*Expert Systems With Applications*, Vol. 5, pp. 267-273, 1992
Printed in the USA.

0957-4174/92 $5.00 + .00
© 1992 Pergamon Press Ltd.

# Maintenance Tools for Knowledge-Based Systems: The MAKE Project

TREVOR BENCH-CAPON AND FRANS COENEN

University of Liverpool, Liverpool, England

**Abstract**—*One of the major obstacles to the routine exploitation of knowledge-based and expert systems, is the difficulty of validating the knowledge base, and of maintaining it in a state which reflects current knowledge. This is of particular importance for systems based on law or regulations, where it is vital that the knowledge base be a true reflection of the legal position, and where there is a constant stream of changes to the correct legal position. Maintenance Assistance for Knowledge Engineers (MAKE) is a project designed to explore these issues, and to build a set of tools which will support the validation and maintenance of knowledge bases deriving from regulations. These tools include facilities to examine the structural features of the knowledge base, so as to guard against redundancy, nonprovability and contradiction; facilities to identify parts of the knowledge base jeopardised by changes in the domain, or in the understanding of the domain; and facilities to perform a variety of "house keeping" tasks. The paper firstly analyses the different types of change that may be required to maintain the knowledge base, and then proceeds to describe the set of tools developed in the MAKE project to accommodate these changes.*

## 1. INTRODUCTION

KNOWLEDGE-BASED SYSTEMS (KBSs) have been in existence for several decades now, but their general acceptance is still limited. It is suggested that one principal reason for this is the difficulty in maintaining such systems due to the nature of the expert knowledge used (Bench-Capon & Coenen, 1991a; Bratley, Fremont, Mackaay, le Poulin, 1991). This tends to be dynamic, in that the knowledge will change as new discoveries are made and expert opinions alter over a period of time. Thus KBSs, if they are not maintained, quickly become obsolete or inaccurate. This is even more the case in legal domains where the knowledge required may change overnight if legislation is altered or a new case decided.

The maintenance of KBSs thus tends to be adaptive maintenance rather than the corrective or perfective maintenance associated with conventional systems (Coenen & Bench-Capon, 1990), a categorisation first proposed in Swanson (1976). The approach to maintenance is therefore not the same since the emphasis is on responding to external changes. Further, the structure of KBSs is such that traditional maintenance techniques are not applicable. Given the above, a surprisingly small amount of research work has been carried out to investigate the maintenance of KBSs. What work has been carried out has largely been concerned with formal consistency checking of rule bases (RBs) such as in the COVADIS system (Rousset, 1988), the debugging of RBs such as provided by the TEIRESIAS system (Davis, 1984), or the syntactical editing of RBs within KBS development toolkits.

In this paper, a number of tools to assist the KBS maintenance engineer are outlined. The tools form a suite of tools under development as part of the Maintenance Assistance for Knowledge Engineers (MAKE) Project. This is a two-year collaborative project between Liverpool University, ICL and British Coal. The project includes the development of a KBS to provide decision support for the claims inspectorate at British Coal's Insurance and Pensions Division. This provides a real application for use as a test bed. The tools are essentially designed to be used in conjunction with KBSs built using the MADE (MAKE authoring and development methodology) development environment and methodology, a brief overview of which is given in Section 2. A more detailed description is given in (Coenen & Bench-Capon, 1991).

Knowledge analysis using MADE results in a class hierarchy (CH) giving a vocabulary for the domain and a set of rules represented in an intermediate representation called MIR (MAKE Intermediate Representation). This is fully described in Bench-Capon and Forder (1991): a brief overview is given in Section 3. The

Revised version of Bench-Capon, T., & Coenen, F., The MAKE project: Maintenance tools for knowledge based systems, pp. 1030-1036, from Liebowitz: *Expert Systems World Congress Proceedings*, copyright 1991, with permission from Pergamon Press Ltd., Headington Hill Hall, Oxford, OX3 0BW, United Kingdom.

