Transactions on Information and Communications Technologies vol 1, © 1993 WIT Press, www.witpress.com, ISSN 1743-3517

Using allspeak to reverse engineer KBS specifications

R. Threadgold[a] & F. Coenen[b]

[a] *RAMJET Software, 54-60 Merthyr Road, Cardiff, South Glamorgan CF4 1DJ, UK*

[b] *Department of Computer Science, Liverpool University, Chadwick Building, P.O. Box 147, Liverpool L69 3BX, UK*

ABSTRACT

The maintenance of Knowledge Based Systems (KBS) is often hampered by inadequate or incorrect system specification documents. It results from software being perfected and updated without similar consideration being given to the original specification. This is particularly so in the manufacturing and production engineering industry where the emphasis is on ensuring that plant continues to be productive, i.e. remain in operation. However, to achieve continuous operation, it is suggested that the operating manuals for such KBS are kept up to date. This observation is exploited, to address the maintenance of inadequate specifications, through a process of reverse engineering whereby the maintenance engineer commences the maintenance task with "up to date" operation manuals and "reverse engineers", using KBS development tools, to produce a correct software specification on which further maintenance can be based. We illustrate the process of reverse engineering using ALLSPEAK (RAMJET Software's proto-typing environment). This is supported by a KBS toolkit based on concepts originally espoused in SSADM (Structured Systems Analysis and Design Methodology). A feature of ALLSPEAK is its ability to produce a proto-type system very early on the KBS life-cycle. This feature can thus be applied to the rebuilt KBS specification and the resulting emulation used to confirm the correctness of the specification.

1. INTRODUCTION

In this paper we demonstrate a technique capable of maintaining inadequate or incorrect Knowledge Based Systems (KBS) specifications. We acknowledge that many authors claim that by definition no meaningful specification can exist for KBSs because they are intended to operate in domains which defy rigorous definition. However, KBSs are problem orientated and thus it can be argued that a specification must exist. If not it can be produced in terms of the attributes of an exceptable (or correct) solution. A similar argument is put forward, with respect to KBS verification and validation, by Geissman[1] (see also Green and Keyes[2]).

Although system specifications are generally "correct" when they are first drawn up, they quickly become out of date if they are not maintained. This is especially so in the manufacturing and production engineering industry where the emphasis is on the continuous operation of plant. However, the needs of industrial competitiveness dictate that systems to control plant must be regularly updated and extended. The result is that the software is often altered while the original specification remains unchanged. The specification will thus start to accumulate errors. In the short-term this may not be especially detrimental, however, in the long-term the results can be disastrous. An example is where an industry wishes to integrate a number of KBSs into a "cooperative" environment in line with the current trend for Cooperating-KBSs, so called CKBSs (see Deen[3]). It is therefore becoming much more essential that software specifications are kept up to date. Most of industry would agree with this statement, however, in practice specifications remain suspect.

If the task of updating a KBS specification in either the manufacturing or the production engineering industry is neglected, it may be possible to retrieve the situation. This stems from the need to ensure that plant remains in operation. It has the consequence that system operating manuals will be up to date, although it is appreciated that in many cases the "updating" will have been implemented using hand written notes in margins.

The observation that operating manuals are correct is the starting point for the approach to maintaining KBS specifications advocated here. The approach centers on the concept of "reverse engineering". This commences with a correct operating manual and works backward to produce a valid specification for the KBS. The operating manual is viewed as a knowledge source which can be analysed, processed and represented in some form, in accordance with the traditional KBS development cycle. Usually the desired representation is based on a concept of logic or objects, or a combination of the two. However, the representation can equally well be couched in terms of a KBS specification.

There are many KBS development techniques, tool-kits, shells, environments and methodologies available. Well known examples include the KADS knowledge analysis methodology (Wielinga[4]), the KEATS development environment (Motta et al.[5]) and the NEXPERT-OBJECT object-oriented KBS development toolkit (Aiken and Sheng[6]). However, to the best knowledge of the authors, none of the currently available KBS development aids support the concept of "reverse engineering". ALLSPEAK, although not specifically a KBS development environment, does, expressly provides such support and has the advantage, in its "forward engineering" mode, that it can produce an operational proto-type KBS direct from the specification very early in the development life-cycle. The significance is that the rebuilt KBS specification which has been produced as a result of the reverse engineering process, can be "tested".

