Model Based Engineering of Specifications by Retrenching Partial Requirements

R. Banach[a], M. Poppleton[b]

[a]Computer Science Dept., Manchester University, Manchester, M13 9PL, U.K.
[b]Dept. of Computing, Open University, Milton Keynes, MK7 6AL, U.K.
banach@cs.man.ac.uk , m.r.poppleton@open.ac.uk

Abstract

In conventional model-oriented formal refinement, the abstract model is supposed to capture all the properties of interest in the system, in an as-clutter-free-as-possible manner. Subsequently, the refinement process guides development inexorably towards a faithful implementation. However refinement says nothing about how to obtain the abstract model in the first place. In reality developers experiment with prototype models and their refinements until a workable arrangement is discovered.

Retrenchment is a formal technique intended to capture some of the informal approach to a refinable abstract model in a formal manner that will integrate with refinement. This is in order that the benefits of a formal approach can migrate further up the development hierarchy. After a presentation of the basic ideas of retrenchment, a simple telephone system feature interaction case study is given to illustrate how retrenchment can relate incompatible partial models to a more definitive consolidated model during the development of the contracted specification.

1. Introduction

Formal refinement, in its various guises, has a long and distinguished history. From the early papers of Wirth, Dijkstra, Hoare [1, 2, 3], it has developed into a large and vibrant field of research. A comprehensive survey would be out of place here, but modern accounts in the spirit of the original work can be found in [4, 5]. In all of these the assumption is that one *knows already* what the abstract model is, and all one has to do is to refine it to a suitable lower level model, gaining a high degree of assurance for the development thereby.

But the reality is, that in most software development, the correct abstract model is by no means obvious at the outset. Anecdotal evidence[1] suggests that this is not only true where one would expect it, namely in the development of large and complex real world critical applications, undertaken using a formal approach because of the belief in the assurance obtainable (or because legislation mandates it), but is even present in the behind the scenes aspects of the development of small textbook or research examples, in which some experimentation is often required before a model that will satisfactorily refine to the desired concrete one is arrived at. (And that last sentence exhibits an undertone that is quite deliberate, because it is frequently true that at the outset one has a firmer idea of what the concrete model looks like than the abstract one, and one reverse engineers the latter from the concrete one to some degree.) The upshot of this is that formal approaches, of the conventionally understood kind, do not help much in the creation of an abstract model that can be contracted to with confidence for further development. Not that they ever claimed to, but in the 'oversold and underused' [6] atmosphere that has often surrounded debate about formal techniques in the past, it is easy to imagine that they might have done.

1. Several private communications.

Retrenchment [7, 8, 9, 10], is a technique that aims to help address this issue by providing a formalism in which the demanding proof obligations (POs) of refinement are weakened, so that models not refineable to the ultimate concrete system, but nevertheless considered useful, can be incorporated into the development in a formal manner. This is not to say that every misconception and blunder that led to the correct abstraction ought to be recorded in some sort of retrenchment audit trail, but that a *sanitised* account[2] of the construction of the abstract model from preliminary but incomplete precursors *that are considered convincing by the domain experts* is of benefit.

The stress on the acquiescence of domain experts is vital. To seek to impose from the outside, an alien development discipline on an already well established engineering mileu is doomed to failure. Yet a naive effort to impose refinement as a software development technique can result in exactly that. To be able to successfuly discharge the refine-

2. Taking some liberties with language, we mean not only 'made sanitary' but 'made sane'.

ment POs can force a development to adopt a structure surprisingly unlike what one might imagine at the outset, especially when interfacing with physical models. Furthermore engineers with an established track record of successful development are seldom sympathetic to the suggestion that all their familiar working practices must suddenly be abandoned in favour of a way of working forced implicitly by the rigidity of the refinement POs.

Retrenchment is a technique that seeks not to disturb well entrenched engineering habits, by allowing models to be developed in a manner more in tune with engineering intuitions. Yet it aims to do so in a manner that can ultimately be integrated with refinement. To do so, the POs of retrenchment must be less exigent than those of refinement, but neverthless have a structure that is close enough to those of the refinement POs to make the reconciliation feasible; we will see the details below. Above all, it is vital that the mathematics of the formalism be the servant and not the master during the development activity.

The development route that retrenchment opens up now appears as follows. In the initial stages of requirements definition and specification design, many preliminary and partial models are built. Some of these may well prove, upon experimentation and further reflection, to be misguided. They can be discarded. Other models will, perhaps after some modifications, contain a sensible account of aspects of the desired behaviour of the intended system. Unfortunately, it is quite likely that not all of these sensible models will be compatible with each other, in that being concerned with only part of the desired behaviour, and above all with clarity and intuitive perspicuity, not all of the complexities of how the part focused on interacts with other parts will have been ironed out. Nor indeed should one expect it to have been. One must understand first the broad intentions before narrowing down on the finer details; details moreover that may only be of concern in limited special cases. On a formal level, the incompatibility we speak of usually manifests itself in the impossibility of accomodating the various models we speak of in a single refinement based development. Retrenchment, being more forgiving of this kind of incompatibility, offers the possibility of retrenching from such a collection of models to a more complicated model that properly takes into account all the requirements, and that can serve both as the basis of a contract between customer and supplier, and as the basis of a subsequent refinement based implementation. We call this latter model the contracted model.

The reflective process involved in reconciling the incompatible partial models with the contracted model, which is partially captured in the retrenchment relations and proof obligations between these models, strengthens the confidence that the right contracted model has been decided upon, an activity that would otherwise be completely informal. At worst this is simply because it *is* a reflective process. *Any* kind of reconsideration of such design decisions from a novel standpoint is bound to be helpful to some degree, simply because two perspectives are always better than one. At best the engineering of the POs of the retrenchments will have brought into sharp focus the most important issues that need to be clarified in firming up the contracted model. One side effect of retrenchment is to provide a formal framework within which such considerations can exist.

