It was done for strictly commercial reasons.

[3] Commercial Licensed Evaluation Facility.

The artifacts developed were subjected to review procedures before release to the evaluators; the evaluators subjected them to independent review. The development proceeded iteratively until all parties were happy. There is a natural tendency to concentrate in academia on the formal artifacts (specifications, designs, and proofs) but these elements arise within an engineering management context. It was a development project that, mathematical squiggles aside, looked and felt much like any other high-quality development.

Although raising Evaluation Observation Reports (EORs) indicated focused and diligent activity, it could not per se serve as proof of adequate “review”. Specification *validation* is only partly a formal activity. “Are we specifying the right thing?” can be answered only with knowledge of the real world. The evaluators were helped in that the properties of interest were essentially *integrity* properties and very clearly appropriate ones at that. (Issues of confidentiality seem inherently more difficult, with many formalised models typically *partially* capturing the underlying *intent*.) As one might expect, the evaluators subjected the formal documents to line-by-line review (checking design representation elements and also that proof rules had been invoked and applied as required, etc.), forming and using checklists of potential formal errors. The formulation of such aids to review arose in part due to experimentation with mutation approaches. These ideas were later built into formal methods tool support with the use of mutation in Statechart design testing [Bur02] (after translation to a state machine representation in Z) and also security analysis via CSP specification mutation [SCP+03] (where consideration of automatically generated scenarios that distinguish the original from the mutant system was proposed for validating requirements). Probing formal specifications and designs with such techniques will prove an important formal testing technique in years to come. (Full blown mutation without tool support is impractical.) The practical issue of specification validation is a challenge for the research community. The more complex the requirements, the more such work will be needed. The purse is a significant, but by no means large system by modern standards. Provision of guidance in reviewing formal artifacts will aid practical take-up for major developments.

The ability to navigate a formal document efficiently is crucial to evaluation effectiveness and cost. The evaluators were helped by the provision of a summary proof tree and helpful naming and structuring conventions, and the recurrence of familiar arguments. Navigability and ease of assimilation are also important when changes are considered. The published purse system description is a reduced version of the system that had evolved over several iterations. When changes were made, prompted either by the developers or by the evaluators, those changes would be reassessed. Rapid re-assimilation of the material was needed, often after a gap of many months. Familiarity is a major factor in effective evaluation.

Finally, the evaluators believe that formal methods *work*; or to be more precise (for this is in the spirit of the piece) formal methods *can be made to work*. Part of the evaluation compared accompanying prose with its formalisation. This in itself is a revealing exercise. Were there other interpretations of the English text? Without the formalisation would the evaluators have been prompted to think in such detail and diligence about the natural language descriptions? The answer here is a clear “No”. If you “speak the language”, formality can be a great help. Although the refinement-proof aspects of Mondex have attracted most attention, we believe that formal specification without proof carries many benefits. In this we are at odds with those who maintain that this gives all the trappings of formality and none of the benefits. Our experience with Mondex and larger formal specifications confirms this.

Finally, the developer-evaluator relationship was not perceived as an adversarial one by either side. The developers were genuinely pleased when the evaluators indicated errors. Formal methods do not work by themselves; it is the people who make the adoption of formal methods in a project a success. The developers’ commitment to the final product quality was impressive. The mindset was right. Evaluation was challenging and hard work, but intellectually a fun affair too.

4. Lessons from a commercial development

The full specification and proof of the security-critical functionality of Mondex has been published [SCW00]. We are very pleased that this has led to other researchers using it as a case study for their own techniques. We have often been asked: “Why did you do X?” or “Why did not you make simplifying assumption Y?” This was usually said with genuine puzzlement, but sometimes with the overtones of “Fancy not thinking of *that!*” The answer has occasionally been, “Because the technology did not exist at the time”. In every other case (so far) it has been: “Because of another constraint”. This is usually the fact that the work was part of a large commercial development. For example, “Why did you do the proofs in such detail?” is answered, “Because they had to be evaluated to ITSEC Level E6”. (and also, “Because they had to be right!”)

To start with, it should be noted that the published version is not the full development:

"In order to produce a case study of a size appropriate for public presentation, much of the real functionality has had to be removed. Some of the structure of the larger specification has remained present in the smaller one, although it might not have been used had the smaller specification been written from scratch. This omitted functionality, whilst important from a business perspective, is peripheral to the central security requirements." [SCW00, p.5]

Most importantly, there are many diverse stakeholders involved in a commercial development, and they do not all perceive suggested "simplifications" in the same light. We noted the following reactions to change requests, and other contextual constraints, during the development work.

The customer may not *allow* you to change the specification. Specifications serve several purposes, including being used for certification, validation, and implementation. Often it is more costly to change a specification than it is to get the formalists just to work a bit harder in order to prove the desired properties. For example, this is what happened to the request to modify the balance enquiry operation to perform an *"abort"*, in order to make the abstract and concrete states readily comparable part-way through a transfer: the knock-on effects to the design and implementation (occurring in parallel) were deemed too costly.

Textbook refinement methods usually assume an initially blank sheet of paper; but reality never starts from a blank sheet. There are often given fixed points in the development process, making a purist approach simply infeasible. For example, in the case of Mondex, the concrete level specification was given, and we had to reverse-engineer an abstraction from it that satisfied the formalisation of the given security properties.