Proto-type development in ALLSPEAK comprises the following stages:

1. LANGUAGE: (After some initial system analysis) a design specification is expressed in natural language.
2. STRUCTURE: The supporting ALLSPEAK KBS toolkit then provides facilities to allow the specification to be represented in electronic form by mapping the specification onto annotated bipartite directed graphs (see Threadgold[7]).
3. REPOSITORY: The incapsulated graphs are stored in a mesh of Cartesian Products (CP's) (Threadgold[8]).
4. APPLICATION: The specification stored in the ALLSPEAK repository is automatically translated into an Object Oriented Program (OOP) KBS. This is achieved through ALLSPEAK's Application Generator. This is essentially an expert system founded on stage zero of SSADM (Structured Systems Analysis and Design Methodology) (NCC Blackwell Ltd.[9]) (see also Ashworth and Goodland[10] and Cutts[11]).
5. RUN-TIME: The resulting executable proto-type for the application in question can the be run and inspected.

The ALLSPEAK environment was originally developed to support early proto-type production in SSADM based projects. The idea was that an end user specification could be prepared and converted automatically into an executable model. This would then demonstrate to the user how the proposed system would "look" and "feel". That the resulting proto-type is an object oriented KBS is significant.

More recently it has been appreciated that the environment can be used for quite a different purpose. This is to assist in the rebuilding of system specifications, which have been lost or have got so far out of date that they are useless, using a reverse engineering process. Initial applications investigated have been centered on conventional software maintenance. However, since the end result is an Object Oriented KBS, it is suggested that the technique is also applicable to KBSs where the specification is no longer valid.

2. OVERVIEW OF ALLSPEAK

The development life cycle used is based on Figure 6 in the "concepts" Chapter of the SSADM manual (NCC Blackwell Ltd.[9]). The Figure is a general one which is capable of covering a spiral life cycle (see Peltu[12]). An adapted version (to suit the needs of the feasibility loop of the spiral) is presented in Figure 1 of this document. In Figure 2 the concept is developed further into a practical scheme. The Figure has been constructed to emphasise the strong ties between the operating manuals at one end of the development cycle and system requirements at the other. Returning to Figure 1 a box called "earlier feasibility work" is included which is not in the SSADM diagram. Figure 3 amplifies this "earlier feasibility work" and details the crucial link between data found in the operating manuals and that developed during a feasibility study. It is this which makes reverse engineering a practical proposition.

The products of the feasibility study (see NCC Blackwell Ltd.[9]) are a Context diagram and a Current Physical Data Flow Model (the latter might take the form of a Document Flow Diagram). The context diagram shows the interactions between the area being studied and other areas of the business. The Data Flow Diagrams (DFDs) associated with the above show dataflows which reflect current operating methods.

Data flow modelling is used to develop the DFDs. It identifies potential independent processes in the new system and the dataflow paths between them. The context diagram is enhanced to show the connectivity. This includes connectivity of the processes with both users and data stores.

The input dataflows to the context diagram processes are determined. They trigger associated functions. These are defined on level 1 DFDs. Detailing consists of the addition of output dataflows. These are to other computer processes, to users and to data stores.

ALLSPEAK has an aspect of integrated CASE (Computer Aided Software Engineering) within the automatic code generator element. This

