Dormorbile: a Vehicle for Metacognition

JOHN SELF
Department of Computing, Lancaster University,
Lancaster LA1 4YR, England
(email: jas@comp.lancs.ac.uk)

Abstract: The purpose of this chapter is to develop a conceptual architecture which can be used as a common language in discussions of the role of student modelling in intelligent learning environments. The architecture attempts to integrate empirical, computational and theoretical aspects of the problem. The architecture, christened DORMORBILE (a DOmain, Reasoning, MOnitoring and Reflection Basis for Intelligent Learning Environments), distinguishes the various levels of agent knowledge. Preliminary applications and desirable extensions of the architecture are discussed.

Introduction and background

This work is based upon a number of premises and hypotheses (you may judge which are which) which should be stated up-front because we do not intend to discuss or justify them in any detail:

1. *That an 'intelligent learning environment' involves students engaging in some goal-directed, problem-solving activity which the system knows something about (thus excluding, for example, an encyclopedia browser).*
2. *That the ILE designer, from the fact that he has chosen to design a relatively open environment compared to a straightforward tutoring system, is emphasising the role of strategic skills rather than 'mere domain knowledge'.*
3. *That when using such ILEs students do not (always or often) engage in the kinds of strategic activity intended by the ILE designer.*
4. *That if an ILE is to make any kind of intervention to promote these activities it must itself in some sense know about them, so that it may monitor students' activities and comment appropriately upon them.*
5. *That an abstract discussion of such issues is unlikely to be effective.*
6. *That such discussions must be linked to the on-going problem-solving activities, through explicit, on the screen representations of the concepts being discussed.*
7. *That the learning of such strategic skills is central to the issue of transfer.*
8. *That the educational and psychological literature on metacognition is not very precise in its implications for ILE design.*

9. *That the work on meta-level architectures in artificial intelligence may provide us with frameworks to think with and maybe even to work with.*

There are words in the above list ('strategic', 'abstract', 'transfer', etc.) which have been contentious for a very long time. In this chapter we cannot realistically expect to resolve the arguments - but we could aim to provide a framework within which the arguments may be made more precise. In the following section we will outline some distinctions between the kinds of knowledge involved and then discuss some applications of the framework developed. In the rest of this introduction, we will briefly review work on 'meta-issues' in cognitive science, computer science and artificial intelligence.

There are many things an agent (student or computer - we will use "it" for an agent) may know in addition to specific facts such as '2+3=5' and 'if you multiply x by y the answer is bigger than both x and y' (as usual, we are using 'know' informally since what is 'known' may be incorrect), for example:

: one's own competence ("I can't do integration"; "I know nothing about art");
: that one's beliefs are inconsistent ("Clinton is a trustworthy adulterer");
: how to explain something to someone ("The way to integrate is .."");
: when to give up ("That equation looks much too complicated .."");
: how to improve performance ("Next time I won't use that substitution ..")

and so on. These issues have been looked at from many angles: a comprehensive literature review is not possible here but some useful distinctions can be made.

Metacognition

Campione, Brown and Ferrara (1982) distinguish between reflective access and multiple access. *Reflective access* refers to the ability to mention as well as use something, e.g. a piece of domain knowledge. (The unrevisionist use of terms like "piece of knowledge" is not meant to imply that knowledge literally has pieces 'in the agent's mind': it is simply a convenient way for us, as observers, to discuss such knowledge). Reflective access is thus concerned with the distinction between tacit, implicit knowledge used in solving problems and one's consciousness of that knowledge. However, mere 'mention' is not likely to satisfy us, as we can easily imagine students using a technical term to describe their problem-solving performance without knowing how the term actually relates to the performance. Let us say that an agent has reflective access to a piece of knowledge if it can access, describe and discuss that knowledge in a way that maps onto its actual use by that agent.

Multiple access refers to the ability to use the same thing (piece of knowledge) in more than one way. It is therefore related to the issue of transfer. It seems plausible that an agent with reflective access is more likely to have multiple access (for if it can bring the knowledge to consciousness it can deliberately consider the different uses to which it may be put). But reflective access is not a prerequisite for multiple access, for we can imagine that different situations somehow (unconsciously) trigger the same knowledge. The more general an item of knowledge is, the more amenable it (presumably) is to multiple access.

Metacognitive knowledge, that is, an agent's knowledge about its own knowledge, seems to involve some assessment of knowledge that may be reflectively or multiply accessed. The term is used to refer to the relatively stable and statable knowledge an agent has about its cognitive skills. For example, Flavell (1984) distinguishes three aspects: *Person variables*, e.g. knowledge that we are better at some problems than others, that memory is fallible, that to err is human, etc.; *Task variables*, e.g. knowledge that a certain task needs to be carried out very carefully (perhaps because it's dangerous or a correct answer is crucial); and *Strategy variables*,

e.g. knowledge that a problem should be read quickly to get an impression of how difficult it is likely to be.

The last of these is perhaps more a *metacognitive process*, that is, a process concerned with the way an agent solves problems rather than a process actually to solve problems. Many aspects are considered under this heading:

: problem recognition, e.g. identifying a problem as similar to an earlier one;
: plan generation, i.e. sketching out a sequence of operations before execution;
: resource allocation, e.g. deciding how much time to allocate to certain processes;
: the modification of cognitive processes, i.e. learning;
: consistency checking, e.g. of incoming or calculated data;
: the shared control between cognitive and metacognitive processes;
: the circumstances under which an agent switches to metacognitive activity;
: the monitoring of a solution process, e.g. asking if the goal is becoming nearer;
: evaluating the solution, e.g. by double-checking with a different method;
: evaluating the solution process, e.g. by commenting on time wasted up a blind alley;

and so on (again). Discussions of these processes are often muddled: terms are never precisely defined and the processes seem clearly to overlap in various ways. The inability to draw a precise boundary between cognitive and metacognitive processes seems to be a particular debate. This may be because metacognitive scientists have not recognised that the knowledge that provides reflective access to knowledge is itself subject to reflective access, or to put it another way the knowledge subjected to reflective access may be knowledge that provides reflective access to some other piece of knowledge. The knowledge may not, however, be partitionable into discrete layers, since a 'lower-level' item may refer to a 'higher-level' one. The important point is not the distinction between cognitive and metacognitive but the fact that the cognitive system, considered as a whole, needs to be *self-referential*.

