Artificial Intelligence and Law **1**, 65–86, 1992

Isomorphism and Legal Knowledge Based Systems

T. J. M. BENCH-CAPON & F. P. COENEN

Department of Computer Science, University of Liverpool, Liverpool, England

(Received 10 October 1991; accepted 13 December 1991)

Abstract. This paper discusses some engineering considerations that should be taken into account when building a knowledge based system, and recommends isomorphism, the well defined correspondence of the knowledge base to the source texts, as a basic principle of system construction in the legal domain. Isomorphism, as it has been used in the field of legal knowledge based systems, is characterised and the benefits which stem from its use are described. Some objections to and limitations of the approach are discussed. The paper concludes with a case study giving a detailed example of the use of the isomorphic approach in a particular application.

Key words: legal knowledge based systems, maintenance, isomorphism

1. Introduction

Michael Jackson begins his influential book *Principles of Program Design*, (Jackson, 1975), with the words

> The beginning of wisdom for a programmer is to recognize the difference between getting his program to work and getting it right. A program which does not work is undoubtedly wrong; but a program which does work is not necessarily right. It may still be wrong because it is hard to understand; or because it is hard to maintain as program requirements change; or because its structure is different from the structure of the problem; or because we cannot be sure that it does indeed work.

These words were directed at programmers working in conventional data processing, but they should also bé reflected on by those whose concern is knowledge based systems. When knowledge based systems techniques were in their infancy it was necessary to show that it was feasible to construct such systems in a variety of domains, and while that was the aim the production of a working system was achievement enough. But we are past that stage now, and we should be considering how to build better engineered systems. Here Jackson's points about comprehensibility, maintainability and validation are all right on point. In this paper we will advocate an approach, *isomorphism*, to building knowledge based systems in the legal domain which will help to satisfy these concerns.

Jackson's important insight was to base the structure of the program on the structure of the data to be manipulated by the program. A chief reason for data driven design is that it provides a starting point which is objective: whereas techniques such as modular decomposition require the designer to exercise skill and judgement in determining the appropriate decomposition, with the result that the consequent design will inevitably contain subjective elements, if we start with the data we build from an objective foundation, and

there is the possibility that the resulting design will be similarly objective. What then is there that corresponds to the data in the case of a legal knowledge based system?

An obvious answer to this question, appropriate to many potential applications, is the legislation governing the area of law under consideration. Even in areas where the practitioners do not habitually refer to the legislation – the adjudication of social security claims in the UK, for example – the materials that they do use are ultimately founded on legislation. Basing a system on a representation of legislation was popularised by work done at Imperial College [Sergot *et al.*, 1986; Bench-Capon *et al.*, 1987]. The motivation for this work was primarily to demonstrate the power of logic programming, and emphasis was placed on the separation of the logic of the problem – the relevant legislation – from the manipulation performed by the inference mechanism so as to apply the legislation in carrying out a particular task grounded on that legislation. The representation was intended to be a declarative rendering of 'some chosen unambiguous interpretation of the selected legal sources' [Sergot, 1991]. It was not necessarily supposed to reflect the structure of these sources. Later developments, however, started to emphasise the additional advantages that would accrue if the representation was as faithful to the sources as possible. For example, it was argued in [Bench-Capon, 1989] that a truly faithful representation could serve some of the purposes of 'deep models' as used in some other areas of knowledge based systems, and [Routen, 1989] argued that the representation needed to reflect the structure of the legislation since 'some of the essential content of a statute is embodied in the organisation of the text'. In the following sections of this paper we will describe what is meant by a faithful representation, and discuss the advantages that result from such faithfulness.

2. Ismorphism

In an effort to give more precision to what is meant by a representation being faithful the term 'isomorphism' was used in works such as [Bench-Capon, 1989], [Karpf, 1989], [Vey Mestdagh, 1990] and [Bench-Capon and Forder, 1991]. Isomorphism has, of course, been used in other fields previously, but in this paper we will concern ourselves only with its use in the context of legal knowledge based systems. In essence the term is intended to capture the notion of creating a well defined correspondence between source documents and the representation of the information they contain used in the system. Karpf [1989] gives a particularly clear statement of what he means by 'isomorphism', and so it is worth recalling his definition here. Karpf lays down five conditions which a representation must fulfil if it is to be isomorphic:

(i) Each legal source is represented separately.
(ii) The representation preserves the structure of each legal source.
(iii) The representation preserves the traditional mutual relations, references and connections between the legal sources.
(iv) The representation of the legal sources and their mutual relations ... is separate from all other parts of the model, notably representation of queries and facts management.
(v) If procedural law is part of the domain of the model then the law module will have representation of material as well as procedural rules and it is demanded that the

whole system functions in accordance with and in the order following the procedural rules.

His first condition is a *sine qua non* but is not a peculiar feature of isomorphic representations: the importance of keeping sources separate and resisting the temptation to 'compile in' expert knowledge with regard to interpretation or operational definition is recognised in places where isomorphism is not an issue, such as [Bench-Capon *et al.*, 1987]. The second raises questions discussed in [Routen and Bench-Capon, 1991], and which will be further discussed below. The third condition reflects the need to keep explicit cross references in the text rather than compiling them away into a flat formalisation, for reasons advanced in [Routen, 1989] and [Johnson & Mead, 1991]. The fourth condition is related to the first, but the fifth raises interesting issues, since it appears to conflate the representation with the manipulation of the representation which gives rise to system behaviour. For that reason we would not insist on the fifth condition being met by any representation claiming to exhibit isomorphism: but the explicit representation of procedural knowledge remains a difficult issue, and the importance of system behaviour will be discussed below.

The important demand made by isomorphism is that there is a clear correspondence between items to be found in the source material and items to be found in the knowledge base. The direction needed is this: that it is possible to say of any item in the knowledge base that it derives from some self-contained unit in the source material. Ideally there would be a one to one correspondence between the knowledge base items and the source material items, but practical reasons may necessitate deviation from this. Where one to one correspondence is not achieved, however, it is important to relax the constraint only so that one source item corresponds to several knowledge base items and to maintain the prohibition on a single knowledge base item capturing the material from several source items. The need to avoid such conflation is perhaps best discussed in Routen & Bench-Capon, [1991]. Why cannot we maintain a one to one correspondence in all cases? There may be good reasons, such as efficiency or the need to use a particular expert system shell, why the executable representation has a less rich syntax that would be required to mirror the syntax of the source. Suppose for example we wished to use horn clauses so that we could execute the system in Prolog: and that the source contains a subsection of the form

S1 A person shall be *P* if, and only if, he is *Q* or *R* and *S*.