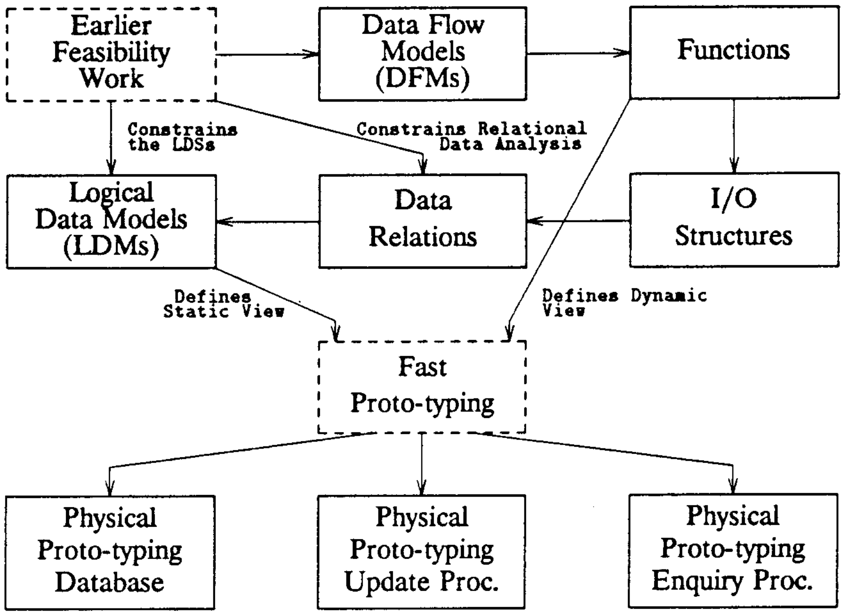
Earlier Feasibility Work
Data Flow Models (DFMs)
Functions
Constrains the LDSs
Constrains Relational Data Analysis
Logical Data Models (LDMs)
Data Relations
I/O Structures
Defines Static View
Defines Dynamic View
Fast Proto-typing
Physical Proto-typing Database
Physical Proto-typing Update Proc.
Physical Proto-typing Enquiry Proc.

Figure 1. Adaptation of 'Concepts'
NOTE: The dashed line boxes have been added to the original SSADM v4 diagram

implements the dataflows in the proto-type by calling on the services of *system builder* modules. These provide Object Oriented methods for the proto-type KBS. A comprehensive list is given in the "Tool constraints on End User language" section of an earlier paper (see Threadgold[13]). This includes descriptions indicating the operational aspects of each module. The methods support all proto-type scheme inter-process messages. Take for example the *Report* module. A requirement for a message between a scheme process and an operator (a second process) appears in the control stream as a call on *Report*. Its parameterisation specifies the type of the report and the structure of information which needs to be reported. The report module code is a set of methods one of which interprets the calling message, creates the consequential report and stores it. A successor *Display* module displays the report to the operator.

Most module definitions include one or more parameters. These need to be dimensioned whenever a call is made. The options which are available are constrained. This is because the data content of the messages must come from or go to a data store which conforms to Logical Data Structure (LDS) requirements.

