A Deidealisation Semantics for KAOS

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

ABSTRACT

KAOS is a goal directed requirements engineering framework based on the decomposition and refinement of goals. Decomposition and refinement continue until a point is reached at which agents, identifiable in the application environment, can be assigned responsibility for operations that manipulate variables over which they have control, and where the information for determining changes in the controlled variables resides in variables which the agent can monitor. Although many of the 'refinements' that arise in the KAOS process can be viewed as acceptable according to one or other Model Based Refinement Formalism, many cannot. Those that cannot correspond to 'deidealisation' steps, not covered by conventional refinement formalisms. It is shown that such deidealisations can be seen as retrenchments, and the smooth interworking between refinement and retrenchment leads to a fuller formalisation of the KAOS process than is otherwise possible.

Categories and Subject Descriptors

D.2.1 [**Requirements/Specifications**]: Languages

Keywords

KAOS, ASM, Refinement, Retrenchment, Tower Pattern.

1. INTRODUCTION

KAOS [1, 2] is a goal directed requirements engineering framework, based on the decomposition and refinement of goals. The decomposition and refinement continue until a point is reached, at which agents, which can be identified in the application environment, can be assigned responsibility for manipulating variables over which they have control, the manipulation being based on values of variables which the agent can monitor. The manipulation is packaged into operations, these being of the same general kind as appear in typical Model Based Refinement Formalisms (MBRFs). As in many areas of requirements engineering, the word 'refinement' is used relatively liberally, referring not only to system description evolutions that would be described as refinements in a typical MBRF, but also to changes of system description that are too permissive to be so described. It is the aim of this paper, to show how the weaker criteria that apply to retrenchment, enable these 'not-really-refinements' to be integrated with the 'real' refinements, to give a more robust formal account of the KAOS process (and, by implication, similar RE approaches).

To say the preceding is not to imply that the KAOS process is without solid formal underpinnings — far from it. However, sprinkled here and there, in [1] for example, one finds comments that such-and-such an issue, . . . 'requires more research' or 'needs more detailed investigation'. Frequently, these issues involve some question of imprecision that arises because some goal is treated in different ways in different parts of the description — in an idealised (and thus too simplistic) manner in one place, and in a more realistic (and thus more complicated) manner elsewhere.

If we accept that the potential for a fully rigorous development is desirable (at least in principle), in that such approaches can ultimately yield systems of the highest quality, then the tension between these two views gives rise to a variety of technical responses. For example, many advocates of formal approaches would stress that the idealised approach ought to have no place in a formal description, saying that the top level spec. ought not to conceal any unavoidable complexities. But this is arguably somewhat naive; we cannot put it better than in [1]:

> "It is actually *desirable* to start from idealised goal definitions. The reason is simple: premature compromises of what is ideally required prevents the identification of further goals and the exploration of alternatives, such as alternative ways of deidealising goals, or to make trade-offs between conflicting goals. Therefore, one should be idealist when writing first-sketch goal definitions."

So there can be an 'idealistic' definition of the system, and also a 'realistic and accurate' definition. Much hinges on where in the system's-to-be modelling hierarchy the 'realistic and accurate' top level spec. is located. When it is buried too deeply in a forest of partial models addressing varying aspects of the system-to-be, then the connection between the realistic definition and the initially conceived goals can be noticeably weakened.

It is with issues such as these, regarding the strength of the formal connection between idealised goals and their more precise but more complex counterparts, that we are concerned in this paper. The beauty of the retrenchment approach (the core of our strategy), is that it allows the idealised and accurate viewpoints to be formally reconciled. Crucial to the enterprise is the smooth interworking between refinement and retrenchment, achieved via the theorems of the *Tower Pattern* [3, 4, 5], which allow both refinement and retrenchment to contribute to a solidly integrated whole.

Our approach is to formalise the operational layer of KAOS using ASM [6] and then to propagate the results to the rest of the KAOS method (insofar as space allows). We choose ASM for reasons of simplicity, and for the fact that ASM provide a concrete

ACM SAC 2010 Sierre, Switzerland

'existential witness' for formalisability. Thus, regarding the latter point, whereas the KAOS literature provides ample clues that a translation to a fully formal model based development process is perfectly feasible (see eg. (13)-(16) below), to this author's knowledge (at least), it is not completely clear that all the needed machinery is actually in place to the extent that one expects in a MBRF. With ASM there is no doubt regarding this point, and by now, many fully mechanically checked rigorous developments have been performed using ASM, in tandem with suitable theorem proving technology. Regarding the simplicity issue, unlike most MBRFs, which insist on executing a single rule or event at a time, ASM insist that '*all rules with true guards fire in the next step (and must be consistent)*', which neatly matches the KAOS convention that '*all operations with a true trigger fire in the next step (and must be consistent)*'. This sidesteps the need to do 'technical plumbing' round this issue.

The plan for the rest of this paper is as follows. In Section 2, we review refinement and retrenchment in the ASM formalism. In Section 3, we give a brief review of KAOS goals, goal refinement and goal operationalisation. In Section 4 we give the relatively straightforward mapping from KAOS operations to ASM rules. In Section 5, we look at obstacles and conflicts, and connect them to retrenchment. In Section 6, we take one of the main running examples from [1], the Mine Pump, and rework some fragments of it using the tools just developed. Section 7 generalises the discussion in order to benefit from the retrenchment *Tower Pattern* and Section 8 extends it to encompass system and requirements evolution. Section 9 concludes.

2. ASM REFINEMENT, RETRENCHMENT

In this section we briefly review what we need of ASM refinement and retrenchment. The standard reference for the ASM method is [6], building on the earlier [7]. In general, to prove an ASM refinement, one verifies so-called (m,n) diagrams, in which m abstract steps are in simulation with n concrete ones. The general policy was made rigorous (and proved formally in KIV [8]) in the work of Schellhorn [9, 10]. For this paper, for application to KAOS (as it is described in [1, 2]), we just need the $(1,1)$ case, which makes ASM refinement look pretty much like any other MBRF (aside from the firing policy noted above). It will be sufficient to focus on the following refinement proof obligations (POs):

$$\forall\ v' \bullet CInit(v') \Rightarrow (\exists\ u' \bullet AInit(u') \wedge R(u',v')) \tag{1}$$

$$\forall\ u,i,v,j,v',p \bullet R(u,v) \wedge In(i,j) \wedge COp(v,j,v',p) \Rightarrow (\exists\ u',o \bullet AOp(u,i,u',o) \wedge R(u',v') \wedge Out(o,p)) \tag{2}$$

In (1), the initialisation PO, it is demanded that for each concrete initial state v', there is an abstract initial state u' such that the retrieve or abstraction relation $R(u',v')$ holds. In (2), the correctness PO, it is demanded that when there is a concrete step $COp(v,j,v',p)$, performed by a concrete operation COp (with before-/after- states v,v' and input/output j,p) such that the retrieve and input relations $R(u,v) \wedge In(i,j)$ hold between concrete and abstract before-states and inputs, then an abstract step $AOp(u,i,u',o)$ can be found to re-establish the retrieve and output relations $R(u',v') \wedge Out(o,p)$. The ASM refinement policy also demands that non-termination be preserved from concrete to abstract, but we will not need that in this paper.

