Knowledge Maintenance: the State of the Art

Tim Menzies*

Artificial Intelligence Department,
School of Computer Science & Engineering,
The University of NSW, Sydney, Australia, 2052
timm@cse.unsw.edu.au
www.cse.unsw.edu.au/~timm

January 25, 1999

Abstract

In the software and knowledge engineering literature we can maintenance strategies offered to maintain seven main types of knowledge: *words*; *sentences*; *behavioral knowledge*; and meta-knowledge. Meta-knowledge divides into *problem solving methods*; *quality knowledge*; *fix knowledge*; *social knowledge*; and processing activities. There are five main ways in which these seven knowledge types are processed: *acquire*; *operationalise*; *fault*; *fix*; and *preserve*. We review systems that contribute to these $7 * 5 = 35$ types of knowledge maintenance to make the following conclusions. Firstly, open issues with the current maintenance research are identified. These include (a) areas that are not being addressed by any researcher; (b) the *recursive maintenance problem*; and (c) drawbacks with *rapid acquire systems* and the *operationalisation KM assumption*. Secondly, a process is described for commissioning a new maintenance tool. Thirdly, a general common principle for maintenance (search-space reflection) is isolated.

*The Knowledge Engineering Review, 1999, To appear

Contents

List of Figures

1 Introduction

A general trend in the twentieth century is an increasing level of doubt about the things we speak or write or try to enter into programs. Many factors have combined to reduce our belief that we can know the "truth" (in some absolute sense) about our world; for example: relativity, Heisenburg's uncertainty principle, the indeterminacy of quantum mechanics, Gödel's theorem [Hofstadter, 1980], the failure of AI to replicate human cognition via manipulation of symbols according to classical logic [Clancey, 1993], chaos theory [Glass & Mackey, 1988], and situated cognition [Menzies & Clancey, 1999]. Popper argues that all knowledge is a hypothesis since nothing can ever be ultimately proved: our currently believed ideas are merely those that have survive active attempts to refute them [Popper, 1963]. Knowledge representation theorists stress that knowledge bases (KBs) are approximate surrogates of reality [Davis *et al.*, 1993, Wielinga *et al.*, 1992a, Bradshaw *et al.*, 1991]; i.e. their accuracy is doubtful. A similar line is taken by Agnew, Ford & Hayes who say that "expert-knowledge is comprised of context-dependent, personally constructed, highly functional but fallible abstractions" [Agnew *et al.*, 1993]. In practice, we know that explicit records of domain knowledge may not evolve to some final stable point, even when the domain itself is stable. Compton reports one expert systems development in which there was always one further important addition, one more significant and essential change [Compton *et al.*, 1989]. Experiments in machine learning confirm that an intelligent agent generalising from experience can always use more experience to improve their specification. Catlett observers that these improvements may imply large-scale reorganisation to that specification [Catlett, 1991].

The premise of this article is as follows. If we doubt our KBs and routinely expect them to change significantly, then we must move the focus of knowledge engineering (KE) from knowledge acquisition (KA) to knowledge maintenance (KM). To facilitate this change, we offer here a review of the state-of-the-art in the emerging field of KM. Techniques from many different communities (e.g. software engineering, requirements modeling, the verification & validation community, case-based reasoning, machine learning, object-oriented databases) will be shown to all contribute to solving the KM problem.

We will say that current KA is focuses on *execution KBs* which acquire and operationalise *wordK* (e.g. atomic terms) and *sentenceK* (e.g. rules and ontologies) using problem solving methods (PSMs). However, we can see a growing body of research into *maintenance KBs* which contain:

- *behaviouralK* storing the known or desired behaviour of the KB;
- *socialK* representing the social context;
- *fixK* representing KB repair strategies;
- *qualityK* representing how the quality of the KB will be assessed.

These different knowledge types are shown in Figure 1. The maintenance KB is itself an execution KB which leads to the recursive maintenance problem (i.e. how do we maintain the maintenance KB? §2.5). Further, we can see in the literature five main strategies for processing these seven knowledge types: *acquire, operationalise, fault, fix* and *preserve*. These five processing strategies

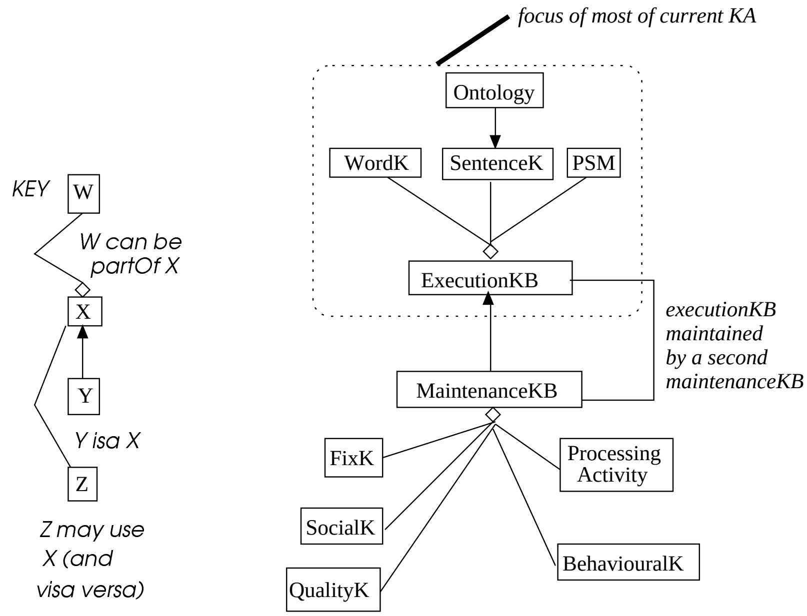

Figure 1: Maintenance KBs are a special kind of execution KBs which access more types of knowledge. Much of modern KA is focused only on execution KBs

Knowledge types (Figure 4)	Knowledge Processing Activities (Figure 23)				
	1.Acquire	2.Operationalise	3.Fault	4.Fix	5.Preserve
1.WordK	All systems	methods in an OO system	repertory grids (Fig 19)	RDF (Fig 30)	RDF (Fig 29)
2.SentenceK (includes ontologies)	MYCIN (Figs 5,6) IO-SCHEMAS (Figs 24,25)	MYCIN (Figs 5,6)	verification tools (Figs 17,18);	CBR (Figs 8 to 32)	RDR (Figure 30)
3.BehaviouralK	UML (Fig 7)	test suites (§2.3.4)	TCAT (§2.3.4)	Zlatereva (§2.3.4)	-
4.PSMs	KADS (Fig 14) (Figure 15)	TINA (Figs 27,28)	Fensel (§3.3.2)	RD-RA (§3.4.2)	RD-RA (§3.4.2), KA scripts (Fig 33)
5.QualityK (includes inconsistencyK)	QARCC (Figs 16,34)	QMOD/HT4 (Figs 9 to 12) CSMs (Figure 20)	-	-	-
6.FixK	SEEK (Fig 21)	machine learning; (Fig 31)	-	-	-
7.SocialK	REMAP (Fig 22)	-	-	-	-

Figure 2: Some example systems that cover the 35 points in the *knowledge management options space.*

CAKE	computer-aided knowledge engineering	CASE	computer-aided software engineering
CBR	case-based reasoning	CSMs	critical success metrics
DSE	data-schema evolution	EBG	explanation-based generalisation
ECS	expert critiquing system	ETM	EXPECT TRANSACTION MANAGER
ILP	inductive logic programming	K	knowledge
KA	knowledge acquisition	KB	knowledge base
KL	knowledge-level modeling	KL-A	KL using a single PSM
KL-B	KL using libraries of PSMs	KM	knowledge maintenance
ML	machine learning	NFR	non-functional requirement
OO	Object-oriented	P	a proof generated by HT4
PSB	Preece, Shinghal, Batarekh	PSCM	problem-space computational model
PSM	problem solving method	RAS	rapid acquire system
RD-RA	ripple-down rationality	RDF	ripple-down functions
RDR	ripple-down rules	RM	requirements modeling
SBF	SPARK/BURN/ FIREFIGHTER	SC	situated cognition
W	a world generated by HT4		

Figure 3: Abbreviations used in this paper

and seven knowledge types define the 35 points of a *knowledge management options space* (see Figure 2).

Due to the fragmented and diverse nature of the KM field, this article does not aim to cover every KM system. Rather, we will mention systems that illustrate portions of the maintenance strategy option space[1]. Many of the systems reviewed here contribute to multiple places within the maintenance strategy option space. Hence, their descriptions may be broken up into different sections. This article will use the term KM to denote some maintenance strategy for processing some combination of our seven knowledge types. Abbreviations used in the paper are described in Figure 3.

The structure of this article is as follows:

- In §2, we explore the seven knowledge types
- In (§3), we review the five processing activities on those types.
- In (§4), we draw the following conclusions. Firstly, a process called *search-search reflection* is central to many KM tools. Search-space reflection will be shown to be a simple extension of Newell's original knowledge-level vision [Newell, 1982, Newell, 1993] (§2.4). Secondly, current knowledge maintenance research is incomplete.

We will see several indicators that current KM research is incomplete. For example, over a third of the 35 points in the knowledge maintenance space are not explored in the literature. Also, many KM systems are *rapid acquire systems* that assume that if knowledge is expressed at a sufficiently high-level, then its flaws are obvious and quick to change. We offer examples below where this is clearly not the case (§3.1.1). Further, some research makes the *operationalisation KM assumption*; i.e. if we can watch a program execute, then we can understand it. We will argue that mere program watching is hardly a complete

[1] KM researchers who feel their system is not accurately reflected in this survey are encouraged to contact the author. It is planned to update this document on a regular basis.

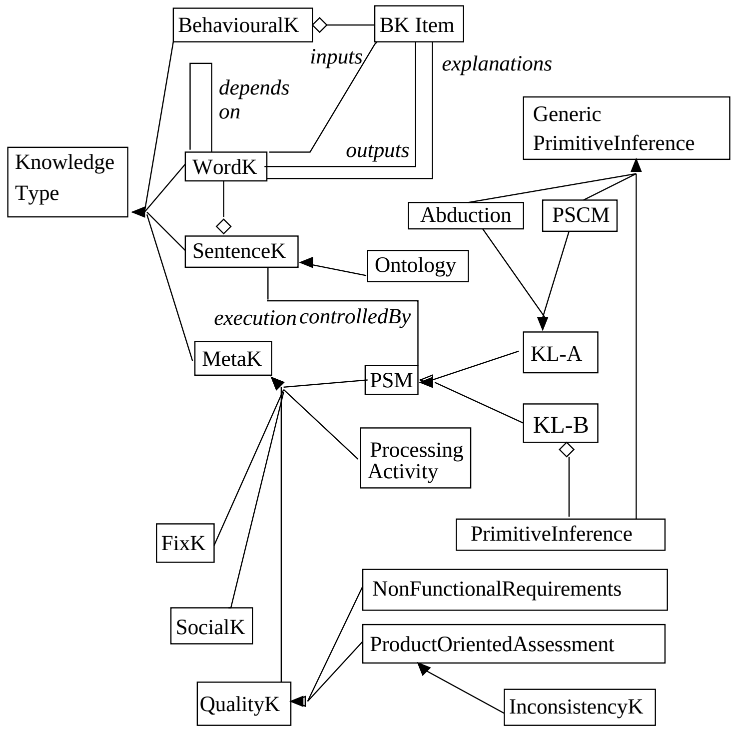

Figure 4: Different knowledge types (in the notation of Figure 1).

KM strategy (§2.3.5). Lastly, most KM research ignores the *recursive maintenance problem* (§2.5). That is, if we maintain KB1 using knowledge stored in some KB2, do we then then need some KB3 to maintain KB2 and a KB4 to maintain KB3 and... The missing third of the KM space are all a result of the recursive maintenance problem. To address this incompleteness in current KM research, our final section discusses how a KM tool should be commissioned.

2 Knowledge Types

Figure 4 summarises our seven knowledge types. KBs contain:

- Words (§2.1) collected into sentences (§2.2). Words may depend on other words, thus forming a dependency network; e.g. some tests may be a precondition for some result or rule left-hand-side words must be true before we can infer rule right-hand-side words. An important class of sentences are *ontologies* [Gruber, 1993, Gomez-Perez, 1996, Neches *et al.*, 1991].

- Known, desired, or past behaviour of a KB is stored in as behaviouralK (§2.3). Members of this library may be:
  - Just KB inputs (e.g. test case generation, §2.3.4);
  - Or input/output pairs (e.g. QMOD/HT4,§2.3.3);
  - Or an "explanation": portions of the dependency network which connects inputs to outputs (e.g. CBR, §2.3.2).

```
if     the infection is meningitis and
       the infection is bacterial and
       the patient has undergone surgery and
       the patient has undergone neurosurgery and
       the neurosurgery-time was < 2 months ago and
       the patient received a ventricular-urethral-shunt
then   infection = eColi (.8) or klebsiella (.75)
```

Figure 5: Rules are sentenceK containing wordK (wordK is underlined)

- In the knowledge-level modeling (KL) paradigm, KB execution is controlled by problem solving methods (PSMs) (§2.4.1). A single PSM is a set of generic primitive inferences. We distinguish two kinds of KL modeling: KL-A and KL-B [Menzies, 1995a]. In KL-A, there is only one PSM which applies preference operator selection control over a traversal of the dependency network (e.g. abduction or the problem-space computational model: PSCM [Yost & Newell, 1989]). KL-B uses libraries containing more than one PSM.
- PSMs are one kind of meta-knowledge. Other kinds of meta-knowledge include:
  - Quality knowledge which assesses a KB via testing for inconsistency (using inconsistencyK); testing the non-functional requirements; or performing product-oriented assessment (§2.4.2);
  - Knowledge of how to fix a broken KB (§2.4.3);
  - Social knowledge describing the agents that interact around the KB (§2.4.4);
  - Maintenance processing activities (§3).
  - Technically, ontologies is also meta knowledge about words. However, they are also sentenceK and so, to avoid an overly-complex classification scheme, we will discuss ontologies under the heading of sentenceK and not meta-knowledge.

All seven types (wordK, sentenceK, behaviouralK, PSMs, qualityK, fixK, socialK) are heuristic knowledge. That is, they are subjective and (potentially) will require change over the lifetime of a KB.

2.1 WordK: Word Knowledge

At the lowest level, a KB connects simple words. When generating explanations, words are the concepts which cannot be decomposed into other concepts. For example, consider the MYCIN rule discussed by Clancey (Figure 5) [Buchanan & Shortliffe, 1984,Clancey, 1992]. Words in this rule are underlines and include `meningitis`, `surgery`, `eColi`, etc. In a logic program, words are the ground terms. In an object-oriented (OO) system, a word could be a call to a method in an object. In a functional system or OO, a word would be a call to some function, perhaps with a simple comparisons (e.g. `age(patient)=old`).
The meaning of words must be maintained since the rest of the knowledge base is a construction that connects these terms. Given a dependency network representing the inference flow in a system, the words would be the vertices

```
subtype(                meningitis,          bacteriaMenigitis       ).
subtype(                bacteriaMenigitis,   eColi                   ).
subtype(                bacteriaMenigitis,   klebsiella              ).
subsumes(               surgery,             neurosurgery            ).
subsumes(               neurosurgery,        recentNeurosurgery      ).
subsumes(               recentNeurosurgery,  ventricularUrethralShunt).
causalEvidence(         bacteriaMenigitis,   exposure                ).
circumstantialEvidence( bacteriaMenigitis,   neurosurgery            ).
```

Figure 6: Some sentences from Figure 5.

(such a dependency network would be computed from knowledge of sentences and PSMs).

2.2 SentenceK: Sentence Knowledge

Knowledge restricts the search space in a domain. The role of sentenceK is to restrict the valid inferences that connect wordK. For example, words may be connected in rules as in Figure 5.

In many domains, there exists knowledge of the legal sentence types. Such knowledge can be expressed as a special kind of sentenceK called ontologies. Gruber defines such ontologies as "an explicit specification of a conceptualisation". This article includes many examples of ontologies:

- Figure 1 describes the types of words used in this article to describe knowledge types.
- Figure 6 shows Clancey's preferred ontology for MYCIN rules. This ontology is instantiated with domain-specific terms from the MYCIN system. For example, according to Clancey, MYCIN rules include a `subsumes`, `causalEvidence` and `circumstantialEvidence` relationships. Further, the word `bacterialMenigitis` "isa" `meningitis`.
- Later in this article, we will meet other ontologies including the REMAP ontology for design discussions (Figure 22) and an intricate ontology describing knowledge processing activities (Figure 23).

In an ontology, abstract terms usual appear high in some *isa* hierarchy while specific domain terms appear lower down the hierarchy.

2.3 BehaviouralK: Behavioural knowledge

We say that when a KB executes (i.e. PSMs §2.4.1 have controlled the application of the sentences from §2.2), then the KB has exhibited some *behaviour*. Once way to assess a KB is to compare the behaviour it can create with the desired behaviour. This implies the presence of behaviouralK: a library of known or desired or past behaviour. BehaviouralK can be found or built in at least four system groups described below: UML (§2.3.1), case-based reasoning (§2.3.2), QMOD/HT4 (§2.3.3), and various tools from software and knowledge engineering (§2.3.4),

We take care to distinguish the term behaviouralK from "functional specification". Functional specifications are generally much larger and more detailed than behaviouralK. In software engineering, for example, a functional specification is a statement of desired program behaviour. These statements are collected

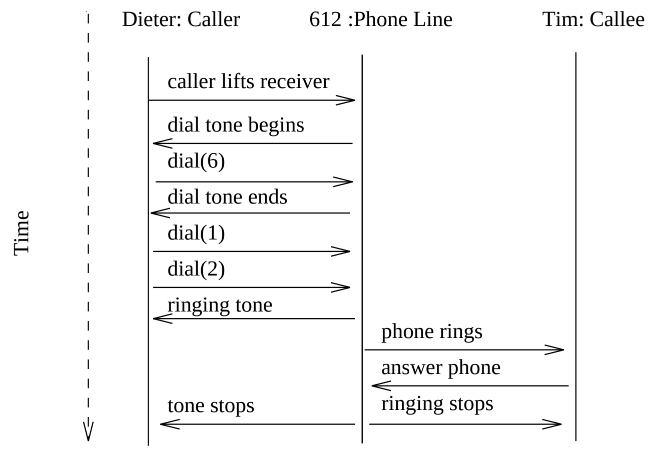

Figure 7: A use case expressed as a sequence diagram.

during system analysis time and are uncontorted, as far as possible, by implementation details. Fensel [Fensel, 1995] argues that function specifications in expert systems can include some implementation knowledge (the problem solving methods of §2.4.1). However, for the most part, functional specifications describe the "what" of the program and ignores the "how". BehaviouralK may not be a full description of the system: it may only be the small portions known at analysis time. Also, behaviouralK may be restricted to only observables on the interface of a program (exception: case libraries in case-based reasoning in §2.3.2).

2.3.1 BehaviouralK in UML

This section notes that behaviouralK can be collected from object-oriented specifications via uses cases [Jacobson *et al.*, 1992,Rumbaugh, 1994,Booch, 1996,Rubin & Goldberg, 1992]. A use case has a very simple text structure. Developers write a short "story" (say less than 2 pages) describing some flow of events within their system. The text of the use case is mapped into classes via *sequence diagram* such as Figure 7. The arrows on a sequence diagram represent the flow of events of the use case text. Each such arrow implies a method at the sender end and a method at the receiver end. A standard sequence diagram does not support conditionals. If a use case divides into N sub-cases, then each such sub-case is a separate use case.

Use cases have become the main driver of OO development [Jacobson & Christerson, 1995]. After dividing up software into multiple small modules, use cases are the "glue" that demonstrate how those modules can be strung together. In OO analysis techniques such as UML [Booch *et al.*, 1997], use cases are used as the basis of specifying the functional requirements, defining software objects, allocating functions to objects, and designing the interface [Jacobson & Christerson, 1995][2].

Note the specific data contained in Figure 7 (dialing of `612`, specific data values like `Dieter` and `Tim`). Use cases can also be used to extract behaviouralK from business users.

[2] For a short tutorial introduction to use cases, see [Rumbaugh, 1994]. For an intricate use of use cases for tracing the link between requirements and code, see [Rubin & Goldberg, 1992].