The rest of this paper is as follows. The next section reviews the basic ideas of retrenchment. In Section 3 we develop a very primitive telephone system model, together with two independent enhancements, call forwarding and call hold. Since there are areas of incompatibility between the primitive model and the enhancements, retrenchment is needed to describe the relationships between the primitive model and the enhanced models. Section 4 considers how the two enhancements may be combined: again there are areas of incompatibility when both enhancements can be triggered. It is shown that given a design decision about how to resolve the incompatibility, retrenchments can relate the two enhancements to the resulting final model. Section 5 considers the two compositions of the enhancements along the two routes from the original model to the final model, and compares these to a retrenchment description of a one step derivation. This attests to the solidity of the retrenchment technique. Section 6 concludes.

2. Retrenchment

In this paper we work in a transition system framework for simplicity. In the context of model based requirements development, we will consider the relationship between two systems: the *abridged* system, expressing an idealised view of a part of the desired system, and the *completed* system, that takes all the necessary details into account[3].

At the abridged level, we have a set of operations, $\mathsf{Ops}_{\mathrm{A}}$ with typical element m_{A}, and our state space, input spaces, and output spaces, are U, $\mathsf{I}_{m_{\mathrm{A}}}$, $\mathsf{O}_{m_{\mathrm{A}}}$, respectively. Meta level values in U, $\mathsf{I}_{m_{\mathrm{A}}}$, $\mathsf{O}_{m_{\mathrm{A}}}$ will be written u, i, o respectively, with primes or subscripts or indices to distinguish different values from the same space. (We will lighten the notation by not subscripting input and output values.) Transitions will be written as u -(i, m_{A}, o)-› u', where u and u' are the before- and after- states, i and o are the input and output, and m_{A} is the operation responsible for the transition. The totality of such transitions makes up the transition or step relation for m_{A}, $stp_{m_{\mathrm{A}}}(u, i, u', o)$.

At the completed level we have state, input and output spaces V, J, P, respectively, with values v, j, p, and similar

3. Most presentations of retrenchment speak of an *abstract* and a *concrete* system, in the spirit of moving towards an implementation.

$$0 \longrightarrow (\varepsilon, Up_{\mathrm{A}}, \varepsilon) \longrightarrow 3$$

$$0 \longrightarrow (\varepsilon, Up_{\mathrm{C}}, Done) \longrightarrow 3$$
$$0 \longrightarrow (\varepsilon, Up_{\mathrm{C}}, Error) \longrightarrow X$$

Fig. 1

conventions, except that we write operation name sets and operation names subscripted with C, eg. m_{C}. We assume each abridged level operation m_{A}, has a corresponding completed level operation m_{C}, but there may also be other completed level operations, so that there is an injection from the set $\mathsf{Ops}_{\mathrm{A}}$ to $\mathsf{Ops}_{\mathrm{C}}$, which associates m_{A} with m_{C}.

The relationship between abridged and completed state spaces is given by the retrieve relation $G(u, v)$. The initialisation operation *Init* at abridged and completed levels establishes G in corresponding after-states (as usual, the free variables are assumed implicitly universally quantified):

$$Init_{\mathrm{C}}(v') \;\Rightarrow\; (\exists\, u' \bullet Init_{\mathrm{A}}(u') \wedge G(u', v')) \qquad (2.1)$$

Turning to the transition relation for operation m_{A}, beyond the retrieve relation G, we have the within relation $P_m(i, j, u, v)$, and concedes relation $C_m(u', v', o, p; i, j, u, v)$. The punctuation indicates that C_m is mainly concerned with after-values, but may refer to before-values too where necessary. These are combined into the retrenchment PO for steps which says that for each m_{A}:

$$\begin{aligned} & G(u, v) \wedge P_m(i, j, u, v) \wedge stp_{m_{\mathrm{C}}}(v, j, v', p) \;\Rightarrow \\ & \quad (\exists\, u', o \bullet stp_{m_{\mathrm{A}}}(u, i, u', o) \wedge \\ & \quad (G(u', v') \vee C_m(u', v', o, p; i, j, u, v))) \qquad (2.2) \end{aligned}$$

This PO affords considerable flexibility in relating different levels of abstraction, see [7, 8, 9, 10] for a discussion.

We consider a toy example, just to set the scene. The abridged level is given by an initialisation operation $Init_{\mathrm{A}}$, and one further operation Up_{A}. We have $\mathsf{U} = \{0, 3\}$, and $\mathsf{I}_{Up_{\mathrm{A}}} = \varnothing = \mathsf{O}_{Up_{\mathrm{A}}}$. $Init_{\mathrm{A}}$ sets u to 0, and Up_{A} is given by 0 -(ε, Up_{A}, ε)-› 3, where ε is the empty input and output; this is the only step in $stp_{Up_{\mathrm{A}}}$. At the completed level we have $Init_{\mathrm{C}}$, and Up_{C}. The state space is $\mathsf{V} = \{0, 3, X\}$, and $\mathsf{J}_{Up_{\mathrm{C}}} = \varnothing$, $\mathsf{P}_{Up_{\mathrm{C}}} = \{Done, Error\}$. $Init_{\mathrm{C}}$ sets v to 0, and $stp_{Up_{\mathrm{C}}} = \{0$ -(ε, Up_{C}, $Done$)-› 3, 0 -(ε, Up_{C}, $Error$)-› $X\}$. The nontrivial steps are illustrated in Fig. 1.

The retrieve relation is given by the inclusion of U into V i.e. equality of abstract and concrete values, and the within relation for Up is $\mathsf{U} \times \mathsf{V}$ (i.e. we have a trivial within relation), where we also ignore the presence of the empty input spaces. There is some scope for choosing the concedes relation C_{Up}. The smallest possibility is:

$$C_1 = \{(u', v', p) \mid u' = 3 \wedge v' = X \wedge p = Error\}$$

while other possibilities include:

$$\begin{aligned} C_2 = \{(u', v', p) \mid\; & (v' = X \wedge p = Error) \vee \\ & (v' = u' \wedge p = Done)\} \end{aligned}$$

$$\begin{aligned} C_3 = \{(u', v', p) \mid\; & (p = Error \Rightarrow v' = X) \wedge \\ & (p = Done \Rightarrow v' = u')\} \end{aligned}$$

Note that $C_2 = C_3$ because of the smallness of the spaces involved. These different possibilities indicate some of what can be expressed using retrenchment in a more syntactically based framework. It is easy to check that the PO (2.2) holds for each of the C_i's.