Meta-logics

Goedel used a self-referential system to prove his famous theorem. Let $E(S)$ be an encoding of a statement S in some formalism. Goedel devised a way of encoding $E(S)$ in the same formalism. If S concerns the formalism itself, then we have a self-referential system. By considering self-referential sentences such as $S : \sim Prov(E(S))$ he was able to show various (negative) results, for example, that such a system is incomplete, cannot prove its own consistency, etc.

Meta-logicians have been struggling to come up with more positive results ever since. Technically, the inconsistencies of self-referential systems can be avoided by avoiding self-reference (not surprisingly) by separating an *object-theory* from a *meta-theory*, and having *reflection rules* which have premises and conclusions in different theories (Weyhrauch, 1980). A standard reflection rule is: $\vdash_O S \iff \vdash_M Prov(E(S))$, i.e. if S can be derived in the object-theory then $Prov(E(S))$ holds in the meta-theory, and vice-versa.

Prov is an example of a *reflection predicate*, i.e. a predicate which takes as argument an encoding of a statement in the object-theory. For *Prov* to be a first-order predicate (which would make things a lot easier than if it weren't) its argument must be a first-order term. This is usually achieved by letting $E(S)$ be some constant associated with S, or some ground term describing the structure of S.

Obviously, we can use the same techniques if we wish to regard the meta-theory as an object-theory and introduce a meta-meta-theory to contain statements about the encodings of meta-theory statements.

Meta-level architectures

A meta-level architecture defines a computational system in terms of an *object-system* and a *meta-system* (Maes and Nardi, 1988). The object-system contains a model for reasoning about the world to solve problems; the meta-system contains a representation of the object-system, which it reasons about. The object-system and its meta-system representation are *causally connected*, meaning that any change in one causes a change in the other. Thus they maintain *introspective integrity*.

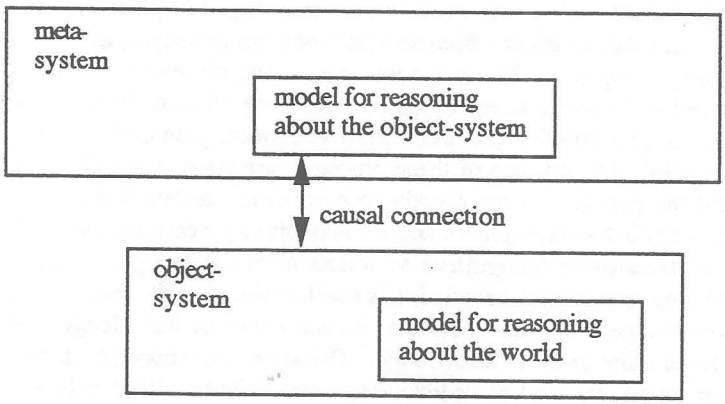

Figure 1. Object-system and meta-system

The purpose of such architectures has mainly been to permit control of object-system computations. In practice the architectures are often realised by means of *meta-circular interpreters*, which are explicit representations in the language itself of the interpreter of the language used to define the object-system and meta-system. Such systems are necessarily causally connected but the disadvantage is that the meta-system representation must be a complete and efficient representation of the operation of the system. Alternatively, in *declarative reflection* the causal connection is maintained by explicit specifications incorporated in the interpretation process.

Some systems carry out all inference in the object-system, having no separate meta-system interpreter and evaluating meta-predicates at fixed points in the computational cycle. Others perform all inference in the meta-system, simulating the object-system interpreter. And of course it is possible to have separate interpreters for the two systems.

Many classical AI systems can be re-described in terms of meta-level architectures, to no obvious purpose except to show that the basic idea (stripped of the jargon) is not particularly radical. For example, TIERESIAS had rules in the meta-system which ordered and pruned rules in the object-system. SOAR has three levels: a 'problem space level', a 'production level' that applies productions from long-term memory to objects in working memory, and a 'preference level' that selects between candidates or signals impasses, which are overcome by recursively applying the three levels to a new sub-goal.

Levels of knowledge

The following discussion is not meant to imply that all agents have or should have some representation in their memory of some or all of the things described, nor to imply that an ILE needs to address all these issues to be useful. We are not aiming for an all-encompassing 'theory of problem-solving, learning, etc.' but just need a framework for organising our thoughts about intelligent learning environments.

From a psychological, theoretical and computational perspective, there is a consensus that it is useful to distinguish an 'object level', which reasons about the problem, and a 'meta level', which reasons about the object level. Beyond that, however, almost everything is unresolved. Among the problems which are still a matter of debate are:

- How are the contents of the meta-level related to the contents of the object-level?
- How independent from the object-level can the meta-level be?
- Are the levels fundamentally separated or is one a part of the other?
- How is control passed between the object-level and meta-level?

In principle, two levels may be enough. Once the relation between the two levels has been sorted out, then clearly the same relation can apply to the meta-level regarded as an object-level and a new meta-meta-level. But there is no a priori reason to assume that the relation between all pairs of levels should be the same. Theoretically and computationally, however, further levels have not been considered much because it opens up the possibility of an infinite number of levels and it is not clear what practical benefit further levels might bring.

However, much of the confusion may have come from trying to pack too many different things into just two levels. The meta-level is supposed, for example, to control the execution of the object-level, to provide metaknowledge of the contents of the object-level, to permit reflection on the performance of the object-level, etc. We may be able to make some progress by disentangling these things. They do not all correspond to new 'levels' and there is no implication that such levels can be definitively separated but to begin with the discussion will be based on various 'levels of knowledge'.

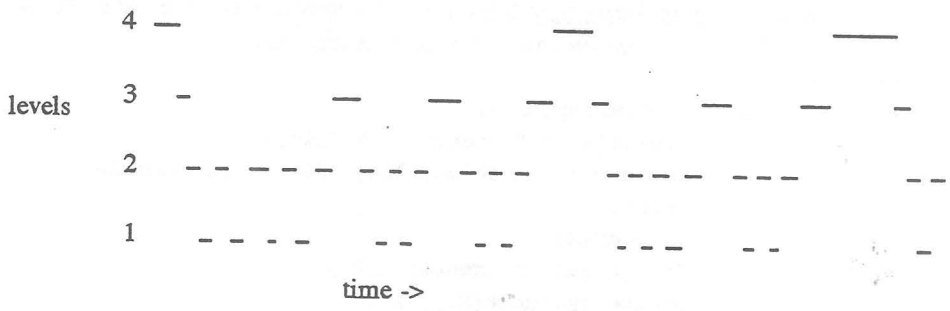

Figure 2. The levels and the time spent in them.

