A METHODOLOGY FOR THE MAINTENANCE OF KNOWLEDGE BASED SYSTEMS

Frans Coenen

Department of Computer Science,
The University of Liverpool, Chadwick Building,
Liverpool L69 3BX, United Kingdom.

ABSTRACT.

A proposed methodology for the maintenance of Knowledge Based Systems (KBS) is discussed. Over the last fifteen years KBSs have undergone a period of steady development and have gained maturity. However they now face a crises similar to that experienced for traditional software systems during the software crisis of the 1960s. Facilities to develop large KBSs exists, however they have still not gained general acceptance. Contrary to initial assumptions the reason for this can be firmly attributed to the difficulties encountered when maintaining these systems. Initially it was assumed that KBSs could be maintained simply by adding new KB elements or replacing existing elements. However this "simplicity" proved to be allusary as indicated by the experience of R1/XCON [Soloway *et. al.* 1987].

In this paper a KBS maintenance methodology is suggested consisting of a number of distinct stages. These include the identification of (a) the area of the KB, both at the global and local levels, that will require attention, (b) the nature of the maintenance action(s) to be implemented and (c) testing the effect of the proposed change. Each stage is considered in detail and the available software support discussed.

KEYWORDS: Knowledge Based Systems, Maintenance, KBS Maintenance Methodology, MAKE Project, KBS Development, KBS Testing

1. INTRODUCTION.

In this paper we outline a proposed methodology for the maintenance of Knowledge Based Systems (KBSs). The methodology was developed as a result of work carried out on the MAKE (Maintenance Assistance for Knowledge Engineers) project, a two year collaborative research project between Liverpool University, International Computers Limited and British Coal. The project was concerned with the specification and development of software tools to support the KBS maintenance task and is described elsewhere; interested readers are referred to [Bench-Capon and Coenen 1991a and Coenen and Bench-Capon 1992a].

We introduce the concept of KBS maintenance by first considering the role of maintenance in traditional software systems and attempt to draw some parallels with KBS. Our conclusion is that the field of KBS maintenance has been soarly neglected and that this is the principal reason why KBSs have failed to gain the general acceptance which was expected when they first came to prominence. We then outline our proposed methodology. By definition a methodology dictates a way of working; however as observed by other authors, for example Motta [1990], no methodology is complete without software support. In the remainder of this paper we consider the stages in the methodology and in particular the software tools that can be used to support each stage. Many of the suggested tools have been developed as part of the MAKE project; others form part of existing KBS development environments and verification and validation systems. Where deficiencies are found additional tools are suggested.

2. TRADITIONAL SOFTWARE MAINTENANCE AND KBSs.

The term maintenance, for traditional software systems, can be traced back to the software crisis of the 1960s when the introduction of micro-chip technology lead to production of the first third generation computers. The power of these machines was several orders of magnitude in excess of the power of the machines of the previous generation. The result was a corresponding increase in the possible applications for software programs encompassing a whole new range of, previously unrealisable, tasks. However programmers quickly got into difficulty as it was discovered that writing these much larger and more powerful programs was not simply a process of scaling up the process used to develop second generation software. While hardware costs were tumbling software costs soared. There was an urgent need for new techniques and methods which would allow the production of large software systems in an economic manner. The result was the discipline of software engineering.

Software engineering practices however do not guarantee the production of systems that are error-free. Thus, periodically, after a system has been delivered to the end user there is a necessity for any errors discovered during its operation to be corrected. This has became known as software maintenance. In addition, to increase the useful life span of a software system, it has to be updated at intervals as a result of changes in the user's requirements and changes in the environment in which it operated. The process of updating software systems has thus also been identified with the overall maintenance task.

The discipline of software engineering and its sub-discipline of software maintenance are now well understood for traditional systems.

KBSs, over the last fifteen years, have undergone a period of steady development and have now gained maturity. However they currently face a crises similar to that experienced for traditional software systems in the 1960s as described above. Facilities to develop large KBSs exists, for example knowledge analysis methodologies such as KADS [Wielinga *et. al.* 1991], environments such as KEATS [Motta *et. al.* 1990], and shells such as NEXPERT OBJECT [Aiken and Sheng 1990]. However KBSs have still not gained general acceptance; there are still a great many applications where the technology has not been applied. Why is this?