3. Features in a Simple Telephone Model

We will illustrate the potential for retrenchment to capture the evolution of an integrated specification from incomplete and contradictory partial models, using feature interaction in telephone systems. There is now a substantial literature on this topic, eg. [11, 12], since the naive combination of novel services on top of the plain old telephone system (POTS) model can be problematic. Since our aim is to illustrate retrenchment and not to advance telephony, our models will be oversimplified in the extreme. Still, they will make the intended points. In this section we start with the simplest model and then consider the addition of call forward and call hold facilities, a well known case in which the naive combination of extra services does not work.

***PHONE*:** In this system the state space is just the set of active calls, a partial injection on the available phones NUM, initialised empty:

$$\begin{aligned} & calls : \mathsf{NUM} \rightarrowtail\!\!\!\!\mapsto \mathsf{NUM} \text{ where} \\ & \quad \mathrm{dom}(calls) \cap \mathrm{rng}(calls) = \varnothing \qquad (3.1) \end{aligned}$$

There are just two operations, $connect_n(i)$ and $break_n$, the former to dial number i from phone n, and the latter to disconnect phone n. We define $\mathrm{free}(n) \equiv n \notin \mathrm{fld}(calls) \equiv \neg\, \mathrm{busy}(n)$, where $\mathrm{fld}(R) = \mathrm{dom}(R) \cup \mathrm{rng}(R)$.

$$\begin{aligned} & calls \text{ -}(i, connect_n, o)\text{-› } calls' \text{ where} \\ & \quad \mathrm{free}(n) \wedge \\ & \quad \text{if } \mathrm{free}(i) \wedge (n \neq i) \\ & \quad \text{then } o = OK \wedge calls' = calls \cup \{n \mapsto i\} \\ & \quad \text{else } o = NO \wedge calls' = calls \qquad (3.2) \end{aligned}$$

$$\begin{aligned} & calls \text{ -}(break_n)\text{-› } calls' \text{ where} \\ & \quad \mathrm{busy}(n) \wedge calls' = \{n\} ⩤ calls ⩥ \{n\} \qquad (3.3) \end{aligned}$$

From this very basic model we construct enhanced services one at a time. First call forwarding.

***PHONE*$_{CF}$:** In this system the state space is the set of active calls as before, plus a table of call forwarding data, the latter being a partial injection on the phones whose transitive closure is acyclic, and also initialised empty:

$$\begin{aligned} & fortab : \mathsf{NUM} \rightarrowtail\!\!\!\!\mapsto \mathsf{NUM} \text{ where} \\ & \quad fortab^{+} \cap \mathrm{id}_{\mathsf{NUM}} = \varnothing \qquad (3.4) \end{aligned}$$

Two new operations $regfor_{CF,n}(i)$ and $delfor_n$ manipulate the table. The former inserts forwarding destinations in the table, the latter removes them. Note that for simplicity we do not mention parts of the state (i.e. *calls*) left unaltered by an operation. Note that $regfor_{CF,n}$ merely overwrites any existing information in the table if it is safe to do so.

$$\begin{aligned}&fortab \text{ -}(i,\, regfor_{CF,n})\text{-}\!\!> fortab' \ \text{ where}\\ &\quad \text{if } (fortab \ntriangleleft \{n \mapsto i\})^+ \cap \mathrm{id}_{\mathsf{NUM}} = \varnothing\\ &\quad \text{then } fortab' = fortab \ntriangleleft \{n \mapsto i\}\\ &\quad \text{else } fortab' = fortab\end{aligned} \tag{3.5}$$

$$\begin{aligned}&fortab \text{ -}(delfor_{CF,n})\text{-}\!\!> fortab' \ \text{ where}\\ &\quad fortab' = \{n\} \ntriangleleft fortab\end{aligned} \tag{3.6}$$

In the presence of this new service, the $connect_n(i)$ and $break_n$ operations need reexamination, as their behaviour potentially changes due to the new requirements. Indulging now and henceforth in a little imprecision regarding definedness of $fortab^+(i)$ and of similar relations, the $connect_n(i)$ operation may be reengineered thus:

$$\begin{aligned}&calls \text{ -}(i,\, connect_{CF,n},\, o)\text{-}\!\!> calls' \ \text{ where}\\ &\quad \mathrm{free}(n) \wedge\\ &\quad \text{if } \mathrm{free}(i) \wedge (n \neq i)\\ &\quad \text{then } o = OK \wedge calls' = calls \cup \{n \mapsto i\}\\ &\quad \text{else if } \mathrm{busy}(i) \wedge fortab^+(i) = z \wedge \mathrm{free}(z)\\ &\quad \text{then } o = OK \wedge calls' = calls \cup \{n \mapsto z\}\\ &\quad \text{else } o = NO \wedge calls' = calls\end{aligned} \tag{3.7}$$

while the $break_n$ operation, it turns out, is unaltered:

$$break_{CF,n} = break_n \tag{3.8}$$

This completes call forwarding. Now for call holding.

***$PHONE_{CH}$*:** In this system the state space is the set of active calls, plus a table of call holding data, this being a subset of the phones, initialised empty:

$$holtab \subseteq \mathsf{NUM} \tag{3.9}$$

Two operations insert and remove elements of this subset.

$$\begin{aligned}&holtab \text{ -}(reghol_{CH,n})\text{-}\!\!> holtab' \ \text{ where}\\ &\quad holtab' = holtab \cup \{n\}\end{aligned} \tag{3.10}$$

$$\begin{aligned}&fortab \text{ -}(delhol_{CH,n})\text{-}\!\!> fortab' \ \text{ where}\\ &\quad holtab' = holtab - \{n\}\end{aligned} \tag{3.11}$$

With this service, $connect_n(i)$ and $break_n$ also need reexamination, for the same reason as above. The $connect_n(i)$ operation simulates rather primitively the infuriating feedback obtainable from most holding services; however there is no attempt to accurately model the resolution of a hold when the call recipient bcomes free:

$$\begin{aligned}&calls \text{ -}(i,\, connect_{CH,n},\, o)\text{-}\!\!> calls' \ \text{ where}\\ &\quad \mathrm{free}(n) \wedge\\ &\quad \text{if } \mathrm{free}(i) \wedge (n \neq i)\\ &\quad \text{then } o = OK \wedge calls' = calls \cup \{n \mapsto i\}\\ &\quad \text{else if } \mathrm{busy}(i) \wedge i \in holtab\\ &\quad \text{then } o = (\text{“Our advisor is busy. Please hold.”})^{100} \wedge\\ &\qquad\qquad calls' = calls\\ &\quad \text{else } o = NO \wedge calls' = calls\end{aligned} \tag{3.12}$$

The $break_n$ operation is unaltered as before:

$$break_{CH,n} = break_n \tag{3.13}$$

completing the call holding model.