This might be represented within the logic programming paradigm, interpreting the condition as Q v (*R* & *S*) rather than (*Q* v *R*) & *S*, as

HC1 $p(X)$:- $q(X)$.

HC2 $p(X)$:- $r(X)$, $s(X)$.

Leaving aside the problem that part of the biconditional in S1 is represented by the treatment of negation as failure within this paradigm (defended in [Kowalski, 1989]), this has done some small violence to the form of the original statement. HC1 and HC2 in fact translate

S2a A person shall be *P* if he is *Q*

S2b A person shall be *P* if he is *R* and *S*

S2c No person shall be *P* unless he satisfies the conditions in S2a and S2b.

Whilst S1 and S2 are logically equivalent, there may well have been a motive in choosing to express the conditions for Pness in the form of S1 rather than S2. Attaining strict isomorphism would thus preclude the use of horn clauses and Prolog. For this reason we prefer in practice to use an intermediate representation which can exhibit one to one correspondence with the sources and which can then be transformed into the desired executable formalism. There is thus a one to one correspondence between source items and intermediate representation items, and, potentially, a one to many correspondence between intermediate representation items and executable knowledge base items. This makes it possible to follow links from source to executable knowledge base. The analysis process which results in the intermediate representation is described in Bench-Capon and Coenen, [1991] and Section 5 of this paper.

2.1. INTERMEDIATE REPRESENTATION

The construction of the intermediate representation, and the analysis which accompanies is, in our view, crucial to the sound development of a legal KBS, and so we will give some account of the nature of the representation here. A full description of the formalism can be found in Bench-Capon & Forder, [1991]. The intermediate representation consists of two parts: the 'class hierarchy' and the rule base. Classes in the hierarchy represent logical types and objects of concern to the domain, as they appear in the domain. Inheritance is by strict specialisation, giving the class hierarchy a sound logical interpretation. These classes also define the attributes which their instances may possess, and the values which these attributes can take. The overall effect is that the class hierarchy defines a vocabulary which can then be used in the development of the rule base. It is of crucial importance that this declaration of a vocabulary takes place, because this makes explicit the nature of the objects in the domain, which would otherwise remain implicit to be inferred by the reader of rules in accordance with his subjective interpretation. That this is needed is well illustrated by Glickfield's 'perpetuities people' [Glickfield, 1991]. In this paper Glickfield argues that perpetuities reasoning takes place in an abstract perpetuities world in which people have properties which run somewhat counter to those that everyday intuition would ascribe to people, such as the 'irrebutable presumption of fertility'. Analysis of the domain requires that the properties that the abstracted objects can have for the purposes of the domain be uncovered, and we would recommend their recording in the explicit form of a domain vocabulary as described above. While the perpetuities domain may be an extreme example, similar abstractions are in practice required in every domain.

Once this well defined vocabulary has been established, rules can be written. These rules will relate the domain objects in the type hierarchy, further defining the properties of those objects and the relations between them. For the most part these rules will be

derived from the source documents, although it may be also necessary to write some to make explicit knowledge latent in the objects, such as that men cannot be pregnant. The flexible syntax of the intermediate representation, and the moving of typing information into the class hierarchy, means that the rules of the intermediate representation can have a one to one correspondence with items in the source, and can mirror the structure of the source very closely.

The development of the intermediate representation is an important step in the construction of the system because it forces the essential analysis of the domain to take place, and ensures a clean separation of the design process from implementation considerations. We believe that the construction of a knowledge base is best seen as *representation* rather than *programming* and first building an intermediate representation ensures that the two processes are kept distinct. Moreover, it ensures that any assumptions or deviations from what might commonly be assumed are made explicit, that conflation of different sources are avoided, and that considerations of control are not allowed to dictate the representation. Finally we emphasise that the intermediate representation is a formalisation: we believe that this use of a formal language is a necessary improvement on a verbatim representation as recommended by Mead & Johnson, [1991], as it forces choices of structural interpretation as in Allen & Saxon, [1991] to be recorded rather than allowing the ambiguities of natural language to pass through to the representation.

3. Advantages of Isomorphism

When isomorphism was first discussed, the advantages were largely thought to lie in the verification and validation of the knowledge bases used, and in maintenance. These advantages largely reflect the kind of software engineering concerns that we noted when quoting Jackson in the introduction. Experience with applying this principle has, however, shown that there are other advantages, and that isomorphism has an impact on the system throughout the whole lifecycle. In the following sections we will discuss these advantages under four headings: the development methodology, the verification and validation of systems, the use of systems and the maintenance of systems. Of course, at this stage we can be no more than persuasive, and comment in the light of our own experiment in using these methods, which has been encouraging. But a single case cannot be a proof and whether these advantages are really to be found is a matter which cannot be settled other than by extensive empirical investigation. We believe, however, that the following sections do make out a persuasive case, and hope that the experiments that will test these arguments will come in the course of time.

3.1. ISOMORPHISM AS A DEVELOPMENT METHODOLOGY

Building a knowledge base by representing legislation and other source documents is a non-trivial and labour intensive task. One solution is to use highly skilled and experienced staff to perform this task, and to rely on their judgement to solve and work around problems. This is not really satisfactory, however: such staff are expensive, and often

reluctant to devote the necessary time and effort to what is essentially a mundane and unexciting business. The problem has been encountered before with respect to conventional software, and there the solution was to use more junior and less experienced staff, but to compensate for their lack of experience by providing them with a methodology which would guide them through the various stages, and encapsulate the best practice of more highly skilled practitioners. Jackson Structured Programming, popular as a methodology for the design of data processing programs [Jackson, 1975], is a good example.

It has been our experience that adopting the principle of isomorphism can help people for whom the representation of sources is a novel area, by guiding their choices, and revealing that they can approach even the largest source text in a methodical, piece by piece, section by section, manner. The basic instruction, to represent the sources in a form as close to the original as the representation language permits, imposes a discipline, and is an instruction that is, when coupled with an expressive intermediate representation language, relatively easy to follow, as it removes a number of concerns that might otherwise intrude. The clear cut, objective, nature of the desired result greatly diminishes the trepidation felt by a novice when confronted with a large source document. Moreover, it provides a framework in which general solutions to problematic constructs can be provided, thus avoiding the difficulties that such constructs present. Examples of such solutions can be found in Routen & Bench-Capon, [1991]).

A further advantage comes when the size of the source documents requires the work of representation to spread across a team of several people. The structure of the sources provides a natural way of dividing the work between team members, but this would avail nothing were we not confident that the resulting pieces could be smoothly integrated into a seamless final product. The principle of isomorphism should promote such integration because there is no conflation and no 'short cuts' (to use Johnson and Mead's useful term).