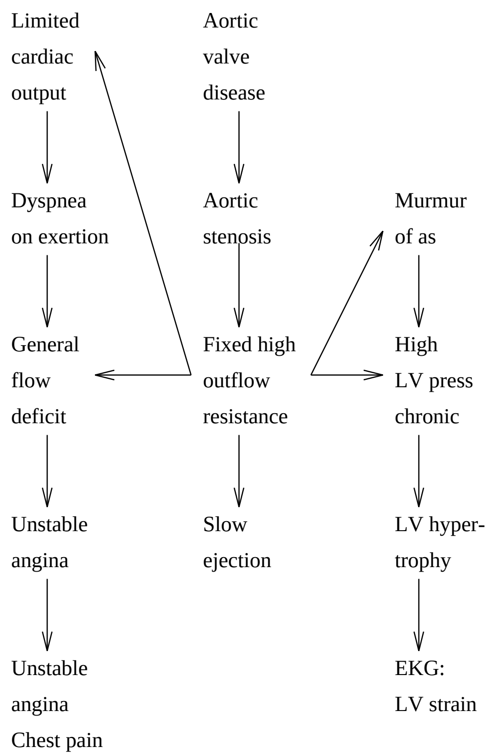

Figure 8: An explanation of David's symptoms using aortic valve disease. From [Kolodner, 1993, p419].

2.3.2 BehaviouralK in Case-based Reasoning

Case-based reasoning (CBR) is an inference strategy where new situations are managed by reviewing prior situations [Kolodner, 1991,Kolodner, 1993]. Knowledge in a CBR system is a library of prior cases. Inference at runtime consists of *matching* prior cases that are most like the new situation, then *adapting* those prior cases to the new situation. CBR fixK controls the finding and adaptation of cases (§3.4.7) .

BehaviouralK typically describes input-output pairs. CBR behaviouralK (the case library) includes details of how these input-output pairs were connected at runtime. Each stored case is generated from prior solutions and may be used to guide future solution generation.

For example, Figure 8 shows a stored case in the CASEY system [Kolodner, 1993, p418] which contains an explanation for the symptoms of the patient David. Note that some of the nodes in Figure 8 are directly observable (e.g. unstable angina) while others are inferred (e.g. aortic valve disease).

2.3.3 BehaviouralK in HT4

In the QMOD/HT4 system [Menzies & Mahidadia, 1997,Feldman *et al.*, 1989], the behaviouralK was a list of input/output pairs. QMOD/HT4 used the behaviouralK to validate neuroendocrinological theories (the study of connections of nerves to glands). For example, consider the task of achieving certain `OUT`puts

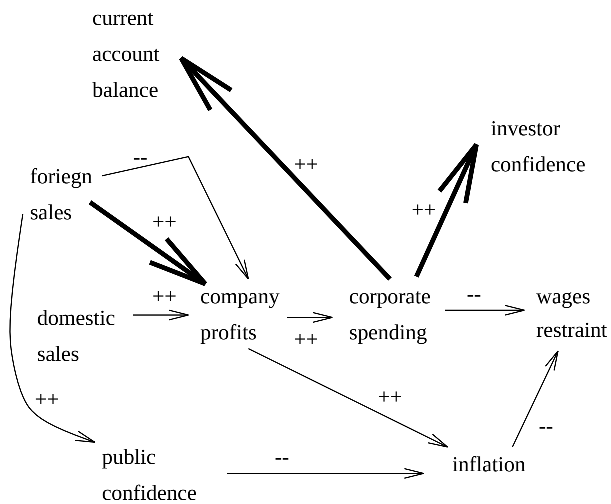

Figure 9: A theory processed by HT4.

```
P[1]:   domesticSalesDown, companyProfitsDown, inflationDown
P[2]:   foriegnSalesUp, publicConfidenceUp, inflationDown
P[3]:   domesticSalesDown, companyProfitsDown, corporateSpendingDown, wagesRestraintUp
P[4]:   domesticSalesDown, companyProfitsDown, inflationDown, wagesRestraintUp
P[5]:   foriegnSalesUp, publicConfidenceUp, inflationDown, wagesRestraintUp
P[6]:   foriegnSalesUp, companyProfitsUp, corporateSpendingUp, investorConfidenceUp
```

Figure 10: Proofs from Figure 9 connecting `OUT`= {`investorConfidenceUp, wagesRestraintUp, inflationDown`} back to INputs= {`foriegnSalesUp, domesticSalesDown`}.

using some `INputs` across the knowledge shown in Figure 9. In that figure:

- `x` $\overset{++}{\rightarrow}$ `y` denotes that `y` being `up` or `down` can be explained by `x` being `up` or `down` respectively;
- `x` $\overset{--}{\rightarrow}$ `y` denotes that `y` being `up` or `down` could be explained by `x` being `down` or `up` respectively.

Figure 9 is a combination of the opinions of two authors: *Dr. Thick* (whose contribution is drawn with thick lines) and *Dr. Thin* (whose contribution is drawn with thin lines). Observe the apparent conflict in the middle of Figure 9 on the left-hand-side. *Dr. Thick* believes `foriegnSales` $\overset{++}{\rightarrow}$ `companyProfits` while *Dr. Thin* believes `foriegnSales` $\overset{--}{\rightarrow}$ `companyProfits`. We will discuss the resolution of this conflict in §3.4.3.

In the case of the observed `OUT`puts being {`investorConfidenceUp, wagesRestraintUp, inflationDown`}, and the observed `INputs` being {`foriegnSalesUp, domesticSalesDown`}, QMOD/HT4 can connect `OUT`puts back to `INputs` using the proofs of Figure 10. These proofs may contain controversial assumptions; i.e. if we can't believe that a variable can go `up` and `down` simultaneously, then we can declare the known values for `companyProfits` and `corporateSpending` to be controversial. Since

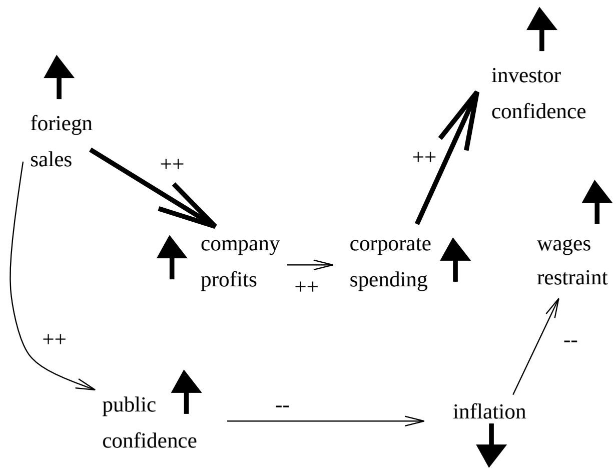

Figure 11: World #1 is generated from Figure 9 by combining `P[2]`, `P[5]`, and `P[6]`. World #1 assumes `companyProfitsUp` and covers 100% of the known `OUT`puts.

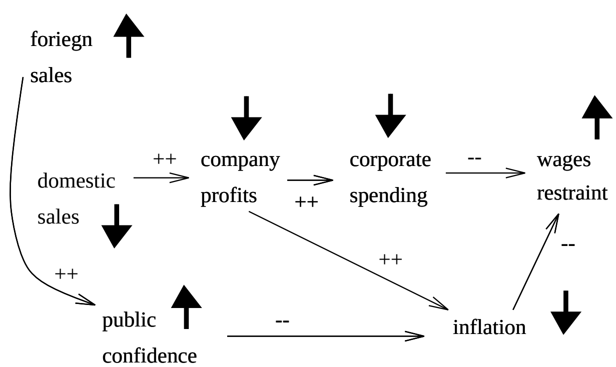

Figure 12: World #2 is generated from Figure 9 by combining `P[1]`, `P[2]`, `P[3]`, and `P[4]`. World #2 assumes `companyProfitsDown` and covers 67% of the known `OUT`puts.

`corporateSpending` is fully dependent on `companyProfits` (see Figure 9), the key conflicting assumptions are {`companyProfitsUp, companyProfitsDown`} (denoted *base controversial assumptions* or `A.b`). We can used `A.b` to find consistent belief sets called worlds `W` using an approach inspired by the ATMS [DeKleer, 1986]. A proof `P[i]` is in `W[j]` if that proof does not conflict with the environment `ENV[j]` (a maximal consistent subset of `A.b`). In our example, `ENV[1]`={`companyProfitsUp`} and `ENV[2]`={`companyProfitsDown`}. Hence, `W[1]`={`P[2]`, `P[5]`, `P[6]`} and `W[2]`={`P[1] P[2] P[3]`, `P[4]`} (see Figure 11 and Figure 12). Note that:

- While the background theory (Figure 9) may be inconsistent, the generated worlds are guaranteed to be consistent.
- QMOD/HT4 has some association to CBR. If we compare Figures 11 & 12 with Figure 8, we see that a CBR case library item could be a QMOD/HT4 world.

QMOD/HT4 defined *cover* to be size of the intersection of a world and the `OUT`put set. The cover of Figure 11 is 3 (100%) and the cover of Figure 12 is 2 (67%). Note that since there exists a world with 100% cover, then all the `OUT`puts can be explained. Feldman & Compton [Feldman *et al.*, 1989], followed by Menzies [Menzies, 1995b, Menzies, 1996a], have shown that QMOD/HT4 could detect previously unseen errors in theories in neuroendocrinology (the study of nerves and glands) published in international refereed journals. Surprisingly, these faults were found using the data published to support those theories

2.3.4 SE/KE BehaviouralK Tools

This section describes a range of testing tools from software engineering [Marick, 1997, Connell & Menzies, 1996] and knowledge engineering [Ginsberg, 1990, Zlatereva, 1992] which use behaviouralK. See §2.4.2 for other testing tools which do not use behaviouralK.

Two standard testing techniques based on behaviouralK is *test suite generation* and *code coverage*. Test suite generation builds the behaviouralK while code coverage tools explores what fraction of the code has been exercised by the behaviouralK.

Test coverage tools use known dependencies between words to augment the source code and/or the interpretor/virtual machine and generate logs of which portions of the code were exercised. For example, Connell & Menzies' TCAT system added calls to logging software into the source code of Smalltalk methods [Connell & Menzies, 1996]. Special browsers were written to allow users to quickly find unused portions of code, or code used very frequently.

A standard technique for behaviouralK generation is to analyse the branches in the source code in order to build test suites which cover all branches. Standard branch analysis makes a single-world assumption in which any combination of forks are compatible. A harder case is multiple-world test suite generation in which incompatibilities may exist within forks. Such mutually exclusive forks must be managed in separate worlds. Further, the source code itself may have to be divided into portions (a.k.a. worlds). Each portion must be internally consistent.

A flavor of such multi-world division of source code was supplied above (§2.3.3). The key conflicting assumptions must be uncovered by a PSM reflecting over the generated proofs. In our example above (Figures 11 & 12), these were the

ENV sets. Each such ENV set would be an interesting test case. Automatic test suite generation is systems which support inconsistencies have been implement by Ginsberg [Ginsberg, 1987, Ginsberg, 1990] and Zlatereva [Zlatareva, 1992, Zlatareva, 1993]. The dependencies between rules/conclusions are computed and divided into mutually consistent subsets. The root dependencies of these subsets represent the space of all reasonable tests. If these root dependencies are not represented as inputs within a test suite, then the test suite is incomplete. Test cases can then be automatically proposed to fill any gaps.

Formally, multiple-world test suite generation is *abduction* [Eshghi, 1993] and abduction is slow. Selman & Levesque show that even when only one abductive explanation is required and the theory is restricted to be acyclic, then abduction is NP-hard [Selman & Levesque, 1990]. Bylander *et. al.* make a similar pessimistic conclusion [Bylander *et al.*, 1991]. Computationally tractable abductive inference algorithms (e.g. [Bylander *et al.*, 1991, Eshghi, 1993]) typically make restrictive assumptions about the nature of the theory or the available data. Such techniques are not applicable to arbitrary theories. Therefore, it is reasonable to question the practicality of multiple-world test suite generation for medium to large theories. Hence the single-world assumption taken by most test suite generators (e.g. PISCES [technlogies, 1997]). However, Menzies has shown that multiple-worlds reasoning is practical at least for a sample of the theories seen in contemporary KE practice [Menzies, 1996b].

2.3.5 The Operationalisation KM Assumption

A common assumption in KM research is that if we can watch a program execute, then we can understand it. This section raises doubts with this *operationalisation KM assumption.*

Supporters of this assumption believe that if we can watch a KB execute, then we see where it goes wrong. However, operationalisation may be an incomplete maintenance strategy without behaviouralK. Recall the tools described in the previous sections. Systematic methods for processing behaviouralK were discussed that supported:

- The collection of libraries of known or desired behaviour using use cases and test suite generation tools.
- The analysis of the execution of those libraries of test data using case-based reasoning, HT4, and code coverage tools.

Note that these tools were much more that "let it run and see what happens". We should not ask the creators of a program to evaluate that program by merely watching it run. The "halo effect" prevents a developer from looking at a program and assessing its value. Cohen likens the halo effect to a parent gushing over the achievements of their children and comments that...

> What we need is not opinions or impressions, but relatively objective measures of performance. [Cohen, 1995, p74].

Unless we have (a) some expectation of appropriate behaviour and (b) analysis tools for the resulting behaviour (i.e. behaviouralK), then we cannot assess if the runtime behavior of a system is adequate.

2.4 Meta-knowledge

Meta-knowledge is knowledge about the structure, assessment, or modification of the different knowledge types. Meta-knowledge is a very broad area and is

```
if     the patient received a ventricular-urethral-shunt
then   infection = e.coli (.8) or klebsiella (.75)
```

Figure 13: The real heuristic knowledge within Figure 5.

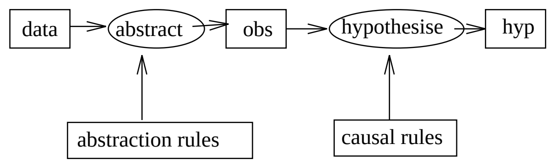

Figure 14: Explicit problem solving (PSM) meta-knowledge: A simple KADS-style PSM for diagnosis. `Abstract` and `hypothesis` are primitive inferences which may appear in other PSMs. From [van Harmelen & Aben, 1996].

the focus of much of the advanced research into knowledge engineering. One characterisation of the difference between software engineering and knowledge engineering is that the former only concerns itself with wordK and sentenceK while the latter supports reflection over that knowledge [Clancey, 1989]. That is, software engineering does not usually model meta-knowledge (exceptions: the *reflection* pattern [Buschmann *et al.*, 1996] and the software engineering product-oriented assessment tools listed in §2.4.2). Meta-knowledge will be discussed below under the headings: problems solving methods (PSMs) (§2.4.1); quality knowledge (§2.4.2); fix knowledge (§2.4.3); and social knowledge (§2.4.4).

2.4.1 PSMs: Problem Solving Methods

In Newell's knowledge-level modeling KL approach [Newell, 1982,Newell, 1993], intelligence is modeled as a search for appropriate *operators* that convert some *current state* to a *goal state*. Domain-specific knowledge in used to select the operators according to *the principle of rationality*; i.e. an intelligent agent will select an operator which its knowledge tells it will lead the achievement of some of its goals. By reverse-engineering KB implementations, we can identify common sets of operators that have been used in many applications. These are called problem solving methods (PSMs).

We can divide research into knowledge level modeling into two broad camps:

1. In the majority KL-B view, a KB should be divided into domain-specific facts (which we call here wordK and sentenceK) and libraries of domain-independent PSMs. For example, Clancey argues that knowledge engineering should separate heuristics like Figure 5 into domain-specific knowledge about the terminology (see Figure 6), meta-knowledge which controls the application of the knowledge (see Figure 15) and true domain-specific

Strategy	*Description*
exploreAndRefine	Explore super-types before sub-types.
findOut	If an hypothesis is subsumed by other findings which are not present in this case then that hypothesis is wrong.
testHypothesis	Test causal connections before mere circumstantial evidence.

Figure 15: PSMs identified by Clancey [Clancey, 1992] within Figure 5.

heuristic knowledge (see Figure 13) [Clancey, 1992]. In the KADS approach, PSMs may be expressed graphically such as in Figure 14 (ovals are functions, rectangles are data structures).

2. In the minority KL-A view, KBs contain wordK, sentenceK, and a single PSM. In Newell's operationalisation of KL, this single PSM is the problem-space computational model (PSCM) [Newell, 1993, Yost & Newell, 1989, Yost, 1993]. Programming the PSCM involves the consideration of multiple, nested problem spaces. Whenever a "don't know what to do" state is reached, a new problem space is forked to solve that problem. The observation that a PSCM system is performing (e.g.) classification is a user-interpretation of a single lower-level inference (operator selection over a problem space traversal) [Yost & Newell, 1989]. Our own maintenance proposal is a KL-A approach since it is based the exception-driven modification of choice operators within a single problem solving method: abduction (§3.4.2).

That is, both KL-A and KL-B use PSMs. However, KL-A uses 1 PSM while KL-B uses N PSMs. In KL-B, each PSM combines a common set of underlying inference mechanisms (called various terms like "knowledge sources" [Wielinga *et al.*, 1992a], "mechanisms" [Marques *et al.*, 1992], etc; e.g. `abstract` and `hypothesis` in Figure 14). KL-B is the major focus of much of the KA community: e.g. generic tasks [Chandrasekaran *et al.*, 1992]; configurable role-limiting methods [Swartout & Gill, 1996,Gil & Melz, 1996]; SPARK/BURN/FIREFIGHTER-hereafter, SBF [Marques *et al.*, 1992]; model construction operators [Clancey, 1992]; CommonKADS [Wielinga *et al.*, 1992a,Schreiber *et al.*, 1994]; the Method-To-Task approach [Eriksson *et al.*, 1995]; components of expertise [Steels, 1990]; MIKE [Angele *et al.*, 1996]). Libraries of PSMs are described in [Benjamins, 1995,Breuker & de Velde (eds), 1994,Chandrasekaran *et al.*, 1992,Motta & Zdrahal, 1996, Tansley & Hayball, 1993]. See the *Related Work* section of [Wielinga *et al.*, 1992a] for a discussion of the differences in some of these techniques. In the terminology of this paper, the KL-B PSM research focuses on acquiring (§3.1) and operationalising (§3.2) PSMs.

A halfway position between KL-A and KL-B is offered by Chandrasekaran *et. al.* [Chandrasekaran *et al.*, 1992] in which:

- PSMs describe *tasks* (e.g. diagnosis) which can be implemented by...
- *Methods* (e.g. classification, simulation) which can be specified in a generic way using the PSCM. Note that methods may be implemented as tasks (which may recursively contain methods).

However, even though Chandrasekaran *et. al.* use the PSCM internally, they argue that the task-level is the best view for understanding the system without using too much low-level detail. We have some sympathy with this view. Our biggest criticism of the PSCM is that there is nowhere to model cliched sets of operators which have proved useful in previous applications.

On the other hand, our biggest criticism of KL-B is that the complexity and diversity of the multiple PSM KL-B libraries may be hard to maintain. A standard assumption in the KL-B community is that the PSM will remain constant (or nearly constant) over the lifetime of the project (but see the exceptions discussed in the next paragraph). For example, Pos *et. al.* [Pos *et al.*, 1997] offer a detailed analysis of "redesign problem solving": the process of selecting or adapting a PSM. They stress in their conclusion that this process would be useful during the early stages of KBS design. We make two comments here:

- Many of the techniques surveyed by Pos *et. al.* are relevant to the general KM problem throughout the lifecycle.

- PSMs are still being developed. For example, we see in the literature at least 8 definitions of "diagnosis": three from the KADS community [Wielinga *et al.*, 1992a,Benjamins, 1995,Tansley & Hayball, 1993]; Clancey's heuristic-classification-as-diagnosis approach [Clancey, 1985]; Fowler's object-oriented approach [Fowler, 1997, cph3]; Menzies graph-based approach based on abduction [Menzies, 1996a]; DeKleer & William's approach based around a distinction between a problem solver and a assumption-based truth maintenance system [DeKleer & Williams, 1987]; and Poole's approach based on Bayesian reasoning [Poole, 1993]. If (e.g.) DeKleer tried to maintain (e.g.) Clancey's system, he might make extensive modifications to the KB. Consequently, it is not clear to us that PSM knowledge is free from the maintenance problem.