Requests for reprints should be sent to Trevor Bench-Capon, Department of Computer Science, University of Liverpool, P.O. Box 147, Liverpool, England L69 3BX.

philosophy behind knowledge analysis using KANT is that the resulting KB will be isomorphic with the source, i.e., it will reflect the structure of the source and vice versa. This desire for isomorphism necessitates the use of an intermediate representation, such as MIR, for reasons described in Bench-Capon (1991a). The advantages to be gained by an isomorphic representation have been discussed in Routen and Bench-Capon (1991) and Bench-Capon and Coenen (1991b).

KBSs developed using MADE can be considered to represent knowledge about the domain which they are designed to address at three different levels. At the topmost level are the source documents themselves. The second intermediate level is in the MIR attribute-rule representation. The third level is the fine grain proposition-clause level produced when the intermediate representation is compiled into CMIR (compiled MIR) or other suitable formalisation. We have earlier argued (Bench-Capon, 1990) that wherever possible maintenance should be carried out at the MIR level. In some cases however, for example inconsistency checking, it is necessary for the proposed maintenance tools to operate at the proposition-clause level. Prior to introducing the proposed tools, a discussion on the nature of KBS maintenance activities is given in Section 4. The proposed tools are then described in Section 5. Finally in Section 6 some conclusions are offered.

## 2. OVERVIEW OF THE MAKE AUTHORING AND DEVELOPMENT ENVIRONMENT (MADE)

MADE is a KBS development environment based on KANT (Knowledge ANalysis Tool). This is a hypertext like knowledge analysis tool originally built to assist in the development of a KBS to provide decision support for Department of Social Security (DSS) Adjudication Officers in the assessment of claims for benefits in local DSS offices (Storrs & Burton, 1989). This system was called the local office demonstrator (LOD) system and was one of three applications built as part of the Alvey DHSS Large Demonstrator Project aimed at demonstrating the viability of KBS decision support in large, legislation-based organisations (Forder & Taylor, 1991). It is ideally suited to the construction of KBSs in domains, such as legal domains, where the source knowledge is comprised of a significant amount of textual material, by assisting the knowledge engineer in the analysis of these source documents. The current version of KANT is CKANT, which is a C++ version of the original.

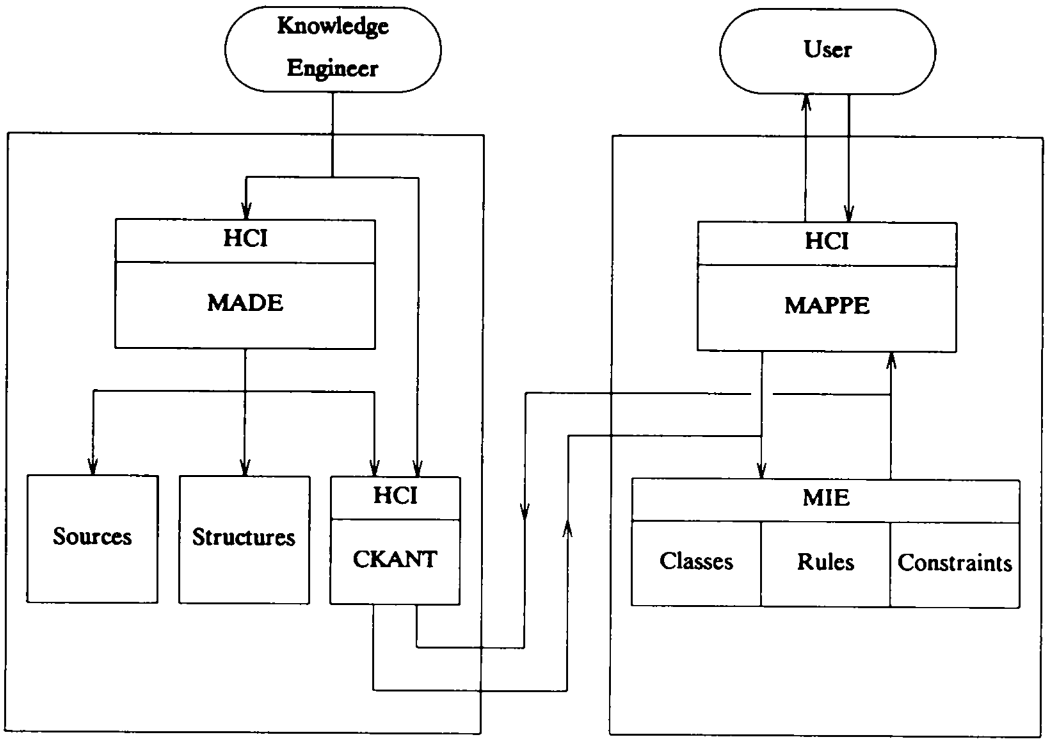
The design of MADE revolves around three base windows, the KANT, MADE and MAPPE (MAKE APPlication Environment) windows. From the KANT base window sources can be selected and analysed. This is where the knowledge analysis of the application is carried out. The result is a rule base and class hierarchy in MIR. The MIR rule base and class hierarchy are then compiled into the target representation used in the MAKE inference engine (MIE). Currently this consists of a clausal form referred to as CMIR (compiled MIR). Other target representations could be used equally well, for example Kappa or NExpert. A schematic illustrating the MADE architecture is given in Figure 1.