Isomorphism provides a style of representation which enables different people to work in a similar fashion, and this further enables the provision of tools which will facilitate and encourage that working style. Some such tools are described in the case study in the later sections of this paper.

Thus isomorphism aids the development of systems by providing a genuine methodology. It is a genuine methodology because it is teachable, its products are objective and do not rely on the developer's flair and insight, it provides a way of decomposing the task into tractable subtasks, and it is capable of being supported by tools. This has been illustrated on the MAKE project Bench-Capon & Coenen, [1991] in that two out of the three developers had not previously built a legal KBS and yet the Pilot was produced in six man months, including time taken to acquire knowledge of the tools described later. Once the use of the tools and techniques had been learned, progress was rapid, and resulted in a knowledge base in which the individual contribution were painlessly integrated.

3.2. ISOMORPHISM AND VALIDATION

Next we need to turn to the issue of whether the system does indeed work. This is of par-

ticular importance with a system concerned with legislation, because there, unlike other areas such as fault diagnosis, it is important not only that the answer be correct, but also that the answer be arrived at in the correct manner, and that the basis of the answer be seen to be correct also. For these reasons it is important that the material be susceptible to validation, not just through the running of test cases to ensure that the correct output is produced, but through an examination of the knowledge base itself to ensure that the correct material is being used to produce that output.

This latter activity can be performed only by an expert scrutinising the knowledge base and comparing it with the original sources which it claims to represent. A number of things are necessary for this to be a practical proposition. First it must be possible to detach fragments of the representation from the knowledge base without changing the meaning of those fragments. Formalisms which permit the context of the rest of the knowledge base and the control strategy which will manipulate the knowledge base to affect the import of items in the knowledge base require the knowledge base to be considered as a whole: we submit that this places an intolerable burden on the validating expert.

Separability of knowledge fragments, is not, however, enough. The validator must be able to tell of any fragment precisely which pieces of the source which it purports to represent, and to have confidence that the fragment is intended to represent this piece of source exhaustively. Maintenance of precise links between source and knowledge base, which is a feature of isomorphism, enables this confidence. The validator will have a good knowledge of the source and its structure, and reflecting this structure in the knowledge base will ease the task of navigating the knowledge base. Finally, at a syntactic level, the validator may well be reluctant to go through the kind of intellectual contortions necessary to recast the source in a normal form, and so it is helpful if the knowledge base syntax is as close as is possible to that of the source. It must, however, remembered that that the representation is unambiguous whereas the source may well contain ambiguities. For a discussion of the structural ambiguities latent in almost any piece of legislation written in natural language see Allen & Saxon, [1991]. It is important that the representation clearly show how these ambiguities have been resolved: this too is facilitated by the structural correspondences found in an isomorphism representation.

This kind of 'by eye' scrutiny of the knowledge base cannot, however, be enough to give complete confidence in the correctness of the knowledge base: it is also necessary to run test cases through the system. Running test cases, however, is not simply a matter of assembling some arbitrarily chosen previous cases and seeing if the correct result is achieved. Test data must be designed, so that each of the test cases will exercise specific areas of the knowledge base, both so that there can be some confidence that the test data has tried out all the various parts, and so that when the system fails a test, there can be some clear idea of which area of the knowledge base requires attention. This in turn means that the designer of the test data must have an understanding of the structure and components of the knowledge base.

Isomorphism can help here also, because the structure of the knowledge base is given by the structure of the sources and so does not need to be discovered independently and because the actual behaviour of the system is largely dictated by the structure of the origi-

nal legislation. This means that when aberrant behaviour is detected it can be noticed more easily, because there is a deviation in structure from what is expected, as well as possibly a wrong answer being produced. Further the offending piece of the knowledge base can then be identified and detached, its parent source fragment associated with it, and corrections made without any need to worry that surrounding parts of the knowledge base or source will be jeopardised by the changes.

3.3. ISOMORPHISM AND MAINTENANCE

It has long been recognised with regard to conventional systems that computer applications are not fixed once and for all upon delivery, but must be capable of being adapted to changing requirements and circumstances, and enhanced to meet extra requrements. The need for maintenance in a legal KBS cannot be overstated, and is now receiving some attention: see [Bench-Capon & Coenen, 1991], [Bratley *et al*, 1991a], [Coenen & Bench-Capon, 1991] and [Bratley *et al*, 1991b]. As an example of the extent of the maintenance problem, we may quote some figures from the application being developed as part of the MAKE project, dealing with claims for compensation from British Coal. The number of changes that will have to be made to this application each year is considerable. Regarding the law itself, each year, there are between 10 and 20 court judgements in British Coal cases and another 5 relating to other employers, but with significance for British Coal. There are up to 20 new relevant Statutory Instruments, and 10 technical instructions issued. In addition the policy of British Coal is modified from time to time, and some 10–15 such policy decisions are made in a typical year. All of these alterations need to be assimilated by the claims officers dealing with the claims. Other changes in the expertise of these employees arise out of changes in medical views, for example the acceptance that a particular substance can cause dermatitis; policy changes by other bodies, as when a particular firm of solicitors may start to issue writs if the claim is not settled in a certain period of time; and changes in the perception of methods of work or occupations. British Coal estimate that will require another 30 changes per year. Note that many of these changes will be relatively minor, but none the less the cumulative effect of these changes indicates the rapidity with which a knowledge base dealing with this sort of application would go out of date. Moreover these changes come in constantly and in an unpredictable manner: a single annual up-dating of the system would simply be unacceptable.

An important criterion for a maintainable system is that small and localised changes to the problem specification should result in small and localised changes to the program. This is the key motivation behind structured programming in conventional systems, and an important concern of later developments such as Object Orientated Programming. It is to help to satisfy this criterion that we need to insist that the structure of the program matches the structure of the problem: if the structures differ, the result is that a small change to the problem may ramify throughout the program. It is, of course, the central idea of the isomorphic approach that the structure of the knowledge base is a faithful reflection of the structure of the sources: this should ensure that limited changes to the source give rise to similarly limited changes to the knowledge base.

Maintenance is thus facilitated by two features of isomorphism: that a given source change can be related to a defined fragment of the knowledge base, and that this fragment can be removed from the knowledge base, altered and replaced with confidence that nothing else will be affected by the changes. Isomorphism enables us to identify the jeopardised parts of the knowledge base given a particular change to the source material, and it enables such jeopardised bits of the knowledge base to be removed cleanly from the knowledge base and worked on in isolation and replaced with all its links to the rest of the knowledge base intact. If, in the case of a legal KBS, a section of source, for example a section in an Act, is changed, an isomorphic representation will allow the maintenance engineer to 'trace' the change from the source through to the final rules representing this source. Any maintenance can then be implemented only on that rule set and, if the representation is truly isomorphic, no other rules will be affected.