Before going on to consider feature interaction, we can ask how these two enhanced models ***$PHONE_{CF}$*** and ***$PHONE_{CH}$***, are related to ***PHONE***. The natural expectation is that they would be refinements of ***PHONE***, but this turns out not to be the case. The reason is that the simple ***PHONE*** system prescribes a specific response for the busy(i) case, this being given by $o = NO \wedge calls' = calls$, a naive model of the engaged tone. Under the same conditions, when appropriate supplementary conditions hold, the two enhanced models prescribe different and incompatible behaviour: in ***$PHONE_{CF}$*** a connection can be made to the forward location, while in ***$PHONE_{CH}$*** an irritating message drones on interminably. This means that the enhanced models cannot be viewed as straightforward extensions of the ***PHONE*** model. But in one sense or another this would be necessary if the relationships between ***PHONE*** and the other systems were to be refinements.

There is no difficulty however in casting these relationships as retrenchments.

PHONE* to *$PHONE_{CF}$*:** We set up the data for the retrenchment as follows, with ***PHONE as the abridged model and ***$PHONE_{CF}$*** as the completed model, and taking some minor liberties with notation:

$$G_{CF}(u, v) = (u = calls \wedge v = (calls, fortab)) \tag{3.14}$$

$$P_{CF,connect_n}(i, j, u, v) = (i = j) \tag{3.15}$$

$$\begin{aligned}&C_{CF,connect_n}(u', v', o, p;\, i, j, u, v) =\\ &\quad (\mathrm{busy}(j) \wedge fortab^+(j) = z \wedge \mathrm{free}(z) \wedge\\ &\quad u' = u \wedge v' = (calls \cup \{n \mapsto z\}, fortab) \wedge\\ &\quad o = NO \wedge p = OK)\end{aligned} \tag{3.16}$$

$$P_{CF,break_n}(u, v) = \mathsf{true} \tag{3.17}$$

$$C_{CF,break_n}(u', v';\, u, v) = \mathsf{false} \tag{3.18}$$

Showing that the POs of retrenchment hold for these data is now easy. The initialisation PO (2.1) is trivial given that all the sets in the states of both models are initialised empty. Also the operation PO (2.2) is easy given that the only case where the actions of $connect_n$ and $connect_{CF,n}$ differ is precisely the case documented in the concedes relation (3.11); also the two break operations are identical.

PHONE* to *$PHONE_{CH}$*:** The abridged model is ***PHONE as before and ***$PHONE_{CH}$*** is now the completed model:

$$G_{CH}(u, v) = (u = calls \wedge v = (calls, holtab)) \tag{3.19}$$

$$P_{CH,connect_n}(i, j, u, v) = (i = j) \tag{3.20}$$

$$\begin{aligned}&C_{CH,connect_n}(u', v', o, p;\, i, j, u, v) =\\ &\quad (\mathrm{busy}(j) \wedge j \in holtab \wedge u' = u \wedge v' = v \wedge\\ &\quad o = NO \wedge p = (\text{“Our … hold.”})^{100})\end{aligned} \tag{3.21}$$

$$P_{CH,break_n}(u, v) = \mathsf{true} \quad (3.22)$$

$$C_{CH,break_n}(u', v'; u, v) = \mathsf{false} \quad (3.23)$$

The POs are as straightforward as previously. The initialisation PO (2.1) is trivial, and the operation PO (2.2) is also similar to the preceding case, and for the same reason.

4. Feature Interaction in Telephony

Having built our basic system, and having separately considered the call forwarding and call holding optional enhancements, we now consider combining the two features. Any combination is based on the assumption that the *calls* state component and the input and output spaces of the two variants of the $connect_n$ and $break_n$ operations are to be identified insofar as possible. (This precludes constructions that incorporate say two *calls* state components and then implement call forwarding in one, and call holding in the other. Formally this might work up to some point, but in practice such solutions are not useful as models of the real world.) Here is our combined system.

$\boldsymbol{PHONE_{CF/CH}}$**:** Here the state space is *calls* as before, plus a table of call forwarding data, plus a table of call holding data:

$$\begin{aligned}
&(\, calls : \mathsf{NUM} \rightarrowtail\!\!\!\rightarrow \mathsf{NUM}\ ,\\
&\ \ fortab : \mathsf{NUM} \rightarrowtail\!\!\!\rightarrow \mathsf{NUM}\ ,\\
&\ \ holtab \subseteq \mathsf{NUM}\ )\ \text{ where}\\
&\mathrm{dom}(calls) \cap \mathrm{rng}(calls) = \varnothing\ \wedge\\
&fortab^{+} \cap \mathrm{id}_{\mathsf{NUM}} = \varnothing
\end{aligned} \quad (4.1)$$

The auxiliary operations to manage the two tables are unchanged:

$$\begin{aligned}
&regfor_{CF/CH,n} = regfor_{CF,n}\\
&delfor_{CF/CH,n} = delfor_{CF,n}\\
&reghol_{CF/CH,n} = reghol_{CH,n}\\
&delhol_{CF/CH,n} = delhol_{CH,n}
\end{aligned} \quad (4.2)$$

as is the break operation:

$$break_{CF/CH,n} = break_{CF,n} = break_{CH,n} = break_n \quad (4.3)$$

The interest lies of course in the $connect_{CF/CH,n}(i)$ operation. Our design is guided by the following principles. Firstly, if the conditions for neither service enhancement hold, then the system should behave like the plain ***PHONE*** service. Secondly, if the conditions for exactly one of the service enhancements hold, then the system should behave according to that enhancement, either $\boldsymbol{PHONE_{CF}}$ or $\boldsymbol{PHONE_{CH}}$ as appropriate. The third case, when the conditions for both the call forwarding and call hold enhancements are valid, requires a design decision. We determine that in this case, the caller should have the option of being forwarded rather than a simple default of being held. To keep things as simple as previously, we do not model the interaction with the caller or the resolution of a hold situation very faithfully, modelling it by a particular message at the output, in line with the unsophisticated nature of all the models in this paper.