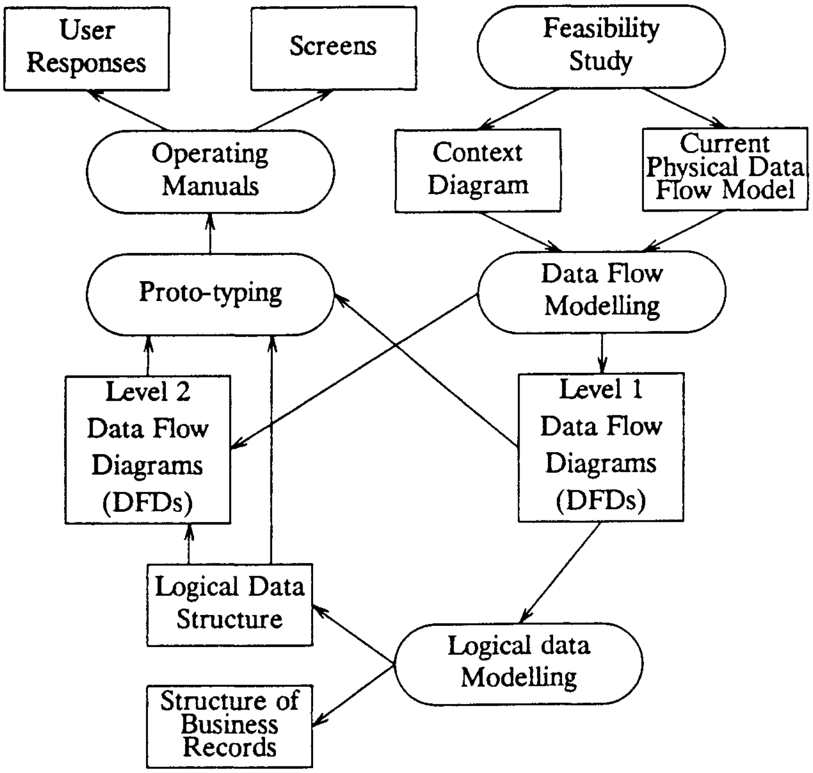

Figure 2. Software Development Cycle

These are specified when the structure of records for the new scheme is determined. This is implemented in the logical data modelling stage of development. Message parameters must match entities and attributes associated with the LDS. An LDS is specified and recorded in a similar way to the level 1 DFDs. The record is used to constrain message parameterisation. Time is saved because it is nor necessary for the designer to refer to a graphical representation of the LDS when a message has its parameters specified. In addition any parameter which is chosen will be consistent with earlier and later parameters in the list. The results are set up in a series of level 2 DFDs.

SSADM products are designed with stepwise refinement in mind (see Wirth[14]). The aim here is the production of a proto-type during the feasibility study. This means that the level 1 DFDs, LDSs and level 2 DFDs must be stepwise refined to the point where inter-process message calls are specified. This is a realistic feasibility stage job.

Transactions on Information and Communications Technologies vol 1, © 1993 WIT Press, www.witpress.com, ISSN 1743-3517

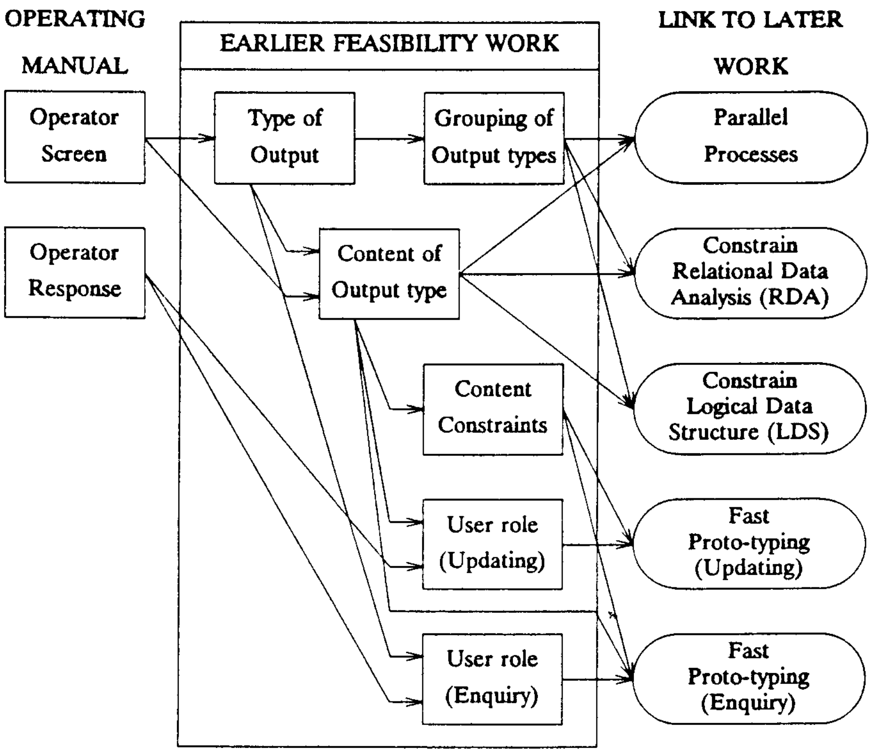

Figure 3. Expansion of 'Earlier Feasibility'
NOTE: This shows the crucial 'reverse engineering' links from operating manual to the first stage of feasibility work.

Once calls on the system builder modules have been parameterised, the specification can be converted to code automatically. This is carried out in two stages.

