Coarse Grained Retrenchment and the Mondex Denial of Service Attacks

Richard Banach
School of Computer Science
University of Manchester
Manchester, M13 9PL, U.K.
banach@cs.man.ac.uk

Abstract

Retrenchment is a framework that allows relatively unrestricted system evolution steps to be described in a way that gives an evolution step some formal content — unlike model based refinement, whence it emerged, which is inapplicable outside some fairly tightly drawn notion of 'progress towards implementation'. In this paper, we introduce a 'coarse grained' version of retrenchment, relating to system behaviours in the large, and exemplify it on the requirements issues surrounding a Denial of Service case study drawn from the Mondex Purse. We show that the coarse grained retrenchment framework gives a good account of this case study.

1. Introduction

Retrenchment [10, 2, 19], which dates back to its first introduction in the context of the B Method [6, 8, 7], aims to extend the kind of rigour achievable using model based refinement techniques [13, 15], to a wider range of scenarios than model based refinement permits. Many system development activities, perfectly justified in their own terms, do not fit the neat monolithic disciplines that typical model based refinement formulations demand. One characteristic element found in such situations, is the introduction of features or details into the process, before their abstractions have been developed at higher levels — refinement methodologies do not normally allow the more concrete to be present before the more abstract. Another characteristic element, is the adaptation (which in logical terms amounts to contradiction) of previously present properties — refinement forbids this, apart perhaps from the mild case of making underspecification more precise during refinement (which can be expressed logically as implication rather than contradiction). Both aspects arise during realistic developments, which are after all, human driven activities (with all that that implies), rather than closed logical theories.

Retrenchment aims to redress this imbalance between idealised developments and real ones. It does this by giving up some of the 'completeness' aspects (to coin a phrase) found in typical refinement formulations, by introducing the capacity for greater laxity in what remains, and by focusing on the 'proof obligations' (POs) or 'verification conditions' (VCs) that formal developments generate.

Notable among the success stories of retrenchment has been the Mondex Purse [21]. A good range of Mondex requirements issues that were treated less than ideally (though cost effectively as regards the techniques of the day), have been well served by being revisited using retrenchment, eg.: the boundedness of purse sequence numbers [9]; the boundedness of purse exception logs [3]; the use of hash functions to validate log clearance [4]; the behaviour of balance enquiries during Mondex protocol runs [5].

This paper continues the Mondex work, but in a direction which is novel in two respects. Firstly, it concerns a new requirements aspect of Mondex, namely its vulnerability to certain kinds of Denial of Service (DOS) attack.[1] Secondly, it requires a new development of the retrenchment framework itself, namely its enlargement to deal with extended behaviours rather than just with effects that concern only individual system events.

The rest of this paper is as follows. In Section 2 we give an outline of the Mondex Purse. In Section 3 we describe the DOS attacks and the fix that mends them (these were first described in the literature in [20], which included a mechanical verification using the KIV theorem prover [1]). In Section 4 we outline the retrenchment tools we use to describe the evolution between the DOS-vulnerable and DOS-immune models. In Section 5 we develop the retrenchment case study itself. Section 6 concludes.

[1] We should say immediately that the attacks that we will discuss below were already envisaged at the time of the original Mondex development, as confirmed by Susan Stepney, author of [21]. However, the decision was taken at the time that these vulnerabilities should be tolerated, given their precise nature (as we discuss below), and most significantly, given the severe technical limitations of the smartcards of the day (i.e. the mid 90s).

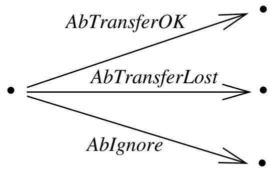

Figure 1. The Mondex atomic protocol.

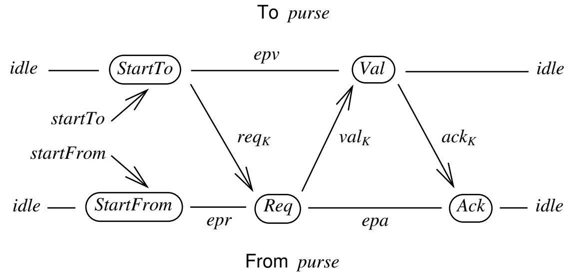

Figure 2. The Mondex concrete protocol.

2. The Mondex Purse

In this section we give an outline of the operation of the Mondex Purse [21]. There are by now several accounts of the Mondex Purse in the literature. See [17] for a good account of the various mechanical verifications that have corroborated the original hand-performed proofs. Our account will differ a little from them since we emphasise those aspects salient to the DOS attacks of interest.

Mondex was an industrial scale development of a smartcard electronic purse application. Mondex money had to be non-forgeable (to maintain users' and underwriters' confidence), but Mondex money was not *itself* concerned with whether the transactions it engaged in were honest or not.

Mondex was (almost) the first real product certified to ITSEC level E6 [14] (these days Common Criteria EAL 7 [16]). Key to this is a refinement from an abstract model to a concrete one. In Mondex, the abstract model described atomic operations for money transfer between purses, whereas the concrete model described a distributed algorithm that closely models the code for the money transfer protocol actually used.