With respect to the RB and CH in the MIR the following definitions, used in the rest of this document, should be noted:

1. *Rule.* This is an expression comprising propositions, typically EAVs, and logical connectives. It consists of a head and a tail separated by an if or iff (read as if and only if). The head contains a single proposition, the truth or falsity of which is determined according to the truth or falsity of the propositions contained in the body and their connectives.
2. *Clause.* This is an expression in standard clausal form, with a proposition as head and a set of conjoined propositions as tail (body). The head proposition is true if all the propositions in the body are true. A single rule will typically be equivalent to several clauses.
3. *Proposition.* A proposition is a triple comprising an entity, an attribute and a value. The entity may be a constant (the name of an instance) or a variable (typed to some class in the CH). The attribute will be a slot associated with the class of which the entity is a member. The value will be a subrange of the possible values for the slot as defined in the class of which the entity is a member. Note that the CH constrains what propositions are possible.
4. *Root Propositions and Attributes.* A root proposition is a proposition which appears in the head of some clause, and does not appear in the tail of any clause. A root attribute is an attribute associated with one or more root propositions. Thus a root proposition represents the task which the RB is designed to establish.
5. *Leaf Propositions and Attributes.* A leaf proposition is a proposition which appears in the body of one or more clauses, and in the head of no clauses. A leaf attribute is an attribute associated with one or more leaf propositions. The significance of leaves is that values associated with them must be supplied to the system, not deduced. Values may be supplied either by explicit user input, or by the association of an instance with a class, the definition of which makes the proposition true.

## 3. OVERVIEW OF MIR

MIR is essentially a simple language to define objects and rules using first-order predicate logic with some extensions, for example to handle arithmetic. In



KEY:-

| | |
|---|---|
| HCI | = Human Computer Interface |
| CKANT | = C+ + version of Knowledge engineers assistANT |
| MIE | = Make Inference Engine |
| MAPPE | = Make APPlication Environment |

**FIGURE 1. Schematic of MADE architecture. HCI: human computer interface; CKANT: C++ version of Knowledge engineers assistANT; MIE: MAKE inference engine; MAPPE: MAKE APPlication environment.**

MADE it is used to represent a CH and RB describing an application domain.

The CH in the MIR consists of a top level class with subclasses branching of it describing different object types. Each class has a unique class name describing an object and a number of slots describing attributes of that object, and the possible values that the attribute may have. A feature of the CH is that subclasses inherit attributes of super classes. Thus a subclass is always a strict specialisation of its super class. In KANT inherited attributes cannot be cancelled as is the case with some other development environments, for example KEE. This permits classes to be interpreted as logical types and avoids the problems raised in Brachman (1985).