There is no *time* to change the specification. Problems can become apparent only at the very end of the refinement proof, when the very last lemma fails to be proved. In an ideal world, one would of course return to the beginning, make the necessary specification changes, and re-perform the entire proof. In reality, there are schedules, deadlines, and budgets, and one must make engineering compromises.

The required specification tweaks often depend on the precise design route chosen. If a single abstract specification is being targeted at several platforms, with diverse design decisions, there may be no single modification that satisfies all implementations.

Specifications are an important means of *communication* between various parties in a development. A refinement structure may not organise the system's requirements in a way that makes sense to domain experts; refinement is essentially a process of conservative extension, and if this structure is at odds with domain experts' approach, then it will fail to communicate.

The suggested change might be unimplementable, given the target platform constraints. For example, the Mondex target platform was so computationally limited, both in space and in speed (by today's standards, at least) that cryptographic functions in particular had to be kept to the bare minimum. Suggestions for improvements requiring extra encryptions were infeasible.

5. Subsequent refinement research

5.1. Backwards refinement rules and i/o refinement rules

In addition to the Mondex specification and proof, the complementary proof rules have also been published [WoD96, SCW98]. This led to a boom in academic research in this area. For example, [DeB01] have greatly extended some of these refinement ideas, and have incorporated them into Object Z; [Dun03] has introduced backwards refinement rules into Banach and his group [BJP+07] have extended retrenchment to include backwards rules and i/o retrenchments.

5.2. Retrenchment opportunities

The classical theory of program derivation used to develop Mondex is based on *refinement* [HHS86, HHH+87]. An abstract specification is transformed, on incorporation of design decisions, into a more concrete specification, and ultimately into an implementation. If the refinement relation holds, then the behaviours of the implementation will be correct with respect to the abstract specification. Depending on the nature of the refinement relation, this might include functional behaviour, termination, and liveness properties.

It is well known that refinement alone is too weak to preserve many non-functional properties of interest [Jac92], security properties being the most commonly quoted example. Additional proof obligations must be discharged to demonstrate that these properties are also preserved. (In the case of Mondex, the security properties of interest were conveniently all functional properties, and so refinement sufficed.)

A less well-advertised limitation of refinement is that it can be too strong a requirement on real-world developments: it can unacceptably limit the degree of abstraction possible. The most commonly quoted example is the inability to refine real numbers to floating or fixed-point numbers: a fundamental problem for the wide range of applications that deal with continuous real-world quantities. Mondex exhibits other, discrete, examples. There are two common engineering solutions to this kind of problem:

1. Require *fully formal refinement*, and hence necessitate the pollution of the abstract model with concrete design decisions (such as introducing fixed-point numbers or MAXINT in the abstract model).
2. Allow *informal steps*, arguing *outside* the formal framework, that the transformation, no longer a formal refinement, is allowable (such as using numerical analysis to argue about the change from real numberss to floating or fixed-point numbers).

Clearly, neither of these solutions is satisfactory in a formal software engineering context, and researchers are following a variety of paths tackling the problem of forms of inexact refinement, e.g. [BoD05].

Retrenchment [BP98] is one approach advocated to bridge this gap, allowing "refinement with concessions". It builds on the solid foundation of refinement, but also allows the kind of development compromises that are an essential part of commercial industrial developments. Although the Mondex refinement was successful, there were several places where subsequent development steps would necessarily be informal, because the abstraction used could not be formally refined to the implementation, or where the specification was distorted into an unnatural shape by the requirement for refinement. Banach and co-workers [BPJ+05a] have used the published Mondex specification as a basis for investigating these as retrenchment opportunities, and further develop retrenchment ideas in the context of a real world application.

1. Each concrete purse has a piece of state representing its sequence number, modelled as a natural number, but implemented as an n-byte number. Informal arguments were used to discount the problem of arithmetic overflow in the operational case, summarised as, *"we need not worry about overflow, since the purse will physically wear out first"* [SCW00, p. 24]. This can be handled by a retrenchment on the single case of the chosen implementation value of $MAXINT$ [BPJ+05b].
2. Each purse has a piece of state representing its exception log, modelled as an unbounded set, but implemented as a *small* bounded set. This requires a different handling of the exceptional cases, which can potentially occur in the operational case [BJP+06a].
3. A cryptographic hash function for signing certain protocol messages is defined to be a total injection, as that property is required in the proof. The implementation is neither total, not injective, but can be argued to be "sufficiently injective". This can be handled by a retrenchment to restrict the domain of the injection to reachable states only, composed with a retrenchment to allow "sufficiently few" collisions on this set [BJP+06b].
4. The abstract and concrete balance enquiry operations do not agree in one particular state during a transfer, expected to happen only rarely in practice. This is handled formally by a modelling trick: observing the balance through finalisation, not through the *BalanceEnquiry* operation output (present in the original specification, but omitted in the published version for clarity). It can be handled more naturally by an output retrenchment [BJP+07].
5. The entire balance transfer has one atomic abstract operation being refined into three concrete protocol steps by a further modelling trick. This can be handled more naturally by a generalised retrenchment [BJP+07].