To answer this question we must look to the philosophy behind KBSs. By definition these systems are designed to behave like an expert in some specific domain of operation. As the expression "knowledge based system" indicates, they are based on knowledge of a specific domain. However all knowledge is in a permanent state of flux [Gardenfors 1988]. Thus for a KBS to be commercially viable it must be able to respond to the changes in the domain knowledge on which it is based, i.e. it needs to be maintained. However the maintenance of these systems has proven to be a difficult task. There are a number of reasons for this:-

- The nature of the representation techniques used do not lend themselves to traditional maintenance techniques.
- The frequency with which knowledge changes in many KBS domains is far in excess of that experienced for traditional software systems.
- There is almost a complete lack of suitable methodologies, techniques and software support for the KBS maintenance task.

The answer to the question of why KBSs have failed to gain more general acceptance can thus be firmly attributed to their lack of maintainability. This has been observed by a number of authors, examples include Bench-Capon and Coenen [1991b] and Bratley *et. al.* [1991].

3. A MAINTENANCE METHODOLOGY.

The KBS maintenance task can be considered to consist of a number of distinct and hence identifiable stages which can be ordered to form a maintenance methodology. This is illustrated in Figure 1.

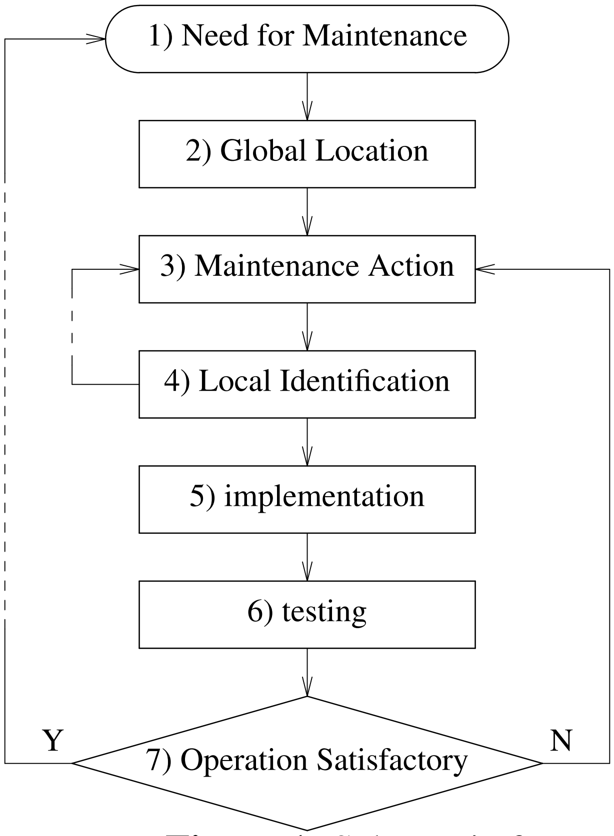

Figure 1: Schematic for KBS Maintenance Methodology.

Generally the end users of a KBS will be the first to become aware of the need for maintenance. The end users of course will not necessarily be domain experts, but they will be familiar with the environment in which the KBS is expected to operate, usually much more so than the maintainers of the system. The need for maintenance is then passed on to the maintainers in the form of bug reports and change requests. The initial detection of the need for maintenance is thus an end user activity and is therefore considered to be outside the scope of the suggested methodology.

Having established that some maintenance is required the next step is to identify, from a global perspective, the section of the KB that will require attention and then determine the nature of the maintenance action that will be required. If we consider a production rule system we can envisage four distinct possibilities (a) adding a rule, (b) removing a rule, (c) replacing a rule and (d) modifying a rule. The distinction between (c) and (d) is that the first consists of the removal of a rule and the adding of a new rule (b and a) while the second involves attention to an individual element of a rule, for example a statement in the condition part. This over simplifies the task, in practice a number of these possibilities will require implementation in conjunction with one another, but for illustrative purposes this classification will suffice.

Having determined the immediate nature of the required maintenance action the next step is to identify, locally, the elements in the KB that will also require attention as a result of the proposed change. Again, considering a production rule system, if the maintenance action identified consists of the removal of a single rule, the rules which call and are called by the rule to be removed will require attention. Alternatively if the rule is a leaf rule the mechanism for handling askable propositions will require attention.

When the local elements of the KB that might require attention have been identified the next step is to consider the further maintenance actions required with respect to these elements. In the above example of the removal of a rule this will require the modification of the rules that call it and are called by it. Thus a feature of the methodology is a loop where steps 3 and 4 are repeated until the extended sub-set of the part of the KB affected by the initial change required has been identified.

The next stage is the implementation stage. This should be carried out in a structured manner following the path mapped out by the looping process just described. In this manner the required maintenance actions will be implemented in a logical and sequential manner.