In Figure 2 the lines are supposed to indicate how control passed between the various levels during problem-solving. Most of the time levels 1 and 2 are proceeding merrily along applying operators to the problem space. Every so often level 3 intervenes to cause some major change in the direction of the problem solution. Even less often level 4 intervenes to consider the progress and fruits of problem-solving. In general, the higher levels take bigger chunks of time on fewer

occasions than lower levels. And in general, each level takes levels lower than itself as data and produces results for those lower levels.

Expressed in this general way it is obvious that the boundaries between levels are arbitrary. There is no hard and fast distinction between a level 3 'intervention' and a level 2 'operation' and similarly for the level 3 and level 4 boundary. Nonetheless, we will dignify the levels with names (Domain, Reasoning, Monitoring and Reflection) and proceed to indicate what might be in them. Where a particular item belongs depends on the expertise of the agent - so we will assume a typical agent (e.g. me). As we climb the levels, the discussion becomes more rarified, i.e. vague. It is one of our aims to develop concrete examples for the higher levels.

The Domain Level

We cannot sensibly discuss meta-anything without first saying what it is meta to. Our use of the term 'domain' as in "the domain of algebra" or "the domain of physics" is of course hopelessly imprecise. So we propose:

Def: A domain D is a set of predicates, functions and literals.

This set is supposed to provide the vocabulary for discussing or solving problems relevant to the domain D. The predicates, functions and literals are (some of) those that occur in propositions used by an agent to solve such problems. D(n) will be used to denote that *n* is the name associated with the domain D. For example,

```
D(astronomy) = {{ Planet, Meteor, Sun, Cold },
                 { mass, temperature, moons_of },
                 { earth, neptune, sun, halley's_comet }}
D(algebra) = {{ Quadratic, Integer, Fractional },
               { sum, product, expansion },
               { 2, 3, x }}
```

An agent possesses 'knowledge':

Def: An agent's knowledge K(a) is a set of propositions which it believes to be true.

Def: An agent's domain knowledge DK(a,D) with respect to a domain D is the subset of the propositions K(a) which contain an occurrence of an element of D.

For example, if

```
K(a) = {  Planet(neptune),
          Planet(x) & Distant(x) -> Cold(x),
          Less(temperature(neptune),100) -> Cold(neptune),
          Cold(2),
          Integer(2),
          Integer(x) -> Greater(x,0),
          Greater(product(x,y),x),
          President(clinton),
          Long_period_of_revolution(x) -> Distant(x),
          Planet(x) -> Orbits(x,sun),
          Orbits(x,y) -> Has_tides(x),
          Orbits(mars,sun) }
```

then

```
DK(a,astronomy) =
        {  Planet(neptune),
           Planet(x) & Distant(x) -> Cold(x),
           Less(temperature(neptune),100) -> Cold(neptune),
```

```
            Cold(2),
            Planet(x) -> Orbits(x,sun),
            Orbits(mars,sun) }
DK(a,algebra) =
        { Cold(2),
          Integer(2),
          Integer(x) -> Greater(x,0),
          Greater(product(x,y),x) }
```

This informal predicate logic notation is for our benefit and should not be taken to imply that agents possess anything analogous in a physical sense. In the case of computer agents we can so design them if we wish.

Domain knowledge is independent of any particular problem. Of course, K(a) also contains *problem-specific knowledge PSK(a)*, that is, propositions relating to the problem at hand (and its ongoing solution), such as

```
Goal(Solve('2x + (x+1)/2 = 5')),
Lhs('2x + (x+1)/2'),
Last_operator_applied(collect_terms,'x + (x+1)/2 + x = 5')
```

Def: An agent's background knowledge BK(a,D) with respect to a domain D = K(a) - DK(a,D) - PSK(a,D).

To solve a problem an agent may need recourse to propositions in BK(a,D). For example, if we have

```
Goal(Prove('Cold(neptune)'))
```

then it may need the proposition

```
Long_period_of_revolution(x) -> Distant(x)
```

which is in K(a) but not in DK(a,astronomy), as defined. In general, propositions in DK(a,D) may refer to predicates, functions and literals which are not in D but which are referred to in other propositions in K(a). It is of course debatable whether such propositions should be considered part of 'domain knowledge'. We will use the term 'domain knowledge' to refer to those propositions which explicitly refer to items in D and use 'implicit domain knowledge' to refer also to such indirectly referred to propositions.

Def: An agent's implicit domain knowledge IDK(a,D) with respect to a domain D = DK(a) + all other propositions in K(a) which contain an occurrence of predicate, function or literal which occurs in IDK(a,D).

For example,

```
IDK(a,astronomy) =
        DK(a,astronomy) ∪
        { Long_period_of_revolution(x) -> Distant(x),
          Orbits(x,y) -> Has_tides(x) }
```

The distinction between DK(a,D) and IDK(a,D) may be helpful in situations where an agent does not bring to bear knowledge that is not explicitly, obviously useful.

Def: An agent's domain knowledge is totally connected with respect to domain D if IDK(a,D) = K(a).

Def: A proposition p in K(a) is irrelevant to D for agent a if p is not a member of IDK(a,D).

For example, D(a,astronomy) is not totally connected since `'President(clinton)'` is not a member of IDK(a,astronomy) - that proposition is irrelevant to D(astronomy). No matter how long a chain of reasoning is carried out, the terms `'President'` and `'clinton'` never occur.

What is designated as a 'domain' is up to us. If we decide to include `'Lifeless'` in D(astronomy) or `'Factorable'` in D(algebra) then of course the corresponding D(a,astronomy)

and D(a,algebra) may change. In particular, if we choose to include any of the terms discussed below in D, as we are free to do, then by definition such terms become part of the domain and any proposition in K(a) referring to it part of domain knowledge. By adding such terms to D we are implying that associated propositions are now considered to be within the 'sphere of knowledge' understood to be referred to by a reference to that domain. Conversely, if we omit a term from D then we are suggesting that associated propositions are not germane to that domain or perhaps that the propositions are related to so many domains that we do not wish to link the term explicitly with this particular domain.

This arbitrariness in defining the boundaries of D is unavoidable and in any case desirable because what we designate to be in D must depend upon the agents concerned. Also of course agents change in their understanding of the 'domain' and hence what we want to consider 'domain knowledge' should change as well.

Most of the AI work on knowledge representation is concerned with the content and structure of K(a). Since domain knowledge is not the specific focus for ILEs there is no need to go into detail here. We should, however, distinguish between *active propositions* and *passive propositions* - the former being available for reasoning with, the latter not.

Def: An agent's active knowledge AK(a) is that subset of K(a) which is available to be reasoned with.