Not all PSM research assumes static PSMs. For example, recall that domain-specific words (e.g. `meningitis`) are mapped into general PSMs- recall Figure 6). This mapping process introduces a degree of flexibility into PSMs: facts can take on different roles in different problem solving contexts. It has hence been argued [Hoffman *et al.*, 1997, Shadbolt & O'Hara, 1997] that PSMs are a technology for addressing the issue of changing knowledge. This may only be partially correct since while domain facts can be mapped into different parts of a PSM, the PSM itself is assumed to be fixed during the lifecycle of the programme. Some research addresses the issue of dynamic configuration of a problem solver. Often, PSM configuration is implemented via a depth-first traversal of a hierarchy describing PSM options (e.g. see [O'Hara & Shadbolt, 1997] and the TINA system described below in Figure 27). In this approach, the depth-first traversal is constrained by the current problem being analysed. This is only a partial solution since while the generated PSMs can vary according to the problem context, the background space of PSM options is fixed. No guidelines are given for how experience with the running program can feedback into modifying the PSM options hierarchy. Van de Velde hints at a more general mechanism in which machine learners learn model review strategies via watching human revise their models [veld93], but such work is only in its infancy.

2.4.2 QualityK: Quality Knowledge

QualityK stores some manner of generating an opinion about the value of the KB. For example, QMOD/HT4 applied a very generalised quality procedure. It computed the worlds with maximum cover (§2.3.3) and assessed the quality of a KB via that maximum coverage figure.

The QMOD/HT4 quality knowledge assumed the presence of behaviouralK (§2.3). If we do not have such a library, we can identify three broad classes of quality knowledge in the literature: non-functional requirements; product-oriented assessment; and inconsistency detection knowledge (this inconsistencyK is a special kind of product-oriented assessment), and critical success metrics. All these forms of quality knowledge are discussed below.

```
``Monitoring and control''
Preconditions:
        ``Monitoring instrumentation''
        ``Control limits''
        ``Algorithms''

Postconditions:
        ``If the function is stable, checks the performance and reports
        it, otherwise stabilises the function by controlling the
        configuration or environment. May also report predicted future
        undesirable states.''

Effects on quality attributes:
        Assurance      : pros ``Avoids undesirable states''
        Performance    : cons ``Needs additional processing in short term''
                         pros ``Improves performance in long term via tuning''
        Timeliness,
        Affordability : cons ``More effort to specify''
                         cons ``More effort to develop''
                         cons ``More effort to verify''
```

Figure 16: NFR quality knowledge: strategy knowledge from QARCC. From [Boehm, 1996].

Non-Functional Requirements: Functional requirements can be measured via executing and measuring a program. Non-functional requirements (NFR) such as portability, evolvability, development affordability, security, privacy, or reusability cannot be assessed with respect to the current version of the working program. For example, consider the NFR of maintainability. Maintainability can only be definitively assessed in retrospect; i.e. only after delivery has occoured and we have some track record of the system's performance in the field. Nevertheless, during initial construction, we may still want to assure ourselves as to the potential maintainability of the system.
Two example of NFR qualityK are Chung & Nixon's *goal graph* [Chung & Nixon, 1995] approach and Boehm's *et. al.* QARCC tool [Boehm, 1996]:

- The QARCC quality KB represents the concerns of different *stakeholders* (e.g. user, customer, developer, developer, maintainer, interfacer, and general public) and the quality attributes which map into those concerns. For example, maintainers are mostly concerned with evolvability and portability while customers and developers are mostly concerned with development affordability and reusability. A *strategy* fragment of a QARCC quality knowledge base is shown in Figure 16. These strategy fragments are mapped into different stakeholders (e.g. developers and customers both worry about `Affordability`). These strategies are then explored looking for conflicts such as `Performance` vs `Affordability` trade-offs.

- Goal graphs contain similar trade off information to the QARCC strategy fragments, but do not explicitly model stakeholders.

Product-Oriented Assessment: Product-oriented assessment knowledge is applied to features extracted from a software product. Knowledge engineering product-oriented assessment tools can be found in the verification literature. Verification is the exploration of the internal syntactic structure of a program.

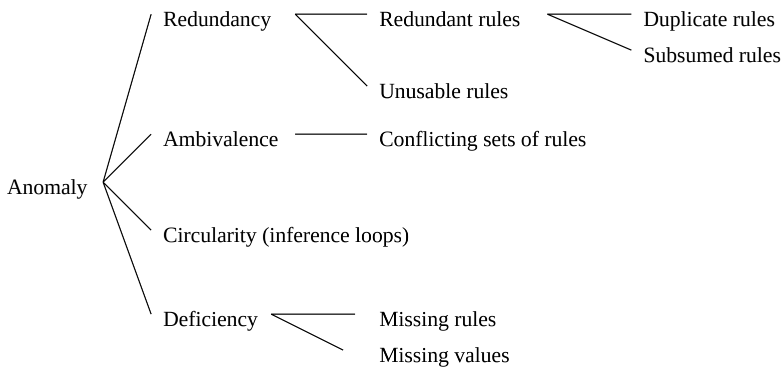

Figure 17: PSB anomalies.

	MMU	TAPES	NEURON	DISPLAN	DMS1
Size (literals)	105	150	190	350	540
Logical subsumption errors	0	$^{5}/_{5}$	0	$^{4}/_{9}$	5
Missing rule errors	0	$^{16}/_{16}$	0	$^{17}/_{59}$	0
Circularities in reasoning errors	0	0	0	$^{20}/_{24}$	0

Figure 18: Ratios of ($^{\text{true errors}}/_{\text{anomalies}}$) in a sample of fielded expert systems. From [Preece & Shinghal, 1992].

Preece, Shinghal and Batarekh [Preece & Shinghal, 1992] (hereafter, PSB) define a taxonomy of structural "anomalies" in rule-based expert systems (see Figure 17) and argue that a variety of verification tools target different subsets of these anomalies (perhaps using different terminology).

PSB stress that the entries in their taxonomy of KBS anomalies may not be true errors. For example, the dependency network from a rule-base may show a circularity anomaly between literals. However, this may not be a true error. Such circularities occur in (e.g.) user input routines that only terminate after the user has supplied valid input. For this reason, PSB call the detected "error" anomalies, not faults. Figure 18 shows the ratio of true errors to detected anomalies for circularities, subsumption, and missing rules. Note the large number of detected anomalies which were not true faults. Knowledge engineering product-oriented assessment should be used as pointers into the system which direct the developer to areas that may require a second glance.

Numerous software engineering product-assessment tools are available. For example:

- Balbo describes software tools for assessing the usability of user interfaces [S, 1995].

- Potentially, high-level product-oriented assessment can be linked back to non-functional requirements [Boehm, 1996]. Architectural and design patterns [Gamma *et al.*, 1995, Buschmann *et al.*, 1996, Fowler, 1997, Coad *et al.*, 1997, Menzies, 1998d] describe common cliches in software engineering. Proposed software designs can be heuristically assessed via known properties of these patterns. For example, layered architectures increase portability but slow down the program. The practicality of this approach is being explored within the QARCC project.

- Haynes, Menzies, and Phipps argue that product-oriented assessment of object-oriented systems can be mapped back to quality attributes; e.g. smaller classes seem to produce fewer bugs that larger classes [Haynes *et al.*, 1995]. The long term goal of research such as the Haynes, Menzies, and Phipps paper is a mapping from product-oriented metrics (i.e. numbers extracted from the source code) back to NFRs such as maintainability and extendibility. Haynes believes that such a mapping can be generated via rigorous statistical means. However, we believe that ultimately this mapping will be subjective; i.e. open to disagreement and therefore will require maintenance using the tools described in this paper (see below §2.5: the recursive maintenance problem).

- A variety of other tools apply software engineering quality criteria to assess systems. For example, the TCAT system of Connell & Menzies reports what percentage of a Smalltalk program was exercised during the execution of a test suite [Connell & Menzies, 1996].

Note that the success of TCAT depends on executing a representative test suite that covers the application's common operations. That is, certain product-oriented assessment software engineering tools require behaviouralK (§2.3). The same can be said for some knowledge engineering product-oriented assessment tools. For example, EXPECT [Gil & Tallis, 1997] detects errors in PSMs in a LOOM representation via a partial evaluation of methods. This partial evaluation is driven by a particular example (a.k.a. an item in the behaviouralK). Errors are detected if a method cannot fire because the types of the input parameters to the methods are not available (formally, this is a variant on PSB "unusable rules" in Figure 18). Note that the completeness of this EXPECT error detector is a function of the completeness of the behaviouralK used to drive the partial evaluator.

InconsistencyK: Our next qualityK performs inconsistency detection. Inconsistency detection is particularly important in the case of knowledge collected from different experts.

Inconsistency detection knowledge may be very simple such as:

```
inconsistent(X, not X).
```

QMOD/HT4 knew that its vertices came from a set of variables that have can be in one of `N` mutually exclusive states. QMOD/HT4 reported an inconsistency if:

```
inconsistent(Var1/State1, Var1/State2) :- not(State1 = State2).
```

However, inconsistency knowledge may be representation-specific. For example:

- In an object-oriented language that supports constraint rules, the constraints on a sub-class should not be violated by a super-class (i.e. any sub-class should be able to "stand in" for the super-class, wherever it is used).

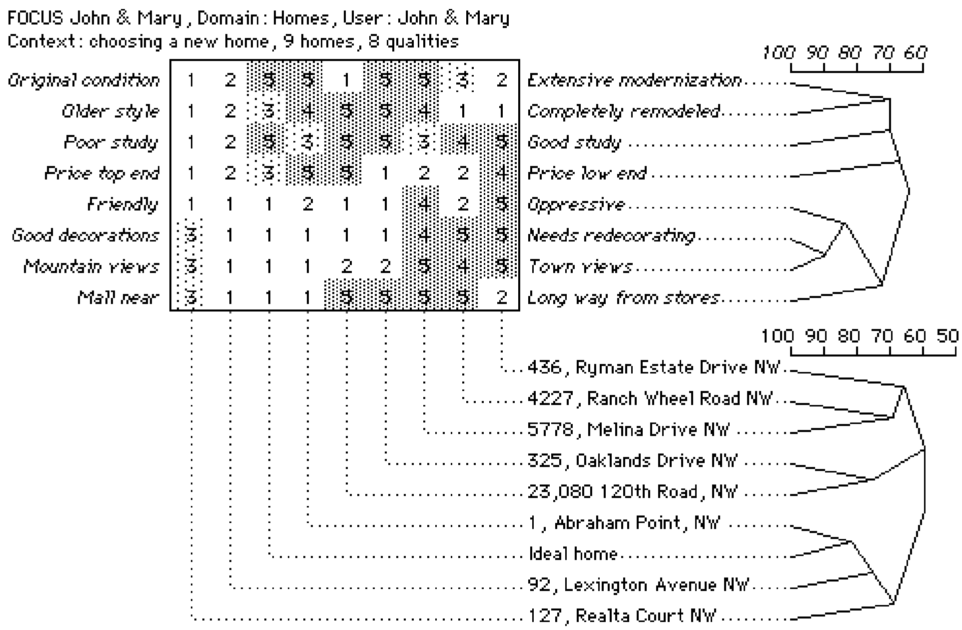

Figure 19: Repertory grids. Generated from the WebGrid WWW server [Shaw, 1997].

- Given state transition diagrams from two authors, report an inconsistency if:
  - A transition exists between two states in one diagram;
  - Those two states appear in the other diagram;
  - The transition does not appear in the other diagram [Easterbrook & Nuseibeh, 1996].
- Shaw's *repertory grids* can detect conflicts in wordK by comparing grids from different experts. Experts are asked to identify dimensions along which items from their domain can be distinguished. The two extreme ends of these dimensions are recorded left and right of a grid. New items from the domain are categorised along these dimensions. This may lead to the discovery of new dimensions of comparisons from the expert which, in turn, will cause the grid to grow [Gaines & Shaw, 1989]. For example, based on how an expert scaled some example houses, we can see from the repertory grid of Figure 19 that the `ideal home` is closest to `1, Abraham Point, NW`. Once the dimensions stabilise, and a representative sample of items from the domain have been categorised, then the major distinctions and wordK of a domain have been defined. Inconsistencies are reported if the categorisations are significantly difference. Using this technique, Gaines & Shaw [Gaines & Shaw, 1989] can detect four classes of inconsistencies in wordK:
  - Consensus: same item, same categorisations;
  - Correspondence: (a.k.a. synonyms) items with different names, but the same categorisation;

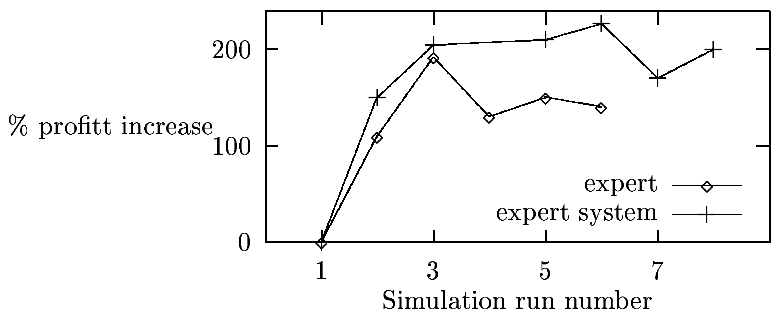

Figure 20: Critical success metrics for PIGE. From [Menzies *et al.*, 1992].

  - True conflict: same items, different categorisations;
  - Contrast: different items, different categorisations

Critical Success Metrics: Our final qualityK are critical success metrics (CSMs). A CSM is a combination of a numeric measure and a threshold value. If the measure exceeds the threshold, then the system is deemed to be a success [Menzies, 1998c]. In some cases, thresholds can be set via some "gold standard" (e.g. [Shahsavar, 1993]).

In our experience, CSMs have several properties. Firstly, they are very domain-specific. Secondly, CSMs are a reflection of the contribution of the behaviour of the software in a particular business context. Hence:

- They typically do not refer to internal properties of a program.
- They cannot be developed by programmers without extensive input from business users.
- They can only be collected once the program is running in its target context.

Thirdly, CSMs may require extra architecture. For example, in one application, we identified "increases sales per day" as the CSM for a dealing room expert system. However, this number was not currently being collected. Sales per day could be estimated from the quarterly statements, but no finer grain data collection was performed at that site. Hence, prior to building the expert system, we had to build a database system to collect the baseline data.

Fourthly, while CSMs are obvious in retrospect, they can take weeks of analysis to uncover. For example, in the PIGE farm management expert system [Menzies *et al.*, 1992], nutrition experts argued for weeks about the merits of different protein utilisation models. Then the marketing people commented that such considerations were irrelevant if it could not be demonstrated that the systems recommendations improved the overall profitability of a farm. Hence, the evaluation focus moved from the protein utilisation models to issues of modeling the farm economics. Figure 20 shows the CSM evaluation of PIGE. Given a particular configuration of the livestock, an optimisation model could infer the annual profit of the farm. Alternate configurations could be explored using

```
seekDiagnosisRule
if   majorSymptoms and
     minorSymptoms and
     tests and
     not exclusions
then diagnosis is true
with confidence
     [definitely or probably
      or possibly]
```

```
seekFixRule
if   the number of cases suggesting
     generalisation is greater than
     the number of cases suggesting
     specialisation
and  the most frequently missing
     component is the major symptoms
then delete some major symptoms
```

Figure 21: SEEK rules.

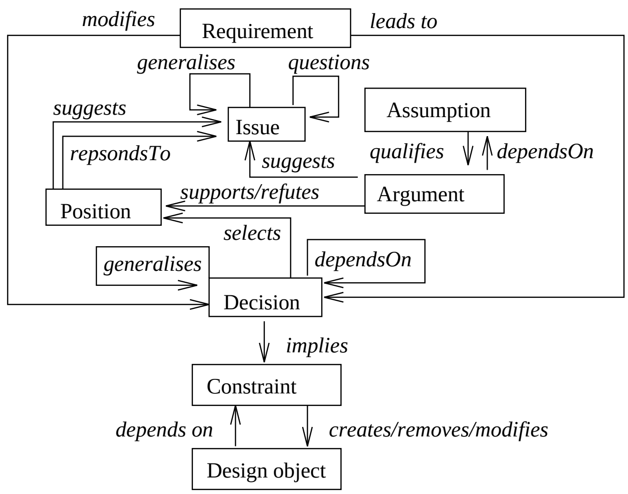

Figure 22: SocialK in REMAP. From [Ramesh & Dhar, 1992].

a simulation model. A user can choose some settings, then run the simulation model to see if the system's performance improved. Figure 20 contrasts this exploration process between the human expert who wrote the rule base and the expert system. Note that this expert system out-performed its author (!!) [Menzies *et al.*, 1992].

2.4.3 FixK: Fix Knowledge

Fix knowledge specifies how to correct errors. Much of the research into machine learning (ML) (§3.4.6) and CBR (§2.3.2) can be characterised as a search for good heuristics for fixK. Since fixing is typically a slow process, fixK is usually expressed algorithmically for reasons of efficiency. One notable exceptions is the SEEK [Politakis, 1985] and SEEK2 systems. SEEK [Politakis, 1985] worked with a human operator to try fix rules of the general form of `seekDiagnosisRule` in Figure 21. SEEK fixK can propose fixes to a human operator; e.g. additions or removals of tests/symptoms, or changes to the confidence value of a rule. SEEK2 [Ginsberg *et al.*, 1988] was a generalisation of SEEK and contained (i) a more general rule format is processed; (ii) a special meta-language for defining specify fix knowledge in a more general form than e.g. `seekFixRule`.

2.4.4 SocialK: Social Knowledge

Some expert systems practioners argue that knowledge cannot be understood with understanding its social context [Winograd & Flores, 1987, Clancey *et al.*, 1996]. Paper documents are often collected which informally describe the organisational context of a system; e.g. the organisational model of KADS [Wielinga *et al.*, 1992b] or the stakeholders of the Olle-126 [Olle *et al.*, 1991]. Some practioners also operationalise their socialK:

- Clancey *et. al.*'s BRAHMS system [Clancey *et al.*, 1996] models the exchange of information amongst a group of agents about functional knowledge (orders, organisations, roles, product flows). BRAHMS includes very detailed descriptions of the actual day-to-day work of those agents. In BRAHMS, the macro-workflow of an organisation is an emergent process that is inferred from all the micro-behaviour of the agents in an organisation.

- Figure 22 shows the socialK within the REMAP system [Ramesh & Dhar, 1992]. REMAP logs design discussions and their inter-connections. If a developer changes their position on some argument, then developers can track the impact of that change to the constraints on the development. Previous discussions can be replayed to generate an historical understanding of how some decision was achieved.

- The REMAP system is a specific example of a more general process. *Design rationales* are a record of why a community decided to change some aspect of a system [Moran & Carroll, 1996]. Design rationales are described in more detail below (§3.5).

2.5 The Recursive Maintenance Problem

The qualityK approach assumes that when assessing some *execution KB* KB1, a second *maintenance KB* KB2 will be created to store the quality knowledge. Internally, this maintenance KB (KB2) may contain wordK, sentenceK, and PSMs. For example, a QARCC PSM might describe how to best resolve conflicting knowledge offered by feuding experts. However, it also contains extra knowledge for fixing, represented agent societies, the behaviouralK, and the processing activities (recall Figure 1). If KB2 is complex subjective, or context-dependent, then it will need maintenance. If we maintain KB2 using qualityK stored in some KB3, do we then then need some KB4 to maintain KB3 and a KB5 to maintain KB4 and...

This is the *recursive maintenance problem*: i.e. how do we maintain the maintenance knowledge. The recursive maintenance problem can be seen in many KM schemes:

- When developing some program (KB1), QARCC and goal graphs are a knowledge base (KB2) which assesses (KB1). If KB2 (e.g. Figure 16) is complex, subjective, or context-dependent, then it may require maintenance as well.

- The mappings searched for by QARCC and Haynes *et. al.* between product-oriented assessment metrics and stake holder quality attributes would have to be represented in some KB2. Such mappings are subjective and may require maintenance.

- Automatic test suite generation present an even tighter recursive maintenance problem than the KB1-KB2 problem. Such generators use KB1 to assess KB1 by generating test suites to exercise all branches of KB1. After an expert describes their world-view in a KB1, that same expert will be asked to specify the results of certain inputs. If the expert then uses KB1 to predict the output, then they would be using a potentially faulty KB to generate a potentially faulty prediction about the output [Menzies, 1996a]. The QMOD/HT4 study (§2.3.3) used real-world observations; i.e. knowledge of valid `IN`put and `OUT`put came from a source external to the knowledge base.

A solution to this (potentially infinite) recursive maintenance problem would need to demonstrate that the recursion terminates; e.g. KB2 is demonstrably smaller or less subjective than KB1. Such demonstrations are not found in the current KM literature (exception: RDR, see §3.4.1).