In the MIR a RB built to support a particular task will consist of a set of rules typically all branching out from a single top level node. It is important to differentiate between rules and constraints at this stage. Rules are task dependent, and are thus arranged in a hierarchy headed by a top level rule representing the goal to be attained: Further rules then branch of this top level rule down to a number of leaf rules. Propositions found in Rules are therefore either "leaves," in which case their truth will be ascertained by direct reference to the CH or the user, "roots," where they are what the system is intended to establish, or "intermediate," where they need to be established from the KB in order to establish some root. Constraints however are task independent. The important issue here is that in a KB redundancy and missing branches are significant, but depend on the notion of a purpose for the rule set in terms of what is to be given and what must be deduced. Different tasks may require different literals in a constraint to be treated as the head. Additionally when considering hard and soft inconsistencies in a task-dependent RB, it may be that an inconsistency is hidden by the restricted set of inferences that may be drawn, but exposed when the rules are considered as constraints. Therefore, some of the tools proposed in this document will be designed to be implemented on the fine grain proposition-clause third level representation.

## 4. KBS MAINTENANCE

It is not the intention of the MAKE project to address major maintenance tasks which may necessitate the entire rebuilding of the system. If changes are sufficiently radical, of course the system will need to be rewritten. The aim of the MAKE project is to address minor adaptive maintenance only, i.e., maintenance resulting from the day to day changes in the source material due to changes in legal texts, the application and operation of the law etc. In this context, the maintenance required can be considered under a number of headings, (a) RB maintenance, (b) CH maintenance, (c) changes to the source data, (d) validation, and (e) maintenance support.

### 4.1. RB Maintenance

The maintenance of the RB will involve one or more of the following activities:

M1 The introduction of a new rule.
M2 The modification of an existing rule.
M3 The removal of an existing rule.

The effect of introducing a new rule may be unwanted redundancy or subsumption, or the creation of a missing branch, a hard contradiction or soft inconsistency. A hard contradiction is simply a logical contradiction, i.e.,

$$A \text{ and not } A.$$

What we term a "soft inconsistency" occurs when some proposition is a consequence of the KB, whereas it is in fact known that its negation is possible. In the simplest possible case, we may have two rules:

$$P => Q,$$
$$P => \text{not } Q.$$

There is no logical contradiction here, but not $P$ is a logical consequence of the KB. If, however, $P$ represented something which we knew to be sometimes true and sometimes false, this would indicate that our KB was in error.

The introduction of a new rule will thus involve checking for the following:

C1 Redundancy or subsumption.
C2 Missing rules or branches.
C3 Hard inconsistency.
C4 Soft inconsistency.

The removal of a rule may also result in the creation of a missing branch or cause a section of the KB to become redundant. Therefore when removing a rule checks C1 and C2 should be implemented.

The modification of a rule has the same effect as removing a rule and introducing another. Hence, the methods outlined above for the introduction and removal of rules can be used in sequence:

Remove old rule
check for C1 and C2
Introduce new rule
check for C1, C2, C3 and C4.

It should be noted that as a result of removing the rule in this case some acceptable redundancy and/or missing branches may temporarily be created until the new rule is added.

It would be rare for a maintenance session to consist of only the removal or introduction of a single rule. In most cases, a maintenance session will involve all three types of KB maintenance, i.e., M1, M2, and M3. It is therefore proposed that on completion of any KB, maintenance checks for C1 to C4 should always be carried out. Two RB maintenance tools are therefore proposed:

T1 The rulemap.
T2 Hard contradiction and soft inconsistency identification.

T1 can be implemented on the RB in its static form and incorporate redundancy and subsumption. It can also be considered to be an RB navigation tool that will allow the user to move through the RB at the MIR level and the fine grain level so that missing rules and branches can be identified and facilitate "by eye" verification.

T2 is designed to address the dynamic aspects of the RB and will be implemented at both the MIR and third level representations as appropriate. The existence of redundant and missing branches will only be significant in a task-dependent RB. Hard and soft inconsistencies can only be identified at the constraint level.

### 4.2. CH Maintenance

In the methodology described, the CH plays an important role as the means by which the vocabulary to be used in writing the rules is described, it also describes the state of an application at any particular time. The discipline that this imposes is important if the representation is to be a faithful reflection of the domain, and hence keep its structure through a period of maintenance. The maintenance associated with the CH may involve:

M4 The modification of an existing slot by introducing a new value.
M5 The modification of an existing slot by removing an existing value.
M6 The modification of an existing class by introducing a new slot.
M7 The modification of an existing class by removing an existing slot.
M8 The introduction of an entire new class.
M9 The removal of an existing class.