1. The level 1 DFDs and the LDSs of the new scheme are converted into control data. This is compiled into a parallel KBS run-time database which defines the following system objects:
   (a) Database names
   (b) Start passwords
   (c) Commands under passwords
   (d) Command batch file names
   (e) Empty files for the names of database parts
   (f) Names of database sets
   (g) Sizes and types of set elements

(h) The limit size of a sector

(i) Overflow inheritances

(j) Database configuration

These cover the various rights of access which are implicit in the requirements. They control proto-type scheme operation. An example is creating a new database sector. This uses the configuration data to define an empty record for the new sector. It acquires the directory name where the sector is to be stored. It records the information as a sectored database part.

2. The level 2 DFDs are converted into a series of proto-type application programs implemented as MS-DOS batch files. The programs comprise lists of calls on builder modules.

The resultant code components implement a working KBS proto-type of which a CP mesh Knowledge-Base (KB) (Threadgold[8]) is an intrinsic part. Its structure is defined in the control files. The live system objects and their features are stored in the mesh. At run-time builder modules are called into use from batch file application programs. Each call involves an object oriented message. It invokes an appropriate object oriented method (in the form of a control thread through a builder module). The selected method causes the transmission of output messages.

When a proto-type system is assessed, one of the most important attributes from the users point of view is the form of the operational screens. Another is the arrangement of the user responses. Other things such as response time and ability to corrupt the system need to be dealt with but initially, users tend to concentrate on usability issues.

Once the information content of the screens and the associated user responses have been agreed, operating manuals can be prepared (this cannot be done until the usability issue is resolved). The production of the manuals is the last stage in the proto-type development cycle.

Screen contents divide into three classifications.

1. System information. This is defined by operators during system operation. The system can not affect it and the content is not a usability issue.
2. Command options. The set of phrases on a screen which describe command options available at a particular point in the operational cycle. as identified in the specification. They are phrases which the

end user defines. The words which appear on the screen are those in the specification. Note that the use of ALLSPEAK support tools intrudes into this area. They have a small number of command options of their own. They are reasonably obvious keyboard descriptors, for example `Esc` for strike the "Esc" key, `Amend` for strike the "A" or "a" key, etc.

3. Type definitions. These are the set of words which appear on the screen whose purpose is to define the "type" of part of the information content. These show users pieces of screen information which are of interest. Thus a report screen dealing with several columns of information has headers. These define column purposes. Another example is a data acquisition screen. This identifies the type of each of the pieces of information which is to be acquired and thus helps concentrate the operators mind when a complex set of information is entered. The impact of "logical data modelling" and "logical system specification" has been described in an earlier paper (see Threadgold[13]). In brief this demonstrates that system data types are also names which are specified by the end user.

Thus it is clear that user involvement in the earliest stages is very beneficial. The words which appear on screens are the users own when the ALLSPEAK design discipline is followed. It means usability issues are able to be resolved easily when the proto-type is generated.

3. THE CONCEPT OF REVERSE ENGINEERING

NOTE: In this section the words: *class*, *object*, *message* and *method* have conventional object oriented programming meanings.

We have noted that when an SSADM level 1 DFD is specified, system waitpoint objects in the new system are defined and dataflow messages in and out of the associated processes are identified. An input message implies the need for a method which transforms it into one or more output messages. In the first instance methods are one to one with control threads. At the top these exit from the system waitpoints and connect through to successor waitpoints. When control flows through a thread the associated method is invoked and the consequential output dataflows occur. Note that lower level detailing could multiply one of these topmost methods into several. Consider a module call within the thread. If this has more than one inner control path, the methods available expand to match.

We bear in mind that the directed graph part of a DFD represents the control paths through the associated SSADM function.

We have noted also that when the waitpoint objects are detailed into SSADM level 2 DFDs, message specification is completed. This signals the end of programming, i.e. a system proto-type can be generated.

A large proportion of any system operating manual is devoted to user screens and consequential operator responses. Each alternative screen corresponds to a waitpoint. Each permitted response defines the need for an associated method.

Tying the above together gives us the start of a reverse engineering discipline. This considers the user screens in turn. Each determines a corresponding waitpoint in the rebuilt specification. Each user response on the screen determines a topmost method (as a control thread in the SSADM level 1 DFD part of the specification).