Def: An agent's passive knowledge PK(a) is K(a) - AK(a).

We might expect a predominance of elements of DK(a,D) and PSK(a) in AK(a). The transfer of propositions from a passive to an active state (and vice versa) is of course crucial but we will not discuss it - instead we will assume that all propositions referred to below are active.

The Reasoning Level

As defined, DK(a,D) is entirely declarative - it is simply a set of propositions, with nothing to indicate how the agent may use them to solve problems. The 'reasoning level' is concerned with this process.

Def: An agent's reasoning knowledge RK(a) is a set of 'reasoners' (a neutral term to avoid suggestive ones such as 'rules of inference' and 'operators'), where a reasoner is a function of K(a) which produces as its result a proposition to be included in K(a).

For example,

```
RK(a) = {  'Planet(neptune)' &
           'Planet(neptune) -> Orbits(neptune,sun)'
              => 'Orbits(neptune,sun)',
           'LHS('x + 2x')' &
           'Applicable_operator(collect_terms)'
              => 'LHS('3x')',
           Predicate1(literal1) &
           Predicate1(literal1) -> Predicate2(literal1,literal2)
              => Predicate2(literal1,literal2),
           'Orbits(neptune,sun)' &
           'Planet(neptune) -> Orbits(neptune,sun)'
              => 'Planet(neptune)' }
```

Consider the reasoner:

```
           'Planet(neptune)' &
           'Planet(neptune) -> Orbits(neptune,sun)'
              => 'Orbits(neptune,sun)'
```

Def: A concrete reasoner is a reasoner which refers explicitly to propositions, by quoting them.

Def: A domain-specific reasoner with respect to a domain D is a reasoner which refers explicitly to propositions which refer to elements in D.

The above reasoner is a concrete and domain-specific reasoner with respect to D(astronomy) since `'Planet(neptune)'` refers to `'Planet'`, which is a member of D(astronomy). However, its application is not entirely straightforward since the second proposition quoted in the reasoner is not in D(a,astronomy) but is an instantiation of an element of D(a,astronomy). Nonetheless, the reasoner is clearly intended to be a procedural reading of the two declarative propositions:

```
Planet(neptune)
Planet(x) -> Orbits(x,sun)
```

to add to K(a) the proposition

```
Orbits(neptune,sun).
```

We might now prefer to extend the definition of 'domain knowledge' to include domain-specific reasoners. We will not do so because such reasoners do not refer to elements of D but to propositions which refer to elements of D. This is a conceivably useful distinction, since it might enable us to distinguish a 'piece of knowledge' from the use to which it is put. In case we need them, we could define:

Def: An agent's domain reasoning knowledge DRK(a,D) with respect to a domain D is the subset of RK(a) which refers to propositions which refer to elements of D.

Def: An agent's background reasoning knowledge BRK(a,D) with respect to a domain D = RK(a) - DRK(a,D).

Def: An agent's implicit domain reasoning knowledge IDRK(a,D) with respect to a domain D = DRK(a) + all other reasoners in RK(a) which refer to propositons which refer to a predicate, function or literal which is referred to by a proposition referred to by a member of IDRK(a,D)..

If we now consider the reasoner:

```
Predicate1(literal1) &
Predicate1(literal1) -> Predicate2(literal1,literal2)
    => Predicate2(literal1,literal2)
```

then we see that it does not refer to propositions explicitly but only implicitly through the variables `Predicate1`, `literal1`, `Predicate2` and `literal2`, which are intended to be instantiated to specific predicates and literals (so that the resultant propositions match elements of K(a)) before the reasoner can be applied. For example, we might instantiate `Predicate1`, `literal1`, `Predicate2` and `literal2` to `Planet`, `neptune`, `Orbits` and `sun` respectively, in which case the reasoner adds `Orbits(neptune,sun)` to K(a). Or, we might instantiate `Predicate1`, `literal1`, `Predicate2` and `literal2` to `Integer`, `2`, `Greater` and `0` respectively, in which case the reasoner adds `Greater(2,0)` to K(a).

Def: An abstract reasoner is a reasoner which is not concrete, that is, it does not refer explicitly to propositions.

The above reasoner is an abstract reasoner. It could not be a domain-specific reasoner since it does not refer explicitly to any propositions. After suitable instantiation, it could be applied to any number of domains, as indicated above. Thus it could be said to be *domain-independent.*

Def: An agent's domain-independent or abstract reasoning knowledge ARK(a) is the set of reasoners in RK(a) which are abstract.

It is arguable that an agent which possesses an abstract reasoner rather than a series of instantiations of it is a more sophisticated problem-solver, showing a deeper understanding of the general processes of deriving conclusions from premises. On the other hand, of course, the

specific instantiations may be more efficient to apply. Acquiring the abstract reasoner might be a step towards 'transfer', the ability to apply skills acquired in one domain to a second domain.

The process of acquiring an abstract reasoner belongs not to the 'reasoning level' but to higher level (discussed below) but it is clear that there are 'degrees of abstraction'. For example,

```
'Planet(neptune)' &
'Planet(neptune)' -> Predicate('neptune',literal)
    => Predicate('neptune',literal)
```

is a concrete reasoner (according to our definition) since it refers explicitly to `'Planet(neptune)'`, although it also contains variables. Similarly, the abstract reasoner discussed above could be made more abstract:

```
Proposition1 &
Proposition1 -> Proposition2
    => Proposition2
```

One of the reasons for isolating and making explicit such reasoners is to give our ILEs the capability of discussing erroneous reasoners, such as

```
'Orbits(neptune,sun)' &
'Planet(neptune) -> Orbits(neptune,sun)'
    => 'Planet(neptune)'
```

The Monitoring Level

Just as the Reasoning level contains functions of its lower lever (the Domain level) which, when applied, produce results which change the lower level, so the Monitoring level contains functions of its lower levels (the Domain level and the Reasoning level) which, when applied, produce results which change its lower levels.

Def: An agent's monitoring knowledge MK(a) is a set of 'monitors', where a monitor is a function of K(a) and RK(a) which produces as its result a proposition or reasoner to be included in K(a) and RK(a) respectively.

RK(a) says what reasoning processes are possible, but it does not say which should actually be carried out. It does not say, for example, whether a domain-specific reasoner should be applied before an abstract reasoner or vice versa. Selecting a reasoner to actually apply is (one of) the function(s) of the Monitoring level.