$$\begin{aligned}
&calls \ \text{-}(i, connect_{CF/CH,n}, o)\text{-}\!\!\rightarrow calls' \ \text{ where}\\
&\quad \mathrm{free}(n)\ \wedge\\
&\quad \text{if } \mathrm{free}(i) \wedge (n \neq i)\\
&\quad \text{then } o = OK \wedge calls' = calls \cup \{n \mapsto i\}\\
&\quad \text{else if } \mathrm{busy}(i) \wedge i \notin holtab \wedge fortab^{+}(i) = z\ \wedge\\
&\qquad\quad \mathrm{free}(z)\\
&\quad \text{then } o = OK \wedge calls' = calls \cup \{n \mapsto z\}\\
&\quad \text{else if } \mathrm{busy}(i) \wedge i \in holtab\ \wedge\\
&\qquad\quad (i \notin \mathrm{dom}(fortab) \vee \mathrm{busy}(fortab^{+}(i))\\
&\quad \text{then } o = (\text{“Our advisor is busy. Please hold.”})^{100}\ \wedge\\
&\qquad\qquad calls' = calls\\
&\quad \text{else if } \mathrm{busy}(i) \wedge i \in holtab \wedge fortab^{+}(i) = z\ \wedge\\
&\qquad\quad \mathrm{free}(z)\\
&\quad \text{then } o = (\text{“Our advisor is busy. Please press 1}\\
&\qquad\qquad \text{to speak to the janitor.”}) \wedge calls' = calls\\
&\quad \text{else } o = NO \wedge calls' = calls
\end{aligned} \quad (4.4)$$

It is clearly at least plausible to say that refinements will not hold either between $\boldsymbol{PHONE_{CF}}$ and $\boldsymbol{PHONE_{CF/CH}}$, or between $\boldsymbol{PHONE_{CH}}$ and $\boldsymbol{PHONE_{CF/CH}}$. For a specific instance this is because in the penultimate of the cases for the $connect_{CF/CH,n}(i)$ operation, the conditions for forwarding and holding are both true, and the behaviour specified is not the same as in either $\boldsymbol{PHONE_{CF}}$ or $\boldsymbol{PHONE_{CH}}$.

Despite this, retrenchment can give a good account of the situation, due to the more flexible proof obligations that characterise it.

$\boldsymbol{PHONE_{CF}}$ **to** $\boldsymbol{PHONE_{CF/CH}}$**:** In contrast to the two retrenchments given previously, this time $\boldsymbol{PHONE_{CF}}$ is the abridged model and $\boldsymbol{PHONE_{CF/CH}}$ is the completed model, illustrating how in a development hierarchy, what is regarded as concrete at one point, becomes abstract when one focuses lower down. This is just as appropriate for the piecewise development of a specification from preliminary models as it is when developing an implementation from an agreed specification.

$$\begin{aligned}
&G_{CF\rangle CH}(u, v) = (u = (calls, fortab)\ \wedge\\
&\qquad v = (calls, fortab, holtab))
\end{aligned} \quad (4.5)$$

$$P_{CF\rangle CH,connect_n}(i, j, u, v) = (i = j) \quad (4.6)$$

$$\begin{aligned}
&C_{CF\rangle CH,connect_n}(u', v', o, p; i, j, u, v) =\\
&\quad (\mathrm{busy}(j) \wedge j \in holtab\ \wedge\\
&\qquad (\neg(fortab^{+}(j) = z \wedge \mathrm{free}(z))\ \wedge\\
&\qquad u' = u \wedge v' = v \wedge o = NO\ \wedge\\
&\qquad p = (\text{“Our … hold.”})^{100})\ \vee\\
&\qquad (fortab^{+}(i) = z \wedge \mathrm{free}(z)\ \wedge\\
&\qquad u' = (calls \cup \{n \mapsto z\}, fortab)\ \wedge\\
&\qquad v' = v \wedge o = OK\ \wedge\\
&\qquad p = (\text{“Our … janitor.”})))
\end{aligned} \quad (4.7)$$

$$P_{CF›CH,break_n}(u, v) = \mathsf{true} \quad (4.8)$$

$$C_{CF›CH,break_n}(u', v'; u, v) = \mathsf{false} \quad (4.9)$$

$$P_{CF›CH,regfor_n}(i, j, u, v) = (i = j) \quad (4.10)$$

$$C_{CF›CH,regfor_n}(u', v'; i, j, u, v) = \mathsf{false} \quad (4.11)$$

$$P_{CF›CH,delfor_n}(u, v) = \mathsf{true} \quad (4.12)$$

$$C_{CF›CH,delfor_n}(u', v'; u, v) = \mathsf{false} \quad (4.13)$$

It is clear that the relvant POs hold. Initialisation is trivial as usual, and the operation POs verify that the cases where the abridged and completed models differ, is adequately documented in the concedes clause.