During the testing phase the implemented changes are verified and validated. Testing should only be implemented on the sub-set of KB elements which will be affected by the change. This will not just consist of the individual elements on which some maintenance action has been implemented but will also involve those elements indirectly effected by the change (it is assumed that this will not involve the entire KB). We refer to this group of elements as the jeopardised sub-set. It may be that, during this testing stage, it is realised that further maintenance is required in which case a return to step 3 should be instigated.

4. GLOBAL LOCATION.

The first task in the proposed maintenance methodology is to identify the area in the KB that will be affected by a required change. How this is achieved will depend on the nature of the required change. We can identify two distinct situations, change requests and bug reports, which will require two distinct approaches.

Change requests result from changes in the user requirements or changes in the environment in which a KBS is intended to operate. These can be referred to, using Swansons classification [Swanson 1991], as perfective and adaptive maintenance respectively although in some situations adaptive maintenance may also result from changes in the environment in which the KBS is expected to operate. There are a number of development philosophies/techniques that can be used to facilitate the detection of the area within a KBS which will require attention. These include isomorphism [Bench-Capon and Coenen 1991c], linking [Coenen and Bench-Capon 1992a] and indexing [Hicks 1990].

Bug reports result from errors in the operation of a KBS, for example where an incorrect result has been produced or where no result has been produced. This is the classic debugging problem to which tools such as rule tracers and steppers have been applied. These tend to operate at a high level, require a lot of interaction, have very low level commands and provide no indication of "how" or "why". A step up from these are automated debuggers and meta-debuggers. These are reviewed in Ducassé and Emde [1988]. An alternative low level solution is to adopt rule base simulation techniques [Coenen and Bench-Capon 1992b].

5. MAINTENANCE ACTION.

Once the nature of the change and the area within the KB affected has been identified we can determine the type of maintenance action required. In most cases a maintenance session will involve the implementation of a number of maintenance actions.

6. LOCAL IDENTIFICATION.

When the nature of the required maintenance has been established and the global area in the KB where it should be implemented identified the next step is to identify locally the elements of the KB that will be affected by the proposed change. If we consider a frame based system where we wish to add a slot we will need to consider the method by which the slot may be set and also, assuming an inheritance mechanism, the sub-frames that may inherit the slot value.

Several techniques for local identification suggest themselves depending on the representation formalism used. There area number of rule base viewing and representation techniques that allow by-eye detection of the connections between elements in a KB. Examples include Petri-Nets, Decision trees, And/Or Graphs and Rule/Frame Maps. There are also some viewing/representation techniques that are usually used for KBS validation and verification purposes but which may also be applicable here. These include evidence flow graph, decision table and incidence matrices techniques. These are all well understood, however they require a certain amount of user interaction and depend on visual inspection for the identification of the affected rules or frames. More desirable will be a mechanism which automatically lists the rules or frames that will be affected. This can be envisaged fairly clearly where (say) a frame is to be removed or altered. How this would operate in the case of (say) the introduction of a new frame is not immediately obvious.

7. IMPLEMENTATION.

Having identified the nature of the required maintenance action(s) the next step is the implementation of the action(s). This usually involves a certain amount of programming in much the same manner as involved during the original development of the system. Most interpreters, development shells and environments include editing facilities. In some cases these are fairly sophisticated, incorporating a certain amount of syntactic and error checking capability.

8. TESTING.

On completion of a maintenance session the maintenance engineer will wish to ensure two things.

Firstly that the action has been implemented correctly, i.e. the KBS operates as required by the change. This is usually carried out by running "suitable" tests cases. Traditionally suitable test cases have been defined as cases that contain some element of the changed rule or frame in the KB. However suitable test cases are not necessarily the same as those that appear to be immediately relevant. Automatic generation of such test cases to test KBSs is still the subject of much research activity.

Secondly the maintenance engineer will wish to ensure that no errors or anomalies have been introduced that may affect the operation of the rest of the KBS. This can be carried out by using similar sets of tools as those proposed by many general KBS verification and validation tool kits [Chang *et. al.* 1991, Nguyen *et. al.* 1991 and Suwa *et. al.* 1984]. There is a certain amount of overlap between the testing proposed here and KBS verification and validation in general. The significant difference is that the verification and validation carried out on completion of a KBS maintenance session impinges only on the sub-set of the elements in the KB identified, whilst general KBS verification and validation focuses on the entire KB. Existing KBS verification and validation tools would however also be applicable here. A criticism of these tools is that they focus very much on the black box aspects of the elements making up a KB. Very little consideration has been given to the white box aspects; as long as the desired output is achieved the element is assumed to be operating correctly. This is clearly not always the case. It is our contention that successful verification and validation of the identified sub-set of jeopardised elements of the KB should focus on white box techniques.

