The Meta-Knowledge Level: A Methodology for Validation

Robert T. Plant

Department of Computer Information Systems
University of Miami
Coral Gables
Florida 33124

ABSTRACT

The aim of this paper is to show that when a development life cycle of representation refinement is utilized, that follows the principles of Newell's Knowledge-Level, then the system will become self validating. This is illustrated through a rigorous development methodology that utilizes formal techniques in the specification of the domain knowledge, the cognitive aspects and the representation. The paper introduces the concept of the Meta Knowledge-Level, a variant of Newell's Knowledge-Level that facilitates the construction of a meta knowledge model. This provides the knowledge engineer with a dynamic perspective of the system which can be used in conjunction with the static aspects found in the intermediate representation, an implementation independent representation that is created through the use of a knowledge filter.

1. Introduction

The focus of research and development in the area of methodologies for knowledge-based systems has primarily been upon the functionality, formal aspects and refinement of this form of software system [Plant.93] [Miller.90]. This has been subsequently reflected in the research area of validation and verification for knowledge-based systems, where the primary focus has been upon the static structures of the systems. In this paper we will attempt to move away from this static perspective towards an alternative philosophy of system design, one that relies not only upon the use of formality and refinement but one that utilizes a variant of Newell's Knowledge-Level [Newell.82] to influence the design, this being the Meta-Knowledge Level. The paper will present an overview of this approach and indicate how this meta knowledge can be obtained and used to assist in the design and validation processes.

2. Background

The creation of knowledge-based systems and their subsequent verification can be considered from two aspects: the *static* and the *dynamic* aspects of the system. The development methods are focused upon the use of increasingly formal aspects of system development, such that the proof obligation for systems can be ultimately determined and met. These development mechanisms are at or approaching the TRILLIUM$_k$ Level 2 capability with certain aspects moving towards Level 3 capability [Preece.93]. This move towards formal functionality in system design has, as Bellman notes, moved the validation research to move in a parallel direction:

"The bulk of rule-based methods concern the systematic analysis of the static structure and dynamic behavior of a rule-base, using analysis based on certain correctness criteria" [Bellman.91]

Examples of such static structure validation mechanisms are those utilized in the EVA tool set [Chang.90]:

- Structure Checker
- Logic Checker
- Omission Checker
- Control Checker
- Rule Refiner

The primary intent of such checkers is to identify inconsistencies and incompleteness states, which Morell has defined in the following way:

> "A system is *inconsistent* if it asserts something that is not true of the modeled domain" [Morell.89]
>
> "A system is incomplete if it lacks deductive capability" [Morell.89]

However, the question thus arises as to what are we to verify our knowledge-based systems against, given the difficulty of obtaining a complete/total specification for the domain. One approach around this is to create a partial or composite specification of the desired system functions that are known. Morell notes:

> "In general there are many desirable properties that need to be specified that can be used as incomplete or semi-specifications. This knowledge about the knowledge base is called *meta-knowledge* and is a vital necessity for verification. [Morell.89]

and this is reflected by Bellman who suggests that the static and dynamic analysis of systems is:

> "Based upon ad-hoc models of rule-based inference, or more generally, on ad-hoc *meta-knowledge* about the rule-base or the rule-inference process" [Bellman.91]

An example set of meta-knowledge categories that a knowledge engineer may utilize are of the form:

- Accuracy
- Applicability
- Assessment
- Consistency
- Completeness
- Disambiguation
- Justification
- Life Span
- Purpose
- Source
- Reliability [Morrell.89]

These categories of meta knowledge can then be utilized in the verification and validation of all aspects of system development - specification, elicitation, representation, implementation, and maintenance. We will later in the paper consider examples of these meta knowledge categories in different aspects of development verification.

A further aspect to the utilization of meta knowledge is the knowledge engineers ability to obtain this meta knowledge, as Bellman notes:

> "This knowledge is even harder to elicit from a domain expert than ordinary knowledge... it is often omitted from original rule base descriptions and provided later by independent evaluators" [Bellman.91]

Thus, we can see that the meta knowledge is often collected (if at all) from many ad-hoc sources in an unorganized manner and subsequently needs validation and verification techniques of its own, to ensure correctness. The remainder of this paper will consider how meta knowledge could be elicited and organized into explicit models, a need originally suggested by Bellman and Walter at the AAAI'88workshop on validation and verification [Bellman.88].