3 Knowledge Processing Activities

The previous section described seven types of knowledge, any of which may require maintenance. This section divides "maintenance" into five knowledge processing activities shown in Figure 23:

- *Gathering "it"* (a.k.a. acquire §3.1) using a variety of technique such as rapid acquire (§3.1.1); software engineering tools (§3.1.2); IO-SCHEMAS (§3.1.3); using PSMs or ontologies to guide the acquire process (§3.1.4); ECS influencers (§3.1.5); amongst other techniques (§3.1.6).

- *Making "it" execute* (a.k.a. operationalise §3.2). Operationalisation can be automated if the KM environment can access the generic primitive inferences used in the PSMs.

- *Finding errors in "it"* (a.k.a. fault §3.3). Manual expert inspection is the usually fault approach used in software development. Automatic support can be offered in a number of ways such as using qualityK (including inconsistencyK) and ontologies. Test suite generation can be used to augment behaviouralK. This behaviouralK can be used to assess test case coverage. If the KM environment can access the wordK dependency network, then (i) the cause of a fault can be localised by tracing upstream from the fault (§3.3.1); and (ii) users can browse-around parts of the knowledge with "how, why, why not" and "what if" queries (§3.3.2).

- *Fixing "it's"errors* (a.k.a. fix §3.4). Ripple-down techniques (ripple-down rules, §3.4.1; or ripple-down functions, §3.4.2; ripple-down rationality, §3.4.2) patch different types of knowledge in an exception-driven manner: ripple-down rules (RDR) patches sentences; ripple-down-functions (RDF) patches words; ripple-down rationality (RD-RA) patches abduction (a KL-A-type PSM). Conflicts can be resolved via negotiation between KB authors (§3.4.3). Case-based reasoning (§3.4.7) or ECS debiasers (§3.4.4) can also be used for fixing. FixK can be used to drive ML. ML systems may be they inductive, deductive (§3.4.6), or generalisers/specialisers (a

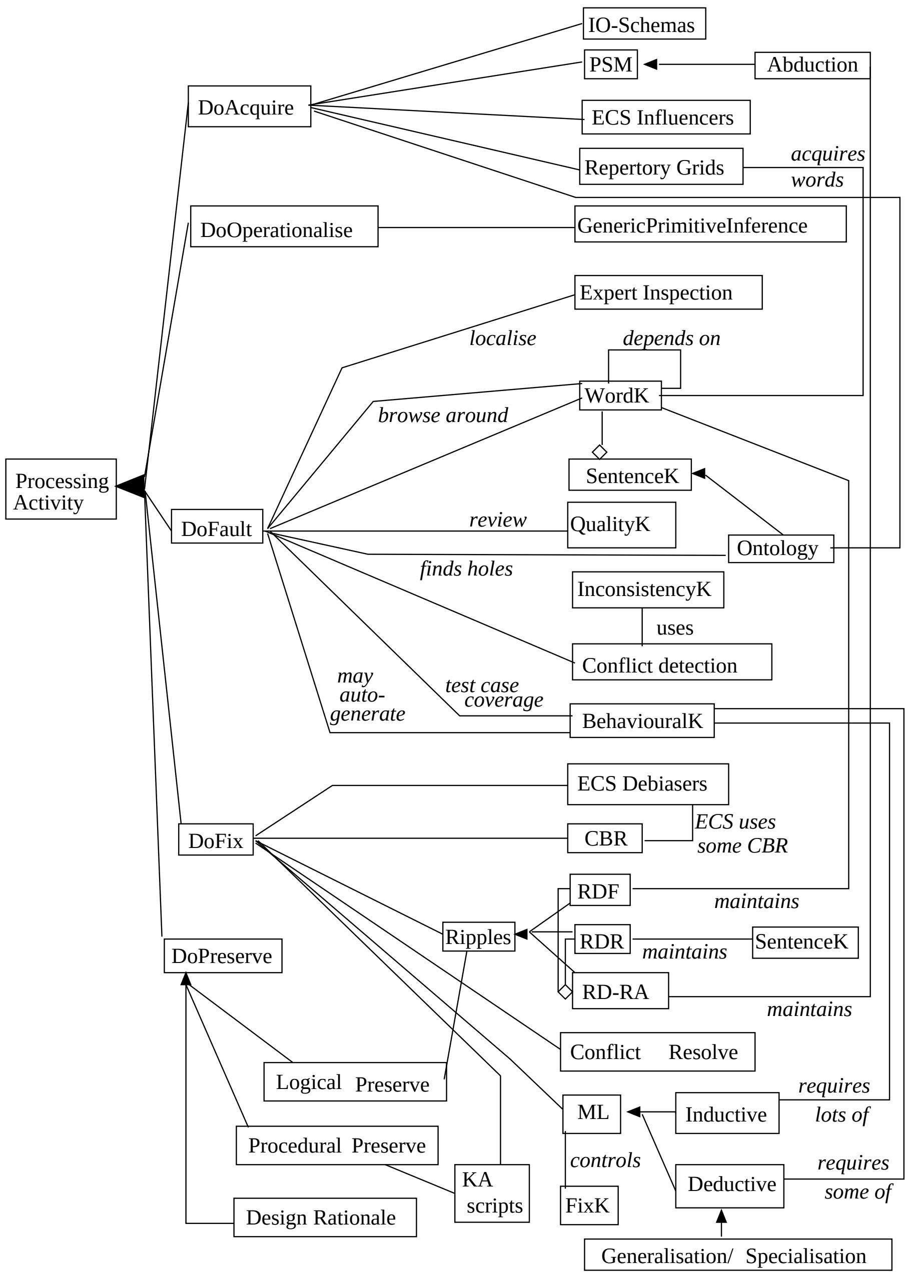

Figure 23: Different knowledge processing activity types (in the notation of Figure 1).

special kind of deductive ML, see §3.4.5). Inductive ML requires far more behaviouralK than deductive ML. KA scripts can be used to ensure that fixes requiring multiple fixes are fully completed (§3.4.8).

- *Ensuring that the changes (i.e. fixes or more acquired knowledge) do not break working parts of "it"* (a.k.a. preserve §3.5). Two broad sub-types of this preserve activity are logical preserver (e.g. ripples) or procedural preserve (e.g. KA scripts).

3.1 Acquire

We use the term "acquire" to denote the process of recording different knowledge types. Typically, the process of acquiring occurs before operationalisation, faulting, and fixing.

In current practice, we can see the following types of acquire tools: rapid acquire systems (§3.1.1); software engineering acquire tool (§3.1.2); IO-SCHEMAS(§3.1.3); PSM-based acquire systems (§2.4.1); expert critiquing systems; and miscellaneous other techniques (§3.1.6).

3.1.1 RAS: Rapid Acquire Systems

In an email survey of the readers of the `comp.ai` news group, we asked for information on maintenance systems. The core technology cited in many of the respondents was some style of rapid acquire system (RAS), plus operationalisation support (§3.2). In RAS systems, developers work in a high-level environment which may include point-and-click graphical editors. The *RAS assumption* presumes that if knowledge is expressed at a sufficiently high-level, then its flaws are obvious and quick to change.

RAS may be an incomplete maintenance strategy. Firstly, merely browsing knowledge may not tell the user how the PSMs will apply the knowledge at runtime (an analogous argument was made above in §2.3.5 when we discussed the operationalisation KM assumption). As soon as such runtime considerations appear, then RAS needs to be augmented at least with behaviouralK.

Secondly, very short descriptions of knowledge may contain faults, even if they are described at a very high level. If the reader doubts this, they are invited to find all the errors in a one line model of population growth: $^{dN}/_{dT} = rN$ where T is time, N is the population, and r is a constant reflecting environmental conditions (positive for benign environments and negative for hostile environments).

This model is wrong. Population growth must taper off as it approaches C, the maximum carrying capacity of the environment; i.e. $^{dN}/_{dT} = rN(1 - {}^{N}/_{C})$. If the reader can correctly answer the following question, then we have anecdotal evidence for believing the RAS assumption: is this new equation correct?

The second equation is incorrect[3]. In the case of a hostile environment and over-population, then our intuition is that population will fall. However, in that case, $N > C$, $r < 0$, and $rN(1 - {}^{N}/_{C}) > 0$; i.e. the maths says that population will increase (example from [Levins & Puccia, 1985]). Our experience has been that the error is not apparent to many people.

If the reader could not find all errors in a one-line model (which they may have studied extensively in high school), then they should be suspicious of the RAS assumption that the truth status of larger models can be accurately determined by visual inspection. Other studies encourage us to be cautious about the RAS

[3] And if the reader cannot see why before reading on, then Q.E.D.

assumption. Myers [Myers, 1977] reports controlled experiments with a 63 line model. 59 experienced data processing professionals hunted for errors in a very simple text formatter (63 line of PL/1 code). Even with unlimited time and the use of three different methods , the experts could only find (on average) 5 of the 15 errors in this 63 line model. We conclude that RAS should be augmented with other KM options. (Note that an analogous conclusion was offered above when we discussed the operationalisation KM assumption.)

3.1.2 Acquire in Software Engineering

Most information systems methodologies (e.g. the Olle-126 [Olle *et al.*, 1991]) focus primarily on on acquiring knowledge (and a little on conflict detection between different stakeholders). Numerous paper-based methodologies have been proposed (some of which are surveyed in the Olle-126, see also [Booch, 1996, Booch *et al.*, 1997]). Many CASE and CAKE tools and are RAS tools for acquiring wordK and sentenceK. Some RAS CASE tools could be said to support meta-knowledge acquisition. For example, RATIONAL-ROSE [Corporation, 1997] allow users to enter sequence diagrams such as Figure 7 and Harel-style state charts [Harel, 1995]. These sequence diagrams and state charts could be used to record PSMs such as Figure 14. However, they are typically not used for knowledge engineering. Rather, they are used in a standard knowledge engineering manner to record specific processing details (and not general PSMs).

3.1.3 Acquire using IO-SCHEMAS

I-SCHEMAS and O-SCHEMAS are a method for structuring the collection of sentenceK for KBs [Debenham, 1998] In this approach, the conceptual model of a KB contains *items* connected by *objects* to describe *data*, *information*, and *rules*:

- Items are described in I-SCHEMAS
- Objects are described by O-SCHEMAS.
- Data are simple variables.
- Information are relations connecting variables.
- Rules that execute over the relations.

Schemas have a uniform format, no matter if they are representing data, information, or rules. They offer constraints and (as we shall see below) normalisation procedures that are analogous to database normalisation procedures. Schemas can be represented formally using the lambda calculus or informally in a simple tabular format. Schemas can contain other schema's recursively. Three i-schemas are shown in the i-schemas of Figure 24. The general structure is shown on the left. A simple example is shown in the middle. There can be no more than 100 part-numbers numbered between 1000 to 9999. A recursive shown on the right. The recursive example uses the middle example within its definition. Parts numbered under 2000 cost no more than 300 dollars. All members of the set parts must be in the set of part/cost-price (see the `forall` symbol). Horizontal lines identify components whose values determine the component marked with an `o`. For example, the last line of `part/cost-price` denotes that `cost-price` is functionally dependent on `part` (in traditional database jargon, `part` is a candidate key).

structure	
item-name	
var-i	var-j
x	y
meaning of item	
constraints on values	
set constraints	

e.g. part
part
-
x
isa(x:part-number)
$1000 < x < 9999$
≤ 100

e.g.part/cost-price	
part/cost-price	
part	cost-price
x	y
-	
cost(x,y)	
$x < 1999$ if $y \leq 300$	
$\forall$	
————-	o

Figure 24: Items in i-schemas.

structure	
object-name	
type-i	type-j
x	y
meaning of object	
constraints on object values	
object set constraints	

e.g. costs	
costs	
D^1	D^1
x	y
costs(x,y)	
$x < 1999$ if $y \leq 300$	
$\forall$	
————-	o

e.g. mark-up-rule		
part/cost-price		
I^2	I^2	D^1
(x,w)	(x,y)	(z)
$(w = z * y)$		
$w > y$		
$\forall$	$\forall$	
————-		o
o	————-	
————-	o	————-

Figure 25: Objects in o-schemas.

When two parts of the KB share the same basic wisdom, i-schemas require the wisdom be represented twice. For example:

- Suppose `part/cost-price` is computed from a function COSTS.
- Suppose also that another function MARK-UP-RULE uses `part/cost-price`.

Rather than represent COSTS twice (once in `part/cost-price` and once in `part/cost-price`), we can use the O-schemas shown in Figure 25.

A *coupling map* shows when schemas share some common structure. For example, the compiling map for `mark-up rule` is shown in Figure 26. This map can be used to control KM. If two items are linked in the map, then modifications to one implies that the other has to be checked again for correctness. Further, these links can be used to simplify maintenance:

- An item is said to be *decomposable* if it may be constructed from other items or objects.
- If all decomposable data, information, and knowledge is discarded, then the knowledge base is said to be *normalised* [Debenham, 1995].
- A normalised system is far simpler to maintain than an un-normalised system (a result first reported in the database community [Date, 1995]). In a normalised system, knowledge used in many places is stored only once. The time required to change knowledge is reduced since that knowledge is uniquely represented in a single place in the system.

3.1.4 Acquire using PSMs and Ontologies

Some CAKE tools support PSM acquiring. For example:

- SHELLEY [Wielinga *et al.*, 1992a] allows users to specify new KADS-style PSMs.

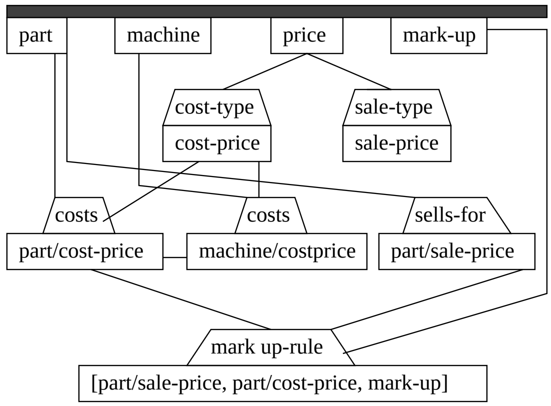

Figure 26: A simple concept map showing dependencies between schema concepts.

.

- van Harmelen & Aben [van Harmelen & Aben, 1996] offer a framework for mapping informal KADS-style PSMs into a formal language. Once expressed in this form, formal methods can be used to explore the specification (an example is given below, in §3.4.5).

However, a more usual use of meta-knowledge is to guide the acquire process. An observation made in the MOLE KA project [Kahn *et al.*, 1985] was that the acquire process can be significantly simplified with some knowledge of the inference structure of the KB. For example, in the MOLE work, it was found that exploring eight specific questions could lead to major improvements in the knowledge base (e.g. when considering X: "what events would rule out X?"). Chandrasekaran argued that these inference structures (later called PSMs) were generic across a range of application tasks [Chandrasekaran, 1983, Chandrasekaran, 1986]. Later research observed that each PSM implies an *ontological commitment*; i.e. requires certain data structures to operate. For example, if we were using the *exploreAndRefine* PSM of Figure 15, then we need to ask our domain experts about the data structures needed for that PSM: super-types and sub-types. Numerous researchers have explored building special-purpose editors which constrain the acquire process to the information required for a range of pre-defined PSMs and their ontological commitments. For example:

- RIME's KB editor [de Brug *et al.*, 1986, Soloway *et al.*, 1987] acquired parts of the KL-A-type meta-knowledge for the XCON computer configuration system [Bachant & McDermott, 1984]. RIME assumed that the KB comprised operator selection knowledge which controlled the exploration of a set of problem spaces. After asking a few questions, RIME could auto-generate complex executable rules.

- SALT's KA interface only collected information relating directly to its KL-B propose-and-revise inference strategy. Most of the SALT rules ($^{2130}/_{3062} \approx 70\%$) were auto-generated by SALT.

- If a user of the SHELLEY CAKE tool [Wielinga *et al.*, 1992a] decides that they are using the KL-B diagnosis PSM of Figure 14, then the acquire process can be directed towards collecting lists of known `obs` and `hyp`, and rules for `abstraction` and `causal` reasoning.

- Benjamin describes TINA, an automatic configuration device for configuring solutions to different problems in the context of KL-B diagnosis [Benjamins, 1994, Benjamins, 1995]. TINA's question generation system can guide the user to the appropriate selection of the PSM (see below, Figures Figure 27 and Figure 28).

- van Harmelen & Aben describe guidelines for mapping informal KL-B PSMs into their preferred style of formal models; e.g. for each inference (i) define a predicate and its arity; and (ii) for each argument of the predicate, define types. Their *transformation tool* guides the user through the application of these guidelines [van Harmelen & Aben, 1996].

Swartout & Gil argue that PSM customisation is the key to flexible acquire environments [Swartout & Gill, 1996]. PSMs and ontologies define the common useful structures and procedures within a domain. Knowledge engineers can build problem-specific shells that are specialised to certain PSMs and their ontological commitments. One the other hand, such problem-specific shells cannot be easily adapted to other tasks [marcus89]. While SALT's propose-and-revise PSM was successful in assisting users acquiring their knowledge, it could not be easily adapted to other PSMs. Tools like PROTEGE-II, and SPARK/ BURN/ FIREFIGHTER [Marques *et al.*, 1992] permit more flexibility in the PSM library. Users of the SBF, for example, can enter their configuration knowledge via a click-and-point editor of business process graphs [marques92]. Knowledge engineers can reconfigure SBF by building new business process graphs and connecting the to a library of reusable knowledge processing mechanisms (`eliminate, schedule, present, monitor, transform-01, transform-02, compare-01, compare-02, translate-01, translate-02, classify, select, dialog-mgr`). Runkel's approach divides acquire into numerous support tools (called *mechanisms for knowledge acquisition*, or MeKA) [Runkel, 1995]. Runkel's MeKAs are hand-built by the knowledge engineering. Marcus & Gil's EXPECT tool uses partial evaluation techniques to automatically detecting dependencies between knowledge and meta-knowledge, thus simplify the task of generating MEKA-like tools.

It is a debatable point if KL-A CAKE tools like RIME suffer from PSM inflexibility. On the one hand, they assume a single PSM (operator selection over a problem space traversal). On the other hand, this single PSM can be applied to numerous KL-B-type tasks by customising the operator selection rules. That is, an environment which supports KL-A operator customisation could be extensible to a wide range of applications (§3.4.2).

3.1.5 Acquire using Expert Critiquing Systems

Silverman discussing representation independent methods for the engineering of expert critiquing systems (ECS) [Silverman, 1990, Silverman, 1992b, Silverman, 1992a, Silverman & Wenig, 1993] which Silverman defines as follows:

> ... programs that first cause their user to maximise the falsifiability of their statements and then proceed to check to see if errors exist.

> A good critic program doubts and traps its user into revealing his or her errors. It then attempts to help the user make the necessary repairs [Silverman, 1992b].

Silverman's research focuses on defining "critiquing" as a add-on to existing systems. That is, Silverman views ECS as an domain-independent analysis technique with domain-dependent connections to existing systems. An ECS is a KB development support environment which offers a set of *cues* to its users. These cues encourage the user to move from useless issues which they are currently considering towards useful issues that they are currently ignoring. An *influencer* cue prevents an error happening. *Debiaser* cues fixes after errors have happened (§3.4.4). Silverman has some experimental results [Silverman & Wenig, 1993] suggesting negative feedback (e.g. the use of a debiaser) is almost always unsuitable, if only used without positive feedback (influencers). However, this results also suggest that as task complexity grows, debiasers are a useful adjunct to influencers.

Cues can be very simple domain-specific techniques such as:

- Help text that focuses user attention on certain issues;
- Colourisation techniques that avoid selective focusing on small problems.

More complex cues are *directors* (called "wizards" in the modern PC applications) that walk a user through a task which may be too difficult to perform on their own.

3.1.6 Other Acquire Techniques

Other miscellaneous acquire techniques include:

- van Harmelen & Aben [van Harmelen & Aben, 1996] proposal to allow users to access indexes on post-conditions defined within a system. So, if a user is seeking a certain function, they can ask "in what post-condition is this function achieved?".
- Object-oriented researchers [Gamma *et al.*, 1995, Buschmann *et al.*, 1996, Fowler, 1997, Coad *et al.*, 1997] have cataloged "patterns": portions of abstract conceptual models that have reappeared in numerous previous applications (KA researchers will recognise "patterns" as a synonym for "ontology"). Such patterns can be used to bootstrap a user some way into a new development. Knowledge-level patterns are PSMs [Menzies, 1998d].