It should, however, be emphasised that for the above maintenance technique to have its best effect, statements in the representation must be truly declarative. While almost all knowledge representation paradigms have declarativeness as an aspiration, in practice some formalisms require that a given fragment needs to be understood in the context of the rest of the knowledge base. If we want to ensure that localised changes to the source material result in correspondingly localised changes to the knowledge base, we must be sure that there are no ramifications of changes resulting from a subtle alteration of the meaning of the statement deriving from its context in the knowledge base. Such problems are discussed at greater length in [Bench-Capon & Forder, 1991].

Finally, by achieving a structural correspondence, we will also be able to record the provenance of all items of knowledge in the intermediate representation. This is not a simple matter in the absence of such isomorphism, but it is vital if changes are to be followed through from source to knowledge base.

3.4. ISOMORPHISM AND USING THE SYSTEM

There is a class of advantages which the isomorphic approach yields which do not have analogues in conventional structured programming. These are the advantages seen from the standpoint of the user, and were perhaps first noted in Vey Mestdagh [1990]. These relate to the the correspondence between the users' conceptualisation of the domain and the way the domain is conceptualised in the knowledge based system. In that paper Vey Mestdagh argues that it will be easier for the user to learn to use such a program, easier for the user to follow to the reasoning of such a program, and that the explanations from such a program will be easier to understand. We concur wholeheartedly with these points: the user will be, or can become, familiar with the domain as expressed in the sources, and the user will therefore expect the system to reflect this conceptualisation. Many of the problems with expert systems come from a mismatch between the rule based conceptualisation of the expert system and the conceptualisation of the user. Such a mismatch will lead to the user failing to recognise why certain questions are asked, and a dissatisfaction with the rule-orientated explanations that form the typical explanation facilities. Isomorphism should give a correspon-

dence between user and system conceptualisations, and hence make the behaviour of the system more transparent to the user.

Making the objective structure of the domain drive the behaviour of the system is further exploited in the MAKE system by the use of structured dialogues as previously used in the Local Office Demonstrator system (Forder 91). Instead of merely presenting the user with a form which requires him to answer questions without giving him a rationale for those questions or an idea of the consequences of his answers, the user can be presented with a dialogue which is tied to a particular section of the source, facilitating orientation, and which is structed in a way that corresponds with the source, allowing the user to employ his knowledge of those sources to select questions which are likely to have the most relevance and impact on outcome.

Isomorphism also helps when the user is faced with the need to come to a decision as to some open textured concept. In systems of the sort under consideration much of the resolution of open texture is thrust on the judgement of the user. It is clearly impossible to anticipate every situation, and the resolution of open texture must therefore be left to the user. The user must, however, resolve the question not in an arbitrary way, but must take into account the guidance provided for him by case law and his employers. Guidance and decisions are typically associated with particular fragments of the legislation, and because the rules reflect this structure, the specific guidance and decisions relevant to the point of the processing which has been arrived out can be selected and presented to the user. Were the rules not isomorphic with the legislation, the associations could not be made. The need for harmonisation of rules, legislation and guidance is further discussed in Bench-Capon Coenen & Dunne, [1991].

The above, of course, assumes that the user *does* conceptualise the domain in terms of the source documents. We would contend that this is generally true – although the source document in question may not be the legislation. Thus U.S. perpetuities law seems to be conceptualised in terms of John Chipman Gray's commentary (Gray, 1886), according to Glickfield (1991), and the UK Social Security law is often conceptualised by adjudicators in terms of the manuals issued by the Department of Social Security rather than the original legislation. If this is the case, and the commentary or manual has a significantly different structure from the legislation, structuring the system on the legislation may lead to a mismatch with the user's conceptualisation. In such cases the correct response is not to abandon isomorphism – for the arbitrary structuring of the knowledge engineer is unlikely to accord with the user's conceptualisation either – but to achieve isomorphism with the favoured source document. This may slow the maintenance process, in that the knowledge base will need to await the revised commentary or manual before it can updated, but when the chosen source is available the maintenance benefits of isomorphism will accrue. Ideally analysis will also discover links between the commentary and legislation also which can be further exploited here. So it may be the case that in order to get the advantages of harmony with the user's conception we need to base our system on a source other than legislation to get the benefits outlined in this section, unless we are willing to educate the users into a new legislation based conceptualisation.

4. Objections to Isomorphism

At this point we should consider an objection to isomorphism. This is due to Marek Sergot, and is latent in Sergot *et al.* [1991], in which the more traditional logic programming method of top down development was followed. The objection was more fully developed in Sergot's presentation of the paper, and in subsequent private discussion. In essence the objection is this: the isomorphism approach is all very well if the legislation is itself well structured. In such a case, the structure of the problem, the structure of the legislation, and the structure of an isomorphic knowledge base would all be in harmony. It is, however, often the case that the legislation is not well structured. Often repeated amendments and 'patching' mean that the legislation is itself a complete mess, and fails to reflect the real structure of the domain. In such a case, basing the structure of the knowledge base on legislation would lead to a poorly structured knowledge base, which failed to correspond to the 'real world' problem.

This is a serious objection, but one which turns on what is meant by 'poorly structured'. It is true that a knowledge base structured according to poor legislation will fail to correspond to the real world problem. However, the legislation is what we have to work with, and although such legislation is in the long run a good candidate for consolidation and re-writing, in the short term it is likely that this poor legislation will itself continue to be patched and amended. If the knowledge engineer tries to produce an interpretation which is the 'simplest and most perspicuous', any subsequent amendments will require a re-interpretation and fail to receive the advantages of limitations of changes that it was an argued an isomorphic representation would have. Moreover, no matter how poor the legislation, it is that structure that is familiar and available to the validating experts and potential users: and their knowledge of such structure will be lost in a wholesale re-interpretation. We therefore maintain that even an inconveniently structured piece of legislation should have its structure respected.

Of course, we are not arguing that the isomorphic approach is the only way to build a legal knowledge based system. As was explained in Bench-Capon, [1991b] there are many entry points to the knowledge required, and the entry point to choose depends on the needs of the application, and the sort of application that needs to emerge. None the less, we maintain that the advantages we have cited for the isomorphic approach will still accrue even when the legislation itself contain structural weaknesses.