3. A Meta-Knowledge Level KBS Development Methodology

3.1 Introduction

Alan Newell showed the relationship between the notion of levels in terms of Computer Science. He described a level to consist of:

> "A *medium* that is to be processed, *components* that provide primitive processing, *laws of composition* that permit components to be assembled into *systems*, and *laws of behavior* that determine how the system behavior depends on the component behavior and the structure of the system." [Newell.82]

A level is defined in two ways: i) "Autonomously without reference to any other level" [Newell.82]
& ii) "Each level can be reduced to the level below. Each aspect of a level - Medium, Components, Laws of Composition, and Behavior, can be defined in terms of systems at the next level. [Newell.82]

This thus provides us with a framework through which we can show how a series of specifications can fit together in a sequence that then act as a series of Knowledge-Levels.

3.2. The Specification of Knowledge-based Systems

The natural point from which to develop any software system is the creation of a specification. The specification should ideally detail every aspect of the system in unambiguous terms that all interested parties can consider. The creation of such a specification for knowledge-based systems is however a far from easy task for any but the most trivial of systems. In light of this problem, knowledge engineers have often been forced to proceed with only a minimal specification or no specification at all. This is a less than ideal situation and a source from which many subsequent developmental problems emanate. In order to overcome the problem of weak specifications in knowledge-based system development we advocate the use of two techniques: *Prototyping & Composite-Specifications* through formal methods.

The first of these techniques: Prototyping, is utilized to achieve the creation of a base-line document: the *Initial Specification*. Following Miller [Miller,90] this phase utilizes prototyping to create an initial specification and this phase does not end until all parties {customer, developer, user} agree that they finally understand what the system is intended to do, and in particular how it is supposed to do it, what Miller terms "The Operational Concept" [Miller,90].

The prototype process is primarily intended to establish the boundaries of the solution space. It is very important that the prototyping is used only to this end, as it is extremely detrimental to consider the more complex development issues at this stage e.g., representation, interface etc. As these decisions would be made on incomplete knowledge of the domain and environment.

Embedded within the initial specification development process is an aspect of *Cognitive Engineering* known as *Cognitive Task Analysis* [Roth,89], where: "Cognitive Task Analysis is used to derive a description of the cognitive demands imposed by a task and the sources of good and poor task performance" [Woods,87].

The aim of cognitive task analysis can then be seen as an attempt by the knowledge engineer to:

> "Definewhat makes the domain problem hard, what errors domain practitioners typically make and how an intelligent machine can be used to reduce or mitigate those errors or performance bottlenecks"[Roth,89]

The use of cognitive engineering techniques, is not however limited to the creation of the initial specification. Wielinga, Breuker and others, have created a development methodology KADS [Breuker,87] [Hesketh,90] that also attempts to model expertise, such that it can be utilized in a knowledge-based software development project.

We shall utilize other cognitive engineering practices later in the methodology to assist in the assessment of validation and verification, quality assurance, in the selection of a representation as well as developing the system interfaces.

The creation of an initial specification provides the knowledge engineer with the first specification in the creation of the composite-specification of the system. A composite-specification being a set of specifications, each of which focuses upon an aspect of the development process: domain specification, representation specification etc. The composite of which enables an approximation of a total specification for the system to be made. Figure I illustrates the six areas where specifications can be derived in a knowledge-based system, to varying degrees of formality.

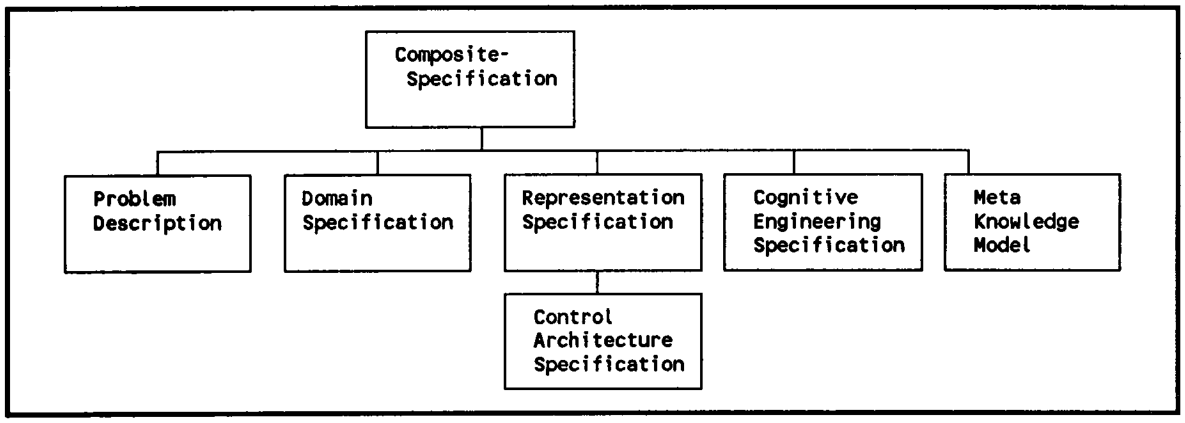
Composite-Specification
Problem Description
Domain Specification
Representation Specification
Cognitive Engineering Specification
Meta Knowledge Model
Control Architecture Specification