All the above acquire systems collect words as a side-effect of their other acquiring activities. Shaw's repertory grids (Figure 19) are a unique KM option since they only acquire words. Other tools assist in acquiring the other knowledge types. QARCC and goal graphs allow for the acquiring of NFR quality knowledge. Easterbrook & Nuseibeh's VIEWER system [Easterbrook & Nuseibeh, 1996] allows for the expression of inconsistency detection knowledge.

QMOD/HT4 acquired its behaviouralK via a literature search over papers reporting experimental results in the field of neuroendocrinology. CBR builds behaviouralK via the incremental caching of parts of previous inferences. Automatic test generation tools can build the `IN`puts of behaviouralK (but some other source of knowledge must be consulted to determine the appropriate `OUT`puts). We argued above that sequence diagrams could be used for acquiring behaviouralK (§2.3.1). However, we note that this is rarely done.

RDR acquires its behaviouralK via *incremental capture*. Experts have a hard time enumerating large behaviouralK at one sitting. However, during some other process, an expert may offer a specific example to illustrate some argument. For example, the experience gained via the execution of a particular example may prompt an expert to offer an opinion on what the program should have done. Whenever such a specific example is generated, it is captured and added to the behaviouralK.

SocialK can be manually acquired via natural language discourse analysis or an anthropological study of individuals within a social organisation. Tools like BRAHMS assist in the storage and organisation of such socialK.

3.2 Operationalisation

We using the term "operationalisation" to denote the process of executing a KB either by direct interpretation or via compilation to some internal form.

Words that are function calls must be operationalised (i.e. the function/ method must be implemented). BehaviouralK are be said to be operationalised when they are used in test engines.

With the exception of the information systems tools and repertory grids, most of the KM techniques support operationalisation of sentences and PSMs. We will demonstrate how this can be done using the TINA system. TINA was a "proof-of-concept" prototype only and is not as sophisticated as (e.g.) PROTEGE-II [Eriksson *et al.*, 1995] or SBF [Marques *et al.*, 1992] (discussed below). However, the TINA technique is quite succinct and therefore ideal for demonstrating PSM operationalisation. In TINA, a solution is a PSM which must be configured for a particular *sub-context* using a set of *primitive inference techniques*. For example:

- Sub-contexts of diagnosis are defined by constraints within the domain such as the availability or absence of `simulation rules`.
- A primitive inference within `prediction based filtering` could be a set `intersection` sub-routine.

A TINA problem is described via *suitability criteria* categorised into a small number of *types*. For example `inference_rules` and `simulation_rules` are suitability criteria with the same type of `constraint_suspension_ method`. The TINA system can automatically reflect over a set of rules describing the transformation process from problems to solutions (or, in the language of TINA, types of suitability criteria into PSMs). A simplified version of some of TINA's rules is given in Figure 27. Types of suitability criteria are shown in the when sections. When executing a then section, if the PSM component is named in another rule, the reflection can recurse. For example, executing `symptom_detection` in `rule1` makes TINA test the suitability of `rule2` and `rule3`.

A sample fragment of TINA output is shown in Figure 28. The `trace_back_method` traces back the dependents of the broken component to find potential contributors to the fault. In the case of multiple contributors, TINA is saying that in this sub-context, they can be simply `intersected`. The resulting contributors set is assessed using the `corroboration method`. Innocent contributors are `deleted` (innocence is computed via running a high-level simulation). The remaining contributors are potentially guilty of the faults and another sub-routine is called to discriminate between them. Note that this `corroboration_method` was generated when TINA explored `prediction_based_filtering` in `rule5` of

```
rule1:diagnosis when prime_diagnostic_method
then symptom_detection and hypothesis_generation  and
     hypothesis_discrimination.

rule2:symptom_detection when ask_user_method
then apply_user_judgment.

rule3:symptom_detection when compare_symptom or detection_method
then generate_expectation and compare.

rule4:hypothesis_generation when empirical_hypothesis_generation_method
then associate and prediction_filter.

rule5:hypothesis_generation when model_based_hypothesis_method
then find_contributors and transform_to_hypothesis_set and
     prediction_based_filtering.

rule6:hypothesis_generation when hypothesis_generation_method
then select_hypothesis and collect_data and interpret_data.
```

Figure 27: Portions of the TINA rules used for converting problems descriptions into solutions. Adapted from [Benjamins, 1994].
.

Figure 27 (using rules not shown in this article). For full details of this example, see [Benjamins, 1994].

As a more sophisticated example of the above process, consider the SPARK/ BURN/ FIREFIGHTER (SBF) toolkit [Marques *et al.*, 1992]. SPARK builds a domain-specific CAKE tool that is tailored to the business information supplied by the user. BURN executes the CAKE tool and conducts a structured interview with the expert. This interview maps the business information offered by the user into a library of inference sub-routines (called mechanisms). The mapping process is guided by the KL-B-style PSM meta-knowledge. At choice points in the mapping, SBF can ask the user questions questions which select different PSMs. Once this mapping has been made, a rule base can be generate which solves the business problem. This is given to the FIREFIGHTER environment which assists the user in executing and debugging the operationalised program. Marques *et. al.* report significantly reduced development times for expert systems using the 13 mechanisms in the SBF toolkit. In the nine applications studied by Marques *et. al.*, development times changed from one to 17 days (using SBF) to 63 to 250 days (without using SBF) [Marques *et al.*, 1992].

3.3 Fault

We use the term "fault" to denote the process of recognising that an operationalised KB has produced the wrong behaviour. There are four standard techniques for faulting sentences and PSMs:

1. *Expert inspection*: A human operator surveying the output recognises some inappropriate output. Expert inspection uses tacit knowledge that has not been captured in our seven knowledge types.

2. *Using quality knowledge*. See §2.4.2.

3. *Using behaviouralK*: A KB is faulted if it can't offer explanations for members of the behaviouralK. This was the validation technique used

```
model_based_hypothesis_generation_method {
   trace_back_method;
   intersection_method;
   corroboration                          }

trace_back_method {
   find_upstream  }

intersection_method {
   intersection     }

corroboration_method {
   select_random;
   simulate_hypothesis;
   compare;
   delete           }
```

Figure 28: After exploring its problems/solution mappings, TINA can automatically generate a PSM for diagnosis. Adapted from [Benjamins, 1994] and converted into a procedural formalism.

by QMOD/HT4 (§2.3.3).

4. *Using inconsistency detection knowledge.*

The maintenance KBs may be heuristic and therefore may need maintenance. Despite this, we know of no research into faulting qualityK, and inconsistencyK. There are at least three ways to check an expert's behaviouralK:

1. The behaviouralK could reflect records of the known behaviour of the domain being modeled. However, in practice, such real-world data sets are rare. Menzies & Compton have argued that many domains tacked by KBS are very data-poor [Menzies & Compton, 1997].

2. A new KB2 could be used (and this would introduce the maintenance recursion problem). This KB2 could some how detect if the behaviouralK does not cover a representative sample of the domain.

3. If a test coverage tool reports that behaviouralK fails to exercise an adequate portion of the KB, then a fault could be reported.

A related activity to "fault" is "fault location"; i.e. determining exactly where within a KB has a problem occoured (§3.3.1). Fault localisation is a special case of "browse-around" (§3.3.2). A special class of "faults" is conflict detection amongst knowledge collected from different sources.

3.3.1 Fault Localisation

A general technique for fault localisation is dependency tracing. If the connections within a KB are known, and the inferencing has arrived in some unexpected part of those connections, then fault localisation can be implemented via a backwards search of the dependencies. This technique was used in many KM systems; e.g.:

- Davis's TEIREISIAS rule editor [Davis, 1976] for the MYCIN [Buchanan & Shortliffe, 1984] system.

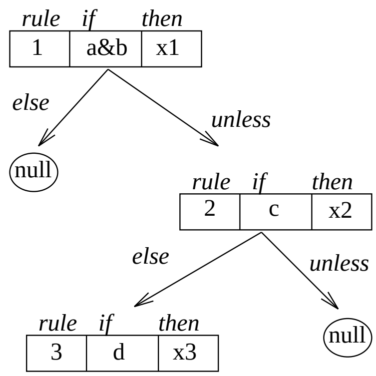

Figure 29: A RDR knowledge base

- Darden [Darden, 1990] discusses theory anomaly localisation based on an analysis of the development of genetic theory in the early part of this century. Anomaly localisation was a process of walking backwards from the final state back towards the initial state across directed causal links, inquiring at each point whether the intermediate state had been entered.
- Fault localisation is a explored extensively in the model-based diagnosis literature [Hamscher *et al.*, 1992].
- RDR is a representation optimised for fault localisation in KBS without PSMs [Compton *et al.*, 1989, Compton & Jansen, 1990, Compton *et al.*, 1992, Gaines & Compton, 1992, Mulholland *et al.*, 1996, Preston *et al.*, 1993, Richards & Compton, 1997, Richards & Menzies, 1997] RDR knowledge is organised into a *patch tree*. If a rule is found to be faulty, some patch logic is added on a *unless* link beneath the rule. The patch is itself a rule and so may be patched recursively. Whenever a new patch (rule) is added to an RDR system, the case which prompted the patch is included in the rule. These *cornerstone cases* are used below when fixing an RDR system (§3.4.1). At runtime, the final conclusion is the conclusion of the last satisfied rule. If that conclusion is faulty, then the fault is localised to the last satisfied rule. More details on RDR are given below (§3.4.1).

3.3.2 Browse-Around

"Browse-around" lets the user manually generate their own explanations of why a variable was/was not set. Browse-around is implemented via queries to the dependency information within a KB. The user starts at some point in the KB and explores the nearby region. For example:

- MYCIN's "how" and "why" queries allowed the user to start at a conclusion or a question and ask "how was that conclusion reached?" or "why are you asking me that question?". Both queries reported the dependency links in the neighborhood of the conclusion or question. How-queries looked upstream back towards the other literals that lead to this literal. Why-queries looked downstream to the word that is the current goal.

- SALT supported how-queries in the MYCIN-sense. However, SALT why-queries just returned canned text since "why" queries are best used in backward-chaining systems like MYCIN (SALT was a forward-chainer). SALT also supported "why not" and "what if" queries. Answers to "why not value X?" were generated by seeking other words that could lead to "X", but which were blocked somehow by known words. The system could answer (e.g.) "if Z=1 was set to Z=2, then we could have achieved X". A what-if-query was a hypothetical look downstream of some temporary setting of a variable. A technical complexity with what-if-queries is that variables set in this hypothetical experiment have to be reset when the hypothetical case is finished.

- Fensel & Schoenegge [Fensel & Schoenegge, 1997, Fensel & Schonegge, 1997] offer an interesting variant on "browse-around". Using an interactive theorem prover (KIV), they let a user browse-around a first-order theory representing the PSMs. If KIV cannot solve a problem, it identifies the missing logical formula that blocked PSM completion and presents this to the user as an assumption to be explored. Users of KIV can hence discover what extra assumptions are required to achieve their desired goals.

3.4 Fix

We use the term "fix" to denote the process of removing a fault in an operationalised KB. A special class of "fix" is conflict resolution for knowledge collected from different sources. General mechanisms for fixing include ripple-down strategies such as ripple-down-rules (§3.4.1) and ripple-down-rationality (§3.4.2), conflict negotiation (§3.4.3), ECS debiasers (§3.4.4), specialisation and/or generalisation (§3.4.5); machine learning (§3.4.6), CBR (§3.4.7); KA scripts (§3.4.8); amongst others, (§3.4.9).

3.4.1 Fixing Via RDR

The RDR representation is optimised for fault localisation and fixing. Once an expert has faulted a conclusion from an RDR system, they then ask the system for a list of possible patches. The system replies with a *difference list* which is calculated as follows. As the *current case* navigates down the RDR tree, if it finds a some satisfied rule, it then checks their *unless* patches (Figure 29). The different between the current case and the cornerstone case of the last satisfied rule is the difference list. For example, Figure 29 showed the rule `if a&b then x1` patched several times. In Figure 29, if `x2` is the correct conclusion when `a&b&c` is true, but incorrect when `c` is false, we add the logic delta `c` to a patch rule in the *unless* branch beneath `rdr1`.

In practice, RDR appears to work very well, at least for single classification problems. For example, the PIERS system at St. Vincent's Hospital, Sydney, models 20% of human biochemistry sufficiently well to make diagnoses that are 99% accurate [Preston *et al.*, 1993]. System development blends seamlessly with system maintenance since the only activity that the RDR interface permits is patching faulty rules in the context of the last error. For a 2000-rule RDR system, maintenance is very simple (a total of a few minutes each day). RDR has succeeded in domains where previous attempts, based on much higher-level constructs, never made it out of the prototype stage (e.g. [Patil *et al.*, 1981]). Further, while large expert systems are notoriously hard to maintain [de Brug *et al.*, 1986], the no-model approach of RDR has never encountered maintenance problems.

RDR is a unique KM strategy in two respects:

- It is the only KM strategy we know that supports the logical preserve knowledge processing activity (§3.5).
- It avoids the recursive maintenance problem of §2.5. An RDR KB is a very low-level structure and its fix knowledge is a single simple procedure (difference list generation). The testing of this single simple procedure can be done manually without requiring other KBs.

Nevertheless, RDR is not a solution to the full KM problem:

- RDR makes no statement on how to recognise and resolve conflicting knowledge from multiple sources.
- RDR does not support KM of the words used in its rules. For example, one medical RDR KB tests for `if tsh_is_high` where `tsh` is the thyroid stimulating hormone and `tsh_is_high` is generated by some feature extractors which are outside of the RDR KM environment. Preston [Preston *et al.*, 1993] describes one method for specifying RDR time-based feature extractors. However, the Preston environment does not include KM facilities beyond rapid acquire (§3.1.1).
- An RDR tree is not compatible with other common representation types (e.g. isa hierarchies, state charts). Consequently, an RDR tree cannot be initialised from pre-existing domain knowledge (but see Richards [Richards & Compton, 1997] and Lee [Lee & Compton, 1996] for experiments in reverse engineering classification hierarchies and causal data-flows from an RDR tree).
- RDR cannot process meta-knowledge such as KL-B PSMs. RDR is focused on the details of the specific case at hand. PSMs may not be expressible with respect to the specific problem at hand. For example, consider an RDR tree maintaining taxi driver knowledge. Our student taxi driver may learn many tricks about navigating from Sydney to Canberra through specific streets. However, in an RDR framework, she may never learn the generalised PSM: "open the map, find your current location, find your destination, compute the shortest distance path from here to there". When Mulholland *et. al.* [Mulholland *et al.*, 1996] used RDR to configure an ion chromatography system, they found they needed an extra control layer on top of a set of ripple trees. This top-level control knowledge was not maintainable within the framework of a standard ripples environment.

3.4.2 Fixing Via Extended Ripple Strategies

We are developing two extensions to RDR to handle:

1. Maintenance of wordK using ripple-down functions, (RDF) [Menzies, 1992]);
2. Maintenance of PSMs using ripple-down rationality, (RD-RA) [Menzies & Mahidadia, 1997]).

In RDF, the wordK within rules are functions that return some result. If, while patching some rule, the function is changed, the system keeps the old

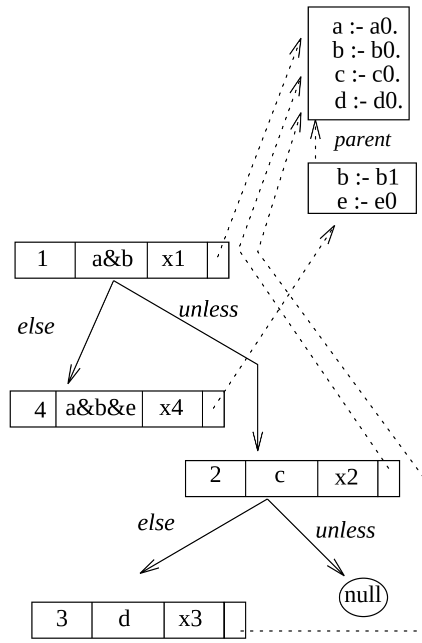

Figure 30: RDF= RDR plus a `Function` stack (top right).

definition plus the new one. Each rule and function is time-stamped and rules only use the functions developed up until they were written. For example, in Figure 30, suppose the `Function b` is found to be faulty in the case of `a&b&e`. A new RDR rule `rdr4` is added to our RDR tree. The resultant tree is shown in Figure 30. Each time stamp is a pointer into a function stack recording the history of function development. `Rdr4` will use the definition of function `a` that was developed prior to the creation of that rule. However, when it calls `b` or `e`, it will use the first definition of those functions it finds in the function stack (i.e. `b1, e0`). Meanwhile, `rdr1` will still use the old definition of `a` and `b`.

Menzies & Mahidadia's RD-RA KM proposal [Menzies & Mahidadia, 1997] is to model KL-A-style PSMs via RDRs that control QMOD/HT4 world generation. Recall that HT4 generates proofs (e.g. Figure 10) from a theory (e.g. Figure 9) and sorts them into consistent worlds (e.g. Figures 11 & 12). We have identified 5 *choice points* within HT4 that control the proof and world generation. Each level j can "see" the results of the levels i ($i < j$). Each level culls possible inferences; i.e. level j only processes the inferences approved by levels $1 \ldots j, j \leq i$:

1. A *level 1 vertex-level choice* looks one edge ahead of the current vertex and culls edges which do not look promising; e.g. their heuristic weight is too high or their heuristic certainty measure is too low.

2. A *level 2 proof-level choice* can access the proof generated generated up to this vertex and may cull an edge if (e.g.) that edge introduced a loop into the proof.

3. A *level 3 proofs-level choice* can access all the proofs generated up to this point and could implement (e.g.) beam-first search.

4. A *level 4 world-level choice* can access a world as soon as it is generated. A level 4 choice might be to stop the inferencing as soon as any sufficing world is generated.

5. A *level 5 worlds-level choice* can compare different worlds. We have argued [Menzies, 1996a] that a wide variety of KL-B tasks can be mapped into different choice operators. For example, validation is a level 5 worlds-level choice that favours the worlds which explain the greatest number of the `OUT`puts. This validation choice operator can also be used for least-cost planning by adding a level 3 proofs-level choice to favour least cost solutions.

Each choice operator is an expert system which classifies proposed inferences as `cull` or `accept`. The core of the Menzies & Mahidadia's KM proposal is a suite of RDR trees that maintain each of these choice operators. Fixing problems solving is this RD-RA framework is a process of maintaining the single classification RDR systems controlling the choice operators. Note that:

- Each choice operator must extract features from the space it is analysing. These feature extractors are maintained using RDF.
- RD-RA could be said to fault and fix PSM knowledge.

3.4.3 Fixing Using Conflict Resolution

This section discusses techniques for handling knowledge collected from different sources based on the following approach:

1. Explicitly represent the different viewpoints of different users;
2. Automatically detect conflicts between these viewpoints;
3. Offer tools for resolution support.

Resolving a conflict implies applying the fix knowledge discussed above (§3.4). Representation-specific strategies can also be used. Easterbrook & Nuseibeh [Easterbrook & Nuseibeh, 1996] discuss conflict resolution between state transition charts from different authors using the VIEWER system. For each inconsistency detection rule, there are associated repair actions. Note that each repair action is offered to the users who then negotiate if they wish to apply that fix. VIEWER supports a range of tools to assist that negotiation such as:

- The ability to view different possible KBs side-by-side;
- A *work record* that records the history of the changes to the KB. This work record can be browsed looking for some past decision that lead to the current conflict. Note that Easterbrook & Nuseibeh might call the REMAP discussion replay technique (§2.4.4) a work record.
- Demons for automating certain tasks; e.g. auto-copy parts of one author's KB into another author's KB. Note that we might call these demons KA scripts (§3.4.8).

It may not always be possible to fully resolve a conflict when it is detected. In this case, the KM problem becomes one of continuing in the presence of inconsistencies. Nuseibeh [Nuseibeh, 1997] reviews a range of domain-specific strategies for this problem. Inconsistencies can be *ignored*, but this is only appropriate for minor errors. If it is known that resolution will be possible in the near future (e.g. when some datum or resource becomes available), then the resolution can be *delayed*. For example, QARCC's conflict handling mechanism divides into three types of *delay* which generates annotation for:

1. "Clarification required": used for simple conflicts and may be resolved with a single email message.

2. "Discussion required": used for middle-sized conflicts which require a meeting.

3. "Analysis required": used for large-scale conflicts which require extensive time for reconsideration.

Note that these conflict handling mechanisms contain a limited model of social interaction knowledge (§2.4.4).