The admission that isomorphism may not always be the best policy leads to the question of when we believe that it should be adopted. We noted some reasons in the last section why the knowledge base might need to be isomorphic with a source other than the legislation, but there may be reasons why even this is not enough of a concession. Considerations such as the scale of the system, the expected use and life time of the system, the intended users of the systems and the relation of this system to other systems all need to be taken into account. Isomorphism, and the associated analysis process may increase the effort required and it might be that such a thorough job is not needed. On the other hand if it is intended that the knowledge base should be transferable between several tasks, an isomorphic foundation is essential. It is partly an empirical matter which

cannot be resolved until a range of systems built using both isomorphic and non-isomorphic methods have been developed and put into practical use. All we argue here is that badly structured legislation is not in itself a sufficient reason to abandon the isomorphic approach.

A second objection concerns the efficiency of the program. Objectors argue that the pressure to produce an isomorphic representation will invariably carry with it a certain amount of baggage which will make the program less efficient than it could be. This objection is related to program optimisation in general, and it it is worth quoting Jackson's two rules on optimisation (Jackson, 1975):

> Rule 1. Don't do it.
> Rule 2 (for experts only). Don't do it yet – that is, not until you have a perfectly clear and unoptimized solution.

There will always be a trade off between optimisation and clarity of design. As Jackson says:

> Two points should always be remembered: first, optimization makes a system less reliable and harder to maintain, and therefore more expensive to build and operate; second, because optimization obscures structure it is difficult to improve the efficiency of a system which is partly optimized.

We therefore recommend that optimisation should not be at the forefront of the mind when constructing a system. Our advocacy of an intermediate representation is intended to re-inforce the idea that such implementation concerns should be kept distinct from the representation process. Of course, when moving from the intermediate to an executable representation it may be that the temptation to optimise is too great to resist: none the less the optimiser should be clear as to the price he is paying for any extra perfórmance he may gain.

5. A Case Study

In this part of the paper, the development of a regulation base Knowledge Based System (KBS) using the Make Authoring and Development Environment (MADE) and methodology for British Coal is presented as a practical case study of isomorphic development. The knowledge base was developed as part of the Maintenance Assistance for Knowledge Engineers (MAKE) project. MAKE is a collaborative project between Liverpool University, International Computers Limited (ICL) and British Coal directed at developing a suite of maintenance tools, suitable for use on regulation based KBS, supported by a methodology that encourages the prodution of maintainable systems. MADE is the result of the latter. The methodology supports KBS maintenance by basing system development on the isomorphic principles advocated above.

5.1. 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, 1989). 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 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 of the analysis is a set of rules, contained in a Rule Base, and a hierarchical set of objects, called the Class Hierarchy. The Class Hierarchy defines the vocabulary of objects, their slots, and the values which can fill these slots. This vocabulary must then be used when constructing rules. It also describes the state of the application at any given time. The Rule Base and the Class Hierarchy are developed in an intermediate representation called MIR (Make Intermediate Representation). This is essentially a simple language to define objects and rules using a language resembling first order predicate logic with some extensions, for example to handle arithmetic. The nature of this intermediate representation and the advantages to be gained are fully described in (Bench-Capon and Forder, 1991).

The MIR Rule Base and Class Hierarchy are compiled into the target representation used in the MAKE Inference Engine (MIE). Currently this executable representation consists of a Clausal form referred to as CMIR (Compiled MIR). Other target representations could be used equally well. The MADE window is the user interface through which the MAKE maintenance tools are invoked. These all operate on the compiled version of the MIR. A full discussion of the tools available under the MAKE project is given in (Bench-Capon and Coenen, 1991).

The MAPPE Base Window provides the end user view of the application. It is organised as a hierarchy of Topics or Question forms which the user is invited to fill in. The answers to questions are 'asserted' as the values of slots and so trigger rules in the Rule base. The structure of this topic hierarchy is used to reflect the natural decomposition of the problem with respect to the task that the application is required to support.

5.2. SYSTEM DEVELOPMENT USING MADE

System development using MADE consists of an iterative process in which repeated cycles are performed. These iterations may proceed indefinitely, in that requirements will change over time and source material will be superseded. Each iteration involves the construction of four levels of (KANT) structures as follows:-

(1) Source Documents: These are imported into the KANT system, preserving their original format, ready for analysis. KANT supports the 'cutting and pasting' of words, phrases etc from these documents into other structures, and also the definition of 'Links' within these sources and between the sources and the new structures.

(2) Freestyle KANT Structures: These can best be described as 'Structured English' notes. KANT readily supports the structuring of such 'notes' (called 'Nodes') through the use of child and sibling relations, and through separable fields within a

node. The author is free to construct any number of such structures, with a variety of contents. However the proposed methodology strongly suggests a number of specific structures, which are detailed later in this document.

(3) MIR Structures: These are yet more KANT structures, but constructed according to the defined syntax of the intermediate representation. They not only define rules and classes, but also the topics and questions to be used in the dialogue with the end user.

(4) Compiled MIR: This is an executable representation of the MIR, in that it can be interpreted by the MAKE Inference Engine (MIE), forming the executable application. It is envisaged that alternative compilers can be written in order to translate MIR into different target representations. Thus if authoring and maintenance tasks can be confined to operate on MIR Structures and preceding structures/sources, then the MADE can be used as a generalised development environment.

For Legislation Based KBS systems the proposed methodology suggests three strands of authoring material as shown in Fig. 1. The dialogue strand results in the user interface and the object classes and rules strands in the Class Hierarchy and Rule Base respectively. As can be seen a number of Freestyle KANT structures are used in the analysis gradually building up to the definition of rules, classes, topics and questions in MIR. It is important to remember that throughout this analysis links are being constructed so that any MIR definition can be traced back to the original source material and vice versa. The method actually encourages the isomorphism between source and MIR definitions

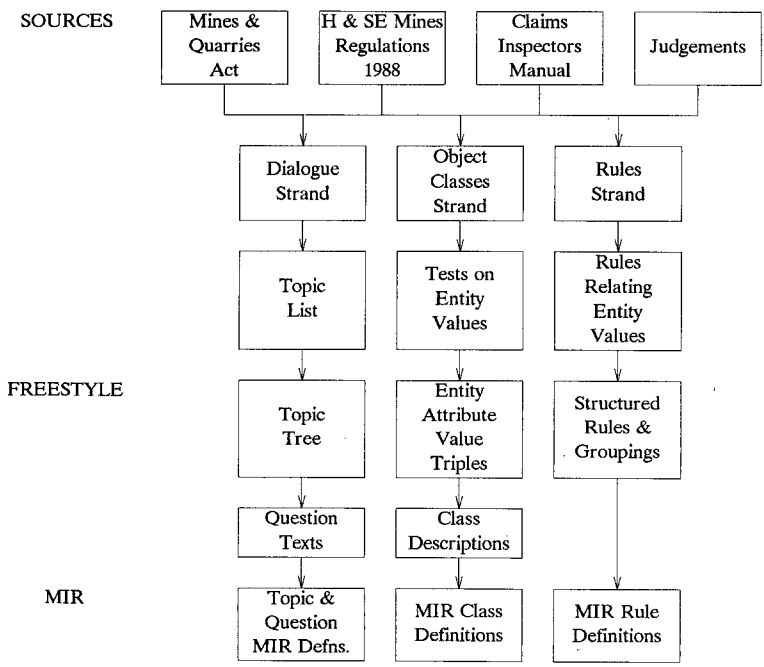

Fig. 1. Strands of authoring material for a regulation based KBS.

through the cut and paste operations offered by KANT. The MIR definitions are then compiled into the target representation, CMIR.