5.3. Development team proof approaches

The original work was done in the mid-1990s, and proof tool support in those times was meagre: the tools were not mature, and the machines on which they ran were relatively feeble. So the proof was done by hand, with all steps merely typechecked. Doing the proof by hand greatly deepened our understanding of why the protocols were correct, and made it relatively easy to discover the cause of the small bug that we did eventually detect. The development was incremental: a first "pilot" phase, and a second "full functionality" phase. This second pass allowed us to uncover several specification and proof refactoring patterns [SFT03], to expose interesting structure, and to capture similar proof stages as lemmas, leading to a better structured presentation. However, much of the reworking was merely tedious, and tools would have been appreciated. We did some subsequent experiments

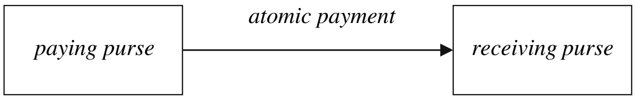

Fig. 1. An atomic transaction in the abstract model

with proof tools, to replicate the hand proofs [Ste98]: essentially, at that time, the tools were inadequate. Times change.

6. An introduction to Mondex

The case study presented in [SCW00] is a reduced version of the real development by the NatWest Development Team (now *platform seven*) of the Mondex smartcard product for electronic commerce. The system consists of a number of *electronic purses* that carry financial value, each hosted on a smartcard. The purses interact with each other via a communications device to exchange value. Once released into the field, each purse is on its own: it has to ensure the security of all its transactions without recourse to a central controller. All security measures have to be implemented on the card, with no real-time external audit logging or monitoring.

6.1. Models

The case study consists of two key models. The first is an *abstract* model, describing the world of purses and the exchange of value through atomic transactions, expressing the security properties that the cards must preserve. The second is a *concrete* model, reflecting the design of the purses that exchange value using a message protocol. Both models are described in the Z notation [Spi92a, WoD96, BSC94], and we prove that the concrete model is a *refinement* of the abstract.

6.2. Abstract model

The abstract model is small, simple, and easy to understand. The key operation transfers a chosen amount of value from one purse to another; the operation is modelled as an atomic action that simultaneously decrements the value in the paying purse and increments the value in the receiving purse (Fig. 1). Operations maintain two key system security properties:

- No value may be created in the system.
- All value is accounted in the system (no value is lost).

The simplicity of the abstract model allows these properties to be expressed in a way that is easily understood by the client.

6.3. Concrete model

The concrete model is rather more complicated, reflecting the details of the real system design. It was a requirement that the concrete level should reflect the real implementation, in particular the protocols (value transfer, and logging), but also things like sequence numbers, etc., in order that the mapping to the subsequent informal development could be carried out and evaluated. The key changes from the abstract are:

- Transactions are no longer atomic, but instead follow an *n*-step protocol (Fig. 2).
- The communications medium is insecure and unreliable.
- Transaction logging is added to handle lost messages.
- There are no global properties—each purse has to be implemented in isolation.

The basic protocol is:

1. The communications device ascertains the transaction to perform.

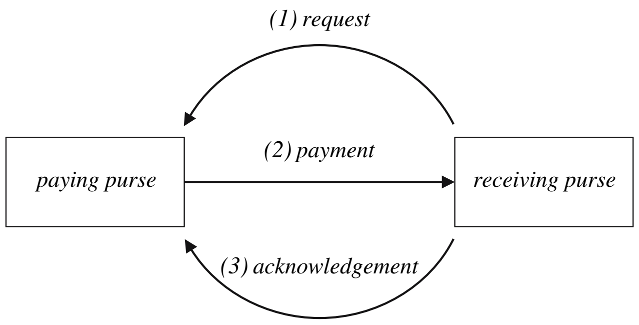

Fig. 2. Part of the n-step protocol used to implement the atomic transaction in the concrete model

2. The receiving purse requests the transfer of an amount from the paying purse.
3. The paying purse sends that amount to the receiving purse.
4. The receiving purse sends an acknowledgement of receipt to the paying purse.

The protocol, although simple in principle, is complicated by several factors: the protocol can be stopped at any point by removing the power from a card; the communications medium could lose a message; and a wire tapper could record a message and play it back to the same or a different card later. In the face of all these possible actions, the protocol must implement the atomic transfer of value correctly, as specified in the abstract model.

6.4. Proofs

All the security properties of the abstract model are *functional*, and so are preserved by refinement. The purpose of performing the proof is to give a very high assurance that the chosen design (the protocol) does indeed behave just like the abstract, atomic transfers. We chose to do rigorous proofs by hand: our experience in 1997 was that current proof tools were not yet appropriate for a task of this size. We did, however, type-check the statements of the proof obligations and many of the proof steps using a combination of *f*UZZ [Spi92b] and Formaliser [FHD90]. As part of the development process, external evaluators also independently checked all proofs.

7. Overview of Mondex model and proof structure

The specification and security proof have the following structure (summarised in Fig. 3):

- Security Properties:
  - The *Security Properties* are defined in terms of constraints on secure operations; they are *formalised* in terms of the appropriate model concepts (see later).
  - In some cases, where it may not be evident that a model captures a particular constraint, the desired property is recast as a *theorem* and proved.
- Abstract model, $\mathcal{A}$: We define an *abstract model*, which forms the *Formal Security Policy Model*; it consists of a global model in terms of a simple *state and operations*:
  - The state is a world of (abstract) purses.
  - The operations are defined on this state.
- Between model, $\mathcal{B}$: Next we build a *between-levels model*. This is the first refinement towards the implementation of purses consisting of local state information only. This model, $\mathcal{B}$, is structured as a promoted state-and-operations model:
  - The state of a single (concrete) purse, and the corresponding single-purse operations, are defined.
  - The purses and operations are *promoted* to a global state and operations. Constraints are put on this promotion to enable the correctness proofs to be performed.