A special case of *delay* is *ameliorate* in which the inconsistency cannot be removed, but steps can be taken to improve the situation. Amelioration requires techniques for reasoning over KBs with inconsistencies. Classical single-world deductive logics are not appropriate. In classical logical, if a contradiction can be detected, the KB is said to be able to prove anything at all; i.e. the KB has become useless. Non-classical multiple-world logics such HT4 (§2.3) handled the inconsistencies that could be generated from theories like Figure 9 by sorting them into compatible worlds (Figures 11 and 12). We have argued that access to the consistent worlds that can be generated from a KB is a powerful knowledge engineering tool [Menzies, 1996a]. Comparing multiple worlds is also a tool for multi-expert conflict resolution [Menzies & Waugh, 1998]. Recall the dispute between *Dr. Thick* and *Dr. Thin* in Figure 9 about the effects of `foriegnSales` on `companyProfits`. The worlds of Figure 11 and Figure 12 tell us:

- *Dr. Thin*'s contributions can be found in two worlds; i.e. with respect to the problem of `OUT`puts= {`investorConfidenceUp, wagesRestraintUp, inflationDown`}, and `IN`puts= {`foriegnSalesUp, domesticSalesDown`}, a single author's opinions are inconsistent.

- Both authors contributions exist in the same consistent world (`W[1]`, Figure 11); i.e. the apparent conflict of *Dr. Thick* and *Dr. Thin* did not matter for the analysed problem. If this was true for all the analysed problems, then we could declare that for all practical purposed, *Dr. Thin* and *Dr. Thick* are not really disagreeing.

- *Dr. Thin* may wish to review their opinion that `foriegnSales` $\stackrel{--}{\to}$ `companyProfits` since, in terms of the studied problem, this proved to explain less of the required behaviour that *Dr. Thick*'s option that `foriegnSales` $\stackrel{++}{\to}$ `companyProfits`.

The multiple-worlds reasoning of HT4 has technical advantages over other conflict resolution approaches:

- Easterbrook [Easterbrook, 1991] lets users enter their requirements into an explicitly labeled viewpoints. He makes the simplifying assumption that all such viewpoints are internally consistent. HT4 has no need for this, potentially, overly-restrictive assumption. HT4 can handle inconsistencies within the opinions of a single user. That is, HT4 can analyse conflicts at a finer granularity than approaches based on manually-entered viewpoints (e.g Easterbrook or Finkelstein *et. al.* [Finkelstein *et al.*, 1994]).

- One of Easterbrook's SYNOPTIC tool only permits comparisons of two viewpoints [Easterbrook, 1991, p113]. HT4 can compare N viewpoints.

- We have found that it easier to build efficient implementations [Menzies, 1996a, Menzies, 1996b] using the above graph-based approach that using purely logical approaches (e.g. [Hunter & Nuseibeh, 1997]).

- HT4 places few restrictions on the representations it can process. It executes over any representation that can be mapped into directed and-or graphs, plus some invariants. Many common knowledge representations can be mapped into such graphs. For example, propositional rule bases can be viewed as graphs connecting words from the rule left-hand-side to the rule right-hand-side. More generally, horn clauses can be viewed as a graph where the conjunction of sub-goals leads to the head goal. In the special (but common) case where the range of all variables is known, this graph can be partially evaluated into a ground form. Once in the ground form, the literals (words) in the ground form can be viewed as vertices in an and-or graph.

3.4.4 Fixing via ECS Debiasers

ECS debiasers are a set of domain-specific cues for fixing problems after they arrive. ECS debiasers may be domain-specific techniques such as:

- Visual feedback in the form of colourisation, highlighting imperfections in the artifact;

- View argumentation including email exchanges to persuade the user that some claim lacks support.

Silverman discusses another class of generalised ECS debiasers that overlaps somewhat with the techniques discussed elsewhere in this paper. For example, ECS "re-use of similar cases" reads like a CBR-variant (§3.4.7).

3.4.5 Fixing via Specialisation and/or Generalisation

Specialisation/ generalisation is a general framework for repairing logic. The framework dates back to at least Shapiro [Shapiro, 1983]. Undesired behaviours can be removed by specialising a pre-condition; i.e. increasing the number of tests in a conjunction. Desired behaviour which was not achieved can be reached via generalising a pre-condition; i.e. decreasing the number of tests in a conjunction.

Representation-specific variants of this framework have appeared in various systems; for example, Shapiro's system, the work of van Harmelen & Aben, and the SEEK/SEEK2 systems. Shapiro's own system was a debugging facility for horn clauses. In that system, specialisation or generalisation means adding or removing (respectively) horn clause sub-goals. van Harmelen & Aben [van Harmelen & Aben, 1996] discuss formal methods for repairing KADS-style PSMs such as Figure 14. For example, Figure 14 can be formally represented as a mapping from data d to an hypothesis h via intermediaries Z and other data R_i (Equation 1).

$$abstract(data(d), R_1, obs(Z)) \wedge hypothesize(obs(Z), R_2, hyp(h)) \qquad (1)$$

There are three ways Equation 1 can fail:

1. We fail to prove $abstract(data(d), R_1, obs(Z))$; i.e. we are missing abstraction rules that map d to observations.

2. We fail to prove $hypothesize(obs(Z'), R_2, hyp(h))$; i.e. we are missing causal rules that map Z' to an hypothesis h.

3. We can prove either subgoal of Equation 1, but not the entire conjunction; i.e. there is no overlap in the vocabulary of Z and Z' such that $Z = Z'$.

Case #1 and #2 can be fixed by adding rules of the missing type. Case #3 can be fixed by adding rules which contain the overlap of the vocabulary of the possible Z values and the possible Z' values. More generally, given a conjunction of sub-goals representing a PSM, fixes can be proposed for any sub-goal or any variable that is used by > 1 sub-goal.

Fixing in the SEEK system used explicit fixK (Figure 21). SEEK rules can be specialised by adding tests/symptoms or deleting exclusions or decreasing its confidence level. Similarly, such rules can be generalised by removing tests/symptoms or adding exclusions or increasing its confidence level. SEEK worked in association with a human operator. Its successor, SEEK2, is a fully automatic system. Automatic specialisation/generalisation tools can be viewed as a special kind of machine learning algorithms (§3.4.6).

3.4.6 Fixing Via Machine Learning

Machine learning (ML) algorithms input some behaviouralK plus a background theory (which may be empty) and output a new theory which covers more of the behaviouralK than the initial background theory. ML techniques may be fully automatic (e.g. [Quinlan, 1982, Quinlan, 1986]) or utilise a human in the revision loop [Sammut & Banerji, 1986] [Winston, 1984, chpt 11]. ML can be divided into two broad camps: deductive and inductive [Michalski, 1993]:

- Inductive learners (e.g. genetic algorithms, neural nets, decision tree learners, belief networks) create a summary theory from the behaviouralK presented to them. Inductive techniques are data hungry: the efficacy of the inductively learnt theory can never be better than the quality of the input examples. The more examples, the better the theory. However, if the behaviouralK grows very large (i.e. thousands of examples or more), then considerable manual knowledge engineering effort may be required to prepare the data for the inductive learner [Williams & Huang, 1996]. Inductive learners typically make little use of a background theory (exception: inductive logic programming [Muggleton, 1991]). That is, a user may be presented with a totally novel theory at the end of an inductive learning session.

- In contrast, deductive learners (e.g. explanation-based generalisation [van Harmelen & Bundy, 1988, Mitchell *et al.*, 1986], chunking in rule-based systems [Laird *et al.*, 1986] and all the systems in §3.4.5) are less data hungry

```
input : eg = array[1..9514] of example
output: rdrError, id3Error

procedure compare begin
  rdrKB    := newRdrTree
  test     := eg[8001..9514]
  training := ∅
  for i := 1 to 8000  begin
    # extend the RDR KB
    rdrKB := rdr(rdrKB, eg[i])

    # create a new ID3 KB
    training := training + eg[i]
    id3KB    := id3(training)

    # assess the new versions of the KBs
    rdrError[i] := test(rdrKB, test)
    id3Error[i] := test(id3KB, test)
end end
```

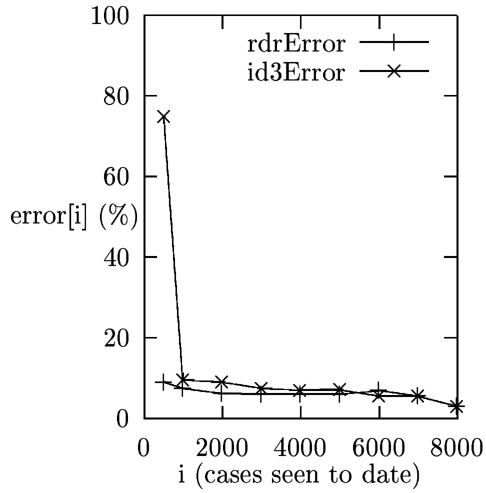

Figure 31: Comparing manual KA (RDR) vs automatic inductive learners (ID3). From [Mansuri *et al.*, 1991].

and make extensive use of the background theory. Deductive learners input the steps taken by some inference engine and output a better set of inference steps. For example, they may cull the middle portions of a long inference procedure to connect inputs directly to outputs.

In the terminology of this paper, machine learners typically operationalise fixing the sentenceK of a KB (§2.2). A common fix technique for inductive machine learners is to rewrite the knowledge from scratch. In the case of totally automatic induction, a case can be made that total rewrites are acceptable. However, in the case where a human is involved in the revision loop, total knowledge rewrites may be unacceptable. Users could treasure their favorite portions of their KB (typically, the ones they have developed and successfully defended from all critics). It would be unacceptable to permit a learning algorithm to scribble all over such treasured knowledge.

Deductive learners are more suited to human-in-the-loop maintenance for two reasons. Firstly, deductive learners can be constrained to generate revised theories that are similar to the initial theory. This minimises the shock to the user when the revised theory is presented. Secondly, inductive learners require more behaviouralK than deductive learners. Such libraries are hard to generate (recall §2.3). Often, these libraries have to be built via incremental caching (§3.1) and this can take many months. Mansuri *et. al.* [Mansuri *et al.*, 1991] compared manual KA using RDR versus automatic inductive learning using ID3 [Quinlan, 1982]. An example set of 9514 examples (from Garvin ES-1 [Compton *et al.*, 1989]) was divided into a test set of 1514 examples and a training set of 8000 examples. For each item in the test suite, an RDR tree was executed. If a human operator detected an incorrect classification, the tree was patched. This item was then added into an ID3 training set. The percentage errors of the current RDR tree and the ID3 tree are compared in Figure 31. This comparison shows that in the case of RDR vs ID3 in the Garvin ES-1 domain:

- Manual KA (RDR) generated KBs with much fewer errors than inductive learners (ID3) when only hundreds of examples were available.
- Only when thousands of examples are available (in this case, 5000) does manual KA and ML perform equally as well.

In general:

- When the behaviouralK is small (say, a few hundred examples), manual knowledge acquisition may generate a better KB than inductive ML.
- Machine learning techniques are not widely used by knowledge acquisition researchers (though see the short list of integrated ML/KA systems in [Webb & Wells, 1996]). Roughly speaking, this is because ML research has focused more on general-purpose inductive learners than deductive learners since the latter relies on more domain-specific modeling assumptions. Hence, ML has focused on the kind of learning algorithm that is least useful for knowledge acquisition.

3.4.7 Fixing Via Case-Based Reasoning

Kolodner [Kolodner, 1993, p. XV] observes that most CBR is simple indexed-based matching to flat files. However, a small number of *model-based CBR* systems offer facilities for retrieving and repairing old models stored in a case library. The fix techniques by CBR used are many and various. In this section, we offer one example of a CBR repair facility. See [Kolodner, 1993] for other facilities.

Recall Figure 8 which explained the symptoms of patient David in terms of aortic valve disease. Such an explanation could become an entry in the behaviouralK of (e.g.) the CASEY CBR system. A new patient called Newman now presents and CASEY recognises that David's case in Figure 8 is an old explanations that best explains Newman's condition. However, David's case does not explain all of Newman's symptoms and so CASEY consults its causal knowledge (§2.2) of cardiac behaviour. This causal knowledge tells CASEY that it is valid to (i) add the dashed edges in Figure 32 to connect David's explanation to Newman's extra symptoms; (ii) remove David's "murmur of as" symptom. This new explanation for Newman is then added to the behaviouralK of CASEY and is available for future inferencing.

Note that this CASEY model-based fix strategy is dependent on the presence of causal knowledge. In general, while CBR offers extensive facilities for fixing cases stored in as behaviouralK, it offers few clues on how to maintain its sentences (e.g. CASEY's causal knowledge) or its fixK (e.g. CASEY's strategies for partial matching of Newman's case to David, or the subsequent attempts at repair).

3.4.8 Fixing Via KA scripts

In the case where numerous changes have to be made to a PSM, if the user does not complete all those changes, then the PSM may be broken. Gil & Tallis [Gil & Tallis, 1997] use a scripting language to control the modification of a KL-B-style PSM to prevent broken knowledge. These KA scripts are controlled

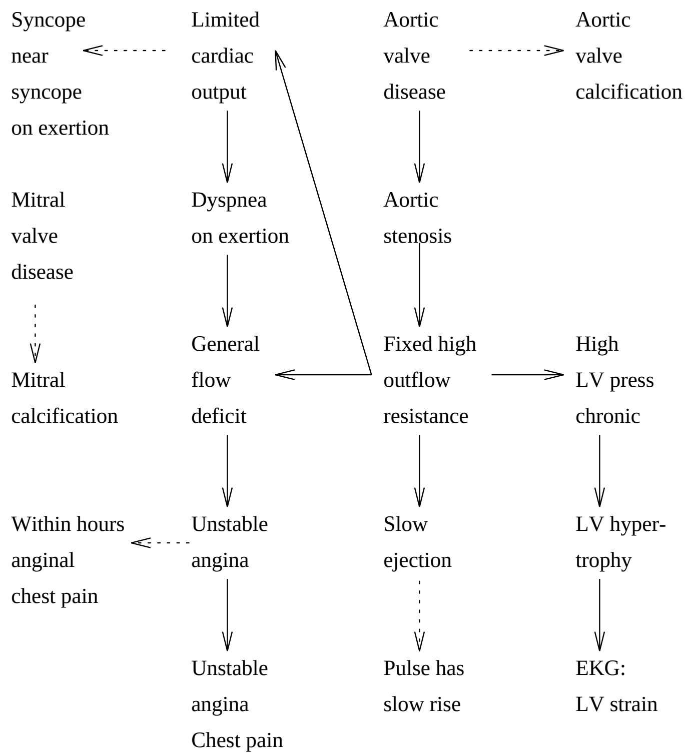

Figure 32: An explanation of Newman's symptoms using an edited version of the explanation of David's symptoms from Figure 8. Added edges are shown as dashed lines. David's "murmur of as" vertex has also been deleted. From [Kolodner, 1993, p419].

	Simple task #1				Harder task #2			
	no ETM		with ETM		no ETM		with ETM	
	S4	S1	S2	S3	S2	S 3	S1	S2
Total time (min)	25	22	19	15	74	53	40	41
Time completing transactions	16	11	9	9	53	32	17	20
Total changes	3	3	3	3	7	8	10	9
Changes made automatically	n/a	n/a	2	2	n/a	n/a	7	8

Figure 33: Change times for ETM with four subjects: S1…S4. From [Gil & Tallis, 1997]

by the EXPECT TRANSACTION MANAGER (ETM) which is triggered when EXPECT's partial evaluation strategy detects a fault. Figure 33 shows some speed up in maintenance times for two change tasks for EXPECT KBS, with and without ETM. Note that ETM performed some automatic changes (last row of Figure 33).

KA scripts are a procedural meta-knowledge (§2.4) which, if they contain bugs, will introduce numerous errors into the EXPECT KBs. If we try to check the KA scripts, then the recursive maintenance problem is encountered (§2.5).

3.4.9 Other Fix Strategies

Shaw (personal communication) reports that "fix" is very simple for wordK. Once the conflicting repertory grids are shown to the experts, they can quickly (i) see why they differ; and (ii) propose some revision to their terminology to resolve the difference in the wordK.

The MYCIN rule editor (TEIREISIAS) applied a clustering analysis to the rule base to determine what parameters where *related*; i.e. are often mentioned together. If proposed rules referred to a parameter, but not its related parameters, then TEIREISIAS would point out a possible error.

3.5 Preserve

Sometimes fixing one bug introduces two more. KM strategies that support the *preserve* task prevent a fix for problem A from introducing problems B,C,D,...

One preserve tool is a *design rationale* annotation that describes the reasons for a fix. Having access to such rationales is very important [Conklin & Begeman, 1988, Fischer *et al.*, 1989, Boy, 1995]. Studies with real-world maintainers show that the most important question a maintainer ever asks is *why did they do this, and not that?* [Moran & Carroll, 1996]. Argumentation structures can record prior debates [Conklin & Begeman, 1988, Lee & Lai, 1996, Klein, 1993, MacLean *et al.*, 1996]. The current state-of-the-art in design rationale defines annotation tools for wordK and sentenceK only.

Creating such argumentation structures can be very costly to build. To reduce that cost, some argumentation systems (e.g. [Fischer *et al.*, 1996]) tightly integrate the argumentation environment with the design environment:

- To check a new argument a simulation module is also offered which allows the user to make what-if queries.
- New arguments can be matched into a library of old arguments. Holes in the new argument can then be filled in automatically and offered back to the user (e.g. 'You said this before, is this what you mean now?').

Note that this functionality can be implemented in a HT4 framework (§2.3.3) as follows:

- Simulation models performing what-if queries is a synonym for generating multiple worlds in abduction (e.g. generating the worlds from the economics theory in Figure 9.).
- One commonly used argumentation representation [MacLean *et al.*, 1996] connects different options to assessment criteria via qualitative statements of *supports* and *objects-to*. To process this kind of argumentation, we need to build worlds containing consistent guesses about the implications of different options.

- Matching new arguments to old arguments is case-based reasoning which Leake argues is an abductive task; e.g. select the worlds containing the most number of things we have used successfully before [Leake, 1993].

Other preserve tools described below are ETM, ripple-down techniques, and data schema evolution (DSE). DSE is a software engineering preserve strategy. The DSE problem is that after some change to the logical model of the program, some parts of the program comply to the former version of the schema. The DSE problem is particularly acute in object-oriented databases used by many applications in which reorganisation is slow (due to the complexity of the schemas and the disc access times) and expensive (due to the cost of changing all the applications). While it is not currently acknowledged in literature, DSE will become a problem in the future for object-oriented KBs.

A common DSE technique is to:

1. Create form a chain connecting versions of the schema for each class;

2. Create coercion functions that map instances of class version i to become an instance of class version j (where j is somewhere later in the chain than i).

Odberg reviews a range of commercial and research OODBMS which offer partial solutions to the DSE problem [Odberg, 1995, chpt2]. He comments that most of these DSE solutions assume that changes are localised to within a class. The more general DSE problem of changes that transcend class boundaries remains an open research issue. Note that RDF (§3.4.2) could be viewed as a restricted version coercion KM technique. When rules are compiled, they can select their relevant functions from a chain of different versions of all functions.

Coercion down a chain of class version is a *procedural preserve* strategy and can be compared to the KA scripts of ETM (§3.4.8). However, ETM KA scripts only manage one schema at a time. When they convert a method to a revised method, they forget the old method. Nevertheless, ETM addresses the multiple class change problem that is generally missed by DSE. Using the dependency knowledge gained from EXPECT's partial evaluation of its KBs, ETM can control the change of numerous classes.

To be truly worthy of the *preserve* label, a KM strategy must demonstrate that what worked before some change will work after that change. Reasoning about procedural knowledge is harder than reasoning about declarative knowledge. Consequently, it is easier to prove the preserve property in a *logical preserve strategy* such as RDR than with procedural preserve strategies such as KA scripts or DSE version coercion. The ripple-down-rule tree representation contains two subtleties that make it an excellent preserve tool:

- Fixing any point in the RDR tree only changes the logic at that point. The rest of the tree is immune to any side-effects of that fix.

- Due to the manner in which the difference list is computed (§3.4.1), the user is constrained to only adding in *relevant* fix logic. That is, logic that reflects the difference between the cases seen previously and the case been examined currently.

Note that RDF is a procedural preserve strategy built on the logical preserve strategy of RDR.