At the abstract level, illustrated in Fig. 1, in which the arrows are transitions and the nodes are states, there are three options. Either the transfer succeeds (*AbTransferOK*), or it fails (*AbTransferLost*), or there is no change of state (*AbIgnore*, Mondex-speak for skip). The security invariants of Mondex are defined in terms of this abstract model. These say that: (i) the sum total of purse balances must never increase; (ii) all funds must be accounted for. While the meaning of (i) is obvious, (ii) means that even in the face of various kinds of protocol failure, the net result has to be such that *it is knowable from the system state* when money has been lost in transit (such knowledge making the situation recoverable in principle). While *AbTransferOK* and *AbIgnore* plausibly preserve (i) and (ii), *AbTransferLost* gives an abstraction (in terms of transfer of money to an overt 'lost' component of the state), that all critically failing protocol runs must provably adhere to.

At the concrete level, more things happen, illustrated in the activity diagram Fig. 2, in which the round nodes are events, the lines are states, and the arrows are messages traveling between the From (donor) purse and the To (recipient) purse. The interface to a purse accepts input messages and emits output messages, reacting to messages according to its internal state.

At the start of a transfer, the two purses receive their orders in the *startTo* and *startFrom* messages. The *startTo* and *startFrom* tags confirm the two purses' roles in the transaction, and the message bodies contain the (public) transaction details, (consisting of the transaction amount, and the transaction's unique identifier (which is a quadruple consisting of the two purses' unique ids and the two purses' internal sequence numbers)). On receipt of the *startTo* and *startFrom* messages, the two purses execute pre-emptive *Aborts*[2] to put them into the *idle* state. Beyond this point, the transaction proceeds in secret.

The two purses execute the *StartTo* and *StartFrom* events. These store the current transaction details and increment the respective internal sequence numbers ready for the *next* transaction (thus freezing the sequence numbers for *this* one). After this, *StartTo* puts the To purse into the *epv* state (expecting payment value) and causes it to send the encrypted req_K message to the From purse, and *StartFrom* puts the From purse into the *epr* state (expecting payment request). The req_K message is assumed unforgeable, and contains the transaction details.

Upon receipt of the req_K message, the From purse decrements its balance, and sends an encrypted val_K message (with the same information as the req_K) to the To purse, going into the *epa* state (expecting payment ack). Upon receipt of the val_K message, the To purse increments its balance, and sends an encrypted ack_K message (with the same information as the val_K) to the From purse, going back into the *idle* state. Upon receipt of the ack_K message, the From purse returns to the *idle* state.

The tag and contents of each message received are checked against the current purse state. If there is anything amiss, the purse can ignore the message or *Abort*.

[2] An *Abort* resets the purse state to *idle* unconditionally, and, if the purse state was either *epv* or *epa* previously, posts the details of the transaction being abandoned in the (small) purse exception log.

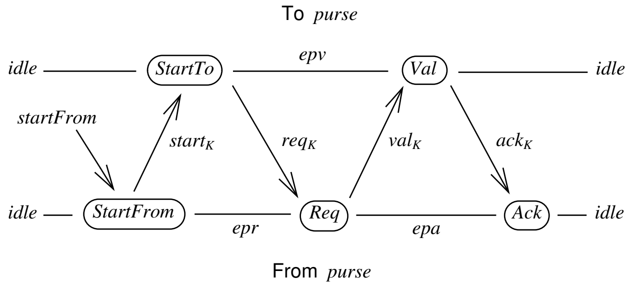

Figure 3. The modified Mondex protocol.

3. The Mondex Denial of Service Attacks

The purse protocol can be proved to preserve the security invariants mentioned earlier [21, 17]. Nevertheless, it admits DOS attacks that adroitly avoid violating them.

The DOS attacks depend on the fact that *all* the needed transaction details are public. Consider the following scenario. An attacker obtains, or plausibly invents, the purse id and sequence number of a genuine From purse, enabling him to impursonate it, becoming a FakeFrom purse. He can offer these details as part of transaction initiation to a victim To purse. The To purse finds the details credible, embarks on a transaction, and emits a req_K message. But the transaction has no genuine counterparty, so the expected reply will not come, and the To purse has no option but to eventually *Abort*. Since it is in the critical *epv* state, it records the transaction details in its log. After a few such episodes, the log, which we noted in footnote 2 is *small*, fills up. At this point the purse becomes disabled, since normal log availability (i.e. the ability to insert another entry) is crucial to the preservation of the security invariants. So, with a full log, new transactions cannot be started in case *they* fail.

We can go further. Suppose an attacker gets temporary access to a genuine From purse, obtaining its id and sequence number. He then uses this in an instance of the preceding scenario, and collects the req_K message that the To purse issues. He then impursonates the To purse, becoming a FakeTo purse. Assuming that the From purse has not in the meantime engaged in any transactions, its sequence number is as before. Therefore the FakeTo purse can initiate a transaction with a credible *startFrom* message, and once the From purse is in the *epr* state, can offer it the previously collected req_K message obtained from the genuine To purse. This is decrypted by the From purse, which finds nothing amiss and thus decrements its balance and sends the money in a val_K message. Of course the attacker can use this to give the money to the genuine To purse (provided it has not yet *Abort*ed[3]), but since he is an attacker, benefiting the To purse is not his aim.