4.2.1. *Introducing or removing a value.* One of the most basic actions in the maintenance of the CH is the addition of a possible value to a slot. In practice a value will be added to a slot as a consequence of the introduction or modification of a rule. The allocation of this additional value to an existing slot will not generally affect the operation of any established rules or the existing CH. The exception to this is if rules exist that use the possible values of a slot to express negation. Thus, the rules that contain the attribute to which a value is to be added need to be identified so that the effect of introducing this value can be determined. For this purpose it will, in some cases, be necessary to go down to the fine grain level of representation.

Removing a value from a slot will jeopardise all rules which make use of that value either in the head or the body of the rule. These rules must therefore be identified and presented to the maintenance engineer so that a decision can be made on whether it is appropriate to remove the rule, remove the atom containing the removed value, or modify the rule.

A tool to allow the identification of jeopardised rules as a result of removing and introducing values to and from slots in the CH is therefore required. A more general to identify jeopardised slots and rules as a result of changes to the source data or changes to the rule base or CH would be more useful. Thus:
T3 Jeopardy tool.

Because the inheritance mechanism used, the CH insists on strict specialisation, a value can only be added to an attribute at the highest level at which that attribute appears. If the rule which motivates the introduction of the new value refers to a class which inherits the attribute from a super class, either the value must be added to the super class, or some new attribute must be created in the class in question. If the value is added to the super class, of course, the rules for that class in which the attribute appears are jeopardised. Therefore, before a new value can be added, the user should be confronted with the class which introduces the attribute to the CH, which may not be the class mentioned in the rule which motivated the introduction of the value, and the value added to this class. If the class to which the attribute is added is not a leaf class, this process would be facilitated by a tool which walks down the CH so that the user is able to determine the correct point, on each path, at which the new value should cease to apply.

Note that the removal of a value will also cause that value to be removed from all subclasses of the class from which it is removed. It will therefore be necessary to walk down the hierarchy repeating the process for all these subclasses, until the value has been "specialised out."

There is thus a need for a tool to provide the maintenance engineer with the facility to walk systematically up or down the CH, focusing on particular attributes. Thus:
T4 CH navigation tool.

4.2.2. *Introducing or removing an attribute.* An attribute can be added to a class in two ways. Either it can be added directly to the class, or it can be added to a super class, and so added to the class in question indirectly by inheritance. Thus if a rule needs to mention a new attribute for some class, the first step should be to walk up the CH to determine the appropriate point at which the attribute should be introduced into the hierarchy. Note, however, that adding it to a super class will cause all the subclasses of that class to take on the attribute, not only those on the path walked up. Once introduced into the hierarchy, rules may be written using the attribute. These can then be subject to the usual checks for new rules already described. The final stage will then be to walk down the CH specialising the values of the attribute as appropriate, until a point range is reached on every downwards path. The process of adding a new attribute to an existing class will thus also involve the use of the CH navigation tool (T4).

The process of removing an attribute from an existing class can be regarded as removing a set of values. Inheritance, however, means that the attribute will also be removed from all the subclasses up to the point in the hierarchy at which it was introduced. The best approach would therefore be to commence the removal at this point, and then to walk down the CH to determine at which point the attribute should be reintroduced into the hierarchy if necessary. Class(es) at which it is now introduced and their subclasses will not be affected. Thus the CH navigation tool will also be of relevance here.

4.2.3. *Introducing or removing a class.* The point at the hierarchy in which the class should be introduced will be best determined by the attributes that need to be associated with the class. If an existing class contains a subset of the desired attributes, then it is a potential super class for the new class. Clearly the class with the largest such subset (i.e., the lowest such class in the hierarchy) is the logical super class to choose. Next, it must be determined which existing classes should be subclasses of the new class. The answer here is that existing classes with attributes which are a superset of the new class should be subclasses of the new class. A tool to determine the relations between sets of attributes is clearly suggested. This may be incorporated into the CH navigation tool. If a suitable super class is not identified, the class can be considered to represent a leaf node, a subclass of the class with the largest subset of desired attributes.