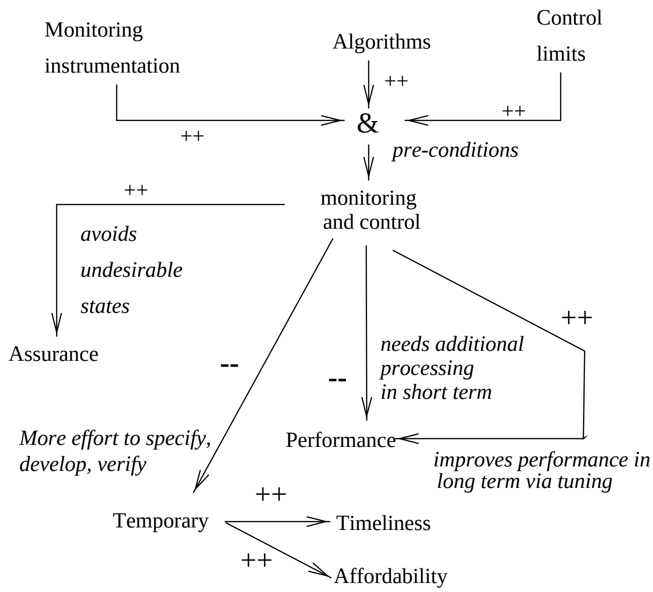

Figure 34: The NFRs of Figure 16 expressed as a dependency graph.

4 Discussion

In this article we have analysed the AI and software engineering literature to reverse engineer 35 points of a “knowledge management strategy options space”. We offer this space as a precise definition of KM. In this section we discuss what general lessons we can take from our exploration of that space:

- A basic requirement for many KM tools is *search space reflection*; i.e. analysing the network of connections between KB concepts (§4.1).

- Current KM research does not cover all the 35 points of the management strategy options space. We hence offer an extension to current KBS practice: commissioning a KM tool (§4.2).

4.1 Search Space Reflection: A General Tool for KM

Numerous KM strategies use a similar data structure; i.e. the dependency network between words in a KB. This section argues that a theme in many of the above techniques is *search space reflection*; i.e. a meta-level analysis of the pathways that an inference engine has or could take through some search space. For example:

- The coupling map of the IO-SCHEMAS approach is clearly a dependency network between KB words (Figure 26).

- Many structures used for KM can be expressed by a dependency network between KB words. For example, the non-functional requirements of Figure 16 can be expressed in Figure 34 without any loss of information.

- The clustering analysis that computed the *related parameters* of TEIREISIAS (§3.4.9) can be mapped into a graph-theoretic processing of a dependency network within a KB.

- The queries "how", "why", "why not" and "what if" can be mapped into traversals of the dependency links in the neighborhood of some word (§3.3.2).

- Fault-localisation is a process of walking backwards from the fault along the dependency network (§3.3.1).

- Inputs for test suites can be generated and assessed by an analysis of this dependency network (§2.3.4).

- Techniques from the verification community can detect anomalies in the KB via an analysis of this network (Figure 17).

- The computational core of many deductive learners (§3.4.6) is a partial evaluator that generates, then edits, the search space relevant to some problem [van Harmelen & Bundy, 1988].

- The core of model-based CBR is a reflective process on portions of the search space which have been useful in the past for generating useful explanations (§3.4.7).

- Swartout & Gil's claim [Swartout & Gill, 1996] of the generality of EXPECT over other KL-B tools such as PROTEGE-II [Musen & Tu, 1993] and SALT [Marcus & McDermott, 1989] rests on the ability of EXPECT to automatically derive the dependencies between domain knowledge and PSMs.

- Sequence diagrams (Figure 7) are pictures of portions of the search space exercised by a particular example.

Sometimes, only parts of the portions of this network may be available to the KM strategy; i.e. only those portions exercised by the current example. Nevertheless, this partial network may be sufficient for:

- Finding errors in the KB; e.g. the EXPECT error detector (§2.4.2).

- Finding assumptions which, if changed, can repair a KL-B PSM; e.g. the Fensel KIV approach (§3.3.2) and RD-RA (§3.4.2)

Both the Fensel KIV approach and RD-RA rely on assumption management. Assumption management in search space reflection is a powerful KM technique. Often there are insufficient examples available to unambiguously generate all of the network. However, this need not block search space reflection. If the meta-level analyser can make assumptions, then portions of the network around the known measurements can be guessed. For example, the core of Fensel's technique is a search for guesses that by-pass a block in the PSM reasoning. In the case of contradictory assumptions, then we need to separate the search space into consistent portions; e.g. the worlds generated in Figures 11 and 12. If we have such worlds available, then we can support a range of conflict resolution tools:

- e.g. the *Dr. Thick* vs. *Dr. Thin* discussion in §3.4.3.

- HT4 could directly process Figure 34 looking for the worlds that cover most of the desired goals; e.g. the most of `assurance` and `performance` and `timeliness` and `affordability`.

Knowledge type	Knowledge Processing Activity				
	Acquire	Operationalise	Fault	Fix	Preserve
WordK	all	most	few	few	few
SentenceK	most	most	many	many	few
BehaviouralK	some	few	few	few	none
PSMs	some	many	few	few	few
QualityK	few	few	none	none	none
FixK	few	many	none	none	none
SocialK	few	none	none	none	none

Figure 35: The 35 points in the *knowledge management options space* are covered by all, most, many, some, one, or none of the systems found by this review.

- The core of RD-RA technique is a structured patching of the control procedures used to process the assumptions. We argue that this process is actually a general technique for unifying software engineering, KA and KM. In the our approach, KL-B is replaced with libraries of multiple world search space control devices maintained by RDF [Menzies, 1996a, Menzies, 1998a].

Note that search space reflection is just an extension to Newell's original knowledge-level proposal (§2.4). If a knowledge-level agent has access to the search space and rationality operators of another knowledge level agent, then the first agent can assist in maintaining the second. However, our reading of Newell's proposal is that Newell viewed rationality as a local-choice procedure (e.g. levels 1 and 2 of RD-RA §3.4.2). Full search space reflection may require non-local reflection (e.g. levels 3,4,5 of RD-RA).

4.2 Commissioning a KM Tool

This section argues that there is an incompleteness in current KM research. To address this problem, a process is described for commissioning a new KM tool. Figure 35 shows the coverage of the 35 kinds of KM which we can find in the literature. To date, no strategy covers all 35 kinds of KM. Many strategies focus on only small portions of the space. Also, no system covers 12 of the 35 possible points; i.e. current research ignores at least $^{12}/_{35} > {}^{1}/_{3}$ of the KM problem. Further, KM research makes the RAS or the operationalisation KM assumptions, which we have argued may be an incomplete approach to KM (§2.3.5, §3.1.1). In our own research, the recursive maintenance problem (§2.5) discourages us from building complicated architectures for KM. Instead, we seek a small number of general mechanisms for KM which we can manually validate (e.g. RD-RA).
Given the framework of this article, commissioning a specific KM tool is a four stage process:

1. A statement of what search space reflection techniques are supported in the tool; i.e. where in the 35 points of the knowledge management options space does this system work?

2. A statement of what position this tool takes on the recursive maintenance problem (§2.5); i.e. what tools are offered for maintaining the maintenance KB?

3. A theoretical demonstration that the tool could support the preserve activity (§3.5).

4. A practical demonstration that:
   - The quality knowledge can assess the KB;
   - KM metrics can be generated from the tool; i.e. we can track the change in the quality of the KB over time.

Point four is particularly important. The empirical demonstration of the merit of the different systems mentioned above is an open and pressing research issue. We have commented elsewhere on the poor state of the art in KA (lack of an active refutable hypothesis; insufficient data collected to satisfy statistical analysis; experiments do not control for process, product, resource variations) [Menzies, 1998b]. We make no further comment here except that current KE practice rarely acquires qualityK. Without this qualityK we cannot assess the success of a KM strategy since even if achieve reuse levels of 100% or development times of a few days, we may still be producing inappropriate systems.

Acknowledgments

The comments of the anonymous referees clarified many points of this paper. Paul Compton has motivated and supported our KM work for many years now. This particular paper grew from (i) discussions with Dieter Fensel on comparing KM techniques; (ii) an attempt to emulate the analysis style of the Pos *et. al.* review [Pos *et al.*, 1997] on redesign techniques (that article focused on early lifecycle issues while this article has tried to expand to cover more of the life cycle). Dieter Fensel was kind enough to offer detailed comments on many aspects of this paper. Simon Goss first articulated the recursive maintenance knowledge problem. Didar Zowghi was my guide to the RM literature.

References

[Agnew *et al.*, 1993] Agnew, N., Ford, K., & Hayes, P. (1993). Expertise in Context: Personally Constructed, Socially elected, and Reality-Relevant? *International Journal of Expert Systems*, 7.

[Angele *et al.*, 1996] Angele, J., Fensel, D., & Studer, R. (1996). Domain and Task Modelling in MIKE. In et.al., A. S., (Ed.), *Domain Knowledge for Interactive System Design*. Chapman & Hall.

[Bachant & McDermott, 1984] Bachant, J. & McDermott, J. (1984). R1 Revisited: Four Years in the Trenches. *AI Magazine*, pages 21–32.

[Benjamins, 1994] Benjamins, R. (1994). On a Role of Probem Solving Methods in Knowledge Acquisition- Experiments with Diagnostic Strategies. In *Proceedings of the European Knowledge Acquisition Workshop, 1994*.

[Benjamins, 1995] Benjamins, R. (1995). Problem-Solving Methods for Diagnosis and their Role in Knowledge Acquisition. *International Journal of Expert Systems: Research & Applications*, 8(2):93–120.

[Boehm, 1996] Boehm, B. (1996). Aims for Indentifying Conflicts Among Quality Requirements. In *IEEE Software*.

[Booch, 1996] Booch, G. (1996). *Object Solutions: Managing the Object-Oriented Project*. Addison-Wesley.

[Booch *et al.*, 1997] Booch, G., Jacobsen, I., & Rumbaugh, J. (1997). *Version 1.0 of the Unified Modeling Language*. Rational. http://www.rational.com/ot/uml/1.0/index.html.

[Boy, 1995] Boy, G. (1995). Supportability-based design rationale. In *Proceedings of the 6th IFAC Symposium on Analysis, Design and Evaluation of Man-Machine Systems*.

[Bradshaw *et al.*, 1991] Bradshaw, J., Ford, K., & Adams-Webber, J. (1991). Knowledge Representation of Knowledge Acquisition: A Three-Schemata Approach. In *6th AAAI-Sponsored Banff Knowledge Acquisition for Knowledge-Based Systems Workshop, ,October 6-11 1991, Banff, Canada*, pages 4.1 – 4.25.

[Breuker & de Velde (eds), 1994] Breuker, J. & de Velde (eds), W. V. (1994). *The CommonKADS Library for Expertise Modelling*. IOS Press, Netherlands.

[Buchanan & Shortliffe, 1984] Buchanan, B. & Shortliffe, E. (1984). *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley.

[Buschmann *et al.*, 1996] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M. (1996). *A System of Patterns: Pattern-Oriented Software Architecture*. John Wiley & Sons.

[Bylander *et al.*, 1991] Bylander, T., Allemang, D., M.C. Tanner, M., & Josephson, J. (1991). The Computational Complexity of Abduction. *Artificial Intelligence*, 49:25–60.

[Catlett, 1991] Catlett, J. (1991). Inductive learning from subsets or Disposal of excess training data considered harmful. In *Australian Workshop on Knowledge Acqusition for Knowledge-Based Systems, Pokolbin*, pages 53–67.

[Chandrasekaran, 1983] Chandrasekaran, B. (1983). Towards a Taxonomy of Problem Solving Types. *AI Magazine*, pages 9–17.

[Chandrasekaran, 1986] Chandrasekaran, B. (1986). Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design. *IEEE Expert*, pages 23–30.

[Chandrasekaran *et al.*, 1992] Chandrasekaran, B., Johnson, T., & Smith, J. W. (1992). Task Structure Analysis for Knowledge Modeling. *Communications of the ACM*, 35(9):124–137.

[Chung & Nixon, 1995] Chung, L. & Nixon, B. (1995). Dealing with Non-Functional Requirements: Three Experimental Studies of a Process-Oriented Approach. In *Proceedings of ICSE '95: the International Conference on Software Engineering*, pages 25–36.

[Clancey, 1985] Clancey, W. (1985). Heuristic Classification. *Artificial Intelligence*, 27:289–350.

[Clancey, 1989] Clancey, W. (1989). Viewing Knowledge Bases as Qualitative Models. *IEEE Expert*, pages 9–23.

[Clancey, 1992] Clancey, W. (1992). Model Construction Operators. *Artificial Intelligence*, 53:1–115.

[Clancey, 1993] Clancey, W. (1993). Situated Action: A Neuropsychological Interpretation (Response to Vera and Simon). *Cognitive Science*, 17:87–116.

[Clancey *et al.*, 1996] Clancey, W., Sachs, P., Sierhuis, M., & van Hoof, R. (1996). Brahms: Simulating Practice for Work Systems Design. In Compton, P., Mizoguchi, R., Motoda, H., & Menzies, T., (Eds.), *Proceedings PKAW '96: Pacific Knowledge Acquisition Workshop*. Department of Artificial Intelligence.

[Coad *et al.*, 1997] Coad, P., North, D., & Mayfield, M. (1997). *Object Models: Strategies, Patterns, and Applications*. Prentice Hall.

[Cohen, 1995] Cohen, P. (1995). *Empirical Methods for Artificial Intelligence*. MIT Press.

[Compton *et al.*, 1992] Compton, P., Edwards, G., Srinivasan, A., Malor, P., Preston, P., Kang, B., & Lazarus, L. (1992). Ripple-down-rules: Turning Knowledge Acquisition into Knowledge Maintenance. *Artificial Intelligence in Medicine*, 4:47–59.

[Compton *et al.*, 1989] Compton, P., Horn, K., Quinlan, J., & Lazarus, L. (1989). Maintaining an Expert System. In Quinlan, J., (Ed.), *Applications of Expert Systems*, pages 366–385. Addison Wesley.

[Compton & Jansen, 1990] Compton, P. & Jansen, R. (1990). A Philosophical Basis for Knowledge Acquisition. *Knowledge Acquisition*, 2:241–257.

[Conklin & Begeman, 1988] Conklin, J. & Begeman, M. (1988). gIBIS: A Hypertext Tool for Exploratory Policy Discussion. *ACM Transactions on Office Information Systems*, 6:303–331.

[Connell & Menzies, 1996] Connell, M. & Menzies, T. (1996). Quality Metrics: Test Coverage Analysis for Smalltalk. In *Tools Pacific, 1996, Melbourne*.

[Corporation, 1997] Corporation, R. S. (1997). Rational-Rose. `http://www.rational.com`.

[Darden, 1990] Darden, L. (1990). Diagnosing and Fixing Faults in Theories. In Sharager, J. & Langley, P., (Eds.), *Computational Models of Scientific Discovery and Theory Formation*. Morgan Kaufmann Publishers Inc.

[Date, 1995] Date, C. (1995). *An Introduction to Database Systems*, volume 6. Addison-Wesley.

[Davis, 1976] Davis, R. (1976). *Applications of Meta-Level Knowledge to the Construction, Mainteance and Use of Large Knowledge Bases*. PhD thesis, Stanford.

[Davis *et al.*, 1993] Davis, R., Shrobe, H., & Szolovits, P. (1993). What is a Knowledge Representation? *AI Magazine*, pages 17–33.

[de Brug *et al.*, 1986] de Brug, A. V., Bachant, J., & McDermott, J. (1986). The Taming of R1. *IEEE Expert*, pages 33–39.

[Debenham, 1995] Debenham, J. (1995). Understanding Expert Systems Maintenance. In *Proceedings Sixth International Conference on Database and Expert Systems Applications DEXA'95, London, September*.

[Debenham, 1998] Debenham, J. (1998). *Knowledge Engineering: Unifying Knowledge Base and Database Design*. Springer-Verlag.

[DeKleer, 1986] DeKleer, J. (1986). An Assumption-Based TMS. *Artificial Intelligence*, 28:163–196.

[DeKleer & Williams, 1987] DeKleer, J. & Williams, B. (1987). Diagnosing Multiple Faults. *Artificial Intelligence*, 32:97–130.

[Easterbrook, 1991] Easterbrook, S. (1991). *Elicitation of Requirements from Multiple Perspectives*. PhD thesis, Imperial College of Science Technology and Medicine, University of London. Available from `http://research.ivv.nasa.gov/~steve/papers/index.html`.

[Easterbrook & Nuseibeh, 1996] Easterbrook, S. & Nuseibeh, B. (1996). Using Viewpoints for Inconsistency Management. *BCS/IEE Software Engineering Journal*, pages 31–43.

[Eriksson *et al.*, 1995] Eriksson, H., Shahar, Y., Tu, S. W., Puerta, A. R., & Musen, M. A. (1995). Task Modeling with Reusable Problem-Solving Methods. *Artificial Intelligence*, 79(2):293–326.

[Eshghi, 1993] Eshghi, K. (1993). A Tractable Class of Abductive Problems. In *IJCAI '93*, volume 1, pages 3–8.

[Feldman *et al.*, 1989] Feldman, B., Compton, P., & Smythe, G. (1989). Hypothesis Testing: an Appropriate Task for Knowledge-Based Systems. In *4th AAAI-Sponsored Knowledge Acquisition for Knowledge-based Systems Workshop Banff, Canada*.

[Fensel, 1995] Fensel, D. (1995). Formal Specification Languages in Knowledge and Software Engineering. *The Knowledge Engineering Review*, 10(4).

[Fensel & Schoenegge, 1997] Fensel, D. & Schoenegge, A. (1997). Hunting for Assumptions as Developing Method for Problem-Solving Methods. In *Workshop on Problem-Solving Methods for Knowledge-based Systems, IJCAI '97, August 23*.

[Fensel & Schonegge, 1997] Fensel, D. & Schonegge, A. (1997). Using KIV to Specify and Verify Architecture of Knowledge-Based Systems. In *Proceedings of the 12th IEEE International Conference on Automated Software Engineering (ASEC-97), Incline Village, Nevada, Nov 3-5*.

[Finkelstein *et al.*, 1994] Finkelstein, A., Gabbay, D., Hunter, A., Kramer, J., & Nuseibe, B. (1994). Inconsistency Handling In Multi-Perspective Specification. *IEEE Transactions on Software Engineering*, 20(8):569–578.

[Fischer *et al.*, 1996] Fischer, G., Lemke, A., McCall, R., & Morch, A. (1996). Making Argumentation Serve Design. In Moran, T. & Carroll, J., (Eds.), *Design Rationale: Concepts, Techniques, and Use*, pages 267–293. Lawerence Erlbaum Associates.

[Fischer *et al.*, 1989] Fischer, G., McCall, R., & Morch, A. (1989). Design environments for constructive and argumentative design. In *CHI '89*.

[Fowler, 1997] Fowler, M. (1997). *Analysis Patterns: Reusable Object Models*. Addison Wesley.

[Gaines & Compton, 1992] Gaines, B. & Compton, P. (1992). Induction of Ripple Down Rules. In *Proceedings, Australian AI '92*, pages 349–354. World Scientific.

[Gaines & Shaw, 1989] Gaines, B. & Shaw, M. (1989). Comparing the Conceptual Systems of Experts. In *IJCAI '89*, pages 633–638.

[Gamma *et al.*, 1995] Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley.

[Gil & Melz, 1996] Gil, Y. & Melz, E. (1996). Explicit Representations of Problem-Soving Strategies to Support Knowledge Acquisition. In *Proceedings AAAI' 96.*

[Gil & Tallis, 1997] Gil, Y. & Tallis, M. (1997). A Script-Based Approach to Modifying Knowledge Bases. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97).*

[Ginsberg, 1987] Ginsberg, A. (1987). A new Approach to Checking Knowledge Bases for Inconsistency and Redundancy. In *Proc. 3rd Annual Expert Systems in Government Conference*, pages 102–111.

[Ginsberg, 1990] Ginsberg, A. (1990). Theory Reduction, Theory Revision, and Retranslation. In *AAAI '90*, pages 777–782.

[Ginsberg *et al.*, 1988] Ginsberg, A., Weiss, S., & Politakis, P. (1988). Automatic knowledge base refinement for classification systems. *Artificial Intelligence*, 35:197–226.

[Glass & Mackey, 1988] Glass, L. & Mackey, M. (1988). *From Clocks to Chaos.* Princeton University Press.

[Gomez-Perez, 1996] Gomez-Perez, A. (1996). Towards a Framework to Verify Knowledge Sharing Technology. *Expert Systems with Applications*, 11(4):519–29.