The net result of this sequence of events is that the From purse has been deprived of the amount of the fake transaction. The process worked, because the attacker in effect played the role of the message transmission medium in a genuine transfer. As far as the Mondex security invariants go, nothing is amiss, for the following reasons. Both purses eventually have to *Abort*. Since the To purse is in *epv* and the From purse is in *epa*, both purses log the transaction when they *Abort*. The two log entries match, which is the key criterion that enables the fact that that the Mondex system lost the val_K message in transit —a situation that is recoverable by design— to be deduced.

The big difference between these scenarios and the loss of a val_K message during a genuine transaction, is that either or both purses' owners may be unaware of what has happened, and may thus eventually need to engage in a nonstandard recovery (i.e. one without the active co-operation of both purses) to restore the missing funds. By contrast, in a standard recovery, both purses readily co-operate in making their logs available to the bank, in order that it may confirm the matching entries characteristic of this eventuality.

So, as in most DOS scenarios, the situation depends on the relationship between the details of what has been formalised, and the wider requirements picture. The one fact that we can pinpoint in the formal structures that singles out the DOS scenarios, is that a genuine To purse may be persuaded to start a transaction without a corresponding start from a genuine From purse. This is a situation that is easy to remedy with a small change in the purse protocol.

Fig. 3 shows what is needed. Instead of both of the start events relying on completely public information, as above, we keep the sequence numbers secret throughout, allowing their use in a challenge-response exchange. Thus there is only one external start message, the *startFrom* message. This contains the purse ids and amount to be tranfered, but not the sequence numbers. Its processing by the From purse during the *StartFrom* event now causes the sending of a $start_K$ message, this time encrypted, to the To purse, containing purse ids, amount and From purse sequence number. Upon decryption by the To purse during the *StartTo* event, the To purse sequence number is added to the $start_K$ message contents to form the req_K message, for use in the ensuing *Req* event. When the req_K message is received by the From purse, it can check that the correct From purse sequence number has been included, confirming that a properly constituted transaction is under way.

4. Coarse Grained Retrenchment

Retrenchment aims to bring some of the rigour characteristic of formal refinement, to situations of system model

[3]Note that once the To purse has *Abort*ed from the *epv* state, the subsequent arrival of the missing val_K message will be ignored.

evolution that fall outside refinement's remit. It does this by noting that the key element shared by all refinement methodologies, is a simulation criterion, that via theoretical considerations specific to the refinement approach under consideration, yield VCs, which turn out to be very similar across approaches. Retrenchment manipulates the simulation idea to make it more flexible and expressive.

Suppose we have an abstract system Abs and a concrete one Conc, both of which are viewed as transition systems. Suppose we have an abstract operation $Op_A(u,i,o,u')$ (with before- and after- states u,u' and input/output i,o) and a concrete operation $Op_C(v,j,p,v')$ (with before- and after- states v,v' and input/output j,p). Suppose the abstract and concrete state spaces are connected via a retrieve relation $R(u,v)$, and the I/O spaces via relations $In(i,j)$ and $Out(o,p)$. Then, provided the ususal abstract-concrete initialisation holds, which demands that for every concrete initial state there is an abstract initial state such that the retrieve relation holds between them, we say that Conc refines Abs if the following forward simulation condition holds:

$$R(u,v) \wedge Op_C(v,j,p,v') \wedge In(i,j) \Rightarrow \\ (\exists\, u',o \bullet Op_A(u,i,o,u') \wedge Out(o,p) \wedge R(u',v')) \qquad (1)$$

Suitably fortified by additional conditions on inputs etc., (1) is typically used in an inductive process that makes every concrete run a realisation of some abstract one. This all amounts to a straightforward form of model based refinement.

In retrenchment, we focus on (1), seeking ways to make it more widely applicable. We recognise that: (i) it may not be appropriate to demand that *all* steps of a concrete operation have abstract counterparts; (ii) some corresponding pairs of steps may not be able to re-establish $R(u,v)$, necessitating an 'escape clause'; (iii) not all properties of interest regarding a corresponding pair of steps arise as pure 'state properties', necessitating the presence of relations featuring more variables. On this basis we generalise (1) to:

$$R(u,v) \wedge W_{Op}(i,j,u,v) \wedge Op_C(v,j,p,v') \Rightarrow (\exists\, u',o \bullet \\ Op_A(u,i,o,u') \wedge ((R(u',v') \wedge O_{Op}(o,p,u'v',u,v,i,o)) \\ \vee C_{Op}(u'v',o,p,u,v,i,o))) \qquad (2)$$

In (2), the previous $In(i,j)$ has become the operation(pair)-specific 'within relation' $W_{Op}(i,j,u,v)$, featuring all the 'before' information.[4] The previous $Out(o,p)$ has become the operation(pair)-specific 'output relation' $O_{Op}(o,p,u'v',u,v,i,o)$, featuring both the 'before' and 'after' information. And there is now also a 'concedes relation' $C_{Op}(u'v',o,p,u,v,i,o)$ in all the variables, occurring disjunctively, allowing the description of exceptional cases. The collection of W_{Op}, O_{Op}, C_{Op} relations, indexed by operation pairs, together with the retrieve relation R, is called the retrenchment data. This gives us conventional retrenchment.