PHONE$_{CH}$ **to** ***PHONE***$_{CF/CH}$**:** Here the abridged model is ***PHONE***$_{CH}$ and ***PHONE***$_{CF/CH}$ plays the part of the completed model, so the role of ***PHONE***$_{CH}$ is different to its role in the other retrenchment in which it appears.

$$G_{CH›CF}(u, v) = (u = (calls, holtab) \wedge v = (calls, fortab, holtab)) \quad (4.14)$$

$$P_{CH›CF,connect_n}(i, j, u, v) = (i = j) \quad (4.15)$$

$$\begin{aligned} &C_{CH›CF,connect_n}(u', v', o, p; i, j, u, v) = \\ &\quad (\text{busy}(j) \wedge fortab^+(j) = z \wedge \text{free}(z) \wedge \\ &\qquad ((j \notin holtab \wedge u' = u \wedge o = NO \wedge p = OK \wedge \\ &\qquad v' = (calls \cup \{n \mapsto z\}, fortab, holtab)) \vee \\ &\qquad (j \in holtab \wedge u' = u \wedge v' = v \wedge \\ &\qquad o = (\text{“Our … hold.”})^{100} \wedge \\ &\qquad p = (\text{“Our … janitor.”})))) \end{aligned} \quad (4.16)$$

$$P_{CH›CF,break_n}(u, v) = \mathsf{true} \quad (4.17)$$

$$C_{CH›CF,break_n}(u', v'; u, v) = \mathsf{false} \quad (4.18)$$

$$P_{CF›CH,reghol_n}(u, v) = \mathsf{true} \quad (4.19)$$

$$C_{CF›CH,reghol_n}(u', v'; u, v) = \mathsf{false} \quad (4.20)$$

$$P_{CF›CH,delhol_n}(u, v) = \mathsf{true} \quad (4.21)$$

$$C_{CF›CH,delhol_n}(u', v'; u, v) = \mathsf{false} \quad (4.22)$$

The POs are as straightforward as previously. The initialisation PO (2.1) is trivial, and the operation PO (2.2) is also similar to the preceding case, and for the same reason.

We note that in both of these retrenchments, the concedes clause for the $connect_n$ operation has to cater for two exceptional conditions. In the case of the *CF›CH* retrenchment, when holding is available, the two actions for forwarding available or not are both incompatible with ***PHONE***$_{CF}$, while in the *CH›CF* retrenchment, when forwarding is available the two actions for holding available or not are both incompatible with ***PHONE***$_{CH}$. Aside from these nontrivial cases, we have a greater proliferation of essentially trivial operation POs, arising from the fact that ***PHONE***$_{CF}$ and ***PHONE***$_{CH}$ have management operations for the forward and hold tables respectively, and these are also present in identical fashion in ***PHONE***$_{CF/CH}$.

5. Compositions of Retrenchments and a Direct Retrenchment Design

Given that we have two routes to get from the simple model ***PHONE*** to the final model ***PHONE***$_{CF/CH}$, the first via ***PHONE***$_{CF}$ and the second via ***PHONE***$_{CH}$, we can examine the compositions of the relevant pairs of retrenchments and compare them, both to each other and to a one step retrenchment obtaining the final design from the original simple ***PHONE*** system.

For the formulation of retrenchment used in this paper, the method of composing retrenchments is examined in detail in [10]. For brevity we just cite the results.

Suppose we have at top level a system given by variables u, i, u', o (for a typical operation). At intermediate level suppose the variables are v, j, v', p (for the corresponding operation). And at lowest level suppose the variables are w, k, w', q (for an operation corresponding to an intermediate level operation with variables v, j, v', p). Suppose a retrenchment is given from top level to intermediate level with retrieve relation $G(u, v)$, and for a top level operation m, within and concedes relations $P_m(i, j, u, v)$, $C_m(u', v', o, p; i, j, u, v)$. Suppose there is also a retrenchment from intermediate level to lowest level whose retrieve relation is $H(v, w)$, and for intermediate level operation m, within and concedes relations $Q_m(j, k, v, w)$, $D_m(v', w', p, q; j, k, v, w)$. Then there is a retrenchment from top level to lowest level whose retrieve relation is:

$$K(u, w) = (\exists\, v \bullet G(u, v) \wedge H(v, w)) \quad (5.1)$$

and whose within and concedes relations for a top level operation m are:

$$\begin{aligned} &R_m(i, k, u, w) = \\ &\quad (\exists\, v, j \bullet G(u, v) \wedge H(v, w) \wedge \\ &\qquad P_m(i, j, u, v) \wedge Q_m(j, k, v, w)) \end{aligned} \quad (5.2)$$

$$\begin{aligned} &E_m(u', w', o, q; i, k, u, w) = \\ &\quad (\exists\, v', p, v, j \bullet \\ &\qquad (G(u', v') \wedge D_m(v', w', p, q; j, k, v, w)) \vee \\ &\qquad (C_m(u', v', o, p; i, j, u, v) \wedge H(v', w')) \vee \\ &\qquad (C_m(u', v', o, p; i, j, u, v) \wedge \\ &\qquad\quad D_m(v', w', p, q; j, k, v, w))) \end{aligned} \quad (5.3)$$

We will now calculate these quantities for the two retrenchment routes from ***PHONE*** to ***PHONE***$_{CF/CH}$. In both cases we only need to check for the top level operations $connect_n$ and $break_n$ because the other operations at the intermediate level get filtered out of the composed retrenchment.

We start with the route via ***PHONE***$_{CF}$, getting a retrenchment that we label with *CF›CH*. Starting with the retrieve relation, we plug (3.14) and a suitably relabelled (4.5) into (5.1) and get:

$$K_{CF›CH}(u, w) = (u = calls \wedge w = (calls, fortab, holtab)) \quad (5.4)$$

Moving to $connect_n$ and the within relation, we likewise plug (3.14) and (3.15) and a suitably relabelled (4.5) and (4.6) into (5.2). Noting that as far as the use of the resulting relation in the operation PO is concerned, we can discard the term $K_{CF›CH}(u, w)$ which arises via (5.2) since both $P_{CF,connect_n}(i, j, u, v)$ and $Q_{CF›CH,connect_n}(j, k, v, w)$ are independent of the state variables and $K_{CF›CH}(u, w)$ is one of the PO antecedents anyway, we get:

$$R_{CF›CH,connect_n}(i, k, u, w) = (i = k) \tag{5.5}$$

Similarly we plug (3.14) and (3.16) and a suitably relabelled (4.5) and (4.7) into (5.3). After some simplification and further manipulation we get:

$$\begin{array}{ll}
 & E_{CF›CH,connect_n}(u', w', o, q; i, k, u, w) = (\mathrm{busy}(k) \wedge \\
[1] & \quad ((k \notin holtab \wedge fortab^+(k) = z \wedge \mathrm{free}(z) \wedge \\
 & \quad u' = u \wedge o = NO \wedge q = OK \wedge \\
 & \quad w' = (calls \cup \{n \mapsto z\}, fortab, holtab)) \vee \\
[2] & \quad (k \in holtab \wedge fortab^+(k) = z \wedge \mathrm{free}(z) \wedge \\
 & \quad u' = u \wedge o = NO \wedge q = OK \wedge \\
 & \quad w' = (calls \cup \{n \mapsto z\}, fortab, holtab)) \vee \\
[3] & \quad (k \in holtab \wedge \neg(fortab^+(k) = z \wedge \mathrm{free}(z)) \wedge \\
 & \quad u' = u \wedge w' = w \wedge o = NO \wedge \\
 & \quad q = (\text{“Our … hold.”})^{100}) \vee \\
[4] & \quad (k \in holtab \wedge fortab^+(k) = z \wedge \mathrm{free}(z) \wedge \\
 & \quad u' = calls \cup \{n \mapsto z\} \wedge w' = w \wedge \\
 & \quad o = OK \wedge q = (\text{“Our … janitor.”})) \vee \\
[5] & \quad (k \in holtab \wedge fortab^+(k) = z \wedge \mathrm{free}(z) \wedge \\
 & \quad u' = u \wedge w' = w \wedge o = NO \wedge \\
 & \quad q = (\text{“Our … janitor.”}))))
\end{array} \tag{5.6}$$

In deriving (5.6) we fully exploited the environment of antecedents of the PO, i.e. $i = j$, $j = k$ and the properties of G and H, removing the existential quantifications $\exists\ v', v, j$ via the one point rule. Also we identified intermediate level outputs with higher or lower level outputs as appropriate when G or H was involved, eliminating the $\exists\ p$ quantification too; this goes slightly beyond what is expressed in the generic operation PO because outputs are discussed only in the concedes clause[4].

Now disjuncts [1] and [2] of (5.6) come from $C \wedge H$ in (5.3), [2] being an artifact of the insensitivity of H to the means by which the state it is mapping was arrived at, i.e. it allows forwarding behaviour to survive when a subsequent design decision has overriden it; [3] and [4] come from $G \wedge D$ with [4] likewise being an artifact; and [5] comes from $C \wedge D$, one of the disjuncts from D generating false.

The other operation figuring in the retrenchment is $break_n$ for which we find, uninterestingly:

$$R_{CF›CH,break_n}(u, w) = \mathsf{true} \tag{5.7}$$

$$E_{CF›CH,break_n}(u', w'; u, w) = \mathsf{false} \tag{5.8}$$

4. To improve matters in this regard one can move to a more expressive if more complicated formulation of retrenchment eg. sharp retrenchment or its close relatives [9, 10].

Now we can turn our attention to the alternative route to $\boldsymbol{PHONE_{CF/CH}}$ via $\boldsymbol{PHONE_{CH}}$. Going through the same procedure we get a retrenchment labelled with $CH›CF$. The retrieve relation is as before:

$$\begin{array}{l} K_{CH›CF}(u, w) = (u = calls \wedge \\ \quad w = (calls, fortab, holtab)) \end{array} \tag{5.9}$$

Similarly, for $connect_n$, we obtain the within relation:

$$R_{CH›CF,connect_n}(i, k, u, w) = (i = k) \tag{5.10}$$

and to obtain the concedes relation, we manipulate (3.19) and (3.21) and a suitably relabelled (4.14) and (4.16) into:

$$\begin{array}{ll}
 & E_{CH›CF,connect_n}(u', w', o, q; i, k, u, w) = (\mathrm{busy}(k) \wedge \\
[1] & \quad ((k \in holtab \wedge \neg(fortab^+(k) = z \wedge \mathrm{free}(z)) \wedge \\
 & \quad u' = u \wedge w' = w \wedge o = NO \wedge \\
 & \quad q = (\text{“Our … hold.”})^{100}) \vee \\
[2] & \quad ((k \in holtab \wedge fortab^+(k) = z \wedge \mathrm{free}(z) \wedge \\
 & \quad u' = u \wedge w' = w \wedge o = NO \wedge \\
 & \quad q = (\text{“Our … hold.”})^{100}) \vee \\
[3] & \quad (k \in holtab \wedge fortab^+(k) = z \wedge \mathrm{free}(z) \wedge \\
 & \quad u' = u \wedge w' = w \wedge o = (\text{“Our … hold.”})^{100} \wedge \\
 & \quad q = (\text{“Our … janitor.”})) \vee \\
[4] & \quad (k \notin holtab \wedge fortab^+(k) = z \wedge \mathrm{free}(z) \wedge \\
 & \quad u' = u \wedge o = NO \wedge q = OK \wedge \\
 & \quad w' = (calls \cup \{n \mapsto z\}, fortab, holtab)) \vee \\
[5] & \quad (k \in holtab \wedge fortab^+(k) = z \wedge \mathrm{free}(z) \wedge \\
 & \quad u' = u \wedge w' = w \wedge o = NO \wedge \\
 & \quad q = (\text{“Our … janitor.”}))))
\end{array} \tag{5.11}$$

Using the same technical tricks as before, this time [2] and [3] are spurious; with [1], [4], [5] agreeing with [3], [1], [5] respectively of (5.6).