The rebuilding of a specification from operating screens involves the synthesis of system data aspects as well as of control aspects. It means that all inter process messages have to be determined and added to the graph. Some are operator to process messages. These release control from a waitpoint and are associated with the user responses mentioned above. The output messages which flow as a consequence, are determined next. Each control thread which exits a waitpoint is considered in turn. The operating manual will normally state the order in which a sequence of screens appears. If it doesn't, the old scheme must be run to determine the order. Thus when a response occurs, the message which causes the next screen to appear can be determined. It can now be associated with the control thread under consideration.

At this stage we remember that we are synthesising a specification which will be converted automatically into an emulation of the original scheme. It means we can refer to the properties of the system builder modules which are available and determine which one will give the desired consequences.

An example is an original scheme screen used to pass information in the KB to the operator. In the rebuilt specification, the ALLSPEAK *Report* module will need to be called to emulate report writing. It needs to be dimensioned with parameters if the correct report is to be created. We know the names of the report fields from a study of the original screen in the operating manual. We use the information to synthesise LDSs for inclusion in the rebuilt specification. This controls the parameterisation of the *Report* module as noted earlier.

Other original scheme screens will be associated with data acquisition.

These contain further information which enhances the synthesised LDS.

As far as operator to process dataflows are concerned we have now specified all the messages which are needed for system emulation. These may be an incomplete subset of system messages. This is because the original scheme may have inner parallel processes which management consider necessary to re-specify and emulate.

Should this kind of extra work be necessary, each output dataflow requirement will need to be looked at. The objective is to determine if operator action has a knock on effect which stimulates an inner independent process. If an inner process is implied there will be an inconsistency on the emerging LDS. This takes the form of an incompleteness where output data isn't matched by that in an incoming dataflow. In the other direction input information will not be matched by earlier outputs. All incompleteness data has to be gathered together and used to deduce the LDS of the inner process(es). Once this has been implemented extra messages to and from the process(es) can be defined and agreed. Parameters specifying the message content will have to be deduced. Some will emerge from the above analysis, some may need deductive work on system code listings. In any case KBS code listings are helpful when it comes to determining inner processes of this kind. They are also useful during the determination of the automatic inter-process dataflows.

4. REVERSE ENGINEERING IN PRACTICE

Scenarios in which specifications are inadequate or lost are common place in the real world. Our theme is that the specification can always be rebuilt from the operating manuals. There are several scenarios where the technique is useful. This section works up to a KBS scenario via a consideration of reverse engineering for conventional Data Processing (DP) systems.

4.1 SCENARIO 1

The loss of a DP specification will often be the result of "good intentions". One problem is the theory of self documentation associated with computer languages such as COBOL (see Norman[15]). This theory holds that a code listing is a viable substitute for a specification. It means that the listings are the only specification material to hand when maintenance is carried out. The original programmer is rarely available hence someone else has to pick up the job. He/she can not proceed without understanding the environment so must first recreate part of the original specification.

The maintainer then discovers that "self documentation" omits vital information. This is because the original developers carried out an analysis to determine program requirements which is no longer in existence. Another problem which may be experienced at this time is that of "spaghetti" code. If the system is in commercial use, it will usually incorporate a substantial amount of unstructured code.

As noted earlier, the specification can be rebuilt from the operating manuals. These must be up to date, include pictures of the screen sequences and describe operator responses.

If such work is undertaken and if the specification is rebuilt with the help of ALLSPEAK proto-typing, we are in the realm of Expert System knowledge elicitation. The knowledge is that which is present in the operator manuals. Elicitation records it into the specification KBS. Standard facilities are used to extract specification components.

An advantage of the form of reverse engineering which is advocated is that the original system can be emulated. This is because the recorded knowledge can be used to produce the equivalent of a design situation proto-type. This is run to see if its properties match those of the original scheme. Being able to perform emulations is useful because it means experiments can be run without affecting live system operation. It gives the user the ability to think about changes and their consequences without upsetting the working system.

4.2 SCENARIO 2

Alternatively a specification may be lost through reasons of "efficiency". At some stage a system outgrows its resources. When this happens palliative solutions may be adopted. One technique adopted is the recoding of part of the program into assembler. Since it is efficiency which is required, the most critical parts of a program are affected. The reason is that performance can be improved if a high level program is substituted by an assembler program. This kind of maintenance is most unlikely to be accompanied by adequate configuration management. If not, the thread of specification continuity so necessary for effective maintenance is broken. The next time a change is needed there is a real problem.