[4] Note that the '$_{Op}$' in W_{Op} is shorthand for the pair (Op_A, Op_C).

Since (2) no longer re-establishes something in the conclusion that previously occurred in the hypotheses, it is less obviously useful for inductively based reasoning — generally we need to add greater levels of detail to get global results.

Thus far, we have tacitly assumed that individual operation calls correspond to individual steps of the transition relation. Operations are thus like the events that we used to describe the Mondex protocol. In *coarse grained* retrenchment we let an operation consist of several occurrences of events, and we widen the range of variables that we permit to occur in the various operation-specific elements of the retrenchment data to include what is needed for writing arbitrary temporal properties.

With this liberalisation, a number of issues that can be quietly ignored in the single step world need to be considered. Do the steps that constitute an individual operation call have to occur consecutively, or can they be interleaved with other system activities? Does the system naturally partition into the activities of a number of (more or less autonomous) agents, or is the perspective whole-system? And if the former, do the steps in an operation describe whole-system state changes, or are they focused on just the local states of the agents? These questions can have different answers according to what is convenient for a given application area. For us, the greatest convenience will come from an agent-centric view, describing global state changes via local, agent-centric state changes, with the assumption that non-local state does not change during a step, and allowing interleaving of other system actvities between the individual steps of an operation *provided they do not modify the state being used by the operation* (which may of course be easy to arrange in practice, or not).

5. The DOS Coarse Grained Retrenchment

The attacks that we described in Section 3 emerged during the formal verification in [20] (to the extent that desirable properties of a secure protocol turned out to be unprovable in the original Mondex protocol). In this paper our principal concern is not that aspect, but with *modelling* the relationship between DOS-vulnerable and DOS-immune versions of the protocol via retrenchment.

We describe the phenomena of interest using a stripped down version of the Mondex protocols. Since attacks come from the environment and purses can only manipulate their own state, it will be sufficient to restrict to a world of two purses only, the earlier From and To purses. This enables us to reduce transaction ids to just the From and To sequence numbers, the purse ids becoming implicit. The state variables have names of the form *XYvar* where: $X \in \{D,\overline{D}\}$ represents DOS-vulnerable and DOS-immune protocols respectively; $Y \in \{F,T\}$ represents the From and To purses respectively; and *var* is the actual variable name. The state variables we will need in our simplified system models are: *XYst* (purse state); *XYpd* (payment details) — which is a data structure consisting of $Fseq, Tseq, amt$ (From and To sequence numbers, and transaction amount); *XYbal* (purse balance); *XYnsn* (the sequence number of the next transaction); and *XYlog* (purse log).

The events have names of the form *XYEvent* where X and Y are as before. When discussing a single event, provided we retain the X and Y in the event name, we can suppress them in the variable names since each event is executed by a single purse, and purses can only manipulate their own state. We write events thus:

$$beforeState \text{-}(input, EventName, output)\text{->}\ afterState \quad \text{where } predicate \tag{3}$$

where the *afterState* variables are primed, and *predicate* contains any facts that cannot be conveniently squeezed into the signature. Variables not mentioned are assumed not to change. With these conventions the *Var* event in the D system model can be written as:

$$(epv, pd, bal)\text{-}(val_K, DTVal, ack_K)\text{->}\ (idle, pd, bal + pd.amt) \quad \text{where } val_K.pd = pd = ack_K.pd \tag{4}$$

This makes it explicit that we are discussing the *Val* event in the DOS-vulnerable model, which is executed by the To purse. It takes an input which is a val_K message, the '$_K$' subscript indicating that it is encrypted, and which contains a payment details data structure. Provided that the To purse's payment details agree with those in the val_K message, the purse increments its balance by the transaction amount and emits an ack_K message, which again contains the same payment details. The pd, nsn and *log* variables don't change. Neither do the state variables of the From purse. Since we only have the two purses, the source and destination of each message can be implicit, so events' inputs and outputs can omit message addressing information.

The above establishes our conventions. We now set out the ingredients of our investigation. We start with the DOS-vulnerable model, in which the most significant events for us are as follows.

$$\begin{aligned}&(idle, pd, bal, nsn)\text{-}((Fseq, Tseq, amt), DFStartFrom,)\text{->}\\ &\qquad (epr, pd', bal, nsn')\\ &\text{where } nsn = Fseq \wedge amt \leq bal \wedge nsn' > nsn \wedge\\ &\qquad pd' = (Fseq, Tseq, amt)\end{aligned} \tag{5}$$

$$\begin{aligned}&(idle, pd, bal, nsn)\text{-}((Fseq, Tseq, amt), DTStartTo, req_K)\text{->}\\ &\qquad (epv, pd', bal, nsn')\\ &\text{where } nsn = Tseq \wedge nsn' > nsn \wedge\\ &\qquad pd' = req_K.pd = (Fseq, Tseq, amt)\end{aligned} \tag{6}$$

$$\begin{aligned}&(st, pd, bal, nsn, log)\text{-}(, XYAbort,)\text{->}\ (st', pd, bal, nsn, log')\\ &\text{where } st' = (|\,log'\,| = \mathrm{MAX}\ ?\ blocked : idle) \wedge\\ &\qquad log' = log \cup (st \in \{epv, epa\}\ ?\ \{pd\} : \varnothing)\end{aligned} \tag{7}$$