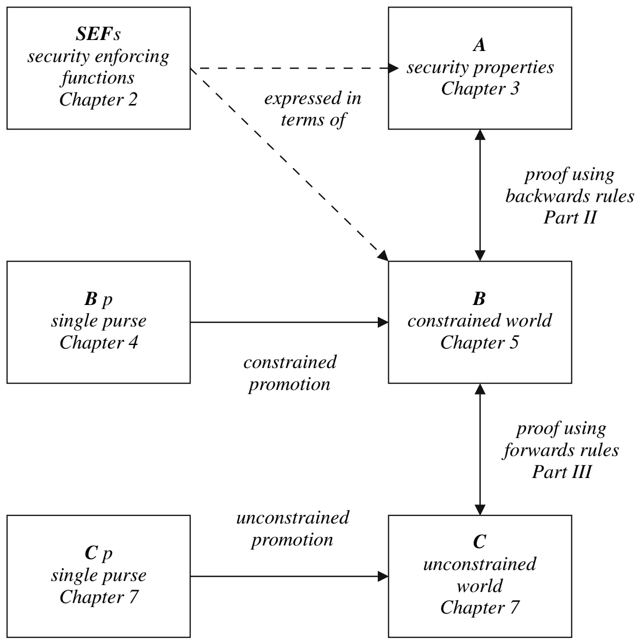

Fig. 3. Overview of document organisation, with model and proof structure

- Concrete model, $\mathcal{C}$: Our final model is the *concrete-level model*, which forms the *Formal Architectural Design*. This model, $\mathcal{C}$, is structured as a promoted state-and-operations model, very similar to $\mathcal{B}$, except it has no constraints on the promotion:
  - The state of a single (concrete) purse, and the corresponding single-purse operations, are defined.
  - The purses and operations are *promoted* to a global state and operations, with *no* constraints.
- Security proof $\mathcal{A}$–$\mathcal{B}$: The *security policy* is proved to hold for $\mathcal{B}$ by proving that $\mathcal{B}$ is a *refinement* of $\mathcal{A}$. This forms the first part of *Explanation of Consistency*.
  - The retrieve relation *Rab*, relating the $\mathcal{B}$ and $\mathcal{A}$ worlds, is defined.
  - The *security policy* is shown to hold for $\mathcal{B}$ by proof that $\mathcal{B}$ refines $\mathcal{A}$, using the backwards proof rules. This proof comprises the bulk of the proof work.
- Security proof $\mathcal{B}$–$\mathcal{C}$: The *security policy* is proved to hold for $\mathcal{C}$ by proving that $\mathcal{C}$ is a *refinement* of $\mathcal{B}$ (and hence of $\mathcal{A}$, by transitivity of refinement). This forms the remaining part of *Explanation of Consistency*.
  - The retrieve relation *Rbc*, relating the $\mathcal{C}$ and $\mathcal{B}$ worlds, is defined.
  - The *security policy* is shown to hold for $\mathcal{C}$ by proof that $\mathcal{C}$ refines $\mathcal{B}$, using the forwards proof rules. These two levels are relatively close, so this proof is relatively straightforward.

7.1. Rationale for model structure

As noted above, the case study [SCW00] has been adapted from a larger, real development. In order to produce a case study of a size appropriate for public presentation, much of the real functionality has had to be removed. Some of the structure of the larger specification has remained present in the smaller one, although it might not have been used had the smaller specification been written from scratch. This omitted functionality, whilst important from a business perspective, is peripheral to the central security requirements.

7.2. Rationale for proof structure

Imagine two specifications $\mathcal{A}$ and $\mathcal{C}$ that describe executable machines. Imagine that, on every step, each machine consumes an input and produces an output. Finally, imagine that every execution of $\mathcal{C}$, viewed solely in terms of inputs and outputs, could equally well have been an execution of $\mathcal{A}$. In this sense, $\mathcal{A}$ can *simulate* any behaviour of $\mathcal{C}$. If this is the case, then we say that $\mathcal{C}$ is a *refinement* of $\mathcal{A}$. This is exactly what we want to prove in our case study: that the concrete model is a refinement of the abstract one.

Refinement is a formal technique for ensuring that essential properties are present in a more concrete specification. It gives us an ordering between specifications that captures an intuitive notion of when a concrete specification implements an abstract one. This allows us to postpone implementation detail in writing our top-level specification, focusing only on essential properties. The cost of this abstraction is the need to refine the specification, reifying data structures and algorithms.

Nondeterminism is used in an abstract specification to describe alternative acceptable behaviours; in choosing a concrete refinement of an abstract specification, some of these nondeterministic choices may be resolved. Since we view $\mathcal{A}$ and $\mathcal{C}$ only terms of inputs and outputs, nondeterminism present in $\mathcal{A}$ may be resolved at a different point in $\mathcal{C}$.

Our abstract model, chosen to represent the difference between secure and insecure transactions very clearly, has nondeterminism in a different place from the implementation. In fact, it has it in a place that precludes proof using the forwards rules of [Spi92a, Sect. 5.6]. For this reason we use the backwards rules to prove against the abstract model.