As expected, for $break_n$ we find:

$$R_{CH›CF,break_n}(u, w) = \mathsf{true} \tag{5.12}$$

$$E_{CH›CF,break_n}(u', w'; u, w) = \mathsf{false} \tag{5.13}$$

Now we can consider what the retrenchment would look like if we built both features into the plain ***PHONE*** model simultaneously. It is not hard to see that this retrenchment is given by:

$$\begin{array}{l} G_{CH/CF}(u, w) = (u = calls \wedge \\ \quad w = (calls, fortab, holtab)) \end{array} \tag{5.14}$$

and for $connect_n$ we get the within relation:

$$P_{CH/CF,connect_n}(i, k, u, w) = (i = k) \tag{5.15}$$

while for the concedes relation we need merely record the cases in which the simple ***PHONE*** model differs from the $\boldsymbol{PHONE_{CF/CH}}$ model, thus:

$$\begin{array}{l}
C_{CH/CF,connect_n}(u', w', o, q; i, k, u, w) = \\
\quad (\mathrm{busy}(k) \wedge u' = u \wedge o = NO \wedge \\
\qquad ((k \notin holtab \wedge fortab^+(k) = z \wedge \mathrm{free}(z) \wedge \\
\qquad q = OK \wedge \\
\qquad w' = (calls \cup \{n \mapsto z\}, fortab, holtab)) \vee
\end{array}$$

$$
\begin{aligned}
&(k \in \mathit{holtab} \wedge \neg(\mathit{fortab}^{+}(k) = z \wedge \mathrm{free}(z)) \wedge \\
&w' = w \wedge q = (\text{“Our … hold.”})^{100}) \vee \\
&(k \in \mathit{holtab} \wedge \mathit{fortab}^{+}(k) = z \wedge \mathrm{free}(z) \wedge \\
&w' = w \wedge \\
&q = (\text{“Our … janitor.”})))) \qquad (5.16)
\end{aligned}
$$

For break_n we find as usual:

$$P_{CH/CF,break_n}(u, w) = \mathsf{true} \qquad (5.17)$$

$$C_{CH/CF,break_n}(u', w'; u, w) = \mathsf{false} \qquad (5.18)$$

With these formulae in place, we are in a position to compare the various retrenchments we have derived. The only places in which they differ are the various concedes relations for the $\mathit{connect}_n$ operation. A little thought shows that $C_{CH/CF,connect_n}$ is a subrelation of both $E_{CF›CH,connect_n}$ and of $E_{CH›CF,connect_n}$; see Fig. 2.

$$
\begin{tikzcd}
& \boldsymbol{PHONE} \arrow[dl] \arrow[dd] \arrow[dr] & \\
\boldsymbol{PHONE_{CF}} \arrow[dr] \arrow[r, phantom, "\supseteq"] & {} & \boldsymbol{PHONE_{CH}} \arrow[dl] \arrow[l, phantom, "\subseteq"] \\
& \boldsymbol{PHONE_{CF/CH}} &
\end{tikzcd}
$$

Fig. 2

It is not hard to see why. The law of composition (5.3) is *inclusive*, in that all the behaviours described by the component concedes relations are effectively preserved and combined in all possible ways in the composed concedes relation. In some cases this yields more than we want, as we have remarked; still it always gives a safe upper bound.

The kind of composition of concedes clauses we have used is appropriate for a *descriptive* approach, in which it is the job of the concedes clauses to *describe* what the systems actually do. In a *prescriptive* approach, in which the concedes clauses must *dictate* what the systems ought to do (and what they ought not to do too), a semantically more incisive law of composition, where a later retrenchment can override the provisions of an earlier one in places where they disagree (just as happens in sequential composition of assignments), might be more appropriate. Much depends on the precise interpretation of the disjunction in (2.2). However the details of such a composition are beyond the scope of this paper.

6. Conclusions

Feature interaction in telephony has attracted a fair amount of attention in recent years, eg. [11, 12]. The burgeoning telecoms industry is always introducing new capabilities in to its systems, mainly because of the flexibility afforded by digitally programmed interconnection exchanges. However, even if a telecoms provider can make a rational reconciliation of all of the enhanced services that it provides itself, it is by no means clear that when one provider's network is interfaced to another provider's network, the results will be as either provider envisaged. This kind of thing has posed a challenge to development techniques (both formal and not so formal).

Amongst these efforts, refinement has been used to to address the problem, but the use of refinement in an area where previously established properties have to be overridden, is frequently an exercise in perversity. One has to search for a way of formulating the problem so that the contradictions inherent in a typical development step do not become exposed during the refinement process, the sophistication of the notion of refinement used notwithstanding. In contrast, the recording of the development decisions made via retrenchment would, we would claim, appear much more natural. However since the denial of previously established properties is fraught with danger if adopted in a development path, we emphasise that such use of retrenchment must be made in a completely transparent manner, and that it not be taken as some miraculous panacea the adherence to the formal structure of which, alone guarantees success. With such a proviso, retrenchment can help to both document and to justify the final design arrived at.

References

1. Wirth N. (1971); The Development of Programs by Stepwise Refinement. Comm. ACM **14**, 221-227.
2. Dijkstra E. W. (1972); Notes on Structured Programming. in: Structured Programming, Academic Press.
3. Hoare C. A. R. (1972); Proof of Correctness of Data representations. Acta Inf. **1**, 271-281.
4. de Roever W-P., Engelhardt K. (1998); Data Refinement: Model-Oriented Proof Methods and their Comparison. CUP.
5. Back R. J. R., von Wright J. (1998); Refinement Calculus, A Systematic Introduction. Springer.
6. Barroca L. M., McDermid J. A. (1992); Formal Methods: Use and Relevance for the Development of Safety-Critical Systems. Computer Journal **35**, 579-599.
7. Banach R., Poppleton M. (1998); Retrenchment: An Engineering Variation on Refinement. *in*: Proc. B-98, Bert (ed.), LNCS **1393**, 129-147, Springer. *See also*: Tech. Rep. UMCS-99-3-2, `http://www.cs.man.ac.uk/cstechrep`
8. Banach R., Poppleton M. (2000); Retrenchment, Refinement and Simulation. *in*: Proc. ZB-00, Bowen, Dunne, Galloway, King (eds.), LNCS **1878**, 304-323, Springer.
9. Banach R., Poppleton M. (1999); Sharp Retrenchment, Modulated Refinement and Simulation. Form. Asp. Comp. **11**, 498-540.
10. Banach R., Poppleton M. (2001); Engineering and Theoretical Underpinnings of Retrenchment. *to be submitted.* `http://www.cs.man.ac.uk/~banach/some.pubs/Retrench.Underpin.ps.gz`
11. Calder M., Magill E. (eds.) (2000); Feature Interactions in Telecommunications and Software Systems VI. IOS Press.
12. Kimbler K. (ed.) (1999); Feature Interactions in Telecommunications and Software Systems V. IOS Press.