The arguments of monitors are more varied than they are for reasoners, and hence there are various different kinds of monitor. The most straightforward perhaps are those which select a reasoner to apply (Warning!: we are here trying to specify what is normally left implicit, that is, the interpreter for something like RK(a): formal notations for this are virtually non-existent, so the following is only suggestive):

```
MK(a) = {   Applicable('Planet(x) -> Orbits(x,sun)')
                => Apply('Planet(x) -> Orbits(x,sun)'),
            Applicable(reasoner) & Simple(reasoner)
                => Apply(reasoner),
            Applicable(reasoner) & Complex(reasoner) &
            Expert(student)
                => Apply(reasoner),
            Applicable(reasoner) & Concrete(reasoner)
                => Apply(reasoner),
            Many_rules_applicable(RK(a),DK(a))
                => Reflect(!),
```

```
Equation(literal) & Complex(literal)
      => Abandon,
Equation(literal) & Previous_equation(literal)
      => Adandon,
LHS(literal) & Surprising(literal)
      => Abandon,
No_rules_applicable(RK(a),DK(a))
      => Abandon_all_hope }
```

Of course, this pseudo-formal notation raises many questions, such as what exactly are `'Apply'`, `'Abandon'`, etc. The general idea is that they are things which when 'executed' result in the application of a reasoner and hence a change in K(a). The extent to which we can (or need to) make these things more precise is unclear. In principle, all sorts of monitors could be included in MK(a). For example, any planning ideas the agent may have (e.g. the production of a provisional outline of how to solve a problem on the basis of the superficial characteristics of the problem) should be included in MK(a) - of course, in practice it is hard to specify such ideas precisely. Similarly, notions of 'resource allocation' (e.g. taking account of how much time there is available to solve a problem, knowing the 'cost' of applying the various reasoners) should be captured in MK(a).

As before, we can distinguish between concrete and abstract monitors.

Def: A concrete monitor is a monitor which refers explicitly to propositions or reasoners.

Def: An abstract monitor is a monitor which is not concrete.

Def: A domain-specific monitor with respect to a domain D is a monitor which refers explicitly to propositons or reasoners which refer to elements in D.

So,

```
Applicable('Planet(x) -> Orbits(x,sun)')
      => Apply('Planet(x) -> Orbits(x,sun)')
```

is a concrete, domain-specific monitor. Such a monitor might indicate that the quoted rule should be applied whenever it can be. The monitor:

```
Applicable(reasoner) & Simple(reasoner) => Apply(reasoner)
```

is an abstract monitor, presumably indicating that any simple applicable reasoner should be applied.

Intuitively, it seems that many of the items in MK(a) will be abstract, i.e. they will be general-purpose problem-solving skills. However, many domain-specific items seem to belong at this level, too, Consider, for example, the heuristic "Whenever I see an expression with brackets I multiply them out". This is a monitor, being concerned with the application of a reasoner (the reasoner itself being "If an expression has brackets then they may be multiplied out"). Again, as discussed above, we can debate whether this is 'domain knowledge' or not. Again, we prefer to regard it as a domain-specific monitor rather than domain knowledge since it refers to propositions which refer to elements of D not directly to elements of D itself. As before, we can define an agent's domain monitoring knowledge, background monitoring knowledge, and implicit domain monitoring knowledge, for example:

Def: An agent's domain monitoring knowledge DMK(a,D) with respect to a domain D is the subset of MK(a) which refers to propositions or reasoners which refer to elements of D.

There are ways in which MK(a) can alter the contents of K(a) and RK(a) other than by selecting a reasoner and applying it. Consider the statement "I know nothing about planets". This is, by our definition, a member of K(a), since it expresses a proposition which is believed. However, it is not a member of DK(a,astronomy), as defined, for although it mentions the concept of 'planet' it does not do so as a predicate - that is, the statement does not express a

proposition about planets but about my knowledge about planets. What is the status of such knowledge? Well, the result (the conclusion that I know nothing about planets) is, as discussed, in the Domain level but the process by which it is derived is not. The process seems to involve the application of a function which takes the set K(a) as argument. The definition of RK(a) above stated that a reasoner is 'a function of K(a)'. At that time this was interpreted to mean that a reasoner applied to individual elements of K(a), not to the set K(a) itself: each reasoner referred, directly or indirectly, to specific items in K(a). However, we could, if we wished, have made the reference to the set K(a) explicit, for example:

```
Member('Planet(neptune) -> Orbits(neptune,sun)',K(a))
```

Thus, we could venture a reasoner:

```
(There is no P) Member(P,K(a)) & Contains(P,'Planet')
    => Know_nothing_about(planets)
```

If we are happy with this, then this belongs in RK(a) (but note that, from the definitions, we may elevate any element of RK(a) to MK(a) if we for some reason (perhaps it takes more time to evaluate than we expect members of RK(a) to take) prefer to think of the item as a monitor rather than a reasoner). However, if we insist that a reasoner only consider individual propositions, not the set of propositions, then perhaps it belongs in the Monitoring level (although we'd have to adapt the definition of MK(a) to allow it there). Let's assume it belongs in RK(a) and define it:

Def: An agent's meta knowledge MTK(a) is the set of reasoners in RK(a) which apply to K(a) (i.e. the set K(a), not the individual items within it) and yield a result to add to K(a).

Def: An agent's meta domain knowledge MTDK(a) is the set of reasoners in RK(a) which apply to DK(a) (i.e. the set DK(a), not the individual items within it) and yield a result to add to K(a).

One particular member of MTK(a) is the reasoner which determines whether K(a) is inconsistent. Knowledge of how to use this fact to initiate a 'belief revision' process:

```
Inconsistent(K(a)) => Revise(K(a))
```

is a variation of the monitor which embarks on some remedial process when something 'surprising' is encountered, and so would appear to belong at the Monitoring level.

Consider now the statement "I don't know how to solve quadratic equations". This appears to be the result of applying a function to RK(a), the result itself being added to K(a). It is a function which does not apply to individual reasoners but to the set of reasoners. The result is clearly a member of K(a), from our definition, but is not domain knowledge (according to our definition) since it is not knowledge about D but knowledge about the reasoners that the agent has available for reasoning in D. The process of acquiring the result is a monitor if we interpret the phrase 'is a function of RK(a)' as allowing the set RK(a) as argument. Let us call this 'meta-reasoning knowledge'.

Def: An agent's meta reasoning knowledge MTRK(a) is the set of monitors in MK(a) which apply to RK(a) (i.e. the set RK(a), not the individual items within it) and yield a result to add to K(a).