At the concrete level, we must describe the purse behaviour in a way that closely mirrors the actual design. An important (and obvious) property of the design is that the purses are *independent*; that is, each purse acts on the basis of its own, local knowledge, and we have no control over the communications medium between purses. This can be expressed cleanly in Z by building a model of an individual purse in isolation, and then *promoting* [WoD96, BSC94, Chap. 19] this model to a world with many purses. To express the fact that we have no global control over the purses or over the communications medium, we must use an unconstrained promotion. This we do in the $\mathcal{C}$ model.

Why do we not, then, do a single backwards proof step from the $\mathcal{A}$ model to the $\mathcal{C}$ model? For technical reasons, the backwards proof rules need the more concrete specification to be tightly constrained in its state space. The form of the proofs forces the description of the state space to include explicit predicates excluding all but valid states. However, these predicates are not expressible locally to purses, and hence cannot be included in specification derived by unconstrained promotion. That is, we cannot express the predicates needed for the proof in the $\mathcal{C}$ model. We therefore introduce an intermediate model, the $\mathcal{B}$ model, which is a *constrained* promotion, and hence can contain the predicates needed for the backwards proofs. We then prove a refinement from $\mathcal{A}$ to $\mathcal{B}$ using the backwards rules. But now the constrained promotion $\mathcal{B}$ is very close to the unconstrained promotion $\mathcal{C}$, and in particular the nondeterminism is resolved in the same place in both models, allowing the forwards rules to be used. This we do in our proof of refinement from $\mathcal{B}$ to $\mathcal{C}$.

The next two sections summarise the definition of the Security Properties. They are formalised in terms of the abstract and concrete models. The full meaning and effect of a security property can be seen only in the context of the model that includes it.

8. Abstract model security properties

The following security properties are expressed in terms of the abstract model $\mathcal{A}$.

Security property 1: No value creation. No value may be created in the system: the sum of all the purses' balances does not increase. This property requires that the sum of the before balances is greater or equal to the sum of the after balances. The abstract model enforces a stronger condition: that transfers change only the purses involved in the transfer and only by the amount stated in the transfer.

Security property 2.1: All value accounted. All value must be accounted for in the system: the sum of all purses' balances and lost components does not change. The concrete level security property 2.2 uses logging to support this security property.

Security property 3: Authentic purses. A transfer can occur only between authentic purses.

Security poperty 4: Sufficient funds. A transfer can occur only if there are sufficient funds in the *from*-purse. The model also ensures that the purse balance and transfer value are non-negative.

9. Concrete model security properties

The following security property is expressed in terms of the between (and concrete) model $\mathcal{B}$.

Secrity property 2.2: Exception logging. If a purse aborts a transfer at a point where value could be lost, then the purse logs the details. The only times the log need be updated are if the purse is in *epv* (having sent the *req* message) or in *epa* (having sent the *val* but not yet received the *ack*). In all other cases the transfer has not yet got far enough for the purse to be worried that the transfer has failed, or has got far enough that the purse is happy that the transfer has succeeded.

10. Abstract model: security policy

The abstract model specification has the following parts:

- State: the abstract world of purses.
- Operations: secure changes from one abstract state to another.
- Initialisation: the abstract world starts off secure.
- Finalisation: a way of observing part of the abstract world to determine that it is secure.

10.1. The abstract state

10.1.1. A purse

An abstract *AbPurse* consists of a *balance*, the value stored in the purse; and a *lost* component, the total value lost during unsuccessful transfers. The unsuccessful, but still secure, transfer is defined elsewhere.

$$AbPurse \ \widehat{=}\ [\, balance, lost : \mathbb{N} \,]$$

10.1.2. Transfer details

Each purse has a distinct, unique name.

$$[NAME]$$

The details of a particular transfer include the names of the *from* and *to* purses and the value to be transferred.

$$TransferDetails \ \widehat{=}\ [\, from, to : NAME;\ value : \mathbb{N} \,]$$

Although it is not permitted to perform a transfer between a purse and itself, the constraint $from \neq to$ is checked during *AbTransfer*, rather than put in *TransferDetails*, since it is permitted to *request* a transfer with $from = to$. Transactions involving zero value are allowed.

10.1.3. Abstract world

The abstract world model contains a mapping from purse names to abstract purses. The domain of this function corresponds to authentic purses, those that may engage in transfers. We allow only a finite number of authentic purses, to ensure a well-defined total value in the system.

$$AbWorld \ \widehat{=}\ [\, abAuthPurse : NAME \twoheadrightarrow\!\!\!\!\!\!\shortmid\ \ AbPurse \,]$$

10.2. Secure operations

Having defined our abstract world, *AbWorld*, we now define operations on the world that respect the relevant security properties. We call these *secure operations*. They comprise:

- *AbIgnore*: securely do nothing.
- *AbTransfer*: securely transfer balance between purses, or securely ‘lose’ the balance.

10.2.1. Abstract inputs and outputs

We are to prove that the implementation is a refinement of the abstract security policy specification. This is made simpler if every operation has an input and an output, and if all operations' inputs and outputs are of the same type.