For retrenchment, [11, 12] give definitive accounts; latest developments are found in [13]. See also [14] for formulations of retrenchment adapted to several specific MBRFs including ASM. Like refinement, retrenchment is also characterised by POs: an initialisation PO identical to (1), and a 'correctness' PO which

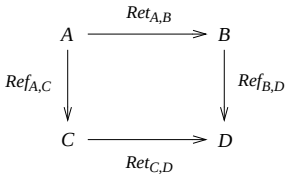

Figure 1: The *Tower Pattern* basic square, with refinements vertical, retrenchments horizontal.

weakens (2) by inserting *within*, *output* and *concedes* relations, W_{Op}, O_{Op}, C_{Op} respectively into (2), to give extra flexibility and expressivity. In particular, the concession C_{Op} weakens the conclusions of (2) disjunctively, giving room for many kinds of 'exceptional' behaviour. The result is:

$$\begin{aligned}\forall\ u,i,v,j,v',p \bullet\ & R(u,v) \wedge W_{Op}(i,j,u,v) \wedge COp(v,j,v',p) \Rightarrow \\ & (\exists\ u',o \bullet AOp(u,i,u',o) \wedge \\ & \quad ((R(u',v') \wedge O_{Op}(o,p,u',v',u,v,i,j)) \vee \\ & \quad C_{Op}(u',v',o,p,u,v,i,j)))\end{aligned} \tag{3}$$

To ensure that retrenchment only deals with well defined transitions, and to ensure smooth retrenchment/refinement interworking, we also insist that $R \wedge W_{Op}$ always fall in the domain of the requisite operations:

$$\begin{aligned}\forall\ u,v,i,j \bullet\ & R(u,v) \wedge W_{Op}(i,j,u,v) \Rightarrow \\ & \mathrm{dom}\, AOp(u,i) \wedge \mathrm{dom}\, COp(v,j)\end{aligned} \tag{4}$$

The smooth interworking between refinements and retrenchments is guaranteed by the *Tower Pattern*, the basic construction for which is shown in Fig. 1. There, refinements are vertical arrows and retrenchments are horizontal, and the two paths round the square from A to D (given by composing the refinement $Ref_{A,C}$ with the retrenchment $Ret_{C,D}$ on the one hand, and on the other, by composing the retrenchment $Ret_{A,B}$ with the refinement $Ref_{B,D}$) are compatible, in the sense that they each define a portion of a (potentially larger) retrenchment from A to D.

The *Tower Pattern* is typically applied when two adjacent sides of a square such as Fig. 1 are available, consisting of three system models with a retrenchment and refinement connecting them, and it is desired to 'complete the square' by building the missing system model, and the retrenchment and refinement that connect it to two of the given ones. The theorems of [4] (subsequently reformulated and revised in [3]) show that such square completions can be made in a generic manner, opening the way to the automatic construction of systems that are defined as (or are seen to naturally arise as) the solutions to such problems.

3. KAOS GOALS, GOAL REFINEMENT, OPERATIONALISATION

KAOS [1, 2] is a richly structured methodology for discovering and precisely defining the requirements of a system-to-be. Many of the 700 or so pages of [2] are deeply involved with strategies for determining the true system goals from a foundation of imperfectly articulated and often flawed user understanding — clearly, we cannot cover all these aspects in this paper.

For us, the starting concept is the system *goal*, howsoever arrived at, which following [1, 2], reduces formally to an expression in temporal logic (TL) in the style of [15]. The logic is over an $\mathbb{N}$-indexed time variable, and centres on a conventional definition of the truth of a temporal formula P at a given index i of a (total)

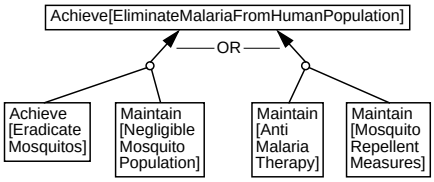

Figure 2: A simple informal KAOS AND/OR goal refinement.

history σ, featuring the usual future and past temporal operators, $\Diamond$, $\Box$, $\bigcirc$, $\blacklozenge$, $\blacksquare$, $\bullet$, parcelled up pairwise in (5)-(7):

$$(\sigma,i) \models \Diamond P \,,\, \blacklozenge P \;\; \textbf{iff} \;\; (\sigma,j) \models P \quad \text{for some } j \geq i \,,\, j \leq i \tag{5}$$

$$(\sigma,i) \models \Box P \,,\, \blacksquare P \;\; \textbf{iff} \;\; (\sigma,j) \models P \quad \text{for all } j \geq i \,,\, j \leq i \tag{6}$$

$$(\sigma,i) \models \bigcirc P \,,\, \bullet P \;\; \textbf{iff} \;\; (\sigma, i+1 \,,\, i-1) \models P \quad \text{, and } i > 0 \tag{7}$$

Beyond the basic temporal primitives, two kinds of generalisation are supported: (*a*) binary operators: *until* U, *unless-later* W, *since* S, *unless-earlier* B; (*b*) deadline-enhanced versions of the quantified primitives in (5), (6), namely: $\Diamond_{\leq d}$, $\Box_{\leq d}$, $\blacklozenge_{\leq d}$, $\blacksquare_{\leq d}$, where for simplicity we can take d to be a number of ticks (though more complex time metrics are obviously also possible). The deadline-enhanced operators restrict the time range within which the governed formula is required to be true to be no more than d ticks away from 'now', and extend to the temporal compounds in the obvious way. We also note the distinction between the 'pointwise' connectives, $P \rightarrow Q$ and $P \leftrightarrow Q$, and their 'strong', or '$\Box$' versions, $P \Rightarrow Q \equiv \Box(P \rightarrow Q)$ and $P \Leftrightarrow Q \equiv \Box(P \leftrightarrow Q)$.