In the following sub-sections each stage in the development of the Class Hierarchy and Rule Base strands will be considered in further detail by considering a fragment of the British Coal application in terms of the four levels of KANT structure created on each iteration. This fragment is concerned with the safety of buildings and structures on the surface of a mine. This fragment will be used to build a sample KB, consisting of a Rule Base and Class Hierarchy, to establish whether the mine manager has complied with the Regulations or not.

5.3. IMPORTATION OF SOURCE DOCUMENTS

Importation is the process of taking raw texts and moving them into a KANT source format but preserving its original appearance. The source documents used in the development of the British Coal application were:

(i) The Mines & Quarries act 1954.
(ii) The H & SE Mines (safety of Exit) Regulations 1988.
(iii) The Claims inspectors Manual (1990).
(iv) A number of significant judgements.

If we consider the fragment of the British Coal application concerned with the safety of surface buildings and structures the relevant section from the Mines and Quarries Act is as follows:

Buildings, Structures, Means of Access, &c.

86. All buildings and structures on the surface of a mine shall be kept in safe condition.

87. (1) There shall be provided and maintained safe means of access to every place in or on a building or structure on the surface of a mine, being a place at which any person has at any time to work.

(2) Where a person is to work at any such place as aforesaid from which he will be liable to fall a distance of more than 10 feet, then, unless the place is one which affords secure foothold and, where necessary, secure handhold, means shall be provided by fencing or otherwise for ensuring his safety.

In its imported form the above will consist of four blocks of text, the title and paragraphs 86, 87.1 and 87.2. These blocks or parts of them can then be copied into KANT Structures where further analysis can take place.

5.4. FREESTYLE KANT STRUCTURES

After importation knowledge analysis is commenced by examining the relevant passages in the source material to gain a broad understanding of the domain. In some cases it is advantageous to collect together the subset of rules and regulations that are applicable to a particular part of the final application into a single KANT Structure file, using the cut and paste facilities provided with KANT. Fig. 2 gives a KANT window showing the relevant legislation in Freestyle KANT Structure format. Note that at the same time links will be created between the nodes and the source. At any time the author may follow these links back to the source or forward to the rules and objects in the final application, when they have been developed.

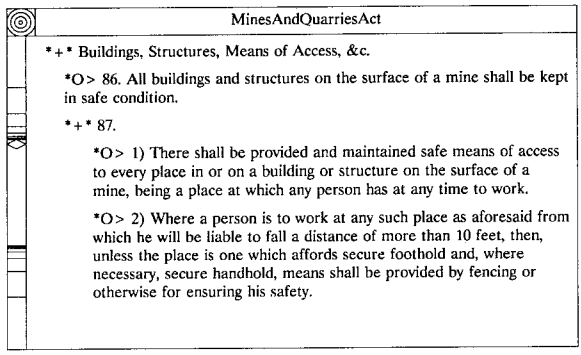

Fig. 2. Source Freestyle KANT Structure.

It is suggested that analysis be implemented by following the steps listed below.

(1) Pass through the source identifying attributes and their entities and the tests applied to them.
(2) Collect the attributes of and tests on each entity into a Freestyle KANT structure.
(3) Collect Entity Attribute Value triples (EAVs) together and form another Freestyle KANT structure.

If we consider the above imported regulations we can identify three principal entities, 'buildings and structures on the surface', 'places where persons work' and 'high places where persons work', i.e. buildings, work places and high work places. It is important to appreciate that high work places are a type of work place and that work places are a type of building or structure. We can also identify a fourth implied entity, namely the mine manager who is responsible for ensuring that the regulations are complied with. The entity 'mine manager' can be considered to be a type of 'manager' which in turn is a type of 'person'. We could identify further entities such as 'means of access', 'foothold' and 'handhold'. However within the example fragment of legislation no tests are carried out on these entities and so they can be considered to not require independent representation. There is also a question of grain size to be considered here. For example we may consider buildings, structures, surfaces and mines to be four separate types of entities. However in the piece of legislation under consideration here no distinction is made between them and thus they can effectively be considered as a single entity since they only occur in the same phrase. Should these entities occur separately elsewhere in the legislation they will, of course, need separate representation as subclasses of the composite derived from this section. But it is important that the composite object exists, also: the rules derived from this piece of the source need to be attached to the composite object. It is partly this desire to locate rules at the correct epistemic level that motivates the use of a type hierarchy. This deviation between the objects discovered by analysis and what might result from everyday intuition is an example of need to abstract and explicitly record an ontology tailored to the domain discussed in 2.1. above.

Having identified the relevant entities contained in the Regulations we can consider the tests that may be performed on them. For example from Section 86 of the Regulations we

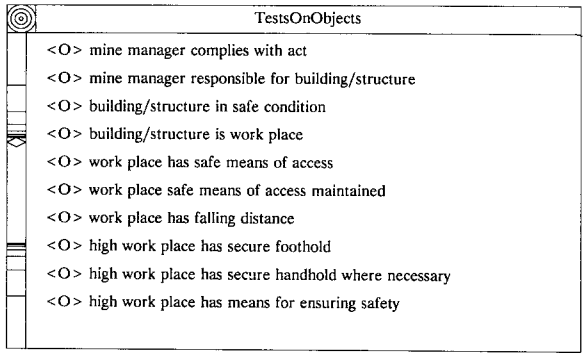

Fig. 3. Tests On Objects Freestyle KANT Structures.

can identify a test to determine whether a building/structure is in a safe condition. Similarly from Section 87.1 we can identify a test to determine whether a building/structure is a work place. At a higher level we can presume the existence of a test to determine whether a manager complies with the act or whether he/she is responsible for the building/structure in question. All relevant tests can be collected into a single KANT structure file and displayed in a KANT window as shown in Fig. 3.