4.3 SCENARIO 3

The lack of adequate configuration management when a commercial KBS system is altered leads to a different kind of maintenance problem. Basically this is due to advances in the knowledge on which the simulation is based. As a result parts of the Expert System become redundant, need improvement or need replacing. The deterioration is much more subtle than that in a poorly maintained DP system. The consequence is that safety critical KBSs must have their specifications maintained to a very high correctness level. Consider a proto-type system, currently under assessed by the medical world, to advise whether an emergency patient will benefit from a proposed course of treatment. If the judgement is that the patient can not and that scarce resources can be saved, the patient may be denied the treatment. If the system is not maintained adequately, its criteria may become severely flawed (consider a breakthrough which makes a previously expensive drug cheap but which has not been added to the rulebase).

4.4 SCENARIO 4

When a DP system specification is rebuilt, specification correctness may not need to be to the same high level at all specification layers. Penetration will usually not be much below system waitpoint level. If such subtle deteriorations occur in a KBS specification, maintenance will penetrate much deeper. An illustration of maintenance to an inadequate depth can be found in the experience of the recent Gulf War where Patriot missiles failed to shoot down Scud missile targets because maintenance activities upgrading the targets from plane targets to missile targets omitted to consider a low level program (see Hunter[16]). The consequence was a miss of 500 metres and ground fatalities.

ALLSPEAK has been designed with coverage of the lower specification layers in mind. Its genesis was in the maintenance of large real time multiprocessing communication systems. The initial work was carried out at RSRE (Royal Signal and Radar Establishment) in the 1970's. It concerned rules for the reduction of directed graphs used to depict software. It has not been published. The rules are formal (Threadgold[7]). They centre around finding the convergence points in a software control flow structure. In the ALLSPEAK case, system waitpoints and startpoints are the natural convergence points which are exploited. The parts of the scheme described so far are based on these. SSADM function definitions are thought of as control flow fragments bounded by waitpoints. ALLSPEAK exploits a rationale for overviewing functions (formally). This is not discussed here but can produce the "across the system signaling" that sits above the functions.

Beneath the functions and implied within them are calls on the top level communication handlers. These always have one example of the next level of natural convergence point within them. This is the return point which is reached when the handler has done its job. The usual case is when ALLSPEAK *system builder* modules are used. These modules are all communication handlers. They allow the user to generate an adequate proto-type from a reverse engineered specification in most situations. In a safety critical situation involving penetration beneath the waitpoint level, one or more extra handlers (and possibly even deeper level software modules) may need to be re-specified.

This can be done. An annotated directed graph representation of a lower level software module can be specified and recorded into a KBS. It can then be converted into proto-type code and used in emulations. Since we are not dealing with operator screens and responses at this lower level, the rules of construction of the graphs are not quite the same as before. Thus nodes are used to specify, for example, program branches and action points on the control side and meta facts and their inter-relationships on the data side. Arcs are used to specify, for example, conditions controlling branching and events consequent on a flow of control through an arc.

Exploitation will vary slightly from case to case. A KBS scheme based on the use of production rules is considered here. An initial study shows that the rules implicit in the use of ALLSPEAK OOP methods do not cover all the needs of the system which is being reverse engineered. A decision is arrived at defining the missing rules to be included in the rebuilt specification. These are then specified into lower level annotated directed graphs. The production rule conditions and actions form the annotations. Each graph is a pictorial representation of a fragment of the full rulebase.

4.5 SCENARIO 5

The ultimate maintenance scenario occurs when two (or more) self standing schemes need to be integrated. They are reverse engineered and emulated separately. The new features demanded by the integration are specified using the forward engineering characteristics of ALLSPEAK. A proto-type of the integrated system is built and assessed. The resulting specification is used as the basis for procuring the new scheme. Today the need is not only to be able to handle KBS integration. More and more, hybrids involving mixtures of DP and KBS schemes are having to be integrated. Witness the current interest in co-operating KBSs (see Deen[3]).