[Gruber, 1993] Gruber, T. (1993). A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 5(2):199–220.

[Hamscher *et al.*, 1992] Hamscher, W., Console, L., & DeKleer, J. (1992). *Readings in Model-Based Diagnosis.* Morgan Kaufmann.

[Harel, 1995] Harel, D. (1995). On Visual Formalisms. In Glasgow, J. & N.H. Narayanan, B. C., (Eds.), *Diagrammatic Reasoning*, pages 235–271. The AAAI Press.

[Haynes *et al.*, 1995] Haynes, P., T.Menzies, & Phipps, G. (1995). Using The Size of Classes and Methods as the Basis for Early Effort Prediction; Empirical Observations, Initial Application; A Practitioners Experience Report. In *OOPSLA Workshop on OO Process and Metrics for Effort Estimation.*

[Hoffman *et al.*, 1997] Hoffman, R., Feltovich, P., & Ford, K. (1997). A General Framework for Conceiving of Expertise in Expert Systems in Context. In Feltovich, P., Ford, K., & Hoffman, R., (Eds.), *Expertise in Context*, chapter 24, pages 543–580. MIT PRess.

[Hofstadter, 1980] Hofstadter, D. (1980). *Gödel, Escher, Bach: An Eternal Golden Braid.* Penguin Books.

[Hunter & Nuseibeh, 1997] Hunter, A. & Nuseibeh, B. (1997). Analysing Inconsistent Specifications. In *International Symposium on Requirements Engineering*, pages 78–86.

[Jacobson & Christerson, 1995] Jacobson, I. & Christerson, M. (1995). A Growing Consensus on Use Cases. *JOOP*, pages 15–19.

[Jacobson *et al.*, 1992] Jacobson, I., Christerson, M., Jonsson, P., & Overgaard, G. (1992). *Object-Oriented Software Engineering: A Use Case Driven Approach.* Addison-Wesley.

[Kahn *et al.*, 1985] Kahn, G., Nowlan, S., & McDermott, J. (1985). Strategies for Knowledge Acquisition. *IEEE Trsansactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-7:511–522.

[Klein, 1993] Klein, M. (1993). Capturing Design Rationale in Concurrent Engineering Teams. *IEEE Computer*, 26(1):39–47.

[Kolodner, 1991] Kolodner, J. (1991). Improving Human Decision Making Through Case-Based Decision Aiding. *AI Magazine*, page 68.

[Kolodner, 1993] Kolodner, J. (1993). *Case-Based Reasoning.* Morgan Kaufmann.

[Laird *et al.*, 1986] Laird, P. S., J. E., R., & Newell, A. (1986). Chunking in SOAR: The Anatomy of a General Learning Mechanism. *Machine Learning*, 1(1):11–46.

[Leake, 1993] Leake, D. (1993). Focusing Construction and Selection of Abductive Hypotheses. In *IJCAI '93*, pages 24–29.

[Lee & Lai, 1996] Lee, J. & Lai, K. (1996). What's in Design Rationale? In Moran, T. & Carroll, J., (Eds.), *Design Rationale: Concepts, Techniques, and Use*, pages 21–52. Lawerence Erlbaum Associates.

[Lee & Compton, 1996] Lee, M. & Compton, P. (1996). From Heuristic to Causality. In *Proceedings of the 3rd World Congress on Expert System (WcES'96)*.

[Levins & Puccia, 1985] Levins, R. & Puccia, C. (1985). *Qualitative Modeling of Complex Systems: An Introduction to Loop Analysis and Time Averaging.* Harvard University Press, Cambridge, Mass.

[MacLean *et al.*, 1996] MacLean, A., Young, R., Bellotti, V., & Moran, T. (1996). Questions, options and criteria: Elements of design space analysis. In Moran, T. & Carroll, J., (Eds.), *Design Rationale: Concepts, Techniques, and Use*, pages 53–106. Lawerence Erlbaum Associates.

[Mansuri *et al.*, 1991] Mansuri, Y., Compton, P., & Sammut, C. (1991). A comparison of a manual knowledge acquisition method and an inductive learning method. In Boose, J., Debenham, J., Gaines, B., & Quinlan, J., (Eds.), *Australian workshop on knowledge acquisition for knowledge based systems, Pokolbin*, pages 114–132. University of Technology, Sydney.

[Marcus & McDermott, 1989] Marcus, S. & McDermott, J. (1989). SALT: A Knowledge Acquisition Language for Propose-and-Revise Systems. *Artificial Intelligence*, 39:1–37.

[Marick, 1997] Marick, B. (1997). The Testing Tools Supplier List `http://www.stlabs.com/marick/faqs/tools.htm`.

[Marques *et al.*, 1992] Marques, D., Dallemagne, G., Kliner, G., McDermott, J., & Tung, D. (1992). Easy Programming: Empowering People to Build Their Own Applications. *IEEE Expert*, pages 16–29.

[Menzies, 1992] Menzies, T. (1992). Maintaining Procedural Knowledge: Ripple-Down-Functions. In *Proceedings of AI '92, Australia*.

[Menzies, 1995a] Menzies, T. (1995a). Limits to Knowledge Level-B Modeling (and KADS). In *Proceedings of AI '95, Australia*. World-Scientific.

[Menzies, 1995b] Menzies, T. (1995b). *Principles for Generalised Testing of Knowledge Bases*. PhD thesis, University of New South Wales. Avaliable from `http://www.cse.unsw.edu.au/~timm/pub/docs/95thesis.ps.gz`.

[Menzies, 1996b] Menzies, T. (1996b). On the Practicality of Abductive Validation. In *ECAI '96*. Available from `http://www.cse.unsw.edu.au/~timm/pub/docs/96abvalid.ps.gz`.

[Menzies, 1998a] Menzies, T. (1998a). Applications of Abduction: A Unified Framework for Software and Knowledge Engineering. Submitted to APWISE '98. Available from `http://www.cse.unsw.edu.au/~timm/pub/docs/98apwise`.

[Menzies, 1998b] Menzies, T. (1998b). Evaluation Issues for Problem Solving Methods. Banff KA workshop, 1998. Available from `http://www.cse.unsw.edu.au/~timm/pub/docs/97eval`.

[Menzies, 1998c] Menzies, T. (1998c). Evaluation Issues with Critical Success Metrics. In *Banff KA '98 workshop*. Available from `http://www.cse.unsw.EDU.AU/~timm/pub/docs/97evalcsm`.

[Menzies, 1998d] Menzies, T. (1998d). OO Patterns: Lessons from Expert Systems. *Software Practice & Experience*. In press. Available from `http://www.cse.unsw.edu.au/~timm/pub/docs/97probspatt.ps.gz`.

[Menzies, 1996a] Menzies, T. (September, 1996a). Applications of Abduction: Knowledge Level Modeling. *International Journal of Human Computer Studies*, 45:305–355. Available from `http://www.cse.unsw.edu.au/~timm/pub/docs/96abkl1.ps.gz`.

[Menzies *et al.*, 1992] Menzies, T., Black, J., Fleming, J., & Dean, M. (1992). An Expert System for Raising Pigs. In *The first Conference on Practical Applications of Prolog*. Available from `http://www.cse.unsw.EDU.AU/~timm/pub/docs/ukapril92.ps.gz`.

[Menzies & Clancey, 1999] Menzies, T. & Clancey, B. (1999). Editorial, Special Issue on Situated Cognition, International Journal of Human-Computer Studies. To appear.

[Menzies & Compton, 1997] Menzies, T. & Compton, P. (1997). Applications of Abduction: Hypothesis Testing of Neuroendocrinological Qualitative Compartmental Models. *Artificial Intelligence in Medicine*, 10:145–175. Available from `http://www.cse.unsw.edu.au/~timm/pub/docs/96aim.ps.gz`.

[Menzies & Mahidadia, 1997] Menzies, T. & Mahidadia, A. (1997). Ripple-Down Rationality: A Framework for Maintaining PSMs. In *Workshop on Problem-Solving Methods for Knowledge-based Systems, IJCAI '97, August 23.* Available from `http://www.cse.unsw.edu.au/~timm/pub/docs/97rdra.ps.gz`.

[Menzies & Waugh, 1998] Menzies, T. & Waugh, S. (1998). On the Practicality of Viewpoint-based Requirements Engineering. In *Proceedings, Pacific Rim Conference on Artificial Intelligence, Singapore.* Springer-Verlag.

[Michalski, 1993] Michalski, R. (1993). Toward a Unified Theory of Learning: Multistrategy Task-adaptive Learning. In Buchanan, B. G. & Wilkin, D. C., (Eds.), *Readings in Knowledge Acquisition and Learning: Automatic Construction and Improvement of Expert System.* Morgan Kaufmann Publishers.

[Mitchell *et al.*, 1986] Mitchell, T., Keller, R., & Kedar-Cabelli, S. T. (1986). Explanation-Based Generalization: A Unifying View. *Machine Learning*, 1:47–80.

[Moran & Carroll, 1996] Moran, T. & Carroll, J. (1996). *Design Rationale: Concepts, Techniques, and Use.* Lawerence Erlbaum Associates.

[Motta & Zdrahal, 1996] Motta, E. & Zdrahal, Z. (1996). Parametric Design Problem Solving. In *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based System Workshop.*

[Muggleton, 1991] Muggleton, S. (1991). Inductive Logic Programming. *New Generation Computing*, 8:295–318.

[Mulholland *et al.*, 1996] Mulholland, M., Preston, P., Hibbert, B., & Compton, P. (1996). An expert system for ion chromatography developed using machine learning and knowledge in context. In *Proceedings of the Sixth International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Edinburgh.*

[Musen & Tu, 1993] Musen, M. & Tu, S. (1993). Probelm-Solving Models for Generation of Task-Specific Knowledge Acquisition Tools. In Cuena, J., (Ed.), *Knowledge-Oriented Software Design.* Elsevier, Amsterdam.

[Myers, 1977] Myers, G. (1977). A Controlled Experiment in Program Testing and Code Walkthroughs/Inspections. *Communications of the ACM*, 21:760–768.

[Neches *et al.*, 1991] Neches, R., Fikes, R., Finin, T., Gruber, T., Patil, R., Senator, T., & Swartout., W. R. (1991). Enabling technology for knowledge sharing. *AI Magazine*, 12(3):16–36.

[Newell, 1982] Newell, A. (1982). The Knowledge Level. *Artificial Intelligence*, 18:87–127.

[Newell, 1993] Newell, A. (1993). Reflections on the Knowledge Level. *Artificial Intelligence*, 59:31–38.

[Nuseibeh, 1997] Nuseibeh, B. (1997). To Be *and* Not to Be: On Managing Inconsistency in Software Development. In *Proceedings of 8th International Workshop on Software Specification and Design (IWSSD-8)*, pages 164–169. IEEE CS Press.

[Odberg, 1995] Odberg, E. (1995). *MultiPerspectives: Object Evolution and Schema Modification Management for Object-Oriented Databases.* PhD thesis, Norwegian Institute of Technology. 408 pages.

[O'Hara & Shadbolt, 1997] O'Hara, K. & Shadbolt, N. (1997). Inerpreting Generic Structures: Expert Systems, Expertise, and Context. In Feltovich, P., Ford, K., & Hoffman, R., (Eds.), *Expertise in Context*, chapter 19, pages 449–472. MIT PRess.

[Olle *et al.*, 1991] Olle, T., Hagelstein, J., MacDonald, I., Rolland, C., Sol, H., Assche, F. V., & Verrijn-Stuart, A. (1991). *Information Systems Methodologies: A Framework for Understanding.* Addison-Wesley.

[Patil *et al.*, 1981] Patil, R., Szolovitis, P., & Schwartz, W. (1981). Causal Understanding of Patient Illness in Medical Diagnosis. In *IJCAI '81*, pages 893–899.

[Politakis, 1985] Politakis, P. (1985). *Empirical Analsis for Expert Systems.* Pitman.

[Poole, 1993] Poole, D. (1993). Probabilistic Horn Abduction and Bayesian Networks. *Artificial Intelligence*, 64(1):81–129.

[Popper, 1963] Popper, K. (1963). *Conjectures and Refutations,*. Routledge and Kegan Paul.

[Pos *et al.*, 1997] Pos, A., Akkermans, H., Straatman, R., & Wijngaards, N. (1997). Redesign Problem Solving. In *Workshop on Problem-Solving Methods for Knowledge-based Systems, IJCAI '97, August 23.*

[Preece & Shinghal, 1992] Preece, A. & Shinghal, R. (1992). Verifying Knowledge Bases by Anomaly Detection: An Experience Report. In *ECAI '92*.

[Preston *et al.*, 1993] Preston, P., Edwards, G., & Compton, P. (1993). A 1600 Rule Expert System Without Knowledge Engineers. In Leibowitz, J., (Ed.), *Second World Congress on Expert Systems*.

[Quinlan, 1982] Quinlan, J. (1982). Learning Efficient Classification Procedures and Thier Application to Chess End-Games. In *Machine Learning*.

[Quinlan, 1986] Quinlan, J. (1986). Induction of Decision Trees. *Machine Learning*, 1:81–106.

[Ramesh & Dhar, 1992] Ramesh, B. & Dhar, V. (1992). Supporing Systems Development by Capturing Deliberations During Requirements Engineering. *IEEE Transactions on Software Engineering*, 18(6):498–510.

[Richards & Compton, 1997] Richards, D. & Compton, P. (1997). Combining Formal Concept Analysis and Ripple Down Rules to Support the Reuse of Knowledge. In *SEKE '97: Proceedings of 1997 Conf. on Software Eng. & Knowledge Eng, Madrid.*

[Richards & Menzies, 1997] Richards, D. & Menzies, T. (1997). Extending Knowledge Engineering to Requirements Engineering from Multiple Perspectives. In Menzies, T., Richards, D., & Compton, P., (Eds.), *Third Australian Knowledge Acquisition Workshop, Perth.*

[Rubin & Goldberg, 1992] Rubin, K. & Goldberg, A. (1992). Object Behavior Analysis. *Communications of the ACM*, 35(9).

[Rumbaugh, 1994] Rumbaugh, J. (1994). Getting Started: Using Use Cases to Capture Requirements. *JOOP*, pages 8–23.

[Runkel, 1995] Runkel, J. (1995). Analyzing Tasks to Build Reusable Model-Based Tools. In *Proceedings of the 9th AAAI-Sponsored Banff Knowledge Acquisition for Knowledge-Based Systems Workshop Banff, Canada.*

[S, 1995] S, B. (1995). Software Tools for Evaluating the Usability of User Interfaces. In Anzi, Y. & Ogawa, K., (Eds.), *Proceddings of the 6th International Conference on HCI, Pacificon Yokohama (Japan).*

[Sammut & Banerji, 1986] Sammut, C. & Banerji, R. (1986). Learning Concepts by Asking Questions. In Michalski, R. S., Carbonell, J., & Mitchell, T., (Eds.), *Machine Learning: An Artificial Intelligence Approach*, volume 2, pages 167–192. Los Altos, California: Morgan Kaufmann.

[Schreiber *et al.*, 1994] Schreiber, A. T., Wielinga, B., Akkermans, J. M., Velde, W. V. D., & de Hoog, R. (1994). CommonKADS. A Comprehensive Methodology for KBS Development. *IEEE Expert*, 9(6):28–37.

[Selman & Levesque, 1990] Selman, B. & Levesque, H. (1990). Abductive and Default Reasoning: a Computational Core. In *AAAI '90*, pages 343–348.

[Shadbolt & O'Hara, 1997] Shadbolt, N. & O'Hara, K. (1997). Model-based Expert Systems and the Explanations of Expertise. In Feltovich, P., Ford, K., & Hoffman, R., (Eds.), *Expertise in Context*, chapter 13, pages 315–337. MIT PRess.

[Shahsavar, 1993] Shahsavar, N. (1993). *Design, Implementation and Evaluation of a Knowledge-Based System to Support Ventilator Therapy Management.* PhD thesis, Department of Medical Informatics, Linkoping University, Sweden.

[Shapiro, 1983] Shapiro, E. Y. (1983). *Algorithmic program debugging.* Cambridge, Massachusetts, MIT Press.

[Shaw, 1997] Shaw, M. (1997). *WebGrid: a WWW PCP Server.* Knowledge Systems Institute, University of Calgary, `http://Tiger.cpsc.ucalgary.ca/WebGrid/WebGrid.html`.

[Silverman, 1990] Silverman, B. (1990). Critiquing Human Judgmet Using Knowledge-Acquisition Systems. *AI Magazine*, pages 60–79.

[Silverman, 1992a] Silverman, B. (1992a). Building a Better Critic: Recent Empirical Results. *IEEE Expert*, pages 18–25.

[Silverman, 1992b] Silverman, B. (1992b). Survey of Expert Critiquing Systems: Practical and Theoretical Frontiers. *Communications of the ACM*, 35:106–127.

[Silverman & Wenig, 1993] Silverman, B. & Wenig, R. (1993). Engineering Expert Critics for Cooperative Systems. *The Knowledge Engineering Review*, 8(4):309–328.

[Soloway *et al.*, 1987] Soloway, E., Bachant, J., & Jensen, K. (1987). Assessing the Maintainability of XCON-in-RIME: Coping with the Problems of a VERY Large Rule-Base. In *AAAI '87*, pages 824–829.

[Steels, 1990] Steels, L. (1990). Components of Expertise. *AI Magazine*, 11:29–49.

[Swartout & Gill, 1996] Swartout, B. & Gill, Y. (1996). Flexible Knowledge Acquisition Through Explicit Representation of Knowledge Roles. In *1996 AAAI Spring Symposium on Acquisition, Learning, and Demonstration: Automating Tasks for Users.*

[Tansley & Hayball, 1993] Tansley, D. & Hayball, C. (1993). *Knowledge-Based Systems Analysis and Design*. Prentice-Hall.

[technlogies, 1997] technlogies, R. S. (1997). PiSCES Automatic Test Case Generator `http://www.rstcorp.com/tools.html#pisces`.

[van Harmelen & Aben, 1996] van Harmelen, F. & Aben, M. (1996). Structure-Preserving Specification Languages for Knowledge-Based Systems. *International Journal of Human-Computer Studies*, 44:187–212.

[van Harmelen & Bundy, 1988] van Harmelen, F. & Bundy, A. (1988). Explanation-Based Generalisation = Partial Evaluation. *Artificial Intelligence*, pages 401–412.

[Webb & Wells, 1996] Webb, G. & Wells, J. (1996). Experimental Evaluation of Integrating Machine Learning with Knowledge Acquisition Through Direct Interaction with Domain Experts. In *Proceedings PKAW '96: Pacific Knowledge Acquisition Workshop.*

[Wielinga *et al.*, 1992a] Wielinga, B., Schreiber, A., & Breuker, J. (1992a). KADS: a Modeling Approach to Knowledge Engineering. *Knowledge Acquisition*, 4:1–162.

[Wielinga *et al.*, 1992b] Wielinga, B., Schreiber, A., & Breuker, J. (1992b). KADS: a Modeling Approach to Knowledge Engineering. *Knowledge Acquisition*, 4:1–162.

[Williams & Huang, 1996] Williams, G. & Huang, Z. (1996). A Case Study in Knowledge Acquisition for Insurance Risk Assessment using a KDD Methodology. In *Proceedings PKAW '96: Pacific Knowledge Acquisition Workshop.*

[Winograd & Flores, 1987] Winograd, T. & Flores, F. (1987). On Understanding Computers and Cognition: A New Foundation for Design: A respose to the reviews. *Artificial Intelligence*, 31:250–261.

[Winston, 1984] Winston, P. (1984). *Artificial Intelligence*. Addison-Wesley.

[Yost, 1993] Yost, G. (1993). Acquiring Knowledge in Soar. *IEEE Expert*, pages 26–34.

[Yost & Newell, 1989] Yost, G. & Newell, A. (1989). A Problem Space Approach to Expert System Specification. In *IJCAI '89*, pages 621–627.

[Zlatareva, 1992] Zlatareva, N. (1992). CTMS: A General Framework for Plausible Reasoning. *International Journal of Expert Systems*, 5:229–247.

[Zlatareva, 1993] Zlatareva, N. (1993). Distributed Verification and Automated Generation of Test Cases. In *IJCAI '93 workshop on Validation, Verification and Test of KBs Chambery, France*, pages 67–77.

[Zlatereva, 1992] Zlatereva, N. (1992). Truth Mainteance Systems and Their Application for Verifying Expert System Knowledge Bases. *Artificial Intelligence Review*, 6.