We are now in a position to consider EAVs. We can identify the attributes associated with the entities and the values those attributes can take by inspection of the structure 'Tests on Objects' and the source. Firstly, from the Tests On Objects structure, we can extract an attribute for the entity 'mine manager' that the manager complies with act and that this attribute can take the value 'true' or the value 'false'. Alternatively if we consider the entity 'in safe condition' which can take the value 'true' or the value 'false', or 'is work place' which can also take the values true or false. The EAV triples identified are stored in another KANT structure file of the form shown in Fig. 4.

In the above EAV and Tests On Object structures care has been take to ensure that each structure or sub-structure bears a direct, one-to-one, correspondence with the source material i.e. isomorphism with the source is maintained. Further note that the above

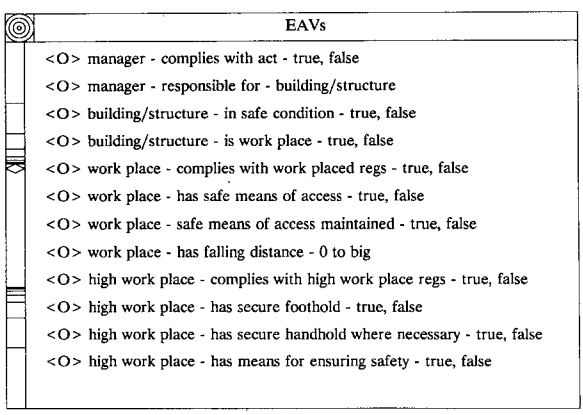

Fig. 4. EAV Freestyle KANT Structures.

analysis stages are designed to minimise and contain the need for redoing the analysis as a result of problems located at later steps. It is recommended that the author uses source material terminology for entity, attribute and value names wherever possible.

5.5. MIR STRUCTURES

In the Intermediate Representation stage the Tests On Objects and EAV structures are developed into a MIR Class Hierarchy and Class Hierarchy. In this case the Classes and Rules are stored in MIR KANT structures.

Considering the Class hierarchy first, we create the hierarchy by collecting the attributes and tests for each entity and defining the attribute set for each entity. At the same time checks should be made for duplication and overlap. The entities are collected together in a Class Lattice, and common attributes are placed up to their highest common level. The result is a formal definition of entities, attributes and types in the MIR language, explicitly recording the appropriate ontology of the domain uncovered by analysis. For the fragment of KB under consideration here the Class Hierarchy in MIR will be of the form given in Fig. 5. Note that the Class Hierarchy is arranged so that subclasses inherit attributes from super-classes. For example a 'HighWorkPlace' inherits the attribute 'keptInSafeCondition'.

With respect to the Rule Base we can pass through the sources, Tests On Objects and EAV structures again and identify rules. Thus we can construct a rule to establish whether the mine manager complies with the Act:

manager CompliesWithAct is true
iff
BuildingStructureInSafeCondition is true
and
  BuildingStructure IsWorkPlace is false
  or

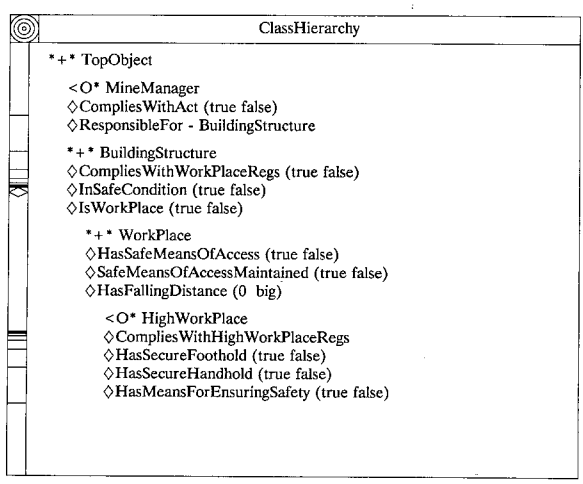

Fig. 5. Class Hierarchy MIR KANY Structure.

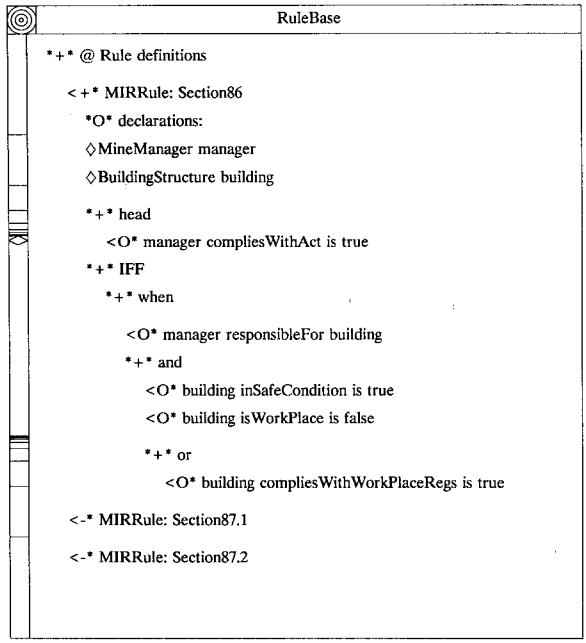

Fig. 6. Rule Base MIR KANT Structure.

BuildingStructure CompliesWithWorkPlaceRegs is true
when
(manager Responsible BuildingStructure)

Here 'manager' and 'BuildingStructure' are variables typed the appropriate class of entity. In MIR this will appear as illustrated in Fig. 6. Note that the links back to the structures from which the rule was derived are shown. The figure also illustrates two additional rules referring to this section of the source in folded form.

5.6. COMPILED MIR

During the compilation stage the MIR Class Hierarchy and Rule Base are compiled into objects and rules in the executable target representation, CMIR. In the case of the Rule Base this will result in a set of clauses. If we consider the rule constructed in the previous section this will compile into the following clauses.

(manager CompliesWithAct is true
if
((BuildingStructure InSafeCondition is true)
and
(BuildingStructure IsWorkPlace is false)
and
(manager Responsible BuildingStructure)))

```
(manager CompliesWithAct is true)
if
((BuildingStructure InSafeCondition is true)
and
(BuildingStructure CompliesWithWorkPlaceRegs is true)
and
(manager Responsible BuildingStructure))

(not manager CompliesWithAct is true)
if
((not BuildingStructure InSafeCondition is true)
and
(manager Responsible BuildingStructure))

(not manager CompliesWithAct is true)
if
((not BuildingStructure IsWorkPlace is false)
and
(not BuildingStructure CompliesWithWorkPlaceRegs is true)
and
(manager Responsible BuildingStructure))
```

These clauses show the inferences that are licensed by the rule in Figure 6. Tools, described in Bench-Capon & Coenen, [1991] and Coenen & Bench-Capon [1991] are available for displaying and animating this version of the knowledge base.