There are also a number of learning mechanisms which we may consider to belong at the Monitoring level. For example, the following is a generalisation monitor:

```
Member(''Planet(neptune)' &
          'Planet(x) -> Orbits(x,sun)'
             => 'Orbits(neptune,sun)'',RK(a)) &
Member(''Integer(2)' &
          'Integer(x) -> Greater(x,0)'
             => 'Greater(2,0)'',RK(a))
```

```
=> 'Predicate1(literal1) &
        Predicate1(variable) -> Predicate2(variable,literal2)
            => Predicate2(literal1,literal2)'
```

The Reflection Level

Like the Reasoning and Monitoring levels, the Reflection level contains functions of its lower levels which, when applied, produce results which change its lower levels.

Def: An agent's reflective knowledge RefK(a) is a set of 'reflectors', where a reflector is a function of K(a), RK(a) and MK(a) which produces as its result a proposition, reasoner or monitor to be included in K(a), RK(a) or MK(a) respectively.

As we might expect, the discussions of the lower levels are echoed for each new level, although it gets progressively harder to give convincingly precise illustrations (at the top-most level, we are after all concerned with the purpose of life and other non-trivial matters).

So, again, we will recognise that MK(a) only says what monitoring processes are possible, not which should actually carried out - so RefK(a) will contain reflectors which select a monitor to apply. This passing of the buck must stop somewhere and for now we will stop at the Reflection level. (However, a 'reflective learner', as opposed to a 'reflective problem solver', will have a fifth level which considers the extent to which the learning mechanisms, principally in RefK, actually succeed, in general in improving problem-solving performance.) We will assert that the interpreter of RefK(a) will evaluate all applicable reflectors simultaneously. Most of the time a 'default reflector' will lead to execution passing to lower levels. Occasionally however the agent will lapse into multiple reflectors (a very worried agent). Lower levels will, most of the time, happily apply reasoners and monitors, setting 'triggers' ("I am confused", "Why am I doing this?", "I have finished", "That took much too long") which may be detected and acted upon by appropriate reflectors. The way in which control is shared among the levels cannot be pre-determined, since it must depend on the nature of the problem and indeed of the agent (some agents are more impulsive, others more reflective).

Again, we may distinguish concrete and abstract reflectors, although we may suspect that not much that is concrete will float up to such levels. Similarly, domain-specific and domain-independent reflectors may be defined, although the former may be uncommon.

We can imagine, even if we cannot easily specify, 'learning by reflection' processes which may, for example, involve the analysis of the performance of the monitoring level and the subsequent modification of MK(a) or RK(a) or DK(a).

We can also try to disentangle sentences such as:

"I know that I know nothing about planets"

"I know nothing about many things"

"I don't know how to tell whether I know nothing about planets"

The first two are the results of analysing the results of applying elements of MTK(a), the agent's meta knowledge. By our definitions, the results are in K(a) and the processes of obtaining the results are in MTK(a) - this suggests that we ought to distinguish the results of MTK(a) from other elements of K(a). The third sentence is the result of analysing the application of MTK(a), thus the process of obtaining this result may be considered an instance of meta meta knowledge.

However we will refrain from defining any more terms until it is clear that we need them and we can provide some convincing illustrations.

The DORMORBILE framework

The discussion above about the four or more levels applies only to a single agent. In order to adapt this to deal with student modelling we need to recognise that we have at least two agents to consider: the system and the student (there may of course be more than one student and also teachers involved). These two agents are building models of each other. In student modelling we generally consider only the model that the system builds of the student. So we can begin by building a three-column framework along the lines of Dillenbourg and Self (1992): in Figure 3, the right-hand column denotes the system agent, the left-hand column the student agent, with the middle column denoting the system's model of the student. This framework is called DORMORBILE[1], a DOmain, Reasoning, MOnitoring and Reflective Basis for Intelligent Learning Environments.

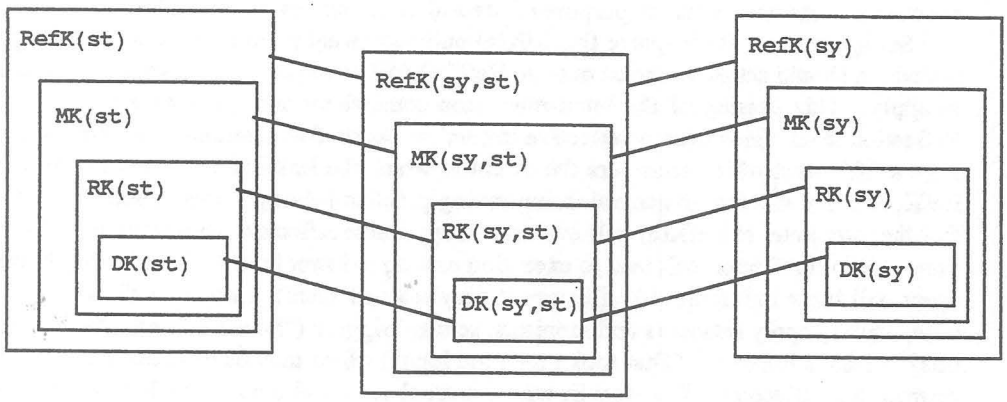

Figure 3. The DORMORBILE framework
(`DK(st)` denotes the student's domain knowledge; `DK(sy,st)` denotes the system's representation of the student's domain knowledge, etc.)

The idea now is that we try to discuss ILEs in terms of the components and transitions of this framework - as we did for standard student modelling in Dillenbourg and Self (1992). As we emphasised there, we are not implying that all research should fully and equally consider all the components - on the contrary, it is the case that different projects have different focusses and this framework should help clarify these.

As a brief example of how the framework may be used, we can consider the work of Bielaczyc, Pirolli and Brown (1993) to determine whether students could be taught to be better self-explainers. There is no system involved, so the right-hand column should be considered to denote the teacher. The experiment involved students 'explaining' to themselves someone else's solution attempt and a teacher overseeing this process and intervening when necessary to clarify what is meant by an explanation.

Since the student is not solving problems in the normal sense we are not concerned directly with DK(st), RK(st) and MK(st) but with the result of the application of these - thus in the student column, the main focus is on RefK(st), the processes that reflect on a problem solution. The teacher is also not directly solving problems, nor in fact reflecting on problem solutions, but is engaged in a fifth level, which oversees the student's reflective processes - and thus the focus in the column is on this new fifth level, OK(t), the teacher's 'overseeing knowledge'. The

teacher is not building a student model in the same explicit sense that a system might but there is clearly some implicit building of a student model, principally focussed on RefK(t,st). So this experiment is mainly concerned with the components indicated in bold in Figure 4.