So we define the inputs and outputs (some being 'dummy' values) using a free type construct:

$$\begin{array}{ll} AIN & ::= aNullIn \mid transfer\langle\langle TransferDetails\rangle\rangle \\ AOUT & ::= aNullOut \end{array}$$

Every abstract operation has the following properties:

$$AbOp \mathrel{\widehat{=}} [\, \Delta AbWorld;\ a? : AIN;\ a! : AOUT \mid a! = aNullOut \,]$$

The output is always *aNullOut* (that is, we are not interested in the abstract output).

10.2.2. Abstract ignore

Any operation has the option of securely doing nothing.

$$AbIgnore \mathrel{\widehat{=}} [\, AbOp \mid abAuthPurse' = abAuthPurse \,]$$

10.2.3. Transfer

The transfer operation changes only the balance and lost component of the relevant purses.

$$AbPurseTransfer \mathrel{\widehat{=}} AbPurse \setminus (balance, lost)$$

The secure transfer operations change at most the *from* and *to* purse states: all other purse states are unchanged.

```
AbWorldSecureOp
  AbOp
  TransferDetails?
  ─────────
  $a? \in \operatorname{ran} transfer$
  $\theta TransferDetails? = transfer^{\sim} a?$
  $\{from?, to?\} \ntriangleleft abAuthPurse' = \{from?, to?\} \ntriangleleft abAuthPurse$
```

A transfer can securely succeed between two purses if they are distinct, both purses are authentic, and the *from* purse has sufficient funds.

```
AbTransferOkayTD
  AbWorldSecureOp
  ─────────
  $Authentic[from?/name?]$
  $Authentic[to?/name?]$
  $SufficientFundsProperty$
  $to? \neq from?$
  $abAuthPurse'\ from? =$
      $(\mu\ \Delta AbPurse \mid \theta AbPurse = abAuthPurse\ from? \land balance' = balance - value? \land$
                          $lost' = lost \land \Xi AbPurseTransfer \bullet \theta AbPurse')$
  $abAuthPurse'\ to? =$
      $(\mu\ \Delta AbPurse \mid \theta AbPurse = abAuthPurse\ to? \land balance' = balance + value? \land$
                          $lost' = lost \land \Xi AbPurseTransfer \bullet \theta AbPurse')$
```

The operation transfers *value*? from the *from* purse to the *to* purse. All the other components of the *from*? and *to*?-purses are unchanged, and all other purses are unchanged. The model is more constrained than required by the security properties, and hence it represents a sufficient, but not necessary, behaviour to conform to the security properties. Hiding the auxiliary inputs gives the *Okay* operation as:

$$AbTransferOkay \mathrel{\widehat{=}} AbTransferOkayTD \setminus (to?, from?, value?)$$

A transfer can securely lose value between two purses if they are distinct, both purses are authentic, and the *from* purse has sufficient funds.

$$\begin{array}{|l}
\hline
\textit{AbTransferLostTD} \\
\textit{AbWorldSecureOp} \\
\hline
\textit{Authentic}[\textit{from}?/\textit{name}?] \\
\textit{Authentic}[\textit{to}?/\textit{name}?] \\
\textit{SufficientFundsProperty} \\
\textit{to}? \neq \textit{from}? \\
\textit{abAuthPurse}'\,\textit{from}? \in \\
\quad \{\, \Delta \textit{AbPurse} \mid \theta \textit{AbPurse} = \textit{abAuthPurse}\,\textit{from}? \wedge \textit{balance}' = \textit{balance} - \textit{value}? \wedge \\
\qquad\qquad \textit{lost}' = \textit{lost} + \textit{value}? \wedge \Xi \textit{AbPurseTransfer} \bullet \theta \textit{AbPurse}' \,\} \\
\textit{abAuthPurse}'\,\textit{to}? = \textit{abAuthPurse}\,\textit{to}? \\
\hline
\end{array}$$

The operation removes *value*? from the *from* purse's balance, and adds it to the *from* purse's *lost* component. All the other components of the *from*?-purse are unchanged, The *to* purse and all other purses are unchanged. Hiding the auxiliary inputs gives the *Okay* operation as:

$$\textit{AbTransferLost} \;\widehat{=}\; \textit{AbTransferLostTD} \setminus (\textit{to}?, \textit{from}?, \textit{value}?)$$

The full transfer operation can also securely do nothing, *AbIgnore*. The full transfer operation is

$$\textit{AbTransfer} \;\widehat{=}\; \textit{AbTransferOkay} \vee \textit{AbTransferLost} \vee \textit{AbIgnore}$$

10.3. Abstract initial state

One conventional definition of the initial state of a system is as being empty; operations are used to add elements to the state until the desired configuration is reached. However, we do not wish to add new abstract purses to the domain of *abAuthPurse*, so we cannot start with a system containing no authentic purses. So we set up an arbitrary initial state, which satisfies the predicate of *AbWorld*′.

$$\textit{AbInitState} \;\widehat{=}\; \textit{AbWorld}'$$

So we say that *AbInitState* has some particular value, we just do not say what that particular value *is*. The particular value chosen is irrelevant to the security of the system; any starting state would be secure.