Goals conform to specific *patterns* of temporal behaviour. KAOS stresses Achieve, Cease, Maintain, Avoid patterns, given by:

$$\textsf{Achieve}[Q] \quad P \Rightarrow \Diamond Q \qquad \textsf{Cease}[Q] \quad P \Rightarrow \Diamond \neg Q \tag{8}$$

$$\textsf{Maintain}[Q] \quad P \Rightarrow \Box Q \qquad \textsf{Avoid}[Q] \quad P \Rightarrow \Box \neg Q \tag{9}$$

where in each case, P is a suitable hypothesis. Goals are also further categorised as eg. *Satisfaction*, *Safety*, *Security*, *Information*, *Accuracy* goals, etc.

The crucial concept in KAOS for progress towards an implementable system is *goal refinement*. Goal refinement consists of growing a shallow (two level) AND/OR subtree under a goal, where each conjunction of goals under any of the disjuncts is itself capable of satisfying the original goal. Although the explorations of alternatives (via the OR) is crucial to the KAOS methodology, we will ignore it for the rest of this paper, since everything we will say will be equally applicable to each alternative. Fig. 2 shows a simple informal example concerning the elimination of malaria from an area infested by an (effectively) isolated, infected mosquito population.

Formally, a set of subgoals $G_1 \ldots G_n$ refines a goal G in the context of a(n application) domain theory *Dom* iff the following hold:

$$G_1 \ldots G_n, Dom \models G \qquad \text{(completeness)} \tag{10}$$

$$G_1 \ldots G_n, Dom \not\models \textsf{false} \qquad \text{(consistency)} \tag{11}$$

$$\forall\, i \bullet (\textstyle\bigwedge_{j \neq i} G_j, Dom \not\models G) \qquad \text{(minimality)} \tag{12}$$

Ultimately, the goals have to be *operationalised*, i.e. turned into collections of operations that can be performed by individual (software or environment) agents. The (unique) agent assigned responsibility to execute a given operation must be capable of monitoring the variables involved in the operation, and must also be capable of (and uniquely responsible for) controlling the variables updated by the operation. The sensitivity of KAOS to such matters of responsibility, and the need to connect operations to the goals that they operationalise, means that KAOS operations *Op* are specified by means of several predicates on states: the *domain preconditions* DomPre, the *domain postconditions* DomPost, the *required preconditions* $RPr \in$ ReqPre, the *required triggers* $RTr \in$ ReqTrig, and the *required postconditions* $RPo \in$ ReqPost. These predicates contribute to the sematics of an operation *Op* in the following manner.

Each operation *Op* is essentially a description of a set of state transitions of a similar kind to those appearing in MBRFs. It is given a temporal semantics via the formula $[|\, Op \,|]$ in (13), which just describes the raw updates to state variables accomplished by the operation at two adjacent ticks via DomPre and DomPost, without concern for *when* the operation occurs. The remaining predicates, *RPr*, *RTr*, *RPo*, are given a temporal semantics via the formulae $[|\, RPr \,|]$, $[|\, RTr \,|]$, $[|\, RPo \,|]$, defined in (14)-(16). (N. B. For (13)-(16), we favour the slightly simpler formulation in [2] over that in [1].)

The required preconditions $RPr \in$ ReqPre, act as guards; (14) defines $[|\, RPr \,|]$ to be the TL formula that says that the operation can take place only if the required precondition *RPr* holds. The required triggers $RTr \in$ ReqTrig, strengthen the required preconditions and *force* operation execution; (15) defines $[|\, RTr \,|]$ to be the TL formula that says that the operation must take place as soon as any required trigger holds. Finally, the required postconditions $RPo \in$ ReqPost, take care of any other conditions for the operation which are not already dealt with earlier; so (16) defines $[|\, RPo \,|]$ to be the TL formula that says that the operation can take place only if the required postconditions hold in the after-state. A history σ contains only valid occurrences of operation *Op* iff all occurrences of *Op* satisfy $[|\, Op \,|]$, and $[|\, RPr \,|]$, $[|\, RTr \,|]$, $[|\, RPo \,|]$ hold on σ. (N. B. $(\forall *)$ indicates quantification over all free variables.)

$$[|\, Op \,|] \equiv \text{DomPre}(Op) \wedge \bigcirc \text{DomPost}(Op) \tag{13}$$

$$\textbf{If } RPr \in \text{ReqPre}(Op) \textbf{ Then } [|\, RPr \,|] \equiv (\forall *)\, [|\, Op \,|] \Rightarrow RPr \tag{14}$$

$$\textbf{If } RTr \in \text{ReqTrig}(Op) \textbf{ Then}$$
$$[|\, RTr \,|] \equiv (\forall *)\, \text{DomPre}(Op) \wedge RTr \Rightarrow [|\, Op \,|] \tag{15}$$

$$\textbf{If } RPo \in \text{ReqPost}(Op) \textbf{ Then}$$
$$[|\, RPo \,|] \equiv (\forall *)\, [|\, Op \,|] \Rightarrow \bigcirc RPo \tag{16}$$

A collection of operations $Op_1 \ldots Op_n$ operationalise a goal G iff the following hold:

$$[|\, Op_1 \,|] \ldots [|\, Op_n \,|] \models G \qquad \text{(completeness)} \tag{17}$$

$$[|\, Op_1 \,|] \ldots [|\, Op_n \,|] \not\models \textsf{false} \qquad \text{(consistency)} \tag{18}$$

$$G \models [|\, Op_1 \,|] \ldots [|\, Op_n \,|] \qquad \text{(minimality)} \tag{19}$$

Note the similarities and differences between (10)-(12) and (17)-(19). There is no domain theory in (17)-(19), since at that level, the environment is itself assumed to be modelled by operations and invariants at an appropriate level of detail. Beyond that, (17)-(18) closely parallels (10)-(11). However (19) differs from (12) because, while goal refinement minimality merely requires that the refinement does not introduce superfluous goals, operationalisation requires that the behaviours of the goal are sufficient to infer the capabilities of the operations. Thus while a goal may be a *proper under*specification of its refinement, an operationalisation of a goal must be *equivalent* (in temporal terms) to the goal that it operationalises.