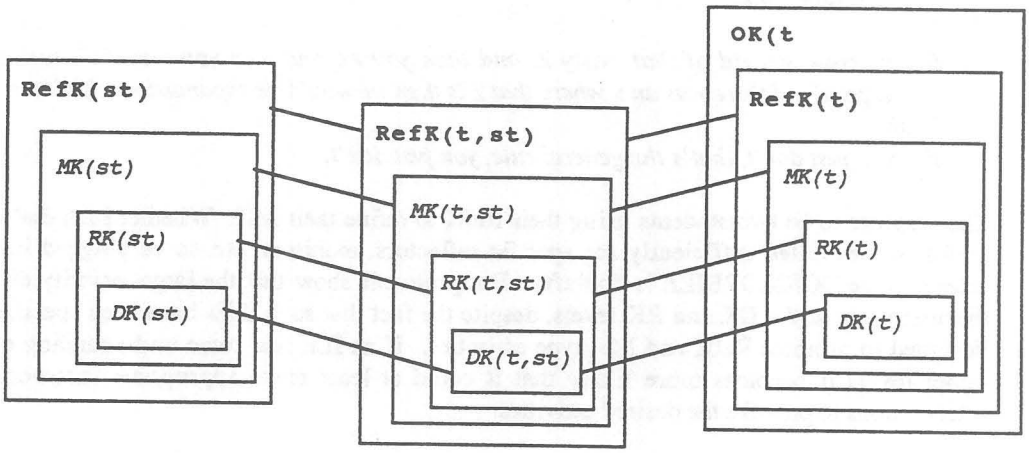

Figure 4. The self-explanation experiment.

Applications of DORMORBILE

DORMORBILE is a conceptual architecture: it may not be implementable in any significant sense. However it is through implementation that we will aim to clarify the contents of and interactions between the various components. This section briefly describes some preliminary efforts in this direction.

Propositional logic

The EPIC system (Twidale, 1991) enables students to prove theorems in propositional logic by specifying and annotating plans. The components of this system map fairly directly onto DORMORBILE: DK(a) is mainly problem-specific knowledge, since formal logic is by definition domain-free; RK(a) corresponds to the various rules of inference which can be applied to expressions; MK(a) corresponds to the various heuristics for selecting rules to apply or plans to adopt and the various annotative actions. A DORMORBILE interpreter can easily execute these components to prove theorems in this domain, and by including faulty reasoners and monitors observed in student protocols can give a reasonable trace of student problem-solving activity. However, propositional logic theorem proving, with its ideal problem-solving processes being close to the rationality implicit in DORMORBILE, is not much of a test of the generality of the architecture.

Algebra

Consider the following extract (Nichols, 1993) from two students contemplating their solution trace from an AlgebraLand-like system (Foss, 1987):

It looked quite easier that way .. than this way [right hand branch], this way obviously took more thinking for us.

Yeah. Cos as a general rule you get all .. the things you dislike on the right hand side first don't you.

Yeah.

In this case, get rid of that nasty 2, and then you expand, cos you wouldn't sort of expand .. if there was an x where that 2 is then we would've expanded wouldn't we.

Yeah.

But you just don't, that's the general rule, you just don't.

..

This appears to be two students using their RefK to refine their MK. Whether such dialogues could be unravelled sufficiently for specific reflectors, monitors, etc. to be plugged into an interpretable DORMORBILE is doubtful. But protocols show that the large majority of such discussions is at the DK and RK levels, despite the fact that such ILEs have been specifically designed to promote RefK and MK type activities. If an ILE had some understanding of the upper levels it becomes more likely that it could at least make appropriate instructional interventions to provoke the desired activities.

Physics

When pairs of students used a qualitative modelling system, it was found that much of their discussion was about how to 'tweak' the model to get the desired result and did not address the beliefs or conceptions on which the model was based (Byard et al, 1992). Thus self-reflection or reflective discussion between students may not be effective in changing beliefs. This requires dialogue with a teacher or ILE. For the latter to be possible, the ILE itself must have reflective knowledge which it can apply to student solution attempts. In a system being designed (SCILAB, Hartley and Ravenscroft, 1993), student comments are analysed for their consistency and completeness and for the plausible inferences which they entail, this analysis forming the basis for an interaction supported by dialogue games (Pilkington, Hartley, Hintze and Moore, 1992). Thus the system's reflective activities may form a model for the student to follow.

Music

Music students do not solve problems in the sense that mathematics and science students do and it is not obvious how DORMORBILE may be relevant to a music ILE. Imagine a student asked to "Discuss Beauvais' Play of Daniel". A satisfactory answer involves the interplay of various kinds of knowledge. The student may consider what is meant by "discuss" (RefK) - describe the plot, explain the musical style, contrast it with other music, etc. Having decided that he should, say, look for contrasts, he must consider which contrasts are important (MK) - secular/sacred, serious/comic, traditional/non-traditional, etc. Having chosen a contrast he must now apply operations to find examples of the contrast (RK), e.g. the use of non-liturgical inclusions in the text to illustrate the secular/sacred contrast. Applying the operations involves a consideration of the features of the specific text and other relevant knowledge (DK). This is a 'top-down' analysis but of course a student who does not know what to discuss may begin by looking at features to see what contrasts occur (just as propositional logic theorem provers often begin by randomly applying rules in the hope that something will turn up). The general point is that a successful 'solution' involves the integration of knowledge of different kinds corresponding approximately at least to the levels of DORMORBILE.

Extensions of DORMORBILE

Sketchy as the above applications of DORMORBILE are, they are sufficient to indicate that the framework is inadequate for addressing many important issues. These are mentioned briefly here to indicate that we are aware of some of the shortcomings, not to imply that we have any significant answers.

The sedimentation of knowledge

As soon as one attempts to specify reasoners, etc. precisely enough to include in an interpreted DORMORBILE it becomes clear that there is considerable latitude in not only defining the items but in deciding where to place them. For one student, an operator (e.g. 'integrate by parts') may be a single reasoner; for another, it might need to be represented by many reasoners. Thus there is a need for the language in which items are defined to evolve as agents learn - this is the 'language shift' process (Dillenbourg, 1992). The items themselves are in the same level; the process through which they change is in a level meta to it. There may be evolution not only within a level but between levels. For example, a monitor which might begin life as a 'rule' which is deliberately evaluated by an agent may be gradually refined (through meta-level processes) and 'compiled down' into a reasoner which is applied more automatically. As far as we aware, the technical difficulties of dealing with migration between the levels of a meta-level architecture have not been addressed.