In the above we left empty slots where I/O was absent, and a single definition suffices for the *Abort*s of both From and To purses, in both the D model and the $\overline{D}$ model below. Note that in these *Abort* events, we have introduced the new purse state *blocked*, which a purse enters when its log fills to the maximum during the execution of the *Abort*, preventing new transactions (which are guarded on purses being in the *idle* state). This goes beyond the modelling of [21], described above, but is obviously relevant to our analysis.

The above D model events are complemented by event:

$$(epr, pd, bal)\text{-}(req_K, DFReq, val_K)\text{->}\ (epr, pd, bal - pd.amt) \quad \text{where } val_K.pd = pd = req_K.pd \tag{8}$$

and event *DTVal* in (4) above, and by event *DFAck*, which just checks the incoming ack_K and returns to idle. Finally, we will need the *XYIncrease* event:

$$(st, pd, bal, nsn)\text{-}(, XYIncrease,)\text{->}\ (st, pd, bal, nsn') \quad \text{where } nsn' \geq nsn \tag{9}$$

which obviously just increases the local purse sequence number, and is the same for both agents and both models.

For the DOS-immune model, aside from what has already been covered, we have the following.

$$\begin{aligned}&(idle, pd, bal, nsn)\text{-}(amt, \overline{D}FStartFrom, start_K)\text{->}\\ &\qquad (epr, pd', bal, nsn')\\ &\text{where } amt \leq bal \wedge nsn' > nsn \wedge\\ &\qquad pd'.(Fseq, amt) = start_K.pd = (nsn, amt)\end{aligned} \tag{10}$$

$$\begin{aligned}&(idle, pd, bal, nsn)\text{-}(start_K, \overline{D}TStartTo, req_K)\text{->}\\ &\qquad (epv, pd', bal, nsn')\\ &\text{where } nsn' > nsn \wedge pd' = req_K.pd =\\ &\qquad (start_K.Fseq, nsn, start_K.amt)\end{aligned} \tag{11}$$

The *Req* event corresponding to the above is:

$$\begin{aligned}&(epr, pd, bal)\text{-}(req_K, \overline{D}FReq, val_K)\text{->}\ (epr, pd, bal - pd.amt)\\ &\text{where } req_K.pd.(Fseq, amt) = pd.(Fseq, amt) \wedge\\ &\qquad val_K.pd = pd' = req_K.pd\end{aligned} \tag{12}$$

and the *Val* and *Ack* events in the $\overline{D}$ model are like their D model counterparts.

This completes our description of the models. Regarding the gap between the two models, there are a number of ways that it can be bridged using retrenchment. We will describe the most useful of them, and comment on alternatives later.

We use coarse grained retrenchment, in which we speak about the properties of specific collections of several events of the underlying system model. In such circumstances there are often very many sequencings of the events of interest that are equivalent as far as what is intended goes, and it is useful to be able to treat them together, regarding them as different facets of the same thing. A convenient way of accomplishing this is to use event structures [23, 18, 22], as a notational device to subsume the different interleavings arising from the execution of independent events. Particularly convenient are *flow event structures* (FES) [12] that

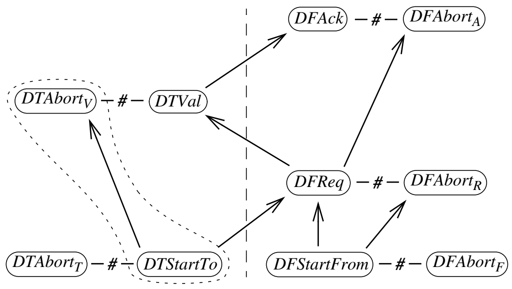

Figure 4. A FES for the original Mondex protocol, highlighting the events of interest.

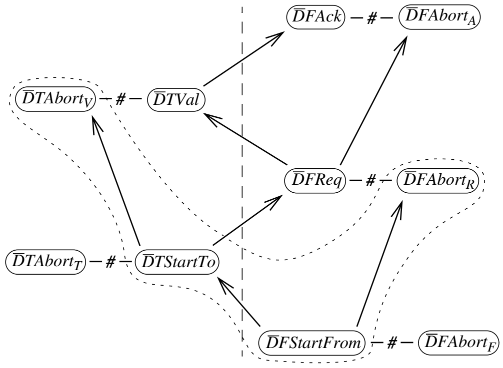

Figure 5. A FES for the modified Mondex protocol, highlighting the events of interest.

succinctly capture the fragments of behaviour of interest.