4. KAOS OPERATIONS AS ASM RULES

As mentioned before, the ASM framework is an MBRF which, unlike the majority of similar formalisms, conforms to the 'all rules with true guards fire immediately' paradigm. This makes it particularly easy to translate KAOS operationalisations into ASM rules. For all the operations in the operationalisation, we merely need to

conjoin the domain preconditions, required preconditions, and the disjunction of the required triggers to form the ASM guard, and to choose an after-value for the state that satisfies the domain and required postconditions, to derive a schematic translation as follows, where *vs* are the state variables of *Op*:

$$
\begin{array}{l}
Op = \\
\textbf{if } \mathrm{DomPre}(Op)(vs) \textbf{ and } \bigwedge_{RPr \in \mathrm{ReqPre}(Op)} RPr(vs) \\
\quad \textbf{and } \bigvee_{RTr \in \mathrm{ReqTrig}(Op)} RTr(vs) \\
\textbf{then} \\
\quad \textbf{choose } vs' \\
\quad \textbf{with } \mathrm{DomPost}(Op)(vs') \textbf{ and } \bigwedge_{RPo \in \mathrm{ReqPost}(Op)} RPo(vs') \\
\quad \textbf{do } vs := vs'
\end{array}
\tag{20}
$$

The above gives the *lazy* execution scheme for KAOS *Op*s: an *Op* does not execute unless one of its triggers is true. Removing the triggers disjunction from (20) gives the *eager* scheme: *Op* executes as soon as its preconditions are true. To allow for executions intermediate between eager and lazy, a (per-operation) flag can be introduced to override the falsehood of the triggers, together with a per-operation, always-enabled (or, for more sophisticated strategies, conditionally enabled), auxiliary rule that non-deterministically assigns it to true or false.

5. OBSTACLES, CONFLICTS AND RETRENCHMENTS IN KAOS

The process of investigating an application-to-be can generate inconsistency in a wide variety of ways. Amongst these we can find that:

• Identified goals may be inconsistent with the application's domain model when formalised (**Obstacles**).
• Domain experts can express views that, when formalised, give rise give to goals which are inconsistent (**Conflicts**).

In KAOS, obstacles and conflicts are characterised by a set of conditions of the form:

$$X, G_1 \ldots G_n, Dom \vDash \mathsf{false} \qquad \text{(conflict)} \tag{21}$$

$$\forall i \bullet (\textstyle\bigwedge_{j \neq i} X, G_j, Dom \nvDash \mathsf{false}) \qquad \text{(minimality)} \tag{22}$$

$$\exists (\sigma, i) \bullet (\sigma, i) \vDash X \qquad \text{(feasibility)} \tag{23}$$

The general picture in equations (21)-(23) is called a divergence, and describes the conflict between goals $G_1 \ldots G_n$ in the presence of X. If there is just one goal, then X is typically referred to as an *obstruction*, while if there are several goals, then X is typically referred to as a *boundary condition* for the conflict.

Suppose that there is an obstacle or a conflict. If the obstacle or boundary condition X involves entities envisaged as being controlled by the application-to-be, then its negation can be added as a new goal, and the elaboration of the application-to-be can continue on that basis. However, if X is not of this nature, (or, if it is considered injudicious to proceed in this way), then it is necessary to deidealise one or more goals in order to arrive at a description of the application-to-be that is satisfiable.

Assuming that a goal is of the form $H \Rightarrow T$, [1] discusses deidealising by: (a) weakening the goal, so that $H \Rightarrow T$ becomes $H \Rightarrow T \vee C$; or (b) strengthening the assumptions made, so that $H \Rightarrow T$ in effect becomes $W \wedge H \Rightarrow T$. In (a) and (b), C and W would be suitably related to the problem condition X. Of course we have paraphrased [1] (with C and W suitably related to X) so that the deidealisation process parallels the introduction of the concedes and within relations during retrenchment.

Although X is a temporal formula in principle, in many cases it reduces to a state formula, since, in a discrete transition system,

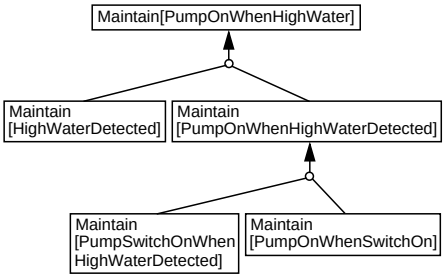

Figure 3: A fragment of the Mine Pump goal refinement.

the problem condition often reveals itself because the current state satisfies some predicate. In this simple situation, the connection between retrenchment, a primarily operation-focused technique, and KAOS, a primarily TL-focused technique, is particularly easy to make, and goes as follows.

As we stressed above, a KAOS operationalisation is equivalent to the TL goal formulation that it instantiates. Thus if retrenchment can express the properties of a relatively arbitrary change in a collection of operations via the retrenchment data and POs, and a collection of operations is equivalent to the TL formulation of to some KAOS goals, then retrenchment can likewise express the properties of a relatively arbitrary change in the KAOS goals. Note that this goes beyond merely saying *WHAT* the change is, it can help illuminate, via the POs and their proofs, *WHY* the change was a good idea, in the same way that a proof of any conclusion illuminates why that conclusion follows from the premises.

The argument just advanced was made in the context of X being a straightforward state formula, for simplicity. But there is no reason to suppose that it will not hold up in the face of more complicated X, simply because the behaviours permitted by an arbitrary temporal formula consist (in our $\mathbb{N}$-indexed model of time) of individual transitions, which will in an operationalisation, amount to the transitions of its operations. Each of these, in turn, can be endowed with its own retrenchment data, and the discharge of the associated POs contributes to the *WHY* of the goal collection morphism (see Section 8) being performed.

The last remarks briefly considered dealing with a multistep X via the fine-grained techniques of single step retrenchment. However there is no reason to restrict to that perspective, and a treatment of a multistep X via coarse-grained retrenchment techniques [16] is perfectly appropriate. The full theoretical consequences of such an observation are beyond the scope of a short paper like this one; they will be detailed elsewhere. We will content ourselves with illustrating the single step version on a familiar case study.

6. THE MINE PUMP IN KAOS AND ASM

We now illustrate the preceding ideas on a well known case study, originally presented in [17], and treated at length as a running example in [1], the Mine Pump. Here is the informal description, taken from [17]. Below, we restrict to a few relevant fragments of the treatment in [1], concentrating on some of the earlier features.

(I) Water percolating into a mine is collected in a sump to be pumped out of the mine. The water level sensors D and E detect when water is above a high and below a low level, respectively. A pump controller switches the pump on when the water reaches the high water level and off when it goes below the low water level. If, due to a failure of the pump, the water cannot be pumped out, the mine must be evacuated within one hour.

(II) The mine has other sensors (A, B, C) to monitor the carbon monoxide, methane and airflow levels. An alarm must be raised and the operator informed within one second of any of these levels becoming critical so that

the mine can be evacuated within one hour. To avoid the risk of explosion, the pump must be operated only when the methane level is below a critical level.

(III) Human operators can also control the operation of the pump, but within limits. An operator can switch the pump on or off if the water is between the low and high water levels. A special operator, the supervisor, can switch the pump on or off without this restriction. In all cases, the methane level must be below its critical level if the pump is to be operated. Readings from all sensors, and a record of the operation of the pump, must be logged for later analysis.