Initialisation also defines the mapping from global (that is, observable) inputs to abstract (that is, modelled) inputs. This is just the identity relation in the $\mathcal{A}$ model:

$$\textit{AbInitIn} \;\widehat{=}\; [\, a?, g? : \textit{AIN} \mid a? = g? \,]$$

10.4. Abstract finalisation

We must 'observe' each security relevant component of the world, in order to determine that the security properties do indeed hold. Observation is usually performed by enquiry operations, and any part of the state not visible through some enquiry operation is deemed unimportant. However, in our case there are no abstract enquiry operations to observe state components, but there are security properties related to them, and so they *are* important. We use *finalisation* to observe them. Finalisation takes an abstract state, and 'projects out' the portion of it we wish to observe, into a global state. Here we choose to observe the entire abstract state. The global state is the same as the abstract state:

$$\textit{GlobalWorld} \;\widehat{=}\; [\, \textit{gAuthPurse} : \textit{NAME} \mathrel{+\!\!\!\!\twoheadrightarrow} \textit{AbPurse} \,]$$

Finalisation gives the global state corresponding to an abstract state. These are mostly the identity relations in the $\mathcal{A}$ model:

$$\textit{AbFinState} \;\widehat{=}\; [\, \textit{AbWorld};\ \textit{GlobalWorld} \mid \textit{gAuthPurse} = \textit{abAuthPurse} \,]$$

Finalisation also defines the mapping from abstract outputs to global (that is, observable) outputs.

$$\textit{AbFinOut} \;\widehat{=}\; [\, a!, g! : \textit{AOUT} \mid a! = g! \,]$$

11. The Grand Challenge in Verified Software

The Grand Challenge in Verified Software is an ambitious, international, long-term programme of research for achieving a substantial and useful body of code that has been formally verified to the highest standards of rigour and accuracy. Tony Hoare initiated the Grand Challenge by calling on the computer science community to develop an integrated, automated toolset that can be used to establish the correctness of software. A workshop on Verified Software was held in Menlo Park in February 2005, and an IFIP Working Conference was held in Zurich in October 2005 on *Verified Software: Theories, Tools, and Experiments*.

The Grand Challenge project will spend fifteen years demonstrating the feasibility of using formal verification technology in industrial-scale software development. The programme has three objectives. (1) It will establish a unified theory of program construction and analysis. (2) It will build a comprehensive and integrated suite of tools that support verification activities, including specification, validation, test-case generation, program refinement, program analysis, program verification, and run-time checking. (3) It will collect together a repository of formal specifications and verified codes. These programs will continue to evolve as verified code, and a measure of success will be when they replace their unverified counterparts in actual use.

Verification should be interpreted in a wide sense. There are generic properties of interest apart from total correctness, such as absence of runtime errors, data consistency, timing behaviour, accuracy, type correctness, termination, translation validation, serialisability, memory leakage, information hiding, representation independence, and information flow. There are also non-functional properties such as dependability and aspects of safety and security. Our goal for the Verified Software challenge is to develop verification technology to a point where it demonstrably enhances the productivity and reliability with which software is designed, developed, integrated, and maintained. The impact of this work will be felt in a number of related areas of software development including software engineering, safety-critical systems, mathematical modelling, and artificial intelligence.

Technical work on the Grand Challenge was launched in January 2006 with a year-long pilot project: the Mondex case study. This short project is intended to demonstrate how different research groups around the world can collaborate and compete in scientific experiments, and to generate artifacts to populate the Repository. We chose as our problem the verification of the Mondex smart card, a product that is now ten years old, to see what the current state of the art is in proof mechanisation. Eight groups took up the challenge (Alloy, ASM, Event-B, OCL, PerfectDeveloper, π-calculus, Raise, Z), and some of their results are published in this special issue of *Formal Aspects of Computing*. Other groups are continuing to work on the project, and we look forward to seeing Mondex become a familiar and useful benchmark.

In the past, many people, often quite justifiably, have said that we cannot verify industrial software on a cost-effective basis. As Tony Hoare has remarked, at the end of the grand Challenge we will be able to tell our sceptics, "You cannot say any more that it cannot be done. Here, we have done it!"

12. Conclusions

The original Mondex work has had valuable consequences. The specification and proof have directly provided a substantial case study for academia to try out a variety of comparative approaches. Moreover, the new refinement laws have revived interest in Z refinement research, in particular demonstrating the use of backwards refinement, and of i/o refinement. Much of this would never have happened if we had not been forced to struggle with the real problem we had been given, but rather had been allowed to "simplify" it in ways subsequently suggested. The difference between commercial developments and academic case studies is that with the former you cannot change the problem to make your life easier. The moral is that not making your life easier may lead to more interesting results. These developments demonstrate a fruitful synergy between academia and industry: each doing what they do best, whilst providing interesting problems and valuable results to the other.

Acknowledgements

We thank our colleagues at NatWest and *platform seven* for the chance ten years ago to participate in such a wonderful project. We thank the anonymous reviewers for their thorough and insightful comments. Finally, we thank all those who have encouraged us to revive Mondex as a pilot project for the Grand Challenge, and those who have already tackled the problem.