Removing a class is, as far as the KB is concerned, effectively like removing a set of attributes. As far as the class hierarchy is concerned, existing subclasses of the class need to become subclasses of its immediate super class. Problems still arise if any specialisation of attribute values, or addition of attributes, were made in the removed class. Clearly such attributes must be reintroduced, or specialisations made, either in the subclasses or the immediate super class, as seems to be most appropriate.

### 4.3. Changes to the Source Data

At a higher level, rules will also be jeopardised by changes in the source material, as when legislation is amended. A feature of the MADE development methodology is that a linking facility is provided to link individual sections in the source material through the various analysis stages to the resulting CHs and RBs in the target representation. This provides a useful basis for identifying rules and classes that may be affected as a result of changes in the source material and can be automated as part of the jeopardy tool (T3).

### 4.4. Validation

So far only the verification of the KB and CH have been considered. However, it is also necessary to validate the KBS after maintenance has taken place. This can be carried out by peopling the rule base and determining what inferences can be made or tracing how inferences are made. Thus:

T5 Rule base animation tool.

## 5. INTRODUCTION TO PROPOSED TOOLS

In the previous section, a number of maintenance tools were identified to address different aspects of KBS maintenance. These are summarised below:

T1 The rulemap.
T2 Hard contradiction and soft inconsistency identification.
T3 Jeopardy tool.
T4 CH navigation tool.
T5 Rules base animation tool.

In the following subsections, each of these tools will be described in greater detail.

### 5.1. The Rulemap

The rulemap is a directed (from left to right) bipartite graph which graphically displays the rule base either at the attributes-rules level or the proposition-clause level. A number of options are provided to allow the user to walk up and down the rule base. By following a path through the rulemap, it is possible to determine the leaf attributes and propositions into which a root attribute ultimately unfolds and vice versa. This gives the user a clear visual view of the rules in the knowledge base. Options are also provided to allow the user to interrogate the rulemap to display the rule and clauses in which attribute and propositions appear or to inspect the values or entities associated with attributes and propositions. There is also a facility to identify redundant or subsumed rules or subsets of rules.

### 5.2. The Class Hierarchy Navigation Tool

This is a class-instance browser designed to allow the user to navigate through a class hierarchy. The tool is intended to give visibility to the author of not just the class hierarchy, but also where slot definitions come from, and will enable the user to determine the best location for new classes, subclasses and attributes related to those classes, and specialisations of attribute values.

### 5.3. Jeopardy Tool

This is a general purpose maintenance tool to identify jeopardised slots and rules as a result of changes to the source data or changes to the rule base or class hierarchy. The tool will incorporate the following facilities:

(a) *Rules Jeopardised by Slot Changes Identification.* Facility to identify rules jeopardised as a result of changes to slots in the class hierarchy.
(b) *Class Definitions Jeopardised by Class Deletions Identification.* Facility to identify the slots jeopardised by the removal of a class because they are typed to that class.
(c) *Rules Jeopardised by Source Changes Identification.* Facility to identify the rules that are affected by changes in the source material.
(d) *Slots Jeopardised by Source Changes Identification.* Facility to identify the slots in the class hierarchy that are affected by changes in the source material.
(e) *Rules Jeopardised by Rule Changes Identification.* In a task-dependent rule base rules above and below an altered rule may be jeopardised. This facility will identify the subset of rules which have been affected by a KB maintenance session.

The jeopardy tool operates using the links that should be included by the knowledge engineer during system development. This is an essential part of the MADE methodology.

5.3.1. *The rule base animation tool.* This tool is similar to the rulemap but addresses the dynamic aspects of the rule base. It allows the user to people the rule base by creating instances and asserting propositions, and then determine what inferences can be made as a result. When the behaviour is unexpected, either because an inference which should not be made is made, or because an inference which was expected fails to be made, this tool will enable the user to locate the precise clause which caused the failure, and from this the rule, analysis, and source from which it was derived. Such animation is a necessary adjunct to the "by eye" validation supported by the static tools, since the practical consequences of a given fragment of the KB may be hard to envisage in the abstract.