The transitions between levels

The conditions under which an agent moves from one level to another (say, from reasoning to monitoring) are obviously crucial for those ILEs which claim to be intended to promote higher-level activities. Our studies show that algebra students spend almost all their time in the lower levels, as manifested by their solution trees, which tend to be long and narrow. Various interventions can be devised to avoid this ('Do you think this would be a better thing to do instead?, 'How far do you think you are from a solution?', etc.). This would be obtrusive of course unless an intelligent environment could determine from its own RefK and MK applied to the on-going solution when it would be opportune to intervene.

The horizontal dimension (agent interventions)

An 'intervention' corresponds to the semantics-free horizontal lines in Figure 3. In fact, the interventions do not have to be 'horizontal'. For example, an ILE might comment on an inconsistency (RefK) in the student's DK intended to activate some monitor in the student's MK; or an ILE might solve a problem (using DK and RK mainly) and ask a student to reflect on the solution process (RefK). Many different kinds of intervention are possible, intended to achieve different effects. We are currently analysing protocols to see what an *experimenter* says to understand or influence what a subject is doing.

The nesting of components

In previous formalisations (Dillenbourg and Self, 1992; Self, 1993) we allowed nested expressions but did not emphasise them, apart from DK(sy,st), etc., since we did not consider that they played a major role in ILE-student interactions. However, in the newer ILEs we are

moving away from systems which present themselves as all-knowing - indeed, in many cases, they deliberately lack knowledge in order to cause students to reflect on what is missing - and in such circumstances students will be much more inclined to consider what the system believes they believe, etc. Recently, for example, Dillenbourg (1993) has found examples of 'reciprocal modelling':

"He (the system) does not stick to the point. I don't like that."

"I think that, since I told him what I wanted and since I did not succeed in doing it before, I think that he will change it by himself."

"He supposes that I wanted the subjects to do something for 40 seconds. I wanted the subjects to do nothing."

"I am sure he will tell me again that I am wrong."

Aptitudes and attitudes

Many student models contain information beyond what the student is believed to know: they may, for example, describe the student along various global dimensions - impulsive-reflective, motivated-unmotivated, serialist-holist, beginner-expert, etc. These kinds of aptitudes and attitudes seem to be descriptive of components of the framework but not within the framework themselves. For example, impulsive-reflective is a property of MK. Its value is determined by RefK, although the value itself does not rightly belong in any particular level since it is not something believed or 'possessed' by the agent. The beginner-expert dimension is usually considered to be in terms of DK and RK, although naive-expert studies show that higher levels differ too. The 'motivation' factor, often considered crucial but never given a satisfactory definition, might be determined as a property of the goals of the top-most level and manifested by the degree of persistence implicit in lower levels.

Conclusions

The DORMORBILE framework distinguishes the different 'levels' of an agent's knowledge and considers the transitions and interactions between the levels of one agent and between those of two agents, specifically, a student and an intelligent learning environment. We have found the framework useful for analysing protocols of students solving problems in various domains, on paper or interacting with an ILE. Beyond serving an analytic function, the framework is also being used to develop system-initiated interventions to promote and support the kinds of monitoring and reflective activities which ILEs are usually intended to activate but often do not. Aspects of the framework are being implemented in order to clarify the contents and inter-relations of the various components.

Note:

1. DORMORBILE is named in honour of the vehicle body company, Dormobile, which went into liquidation in early 1994. Its main product, the Dormobile, embodied the British sense of freedom in the 1960s by realising the common ambition to be a cosy snail. Its initial 1951 model, essentially a van with seats which could be folded into beds (or a bed), evolved into a mobile bungalow which is a fondly remembered feature of British lay-bys.

References

Bielaczyc, K., Pirolli, P. and Brown, A.L. (1993). Strategy training in self-explanation and self-regulation strategies for learning computer programming, Technical Report CSM-5, UC Berkeley.

Byard. M. et al (1992). Conceptual change in science, Final Report to the ESRC, Computer Based Learning Project, University of Leeds.

Campione, J.C., Brown, A.L. and Ferrara, R.A. (1982). Mental retardation and intelligence, in J.S. Sternberg (ed.), *Handbook of Human Intelligence*, Cambridge: Cambridge University Press.

Dillenbourg, P. (1992). The language shift: a mechanism for triggering metacognitive activities, in M. Jones and P. Winne (eds.), *Adaptive Learning Environments*, Berlin: Springer.

Dillenbourg, P. (1993). Analysis of a protocol, Proceedings of SMILE workshop, Department of Computing, Lancaster University.

Dillenbourg, P. and Self, J.A. (1992). A framework for learner modelling, *Interactive Learning Environments*, 2, 111-137.

Flavell, J.H. (1984). Annhmen zum begriff metakognition, in F. Weinert and R. Kluwe (eds.), *Metakognition, Motivation und Lernen*, Stuttgart: Kohlhammer.

Foss, C.L. (1987). Learning from errors in AlgebraLand, Technical Report IRL 3, Institute for Research on Learning, Palo Alto.

Hartley, J.R. and Ravenscroft, A. (1993). Computer-aided reflection, Proceedings of SMILE workshop, Department of Computing, Lancaster University.

Maes, P. and Nardi, D., eds. (1988). *Meta-Level Architectures and Reflection*, Amsterdam: North-Holland.

Nichols, D. (1993). AlgebraLab protocols, Proceedings of SMILE workshop, Department of Computing, Lancaster University.

Pilkington, R.M., Hartley, J.R., Hintze, D. and Moore, D. (1993). Learning to argue and arguing to learn, *J. of Artificial Intelligence in Education*, 3, 275-295.

Self, J.A. (1994). Formal approaches to student modelling, to appear in G. McCalla and J. Greer (eds), *Student Modelling*, Berlin: Springer.

Twidale, M.B. (1991). Improving error diagnosis using intermediate representations, *Instructional Science*, 20, 359-387,

Weyhrauch, R. (1980). Prolegomena to a theory of mechanized formal reasoning, *Artificial Intelligence*, 13, 133-170.

Acknowledgements

This work has been carried out within the "Student modelling in intelligent learning environments" (SMILE) project funded by the UK Research Council's initiative on cognitive science and human-computer interaction. I am grateful to colleagues associated with the project: Roger Hartley, Pierre Dillenbourg, David Nichols, Michael Twidale, Michael Pengelly, Ana Maria Paiva, Andrew Ravenscroft and Rachel Pilkington.