Figure I: A Composite-Specification

From Figure I we can see that there are two distinct types of specification present: The *dynamic* specifications and the *static* specifications. Dynamic specifications refer to aspects of the system that are under constant change or for which the interaction of the components are undetermined due to their combinatorial complexity. Static specifications refer to those aspects that do not change, but rather remain consistent over a period of time. Thus, there are four static specifications:

The Specification of the Domain Knowledge

We can easily model this aspect as the knowledge elicited from the domain expert(s) and knowledge source(s) is finite and that though the use of transformational processes this can be specified formally in a language such as "Z"[Spivey,90]. From a specification in a language such as this, we obtain several advantages. Firstly, the Z notation in which it is written is clear, concise, unambiguous and allows for both a technical and non technical readership. Secondly, the use a formal notation has significant maintenance benefits, such as allowing knowledge engineers to keep a correct document of the domain information in an implementation independent form, allowing the implementation language to vary if necessary.

The Specification of the Representation

The aim of this specification is to allow the knowledge engineer an opportunity to consider and identify those aspects of the knowledge representation language to be used and specify them in a formal manner. The selection of a representation is a difficult consideration, that we shall discuss further in the next section, however once a representational form has been selected then it is imperative that this be specified fully in terms of its denotational semantics and its syntax. For without these, it is extremely difficult to reason about a domain description/representation with any certainty. Included in this specification is the *Specification of the Control Architecture* to be used in the system.

Specification of the Cognitive Engineering Aspects

The cognitive engineering aspects of the system definition are those that involve:

- Specification of the man-machine interface
- Cognitive Task Analysis
- Knowledge-Encoding
- Competence modeling
- Performance modeling

The man-machine interface can be subjected to formal specification techniques, as demonstrated by Sufrin et al [Sufrin,90], who use the Z notation to specify an interface, and Jacob who formally specifies a man-machine interface [Jacob,83]. The Z notation is again a superior form of specification to the pseudo-code, or natural language descriptions that are usually used.

As mentioned, Cognitive Engineering has an impact upon many aspects of system development, from the initial specification, through acquisition, elicitation, quality assurance to validation and verification. We shall consider each of these aspects later.

In addition to the static specifications, there are those aspects that are dynamic in nature.

Specification of the Problem Description.

A principal aspect of the systems dynamic aspect is the specification of the problem description. The nature of the techniques for formally specifying systems are however limited to static aspects of a system and do not facilitate full specifications of the dynamic aspects and thus we are unable to fully define the system in its entirety hence this necessitated the utilization of prototyping in the creation of the initial specification in addition to the utilization of the composite-specification technique and the design tenets identified earlier.

Meta Knowledge Model (Specification)

In the development of a knowledge-based system the knowledge engineer is obligated to not only elicit the static domain specific knowledge, but to incur the overhead of eliciting the dynamic meta knowledge associated with the domain knowledge. This knowledge is inclined to change over time and thus there is a need to utilize a separate specification of this knowledge, this is the role of the meta knowledge model. This model should utilize as strong a degree of formality as is possible, again the use of a notation such as Z or VDM is advocated [Jones.80] as this will enable the relationships between the static domain knowledge and the meta knowledge be identified and assist in showing their correctness, completeness and consistency.