In a FES, there are events, connected by arrows and hashed links. An arrow from one event to another indicates causal dependence: the event at the head of the arrow cannot execute until the event at the tail has executed. A hashed link represents conflict: once the event at one end of the link has executed, the event at the other end remains blocked forever. Events may be executed as soon as all their needed prior causes have been executed, and provided no event with which they are in conflict has executed earlier. In this way, a single FES diagram can represent a large number of executions of sequences of events, depending on which choices are made of which event to execute next, at various points.

Fig. 4 shows a FES for the original Mondex protocol, with the events in rounded boxes. A line of dashes separates the To purse events and the From purse events. We have also separated the overall *Abort* event into its various cases, depending on which 'normal' event it replaces (i.e. is in conflict with). One can see how a successful run of the protocol starts with the two *Start* events in either order, and then proceeds sequentially.

Fig. 4 highlights the $DTStartTo$ and $DTAbort_V$ events (the latter being in conflict with the $DTVal$ event). These are the events we need for the DOS scenario in the DOS-vulnerable model.

Apparently, we gain nothing by using FESs here since these two events are causally dependent. However, below, we will see that not only are these two To purse events needed, but we also need to be able to match up the change in sequence number in the From purse in the other model. Accordingly, we invite an occurrence of $DFIncrease$ to the party. As is clear from (9), the $DFIncrease$ event is independent of $DTStartTo$ and $DTAbort_V$, so it can occur before, between, or after them. Now, this collection of events, the original protocol event plus the addition of $DFIncrease$, can make up the operation Op_D, which we can simply write as a set: $Op_D \equiv \{DFIncrease, DTStartTo, DTAbort_V\}$, since the FES implicitly gives the allowed orderings.[5]

Fig. 5 similarly shows a FES for the modified Mondex protocol, the events of interest being again highlighted. Here the FES is more obviously useful, since it neatly captures the various ways in which $\overline{D}FAbort_R$ can be scheduled with respect to $\overline{D}TStartTo$ and $\overline{D}TAbort_V$, thus encoding three separate interleavings. From these ingredients we create an operation in the DOS-immune model: $Op_{\overline{D}} \equiv \{\overline{D}FStartFrom, \overline{D}FAbort_R, \overline{D}TStartTo, \overline{D}TAbort_V\}$.

We set up a coarse grained retrenchment from Op_D to $Op_{\overline{D}}$. The issues to be confronted when creating a coarse grained retrenchment are: (i) what are the useful coarse grains to focus on — this has already been addressed in our choice of Op_D and $Op_{\overline{D}}$; (ii) what are the retrenchment data for the retrenched operations — to which we turn now.

The retrieve relation gives a good measure of the extent to which, when comparing behaviours of two different system models, we want them to stay comparable, despite the inevitable differences that will arise. Here is ours, R_{Op}:

$$R_{Op} \equiv DYvar_1 = \overline{D}Yvar_1 \wedge DYvar_2 = \overline{D}Yvar_2 \wedge \ldots \qquad (13)$$

This expresses equality between all D model variables and their $\overline{D}$ model counterparts.

The within relation says what further facts we need concerning before-data (particularly as regards inputs) in order that the scenario that we wish to describe using this retrenchment is properly set up to execute. Here is ours, W_{Op}:

$$W_{Op} \equiv i_D = (Fseq, Tseq, amt) \wedge j_{\overline{D}} = amt \qquad (14)$$

This expresses two facts: firstly that i_D, the input to Op_D (which is the input to $DTStartTo$), is $(Fseq, Tseq, amt)$, the

[5] When a collection of events makes up an operation, intermediate data values need to be handled suitably: after-values of one event must be the before-values of its successor; outputs of one event which are input by another are similar. External I/O must be interpreted as I/O of the whole operation. And so on.

complete transaction details; secondly that $j_{\overline{D}}$, the input to $Op_{\overline{D}}$ (which is the input to $\overline{D}TStartTo$), is *amt*, i.e. just the transaction amount, without the sequence numbers. So $j_{\overline{D}}$ is a projection of i_D.

The other two relations of the retrenchment data, say something about the state of affairs when the courses of events in the two models spoken of in the retrenchment have taken place.

The output relation gives a description of the state of affairs, after the facts, under 'normal running', i.e. when the retrieve relation has been re-established. Ours is O_{Op}:

$$\begin{aligned} O_{Op} \equiv\ & |DTlog'| = |DTlog| + 1 \wedge \\ & |\overline{D}Tlog'| = |\overline{D}Tlog| + 1 \wedge \\ & DTst' = \overline{D}Fst' = \overline{D}Tst' = idle \wedge \\ & DFst' = DFst \wedge \\ & \#DFStartFrom' = \#DFStartFrom \wedge \\ & \#\overline{D}FStartFrom' = \#\overline{D}FStartFrom + 1 \end{aligned} \tag{15}$$

This expresses the following facts: (i) in both models, both To logs have an additional entry corresponding to the failed To purse (spoof) transaction; (ii) nevertheless, after the way that we have matched the scenarios in the two models, the To purses in both models have returned to the *idle* state, as has the From purse in the $\overline{D}$ model, whereas the From purse in the D model has remained in the state it was in before; (iii) the number of occurrences of the *DFStartFrom* event in the D model has remained constant, whereas the number of occurrences of the $\overline{D}FStartFrom$ event in the $\overline{D}$ model has incremented.