5.CONCLUSION

The paper introduces the idea that a software system specification which is out of date or lost can be rebuilt from other system documentation. The rationale is that an operating manual will exist, will be up to date and can be reverse engineered to create a new specification. Using the facilities provided by ALLSPEAK a working prototype KBS can be created and inspected. The supporting toolkit also allows the user to amend and retest the specification as required.

Conventional DP system specifications have had to be reverse engineered in the past. This is primarily because they are often neglected. When maintenance is needed they are found to be deficient. The need to maintain commercial KBSs is a current problem. Often, the impact of poor maintenance on a KBS is very much worse than on a DP system. This is because a KBS rulebase deteriorates in a much more subtle way. Very often the owners are unaware that they even have a problem. This is particularly true of safety critical systems. There have been several high profile reports of the fatal effects that result when such systems are poorly maintained.

The ALLSPEAK development environment for proto-type systems can be used when a system specification needs to be rebuilt. The advantages are:

- Clear and simple specification documentation
- Formally linked specification parts
- The ability to generate a working emulation
- The ability to experiment realistically in isolation (i.e. the operational system is undisturbed)
- Easy to use tool set

ACKNOWLEDGEMENTS

Figures 1 to 3 have been extracted from tool handbooks. Some parts of the section headed 'Overview of ALLSPEAK' back up the figures. These too have been extracted from a handbook. The copyright of these is with RAMJET Software. They may be freely copied providing a suitable acknowledgement is made.

REFERENCES

1. Geissman, J. (1988). Verification and Validation for Expert Systems: A Practical Methodology. Proceedings, 4th Annual Artificial Intelligence and Advanced Computer Technology Conference, p344-51.
2. Green, C.J. and Keyes, M.M. (1987). Verification and Validation of Expert Systems. WESTEX 87, Proceedings of the Western

Conference on Expert Systems, p38-43.

3. Deen, S.M. (1992). Cooperating Knowledge Based Systems. Presentation at the inaugural meeting of the CKBS-SIG, Queen Mary College, London, 3 June 1992.
4. Wielinga, B.J., Schreiber, A.T. and Breuker, J.A. (1992). KADS: A Modelling Approach to Knowledge Engineering. Knowledge Acquisition (Special Issue: The KADS approach to knowledge engineering), Vol 4, No 1, March, p5-54.
5. 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.
6. Aiken, M.W. and Sheng, O.L. (1990). Nexpert Object. Expert Systems, February, p54-57.
7. Threadgold, R. (1990). The Specification of Real Time Software, Proceedings of Expert Systems 90, the Tenth Annual Technical Conference of the British Computer Society Specialist Group on Expert Systems, London, September 1990, Cambridge University Press 1990.
8. Threadgold, R. (1991). Controlling Database Integrity. Applications of Artificial Intelligence in Engineering VI, AIENG'91, Oxford, July 1991, Computational Mechanics Publications, Elsevier Applied Science, 1991.
9. NCC Blackwell Ltd (1990) SSADM Version 4 Reference Manual. NCC Blackwell Ltd. 108 Cowley Road, Oxford, OX4 1JF. 1990.
10. Ashworth, C. and Goodland, M. (1990). SSADM: A Practical Approach. McGraw-Hill.
11. Cutts, G. (1991). Structured System Analysis and Design Methodology. Blackwell Scientific, 2nd Edition.
12. Peltu, M. (1993). Spiral of Success. Computing, 28 January, p24.
13. Threadgold, R. (1992). Adapting a KBS in the face of change, Applications of Artificial Intelligence in Engineering VII, AIENG'92, Waterloo - Toronto, July 1992, Computational Mechanics Publications, Elsevier Applied Science, 1992.
14. Wirth, N. (1971). Program Development by Stepwise Refinement. Communications of the ACM, Vol 14, No 4, pp221-227.
15. Norman, R.W. (1991). Essential COBOL: A First Course in Structured COBOL (ANSI 1985). McGraw-Hill.
16. Hunter, S. (1992). Battling on with Veteran Computers. New Scientist, 14 December, Issue no 1847.