6. Summary of the Case Study

The MAKE Application Development Environment and Methodology have been described in terms of

- The way in which it preserves an isomorphism between source material and, ultimately, the end user application.
- The use of the MAKE Intermediate Representation as a general re-targetable step before delivery in a specific language or environment.
- An authoring process, supported by Freestyle KANT structures, which exploits audit trail information, and thus encourages the preservation of the aforementioned isomorphism.
- The linking of structures to form an audit trail which will later be used in various maintenance activities.
- The heavy use of graphical presentation of the material produced.

It is argued that all these features put together, as in the MADE system, combine to make a very practical development environment for regulation based KBSs which addresses the whole application life cycle.

7. Conclusion

In this paper we have discussed the isomorphic approach to the construction of legal knowledge based systems, and have argued that a number of gains come from this approach, and drawn comparisons between this approach, and its benefits to the best

practice in the software engineering of conventional systems. We have illustrated the approach and some of its gains by a detailed case study of the approach in operation. We emphasise that we do not believe that this is the only way to build such a system: we do maintain, however, that for a large class of potential applications it is the best approach.

Acknowledgements

The work described above was 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 International Computers Limited, the University of Liverpool and British Coal. In addition the authors would like to express special thanks to Charlie Portman for his working paper 'Authoring in MAKE.' We greatly benefited from a number of discussions with a variety of people at the Third International Conference on AI and Law held in Oxford in June 1991: especially Marek Sergot, Paul Bratley and Kees de Vey Mestdagh. Finally we would like to thank Don Berman for his comments on an earlier draft of this paper.

References

Allen, L. E. & Saxon C. S. 1991. More IA Needed in AI: Interpretation Assistance for Coping with the Problem of Multiple Structural Intepretations. In Proceedings of *The Third International Conference on AI and Law*, 53–61. Oxford:ACM Press.

Bench-Capon, T.J.M., Robinson, G. O. Routen, T. W. & Sergot, M. J. 1987. Logic Programming for Large Scale Applications in Law: A Formalisation of Supplementary Benefit Legislation. In Proceedings of *The First International Conference on AI and Law*, 190–198. Boston:ACM Press.

Bench-Capon, T. J. M. 1989. Deep Models, Normative Reasoning and Legal Expert Systems. In Proceedings of *The Second International Conference on AI and Law*, 37–45. Vancouver:ACM Press.

Bench-Capon, T. J. M & Forder, J. M. 1991. Knowledge Representation for Legal Applications. In *Knowledge Based Systems and Legal Applications*, ed., T. J. M. Bench-Capon, 245–264. Academic Press.

Bench-Capon, T. J. M. & Coenen, F. P. 1991. Practical Application of KBS to Law: The Crucial Role of Maintenance. In *Legal Knowledge Based Systems, Aims for Research and Development*, C. van Noortwyk, A. H. J. Schmidt, R. G. F. Winkels, 5–17. Lelystad, The Netherlands:Koninklijke Vernande BV.

Bench-Capon, T. J. M. & Coenen F. Exploiting Isomorphism: Development of a KBS to Support British Coal Insurance Claims. In Proceedings of *The Third International Conference on AI and Law*, 62–68. Oxford:ACM Press.

Bench-Capon, T. J. M. 1991. KBS Applied to Law: A Framework for Discussion. In Bench-Capon T. J. M. (ed), *Knowledge Based Systems and Legal Applications*, ed. T. J. M. Bench-Capon, 329–342. Academic Press.

Bench-Capon, T. J. M., Coenen, F. P. & Dunne, P. E. S. Integrating Legal Support Systems Through Document Models. *Expert Systems with Applications* (to appear).

Bratley, P., Fremont, J., Mackaay, E. & Poulin, D. 1991. Coping with Change. In Proceedings of *The Third International Conference on AI and Law*, 69–76. Oxford:ACM Press.

Bratley, P., Poulin, D. & Savoy, J. 1991. The Effect of Change on Legal Applications. In *Proceedings of DEXA Berlin*, ed. D. Karagiannis, 436–441.

Coenen, F. & Bench-Capon, T. 1991. A Graphical Interactive Tool for KBS Maintenance. In *Proceedings of DEXA Berlin*, ed. D. Karagiannis, 166–171.

Glickfield, Barnett W. 1991. Perpetuities Reasoning Captured and Automated in a Logic Program. In Proceedings of *The Third International Conference on AI and Law*, 128–136. Oxford:ACM Press.

Gray, J. 1986. *The Rule Against Perpetuities*. Boston: Little, Brown and Company.

Jackson, M.A. 1975. *Principles of Program Design*. Academic Press.

Johnson, P. & Mead, D. 1991. Legislative Knowledge Based Systems for Public Administration – Some

Practical Issues. In Proceedings of *The Third International Conference on AI and Law,* 108–117. Oxford: ACM Press.

Karpf, Jorgen. 1989. *Quality Assurance of Legal Expert Systems.* Jurimatics No 8, Copenhagen Business School.

Kowalski, R. A. 1989. The Treatment of Negation in Logic Programs Representing Legislation. In Proceedings of *The Second International Conference on AI and Law*, 11–15. Vancouver:ACM Press.

Routen, Tom. 1989. Hierarchically Organised Formalisations. In Proceedings of *The Second International Conference on AI and Law*, 242–250. Vancouver:ACM Press.

Routen T. W. & Bench-Capon, T. J. M. 1991. Hierarchical Formalisations. *International Journal of Man Machine Studies* 1:69–93.

Sergot, M. J., Sadri, F., Kowalski, R. A., Kriwaczek, F., Hammond, P. & Cory, H. T. 1986. The British Nationality Act as a Logic Program. *Communications of the ACM* 29:5 370–386

Sergot, M. J. 1991. The representation of Law in Computer Programs. In *Knowledge Based Systems and Legal Applications*, ed. T. J. M. Bench-Capon, 3–68. Academic Press.

Sergot, M. J. Kamble, A. S. & Bajaj, K. K. 1991. Indian Civil Service Pension Rules: A Case Study in Applying Logic Programming to Regulations. In Proceedings of *The Third International Conference on AI and Law*, 118–127. Oxford:ACM Press.

Storrs, G. E. & Burton, C. P. 1989. KANT, A Knowledge Analysis Tool. *ICL Technical Journal* 6:572–581.

de Vey Mestdagh, C. N. J. 1990. How Artificial Should Artificial Intelligence Be? In *Legal Knowledge Based Systems: An Overview of Criteria for Validation and Practical Use*, eds. Kracht D., de Vey Mestdagh, C. N. J. and Svensson, J. S. 93–104. Lelystad, The Netherlands:Koninklijke Vermade.