Finally the concedes relation. The concedes relation says something about the state of affairs when the execution of the events in the operations in question cannot re-establish the retrieve relation. Ours is C_{Op}:

$$\begin{aligned} C_{Op} \equiv\ & |DTlog'| = |DTlog| + 1 \wedge \\ & |\overline{D}Tlog'| = |\overline{D}Tlog| + 1 \wedge \\ & DTst' = \overline{D}Fst' = \overline{D}Tst' = blocked \wedge \\ & DFst' = DFst \wedge \\ & \#DFStartFrom' = \#DFStartFrom \wedge \\ & \#\overline{D}FStartFrom' = \#\overline{D}FStartFrom + 1 \wedge \\ & \mathsf{RestSame...} \end{aligned} \tag{16}$$

This is very similar to the output relation (15). The notable differences are that the expicitly mentioned state is *blocked* rather than *idle*, and that we have included RestSame... This is a shorthand for equalities for the after-values of the remaining state variables not mentioned in (16), which would have been taken care of routinely via R in the conjunction $R \wedge O_{Op}$ in (2), in the previous case.

The preceding gives the bare bones of our retrenchment. It is not hard to see that the VC (2) becomes provable with these ingredients. Note that we have said nothing about what we assume to be the appropriate relationship between behaviours in the D and $\overline{D}$ models when our identified operations are *not* executing. We have to fix a position on this before we can make sensible statements about D and $\overline{D}$ model behaviours in the large. Given the closeness of the two models, the most natural thing to require is that other than during the execution of the two big operations, events in the two models correspond in a 1-1 manner.

With the relationship between the D and $\overline{D}$ models now fully defined, we have a means of translating runs in one model into runs in the other — only when this is precisely known can we begin to address the question of what a property of runs in one model 'becomes' in the other model, i.e. how *system properties translate*.

The orientation of the VC (2), in which hypotheses about $\overline{D}$ imply the existence of something in D, enables us to translate runs of the $\overline{D}$ model into runs of the D model as follows. Firstly, for simplicity, we restrict to runs which go to completion; i.e. once a transaction has started it continues through, either to success, or to the requisite *Abort*s. Secondly, given the 'completion' assumption, we can partition the events in the run into sets, each of which consists of the events for exactly one transaction. Thirdly, we observe that while the end of one transaction can overlap with the beginning of the next, transactions are effectively serial and we have a well defined notion of 'the next transaction'.[6]

Fourthly, one builds a simulation in the D world, of the $\overline{D}$ run as follows. For each successive transaction in the $\overline{D}$ run, if it consists of a set of events *other than* those constituting an $Op_{\overline{D}}$, we replace each $\overline{D}$ event by its D world counterpart, thus creating a D world transaction that accurately mimics the $\overline{D}$ world one. If however, the transaction consists of a set of events *which is* an instance of $Op_{\overline{D}}$, then we have a choice. *Either* we proceed as in the previous case, accurately reflecting the events of the $\overline{D}$ world one by one in the D world, *or* we use the retrenchment of *Op*, and replace the $\overline{D}$ world $\overline{D}FStartFrom$ and $\overline{D}FAbort_R$ events by the D world *DFIncrease* event, introducing an instance of DOS-inducing behaviour into the D world run.

Fixing this relationship between the worlds allows us to write down some facts relating the D and $\overline{D}$ worlds. In a homespun temporal notation we could have for instance:

$$\mathbf{A}_{\overline{D}}\ \mathbf{E}_D\ \#Op_{\overline{D}} = \#Op_D \tag{17}$$

This says that for all $\overline{D}$ world runs ($\mathbf{A}_{\overline{D}}$), there exists a D world run ($\mathbf{E}_D$) such that $\#Op_{\overline{D}}$ (the number of occurrences of the $\overline{D}$ world $Op_{\overline{D}}$) equals $\#Op_D$ (the number of occurrences of the D world Op_D). This is true since we could replace *every* $Op_{\overline{D}}$ by an Op_D in the preceding simulation. On the other hand, since we do not need to replace *all* of them, equally true is:

[6] Restricting to a world of exactly one To purse and exactly one From purse, and unidirectional money transfers, obviously facilitates this, but even in a more complex setup, with many purses, each playing To and From roles at various times, a suitable serial property survives, as one would expect for a financial application. See [11].

$$\mathsf{A}_{\overline{D}}\ \mathsf{E}_{D}\ \ \#Op_{\overline{D}} \geq \#Op_{D} \tag{18}$$

Another property of interest concerns the *blocked* state:

$$\mathsf{A}_{\overline{D}}\ \mathsf{E}_{D}\ \ \Diamond\, \overline{D}Tst = blocked\ \Rightarrow \Diamond\, DTst = blocked \tag{19}$$

This says that for all $\overline{D}$ world runs that eventually ($\Diamond$) reach a *blocked* state for the To purse ($\overline{D}Tst = blocked$), there exists a D world run that eventually also reaches *blocked* for the To purse. (N. B. In both cases, the *blocked* state effectively ends the run since only *Abort* and *Increase* remain enabled in the *blocked* state in our models.)