6.1 Accuracy of Measurement

Consider the goal refinement fragment shown in Fig. 3, and focus on the top goal Maintain[PumpOnWhenHighWater], which, hypothesising a suitable *DomainTheory*, is formalised in an ideal way as:

$$\mathsf{WaterLevl} \geq \mathsf{High} \Rightarrow \mathsf{PumpMotor} = \text{`On'} \tag{24}$$

But of course, the water detection mechanism is not infinitely precise, so a deidealised version of the goal might read:

$$\mathsf{WaterLevl} \geq \mathsf{High} + \mathsf{Dev} \Rightarrow \mathsf{PumpMotor} = \text{`On'} \tag{25}$$

where Dev describes a tolerable deviation between the actual water level and the pump controller's perception of it. In fact [1] explores the issue further, introducing a variable WaterMeasure corresponding to the level of water measured, and discusses various goals that relate the ideal of (24) to more realistic possibilities. In this paper, the relationship between (24) and (25) will be sufficient to make our point.

In (26) below, in ASM notation, we see idealised and realistic operationalisations (subsripted '$_I$' and '$_R$') of the switching on and the switching off of the pump motor.

$PumpOn_I =$
if $PumpMotor_I = Off$
  and $WaterLevl_I \geq High$
then $PumpMotor_I := On$
	$PumpOn_R =$
	if $PumpMotor_R = Off$
	  and $WaterLevl_R \geq High + Dev$
	then $PumpMotor_R := On$

$PumpOff_I =$
if $PumpMotor_I = On$
  and $WaterLevl_I \leq Low$
then $PumpMotor_I := Off$
	$PumpOff_R =$
	if $PumpMotor_R = On$
	  and $WaterLevl_R \leq Low - Dev$
	then $PumpMotor_R := Off$

(26)

In [1], in connection with the discussion of the nature of triggers, a distinction is drawn between the fact that if the water level gets high then the motor *must* come on, whereas once the water level ceases to be high then the motor *may* go off, leading to an absence of triggers and a different kind of operationalisation in the latter case. But taking note of the fact that once the water level becomes too low, then the motor *must* go off, whereas (with analogous thinking) once the water ceases to be too low then the motor *may* come on, leads to two versions each of the *PumpOn* and *PumpOff* operations, one triggered and one not triggered. If we disregard for now any consequences of paragraphs (II) and (III) of the problem description, keeping both versions of each operation would be wasteful, so in (26) we merged the untriggered and triggered versions, leading to a pure 'bang bang' control model. Such a merging is of course perfectly permissible in KAOS.

Turning to the relationship between the idealised and realistic models, it will not be a goal refinement in the normal KAOS sense. For suppose that the water level is somewhere between *High* and $High + Dev$. Then (24) would demand that the pump be on, while (25) would not, so the usual implicative relation, from the TL formula(e) expressing the refining goal(s), to the TL formula expressing the refined goal, would not hold. As regards the pure operationalisation, we might be able to 'get away with it' to an extent, if we posit a retrieve relation that is piecewise linearly bijective between the ideal and realistic water levels, matching up intervals $[0..Low]$, $[Low..High]$, $[High..\infty]$ of the ideal water level, respectively to intervals $[0..Low - Dev]$, $[Low - Dev..High + Dev]$, $[High + Dev..\infty]$ of the realistic water level. However, even if this were made to work technically, it would be highly detrimental to the wider requirements context — consider for instance what would happen to the laws representing the inflow of water into the mine in the two models (and the relationship between these formulations) if we adopted this correspondence between ideal and realistic water levels. So such a strategy is ruled out as a thoroughly bad idea — not all technically achievable refinements turn out to necessarily be good in reality.

Another aspect of the relationship between the ideal and realistic models, is the fact that the visible lack of accuracy in the correspondence between ideal and realistic triggering water levels, leads to a breakdown in synchronisation between the two models when we wish to set up an extended simulation between them. Thus, starting from the same initial conditions and assuming the same steady inflow of water into the mine, the realistic pump will come on slightly later (by δ secs. say) than the ideal one. The two will empty their sumps at the same rate, and then the realistic pump will go off 2δ secs. after the ideal one goes off, etc. Later, the ideal and realistic models will be completely out of phase, and even later, they will return to being in phase, the realistic pump having lost a whole cycle relative to the ideal pump.

A solution to both of these difficulties can be found in retrenchment. In (27) we present the retrenchment data for a simple retrenchment from $PumpOn_I$ to $PumpOn_R$, and also a similar one from $PumpOff_I$ to $PumpOff_R$, in an *ad hoc* ASM-like notation.[1]

retrenchment $PumpOn =$
retrieves
  $\lvert WaterLevl_I - WaterLevl_R \rvert \leq 2Dev$
within $WaterLevl_I = High$
  and $WaterLevl_R = High + Dev$
output true
concedes false
	retrenchment $PumpOff =$
	retrieves
	  $\lvert WaterLevl_I - WaterLevl_R \rvert \leq 2Dev$
	within $WaterLevl_I = Low$
	  and $WaterLevl_R = Low - Dev$
	output true
	concedes false

(27)

In (27) we see that the retrieve relation expresses a relatively 'honest' relationship between ideal and realistic water level values, on the satisfaction of which, we expect that something sensible might be said about the two pairs of operations (or indeed regarding other aspects of the Mine Pump problem). The within relations strengthen the retrieve relation (in an operation-specific way) to the precise form appropriate for the comparison of the two pairs of operations.

Retrenchments work on a per-operation (pair) basis, and the POs (3) and (4) do not insist on re-establishing in the after-state the same conditions that held in the before-state. This immediately avoids the problem of the failure of refinement and of extended simulation noted above. The notion that replaces (extended) simulation in the retrenchment context is *punctured simulation* (see [11, 18]). In a punctured simulation, the normal simulation criteria that ensure that the two parties in the simulation stay in step, can break down periodically due to the weaker constraints demanded of them. As a result, whereas the comparison between operations during conventional refinement implicitly assumes that we compare occurrences of them that are suitably 'in synch', the comparison between operations during retrenchment is unable to carry any such connotations without additional assumptions. Thus (27) allows a comparison of how the ideal and realistic operations work, without any presump-

[1] Obvious extensions of the notation can cater for cases where the operations in question have different names, etc.

tion of synchronisation.

Of course, the reason why the ideal and realistic systems both give a credible account of the operation of the pump, despite the breakdown of conventional simulation criteria, is that higher level invariants are at play. For instance either system satisfies the goal Maintain[WaterLevelNeverTooHighUnlessException], that we can formalise as:

$$P \Rightarrow \mathsf{WaterLevl} \leq \mathsf{High} + \mathsf{Dev} \tag{28}$$

where P excludes all the circumstances in which the water level could indeed get too high, such as those that follow from paragraphs (II) and (III) of the Mine Pump (and furthermore taking into account equipment failure modes etc.).