5.3.2. *Hard contradiction and soft inconsistency identification.* When adding or modifying rules in a KB during a maintenance session a hard contradiction may be introduced. In logical terms, this means that there can be no model for the knowledge base, so the knowledge base cannot be correct. Soft inconsistency is a modified phenomenon and occurs when some proposition is a consequence of the KB when it is in fact known that its negation is possible. This means that the knowledge base excludes some models which are known to occur, and again suggests a defect in the knowledge base. Thus a minimal validation of the knowledge base will involve ensuring that neither of these situations exist. Tools are under development in the MAKE project which will allow such inconsistencies in a knowledge base to be detected.

## 6. CONCLUSIONS

In this paper we have given a brief overview of the MAKE project and identified a number of maintenance tasks that may be required for systems developed in this environment. To assist in these maintenance tasks, a number of maintenance tools have been suggested and briefly described. These descriptions do not in any way represent a precise specification. They are however an indication of how the proposed tools may be implemented.

For many of the proposed tools prototypes have been built. If not, the tools are at least well understood. Where prototypes have been built, these have been implemented on a pilot version of the British Coal KBS which has been developed as part of the MAKE project and have been well received.

**Acknowledgements**—The work described here is being carried out as part of the MAKE Project, supported by the Information Engineering Directorate of the UK Department of Trade and Industry and the UK Science and Engineering Research Council. The project collaborators are ICL, the University of Liverpool and British Coal.

## REFERENCES

Bench-Capon, T.J.M., & Coenen, F. (1990). *Role of intermediate representation in maintenance of KBS* (MAKE Report 4/90). Liverpool, England: Liverpool University.

Bench-Capon, T.J.M., & Coenen, F. (1991a). Practical application of KBS to law: The crucial role of maintenance. In C. van Noortwijk, A.H.J. Schmidt, & R.G.F. Winkels (Eds.), *Legal knowledge based systems: Aims for research and development.* Lelystad, Netherlands: Koninklijke Vermande BV.

Bench-Capon, T.J.M., & Coenen, F. (1991b). Exploiting isomorphism: Development of a KBS to support British coal insurance claims. *Proceedings of the 3rd International Conference on Artificial Intelligence and Law* (pp. 62–68). Oxford: ACM.

Bench-Capon, T.J.M., & Forder, J.M. (1991). Knowledge representation for legal applications. In T.J.M. Bench-Capon (Ed.), *Knowledge based systems for legal applications* (pp. 245–264). New York: Academic Press.

Brachman, R.J. (Fall, 1985). I lied about the trees. *AI Magazine,* 6(3), 80–93.

Bratley, P., Fremont, J., MacKaay, E., & Poulin, D. (1991). Coping with change. *Proceedings of ICAIL* (pp. 69–76).

Coenen, F., & Bench-Capon, T.J.M. (1990) *Initial analysis of maintenance issues associated with knowledge based systems* (MAKE Report 2/90). Liverpool, England: Liverpool University.

Coenen, F., Bench-Capon, T.J.M., & Smith, M.J. (1991). KBS *development using X windows: The MADE development methodology.* Proc. UKUUG Summer Conference (pp. 64–72).

Davis, R. (1984). Interactive transfer of expertise. In B.G. Buchannan & E.H. Shortliffe (Eds.), *Rule-based expert systems: The MYCIN experiments of the Stanford Heuristic Programming Project* (pp. 171–205). Reading, MA: Addison-Wesley.

Forder, J.M., & Taylor, A.D. (1991). The local office system. In T.J.M. Bench-Capon (Ed.), *Knowledge based systems for legal applications.* (pp. 139–164) New York: Academic Press.

Rousset, M. (1988). On the consistency of knowledge bases: The COVADIS system. 8th Int. Conf. on Expert Systems & Their Applications (pp. 79–83). Avignon EC2 Nanterre, Cedex, France.

Routen, T.W., & Bench-Capon, T.J.M. (1991). Hierarchical formalisation. *International Journal of Man-Machine Studies.*

Storrs, G.E., & Burton, C.P. (1989). KANT, A knowledge analysis tool. *ICL Technical Journal,* 6(3), 572–581.

Swanson, E.B. (1976). The dimensions of maintenance. *Proceedings, 2nd International Conference on Software Engineering* (pp. 492–497). IEEE, Long Beach, CA.