6. Conclusions

In this paper we developed a coarse grained approach to retrenchment, and applied it to the DOS attacks that the Mondex protocol is vulnerable to. We saw that the technique gave us a good way of capturing the evolution of certain system properties. Lack of space prevented us exploring this as extensively as we would have liked, but the work here showed that the approach we initiated holds great promise for investigating the evolution of coarse-grained and temporal system properties through relatively arbitrary changes in the system model. This is, in general, certainly a very challenging question, and one that will be pursued more extensively elsewhere.

A final remark. In the whole of this paper we have been exclusively concerned with functional properties. Can similar techniques be used for non-functional properties of applications, such as the famed '-ilities' beloved of the aspect approach? The view of the author is that they can, provided one can *quantify* and *compose* the relevant aspect information in a suitable way. This would open the door to a principled propagation of the aspect-relevant information throughout the system structure, and as the system evolves. The details of this also remain as future work.

References

[1] *KIV Homepage*. http://www.informatik.uni-augsburg.de/lehrstuehle/swt/se/kiv.

[2] R. Banach, C. Jeske, and M. Poppleton. Composition Mechanisms for Retrenchment. *J. Log. Alg. Prog.*, 75:209–229, 2008.

[3] R. Banach, C. Jeske, M. Poppleton, and S. Stepney. Retrenching the Purse: Finite Exception Logs, and Validating the Small. In *Software Engineering Workshop 30*, pages 234–245, Layola College, Columbia, MD, 2006. IEEE.

[4] R. Banach, C. Jeske, M. Poppleton, and S. Stepney. Retrenching the Purse: Hashing Injective CLEAR Codes, and Security Properties. In *Second International Symposium on Leveraging Applications of Formal Methods*, pages 82–90, Paphos, Cyprus, 2006. IEEE.

[5] R. Banach, C. Jeske, M. Poppleton, and S. Stepney. Retrenching the Purse: The Balance Enquiry Quandary, and Generalised and (1,1) Forward Refinements. *Fundamenta Informaticae*, 77:29–69, 2007.

[6] R. Banach and M. Poppleton. Retrenchment: An Engineering Variation on Refinement. In *2nd Int. B Conference*, volume 1393 of *LNCS*, pages 129–147. Springer, 1998.

[7] R. Banach and M. Poppleton. Retrenchment and punctured simulation. In *Proc. IFM'99:Integrated Formal Methods 1999*, pages 457–476. Springer, June 1999.

[8] R. Banach and M. Poppleton. Sharp Retrenchment, Modulated Refinement and Simulation. *Formal Aspects of Computing*, 11:498–540, 1999.

[9] R. Banach, M. Poppleton, C. Jeske, and S. Stepney. Retrenching the Purse: Finite Sequence Numbers and the Tower Pattern. In *Formal Methods 2005*, volume 3582 of *LNCS*, pages 382–398. Springer, 2005.

[10] R. Banach, M. Poppleton, C. Jeske, and S. Stepney. Engineering and Theoretical Underpinnings of Retrenchment. *Science of Computer Programing*, 67:301–329, 2007.

[11] R. Banach and G. Schellhorn. Atomic Actions, and their Refinements to Isolated Protocols. *Formal Aspects of Computing*, 2009. to appear.

[12] G. Boudol. Flow Event Structures and Flow Nets. In *Semantics and Systems of Concurrent Processes, Proc. LITP-90*, pages 62–95. Springer LNCS 469, 1990.

[13] W.-P. de Roever and K. Engelhardt. *Data Refinement: Model-Oriented Proof Methods and their Comparison*. Cambridge University Press, 1998.

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

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

[16] ISO 15408, v. 3.0 rev. 2. *Common Criteria for Information Security Evaluation*, 2005.

[17] C. Jones and J. Woodcock (eds.). Special Issue on Mondex Verification. *Formal Aspects of Computing*, 20(1):1–139, 2008.

[18] M. Nielsen, G. Plotkin, and G. Winskel. Petri Nets, Event Structures and Domains. *Journal of Theoretical Computer Science*, 13:85–108, 1981.

[19] Retrenchment Homepage. http://www.cs.man.ac.uk/retrenchment.

[20] G. Schellhorn, H. Grandy, D. Haneberg, N. Moebius, and W. Reif. A Systematic Verification Approach for Mondex Electronic Purses using ASMs. In *Dagstuhl Seminar on Rigorous Methods for Software Construction and Analysis*, volume 5115 of *LNCS*. Springer, 2008.

[21] S. Stepney, D. Cooper, and J. Woodcock. An Electronic Purse: Specification, Refinement and Proof. Technical Report PRG-126, Oxford University Computing Laboratory, 2000.

[22] G. Winskel. Event Structures. In Brauer and Reisig and Rozenberg, editor, *Advances in Petri Nets*, pages 325–392. Springer LNCS 255, 1986.

[23] G. Winskel and M. Nielsen. Models for Concurrency. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science Volume 4 Semantic Modelling*, pages 1–148. Oxford Univ. Press, 1995.