Both the ideal and realistic models refine (28) even though one is not a refinement of the other. That is not to say that either the ideal or realistic model should be discarded in favour of the other — as already observed in the Introduction, both can make a valuable contribution to requirements development. However, we can take advantage of retrenchment to tie them together formally.

The retrenchment treatment worked principally via the details of the operationalisations of the goals in question. This is in slight contrast with [1, 2] which promote working with the TL formulation of goals over working with the details of the operationalisations. Normally this is perfectly justifiable. However our present business is with *deidealisation*, which (in essence) deals with the breakdown of invariants. In a discrete time framework such as the KAOS one, if an invariant is true at one time, but is not true at some later time, it can only be because there was some specific moment at which the invariant failed, i.e. *some event* caused the breakdown of the invariant. In such situations, it is the present author's contention that a precise understanding of the event causing the invariant breakdown is just as important as the bigger picture obtained via the TL formulation.

6.2 Accuracy of Timing

The bottom left goal in Fig. 3, Maintain[PumpSwitchOnWhen HighWaterDetected], can be formalised in an ideal way as:

$$\mathsf{HighWater} = \text{`on'} \Rightarrow \mathsf{Switch} = \text{`on'} \tag{29}$$

In (29), HighWater is the pump controller high water signal and Switch is the pump on/off signal.

Of course the reaction of the pump switch to the controller is not instantaneous, so a deidealised version of the goal might read:[2]

$$\blacksquare_{\leq \mathsf{PumpDelay}}\ \mathsf{HighWater} = \text{`on'} \Rightarrow \mathsf{Switch} = \text{`on'} \tag{30}$$

As in the previous case, we have a failure of refinement, since, whenever the high water signal is on for less than *PumpDelay* secs., the realistic switch is not required to be on, whereas the ideal switch *is* required to be on.

To capture the dissonance between the ideal and realistic systems, we consider the operationalisations. It is clear from (29) and (30) that there is no difference at all between the corresponding operations in the two models (since the delay will be due the finite rate of working of the apparatus), so we bundle their descriptions as follows:

$$\begin{array}{|l} \mathit{SwitchOn}_{I/R} = \\ \textbf{if } \mathit{Switch}_{I/R} = \mathit{off} \textbf{ and } \mathit{HighWater}_{I/R} = \mathit{on} \\ \textbf{then } \mathit{Switch}_{I/R} := \mathit{on} \end{array} \tag{31}$$

[2] In [1], the formalisation corresponding to (30) has an additional $\bullet$ in the temporal quantifier, to cater for the fact that before- and after- states differ by one tick. We will ignore this here, though it is obviously easy to take it into account.

The interest in this deidealisation lies in how to describe the discrepancies in behaviour despite the identity of the operations — for this we allow the needed retrenchment to become coarse-grained.

In the conventional retrenchment of (1), (3), (4), the variables that are allowed to appear in the retrenchment data are spelled out explicitly — they are the variables that the operations explicitly touch. In coarse-grained retrenchment [16], we relax this and allow arbitrary TL formulae to appear in the retrenchment data. With such a proviso, we can draw up a retrenchment for this deidealisation as follows:

$$\begin{array}{|l} \textbf{retrenchment } \mathit{SwitchOn} = \\ \textbf{retrieves } \mathit{HighWater}_I = \mathit{HighWater}_R \textbf{ and } \mathit{Switch}_I = \mathit{Switch}_R \\ \textbf{within } \mathit{now}_R \geq \mathit{now}_I \textbf{ and } \mathit{now}_R \leq \mathit{now}_I + \mathit{PumpDelay} \\ \quad \textbf{and } \blacksquare_{\mathit{now}_R - \mathit{now}_I}\ \mathit{HighWater}_R = \mathit{on} \\ \textbf{output } \mathsf{true} \\ \textbf{concedes } \mathsf{false} \end{array} \tag{32}$$

As well as things that we have seen already, the retrenchment in (32) features the now_I and now_R time variables, to permit the capturing of information about the instants of time that the operations took place at (i.e. the before-ticks). For the purposes of illustration, we also allowed the retrenchment to encompass situations in which the realistic switch could react early if the command to switch the pump on arrived before the full *PumpDelay* period has elapsed, this being achieved via the pair of inequalities relating the now_I and now_R time variables.

The coarse-grained retrenchment approach not only allows more general statements to be made about corresponding pairs of *single* operations, as in (32), but also permits saying things about longer sequences of operations. To cater for possible interleavings of operations of interest with other independent activities taking place in the system as a whole, the approach in [16] captures the operation sequences using flow event structures [19].[3] These conveniently allow the description of a variety of incompatible outcomes resulting from the same starting conditions — as can easily result when one is contemplating non-trivial changes to system behaviour.

With this in mind, suppose we complemented our $\mathit{SwitchOn}_{I/R}$ operations with corresponding $\mathit{SwitchOff}_{I/R}$ operations, and considered an entire cycle consisiting of: switching the pump on, running the pump for a while (until a low water indication became true), and switching the pump off — and we called this sequence of events $\mathit{SwitchOnOffCycle}_{I/R}$. Then we could express the fact that the ideal and realistic cycles will take the same amount of time (since in the current deidealisation we are only looking at delays in pump switching on and off, and are insensitive to the accuracy issues of Section 6.1) using the following retrenchment of *SwitchOnOffCycle*:

$$\begin{array}{|l} \textbf{retrenchment } \mathit{SwitchOnOffCycle} = \\ \textbf{retrieves } \mathit{HighWater}_I = \mathit{HighWater}_R \textbf{ and } \mathit{Switch}_I = \mathit{Switch}_R \\ \textbf{within } \mathit{ini}_R \geq \mathit{ini}_I \textbf{ and } \mathit{ini}_R \leq \mathit{ini}_I + \mathit{PumpDelay} \\ \quad \textbf{and } \blacksquare_{\mathit{ini}_R - \mathit{ini}_I}\ \mathit{HighWater}_R = \mathit{on} \\ \textbf{output } \mathit{fin}_R \leq \mathit{fin}_I + \mathit{PumpDelay} \\ \textbf{concedes } \mathsf{false} \end{array} \tag{33}$$

In (33) $\mathit{ini}_{I/R}$ are the start ideal/realistic times of the entire cycle and $\mathit{fin}_{I/R}$ are the corresponding end times, and we assume that the delays for transmitting the commands to switch on and off are the same. Note that in (33) we are able to capture a global property of the entire on/off cycle in the output relation (the one that strengthens the retrieve relation in the after-state). In this example the realistic system does not deviate dramatically from the ideal one, so the concedes capability was not needed.