9. CONCLUSIONS.

Maintenance, and KBS maintenance in particular, is perceived as being distinctly unglamorous. In this paper we have tried to redress the balance by stressing the importance of maintenance as part of the KBS life-cycle. We suggest that the lack of general acceptance of KBS technology can be firmly attributed to the difficulties encountered when trying to maintain such systems and therefore suggest a possible maintenance methodology based on work carried out as part of the MAKE project. Each stage is discussed in detail and suitable software support tools suggested. It is hoped that the work described will provide a basis upon which further research concerning the maintenance of KBS can be founded.

REFERENCES.

1. **Aiken, M.W. and Sheng, O.R.L. (1990).** *Nexpert Object.* Expert Systems, February, p54-57.

2. **Bench-Capon, T.J.M. and Coenen, F. (1991a).** *The Make Project: Maintenance Tools For Knowledge Based Systems.* Proceedings World Congress on Expert Systems, Orlando, Pergamon Press, p1030-1036.

3. **Bench-Capon, T.J.M. and Coenen, F.P. (1991b).** *Practical Applications of KBS to Law: The Crucial Role of Maintenance.* In van Noortwijk, C., Schmidt, A.H.J., and Winkels, R.G.F. (eds). *Legal Knowledge Based Systems, Aims for Research and Development.* Koninklijke Vermande BV, Lelystadt, Netherlands, p5-17.

4. **Bench-Capon, T.J.M. and Coenen, F. (1991c).** *Exploiting Isomorphism: Development of a KBS to Support British Coal Insurance Claims.* Proceedings Third International Conference on AI and Law, Oxford, ACM Press, p62-68.

5. **Bratley, P., Fremont, J., MacKaay, E. and Poulin, D. (1991a).** *Coping With Change.* Proceedings Third International Conference on AI and Law, Oxford, ACM, p69-76.

6. **Chang, C.L., Combs, J.B. and Stachowitz, R.A. (1991).** *A Report on the Expert Systems Validation Associate (EVA).* Experts Systems with Applications, Vol 1, No 3, p219-230.

7. **Coenen, F. and Bench-Capon, T.J.M. (1992a).** *Maintenance and Maintainability in Regulation Based KBS.* ICL Technical Journal, May, p67-84.

8. **Coenen, F. and Bench-Capon, T.J.M. (1992b).** *Maintenance and Maintainability in Regulation Based KBS.* Proccedings AIENG92, Waterloo, Ontario. Computational Mechanics Publications and Elsevier Applied Science.

9. **Ducassé, M. and Emde, A. (1988).** *Opium+, a Meta-Debugger for PROLOG.* Proceedings ECAI, Munich, p272-277.

10. **Gardenfors, P. (1988).** *Knowledge in flux.* MIT Press.

11. **Hicks, R.C. (1990).** *A Composite Methodology for Low Maintenance Expert System Development.* Proceedings of the 3rd Annual Hawaii International Conference on System Science, Vol 3, Computer Society Press, p293-302.

12. **Soloway, E., Bachant, J. and Jensen, K. (1987).** *Assessing the Maintainability of XCON-in-RIME: Coping with The Problems of a Very Large Rule-Base.* Proceedings of AAAI-87, Vol 2, p824-829.

13. **Motta, E., Rajan, T., Domingue, J. and Eisenstadt, M. (1990).** *Methodological Foundations of KEATS, The Knowledge Engineer's Assistant.* In Wielinga, B., Boose, J., Gains, B., Schreiber, G. and van Sommeren, M. (eds). *Current Trends in Knowledge Acquisition.* IOS Press, p257-275.

14. **Nguyen, T.A., Perkins, W.A., Laffey, T.J. and Pecora D. (1987).** *Knowledge Base verification.* AI Magazine, summer, p67-75, 1987.

15. **Suwa, M., Scott, A.C. and Shortliffe, E.H. (1984).** *Completeness and Consistency in a Rule-Based System.* In Buchanan, B.G. and Shortliffe, E.H. (eds). *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.* Addison-Wesley, p159-170.

16. **Swanson. E.B. (1991).** *The Dimensions of Maintenance.* Proceedings of The 2nd International Conference Software Engineering, October, p492-497.

17. **Wielinga, B.J., Schreiber, A.T. and Breuker, J.A. (1991).** *KADS: A Modelling Approach to Knowledge Engineering.* University of Amsterdam.