References

[BJP+06a]	Banach R, Jeske C, Poppleton M, Stepney S (2006) Retrenching the purse: finite exception logs, and validating the small. 30th Annual IEEE/NASA Software engineering workshop. Columbia, April 2006
[BJP+06b]	Banach R, Jeske C, Poppleton M, Stepney S (2006) Retrenching the purse: hashing injective CLEAR codes, and security properties. In: 2nd International symposium on leveraging applications of formal methods, verification and validation (ISoLA 2006). Cyprus, November 2006. IEEE, 2006
[BJP+07]	Banach R, Jeske C, Poppleton M, Stepney S (2007) Retrenching the purse: the balance enquiry quandary, and generalised and (1, 1) forward refinements. Fundam Inform 77:1–41
[BP98]	Banach R, Poppleton M (1998) Retrenchment: an engineering variation on refinement. B-98. Lecture notes in computer science, vol 1393. Springer, Heidelberg
[BPJ+05a]	Banach R, Poppleton M, Jeske C, Stepney S (2005) Retrenchment and the Mondex electronic purse (extended abstract). In: Proceedings 12th international workshop on abstract state machines (ASM'05). Paris, March 2005
[BPJ+05b]	Banach R, Poppleton M, Jeske C, Stepney S (2005) Retrenching the purse: finite sequence numbers and the tower pattern. In: Proceedings FM05. Lecture notes in computer science, vol 3582. Springer, Heidelberg, pp 382–398
[BSC94]	Barden R, Stepney S, Cooper D (1994) Z in Practice. BCS Practitioners Series. Prentice Hall, Englewood Cliffs
[BoD05]	Boiten EA, Derrick J (2005) Formal program development with approximations. ZB 2005. Lecture notes in computer science, vol 3455. Springer, Heidelberg, pp 374–392
[Bur02]	Burton S (2002) Automated testing of high integrity test suites from graphical specifications. Ph.D. thesis. Department of Computer science, University of York
[CSC05]	Clark JA, Stepney S, Chivers H (2005) Breaking the model: finalisation and a taxonomy of security attacks. REFINE 2005, Surrey. Electron Notes Theor Comput Sci 137(2):225–242
[CSW02]	Cooper D, Stepney S, Woodcock J (2002) derivation of Z refinement proof rules: forwards and backwards rules incorporating input/output refinement. Technical report YCS-2002-347, December, University of York
[CoCURL]	http://www.niap-ccevs.org/cc-scheme/cc_docs/
[DeB01]	Derrick J, Boiten E (2001) Refinement in Z and Object-Z. Springer, Heidelberg
[Dun03]	Dunne S (2003) Introducing backwards refinement into B. In: ZB2003: third international conference of B and Z Users, Turku, June 2003. Lecture notes in computer science, vol 2651, Springer, Heidlberg, pp 178–196
[FHD90]	Flynn M, Hoverd T, Brazier D (1990) Formaliser—an interactive support tool for Z. Z UserWorkshop. In: Proceedings of the 4th annual Z user meeting, workshops in computing, Springer, Hiedelberg, pp 128–141
[GCH97]	E6: Use of formality discussion. G3A Tape No 68. Unclassified. Government Communications Headquarters (GCHQ). 22 October 1997
[HHH+87]	Hoare CAR, Hayes IJ, He J, Morgan C, Roscoe AW, Sanders JW et al (1987) The laws of programming. Commun ACM. 30
[HHS86]	Jifeng H, Hoare CAR, Sanders JW (1986) Data refinement refined: resume. ESOP 86. Lecture notes in computer science, vol 213. Springer, Heidelberg, pp 187–196
[ITS91]	Information Technology Security Evaluation Criteria (ITSEC): Preliminary Harmonised Criteria. Document COM(90) 314, Version 1.2. Commission of the European Communities. June 1991
[Jac92]	Jacob JL (1992) Basic theorems about security. J Comput Secur 1(4):385–411
[SCP+03]	Srivratanakul J, Clark J, Polack F, Stepney S (2003) Challenging formal specifications with mutation: a CSP security example. 12th IEEE Asia Pacific Software Engineering Conference (APSEC)
[SCW00]	Stepney S, Cooper D, Woodcock J (2000) An electronic purse: specification, refinement, and proof. Technical monograph PRG-126, Oxford University Computing Laboratory, July 2000
[SCW98]	Stepney S, Cooper D, Woodcock J (1998) More powerful Z data refinement: pushing the state of the art in industrial refinement. In: ZUM '98: 11th international conference of Z users, Berlin, September 1998. Lecture notes in computer science, vol 1493. Springer, Heidelberg, pp 284–307
[SFT03]	Stepney S, Polack F, Toyn I (2003) Patterns to guide practical refactoring: examples targetting promotion in Z. In: ZB2003: third international conference of B and Z Users, Turku, June 2003. Lecture notes in computer science, vol 2651. Springer, Heidelberg, pp 20–39
[Spi92a]	Spivey JM (1992) The Z Notation: a reference manual, 2nd edn. http://spivey.oriel.ox.ac.uk/∼mike/fuzz
[Spi92b]	Spivey JM (1992) The *f*UZZ Manual. Computer Science Consultancy, 2nd edn. Prentice Hall, Englewood Cliff
[Ste01]	Stepney S (2001) New horizons in formal methods. The Computer Bulletin, pp 24–26. BCS, January 2001
[Ste98]	Stepney S (1998) A tale of two proofs. BCS-FACS third Northern formal methods workshop, Ilkley, September 1998. Electronic Workshops in Computing
[WoD96]	Woodcock J, Davies J (1996) Using Z: specification, refinement, and proof. Prentice Hall, Englewood Cliff

Received 19 January 2007
Accepted in revised form 9 November 2007 by C. B. Jones
Published online 12 December 2007