[3] More global aspects of behaviour can also be addressed using fluent temporal logic, eg. [20].

6.3 Goal Conflict

We now return to Fig. 3 and take note of some of the provisions of paragraph (II) of the Mine Pump. The remark about the critical methane level indicates that there is a conflict between the goal Maintain[PumpOnWhenHighWater] and one that could be expressed as Maintain[PumpOffWhenCriticalMethane] [1]. The conflict bites when the boundary condition:

$$\mathsf{WaterLevl} \geq \mathsf{High} \wedge \mathsf{MethaneLevl} \geq \mathsf{CriticalMethane} \qquad (34)$$

becomes true. A little further investigation shows that it is not the emptying of the water by the pumping mechanism that gives rise to the hazard in the presence of critical methane, but the sparks potentially originating in its electric motor.[4] Accordingly, at the operational level, the conflict is between turning the pump switch on when the high water signal becomes true, and keeping it off because the methane level is critical. This impels us to revisit the *SwitchOn* operation considered before. Here is the new realistic operation:

$$\begin{array}{l} \mathit{SwitchOn}_R = \\ \textbf{if } \mathit{Switch}_R = \mathit{off} \textbf{ and } \mathit{HighWater}_R = \mathit{on} \\ \textbf{then if } \mathit{HighMethane}_R = \mathit{off} \\ \qquad \textbf{then } \mathit{Switch}_R := \mathit{on} \textbf{ else } \mathit{MethaneAlarm} \end{array} \qquad (35)$$

In (35) we have an '**else** *MethaneAlarm*' option (not detailed further), since the controller will certainly not ignore the fact that the methane level has got high.

The discrepancy between the ideal and realistic *SwitchOn* is now greater than before, and the retrenchment relating them now has two options. Firstly, we can focus it on the parts of the behaviours that are common, by suitably restricting the within relation. This works technically, but has the demerit of excluding significant parts of the behaviours of the two systems from scrutiny by the retrenchment. Secondly, and better, we can include all the possible behaviours in the retrenchment relationship, and use the concession (which we can make mutually exclusive to the output relation to prevent confusion between safe and unsafe modes of operation) to highlight the contradictory parts in the two systems' behaviours.

$$\begin{array}{l} \textbf{retrenchment } \mathit{SwitchOn} = \\ \textbf{retrieves } \mathit{HighWater}_I = \mathit{HighWater}_R \textbf{ and } \mathit{Switch}_I = \mathit{Switch}_R \\ \textbf{within } \mathit{now}_R \geq \mathit{now}_I \textbf{ and } \mathit{now}_R \leq \mathit{now}_I + \mathit{PumpDelay} \\ \quad \textbf{and } \blacksquare_{\mathit{now}_R - \mathit{now}_I} \mathit{HighWater}_R = \mathit{on} \\ \textbf{output } \mathit{HighMethane}_R = \mathit{off} \\ \textbf{concedes } \mathit{HighMethane}_R = \mathit{on} \textbf{ and } \mathit{MethaneAlarm} \end{array} \qquad (36)$$

Note that we created (36) by adaptating (32) to take account of the the additional circumstances. However, (36) could also have been obtained by creating an independent retrenchment to deal with methane alone, and then composing it with the earlier one using conjunctive fusion composition [12].

7. PROPAGATING MODIFIED GOALS, AND THE TOWER PATTERN

In [1], the examples of deidealisation that we remodelled as retrenchments above, occur in the midst of the Mine Pump goal elaboration. As it then says in [1]: "*Once the goal is weakened, the transformation of the of the goal definition is propagated up and down along the goal refinement and operationalisation links.*" Since, regarding (just) the goal variables, all the goal refinements in [1] amount to inclusions of parent goal variable sets into offspring goal variable sets, this propagation is relatively straightforward.

[4]Hypothetically, one might operate the pump safely even while the methane was critical, if the electric motor was on the surface and the pump was operated using a suitable arrangement of drive belts and wheels.

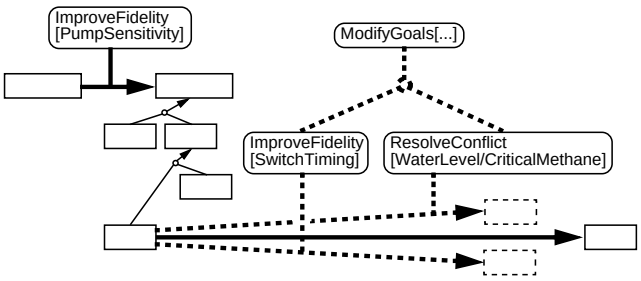

Figure 4: Goal morphisms describing goal deidealisations via retrenchments.

Viewed through retrenchment glasses, these propagations are instances of the *Tower Pattern*, Fig. 1, specifically of the pattern's *Lifting Theorem* and *Postjoin Theorem*. The former starts with the 'L' shape made by systems A, C and D, and completes the square by constructing B together with its relationships to A and D, and is useful for propagating changed goals upwards, towards ancestor goals. The latter starts with the 'Γ' shape made by systems A, B and C, and completes the square by constructing D together with its relationships to C and B, and is useful for propagating changed goals downwards, towards descendant goals or their operationalisations. The advantage of viewing the process as an instance of the *Tower Pattern* is that the latter can deal with correspondences between levels of abstraction mediated by relationships between variables that are more complicated than mere inclusions, giving the whole modelling process more options.

Thus suppose that at a high level of abstraction the most suitable variables are $u_1 \ldots u_n$. It may be that as goals are refined (not to mention retrenched), a more suitable family of variables to express the descendant goals is $v_1 \ldots v_m$. In a standard KAOS development, $v_1 \ldots v_m$ would have to be a superset of $u_1 \ldots u_n$ since standard KAOS goal refinement (as described in the literature) does not cater for non-trivial 'data' refinement during goal refinement. However, the use of the full capabilities of the Tower Pattern not only permits the use of data refinement during goal refinement (something that would admittedly not be hard to achieve provided one stuck to pure refinement and eschewed deidealisation in any form), but also allows the integration of this with the proper formal treatment of deidealisations via retrenchment.

8. GOAL / REQUIREMENTS / SYSTEM EVOLUTION — GOAL MORPHISMS

The KAOS methodology highlights the benefits of analysing the requirements of a system-to-be, by first identifying and decomposing the goals that the system should satisfy, a perspective that is frequently stressed in the generally excellent [2]. This amounts to focusing on the *WHY* before getting down to the *WHAT* and *HOW*.

Above, we showed in a few small examples how retrenchment could help explain (via its POs) the *WHY* of a goal deidealisation. But the mathematical basis of what we did is insensitive to the relatively small scale of the changes required in those small examples. The techniques used could address much bigger scale change-of-goals issues. In other words, one could reasonably extend the retrenchment perspective to the activity of *system requirements evolution in the large* too.

Interestingly, in [2], a work in which the reader is left in no doubt at all that the informal discussion has solid rigorous underpinnings (despite the fact that formality is downplayed somewhat until the

final chapters), the chapter on requirements evolution is the one that betrays the least connection with rigorous methods (a property it shares with all other writings on requirements evolution that the present author is acquainted with). The focus in the relevant chapter of [2] is on: *evolution types and causes*, *traceability and dependency*, and *change management*. What is missing (in the present author's opinion) is the *WHY* of the evolution, in other words the goal of the change of goal(s) — or to give it a more telling name, the goal of the *goal morphism*. A goal morphism would be the concept that drives the change from one set of goals to another, i.e. that changes the before-goals into the after-goals. Our thesis in this paper is that retrenchment can help address this issue.

In Fig. 4 we show what such an approach might contribute. The top left empty box is the original, ideal Maintain[PumpOnWhen HighWater] goal. A fat arrow, representing the retrenchment that implements its deidealisation, and labelled by the ImproveFidelity [PumpSensitivity] goal morphism, connects it to its more realistic counterpart, underneath which, the rest of the goal refinement shown in Fig. 3 has been reproduced using empty boxes, on the assumption that the ImproveFidelity[PumpSensitivity] deidealisation has been appropriately propagated to the refinement using the *Tower Pattern* Postjoin construction.

The bottom left empty box is the original bottom left goal of Fig. 3, the Maintain[PumpSwitchOnWhenHighWaterDetected] goal, deidealised as just described, but before the deidealisations of Sections 6.2 and 6.3 have been applied. The dashed fat arrows, labelled by the goal morphisms ImproveFidelity[SwitchTiming] and ResolveConflict[WaterLevel/CriticalMethane], represent the individual component retrenchments discussed briefly in Section 6.3. The fact that they can be combined by conjunctive fusion composition corresponds to the higher goal morphism ModifyGoals[...], and the long solid fat arrow represents the composed retrenchment, which deidealises the Maintain[PumpSwitchOnWhenHighWater Detected] goal to its realistic counterpart discussed in greater detail in Section 6.3.

With the approach just outlined, the goal elaboration process can take on something of a 'staircase' appearance, with genuine refinement steps going in a downward direction, and deidealisation and similar steps going horizontally. The tower constructions can be used to propagate the effects of decisions internal to the goal elaboration strategy up and down the abstraction hierarchy, ensuring consistency of all parts of the process.

9. CONCLUSIONS

In the preceding sections, we briefly overviewed key aspects of the KAOS requirements engineering methodology, and the ASM model based refinement formalism (the latter chosen mainly for the convenient fit of its firing policy to the KAOS one). We saw that KAOS operationalisations can readily map to ASM rules, and we brought in the ASM formulation of retrenchment to enable the phenomena inherent in goal deidealisation and conflict to be more precisely reasoned about, principally at the operationalisation level. This more precise reasoning can be viewed as an extension of the KAOS methodology via the concept of *goal morphism*, which retrenchment can be seen as giving a precise semantics to. The emphasis on operationalisations in connection with such phenomena (compared with the emphasis on the temporal formulation during refinement) was argued to be appropriate given that the breakdown of invariants will always be rooted in the properties of some specific events in a discrete time framework.

In this paper, we illustrated the the approach in the small, using a few examples of goal deidealisation taken from [1] for the Mine Pump case study. We saw that this generates a staircase shaped development route, which the *Tower Pattern* can fill out to propagate the effects of retrenchment-described deidealisations to higher and lower levels of abstraction. What we additionally claim in this paper, is that the approach extends to encompass the description of much more large scale system and requirements changes —i.e. system, or requirements *evolution*— since the mathematics utilised is insensitive to such size issues.

10. REFERENCES

[1] Letier, E.: Reasoning about Agents in Goal-Oriented Requirements Engineering. PhD thesis, Dépt. Ingénierie Informatique, Université Catholique de Louvain (2001)

[2] van Lamsweerde, A.: Requirements Engineering: From System Goals to UML Models to Software Specifications. Wiley (2009)

[3] Banach, R., Jeske, C.: Retrenchment and Refinement Interworking: the Tower Theorems. Submitted.

[4] Jeske, C.: Algebraic Integration of Retrenchment and Refinement. PhD thesis, University of Manchester (2005)

[5] Banach, R., Poppleton, M., Jeske, C., Stepney, S.: Retrenching the Purse: Finite Sequence Numbers, and the Tower Pattern. In: Proc. FM-06, LNCS. Volume 3582. (2005) 382–398

[6] Börger, E., Stärk, R.: Abstract State Machines. A Method for High Level System Design and Analysis. Springer (2003)

[7] Börger, E.: The ASM Refinement Method. FAC **15 (1-2)** (2003) 237–257

[8] The Karlsruhe Interactive Verifier. http://www.informatik.uni-augsburg.de/lehrstuehle/swt/se/kiv/.

[9] Schellhorn, G.: Verification of ASM Refinements Using Generalized Forward Simulation. J.UCS **7**(11) (2001) 952–979

[10] Schellhorn, G.: ASM Refinement and Generalizations of Forward Simulation in Data Refinement: A Comparison. Theor. Comp. Sci. **336** (2005) 403–435

[11] Banach, R., Poppleton, M., Jeske, C., Stepney, S.: Engineering and Theoretical Underpinnings of Retrenchment. Sci. Comp. Prog. **67** (2007) 301–329

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

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

[14] Banach, R.: Model Based Refinement and the Design of Retrenchments. Submitted.

[15] Koymans, R.: Specifying Message Passing and Time-Critical Systems with Temporal Logic. LNCS. Volume 651, Springer (1992)

[16] Banach, R.: Coarse Grained Retrenchment and the Mondex Denial of Service Attacks. In: Proc. IEEE TASE-09. (2009) 103–110

[17] Joseph, M.: Real-Time Systems: Specification, Verification and Analysis. Prentice-Hall Int. (1996)

[18] Banach, R., Poppleton, M.: Retrenchment and Punctured Simulation. In: Proc. IFM-99, Springer (1999) 457–476

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

[20] Letier, E., Kramer, J., Magee, J., Uchitel, S.: Fluent Temporal Logic for Discrete-Time Event-Based Models. In: Proc. ESEC/FSE-05. (2005)