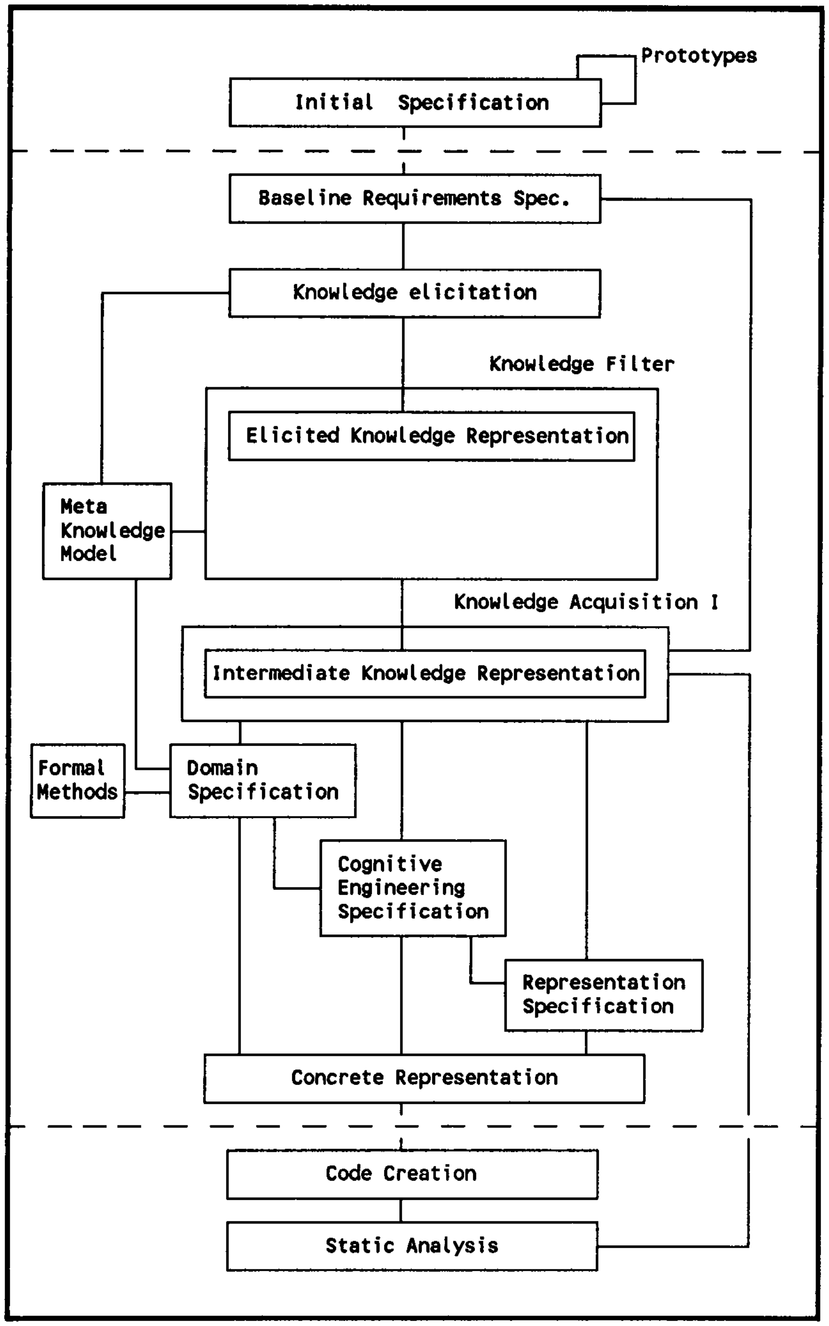

Figure II: Design of Knowledge-Based Component

Thus, we have identified the need for specifications in the creation of knowledge-based systems. We now discuss how these specifications can be brought together through the use of a rigorous development methodology and how these specifications can be equated to Knowledge-Levels. The methodology as a whole can be introduced by considering Figure II above. These stages will now be examined in greater detail.

As we have already seen, the initial specification is a document that can act as a baseline for the remainder of the systems development. Each of the resultant phases can be compared to the objectives and system specifications laid down in this document.

3.3 The Knowledge Elicitation Process

The unique nature of knowledge-based systems is that they utilize domain specific information that is *"expert"* in nature. This has several implications. The information may in itself be unique, scarce or uncommon, however it is the way that the expert employees that information that makes the information valuable. Thus, one of the most important tasks befalling the knowledge engineer is to ensure that he elicits as much structural, control and relational knowledge from the expert source as possible. This thus forms the basis of the knowledge elicitation task and the knowledge-based system development process itself. As later in the process the knowledge engineer will have to consider specifying the static domain knowledge {facts, rules, heuristics etc.,} and select a representation in which to manipulate this knowledge, which entails consideration of such factors as structural, control and hierarchical knowledge types. Thus the knowledge engineers task in knowledge elicitation can be seen as falling into two categories:-

- Elicitation of static knowledge
- Elicitation of dynamic knowledge

The knowledge engineer has several different approaches to the knowledge elicitation process, [Roth,89] [Welbank,83], for example:

- Verbal transfer of knowledge e.g., Interviewing - structured, focused and unstructured.
- Reporting techniques: e.g., On-line, Off-line and Hybrid.
- Psychological techniques: e.g., Repertory grid, critical incident, Inference structure, Goal decomposition & Distinguishing evidence.
- Knowledge engineer investigates literature.

The choice of elicitation technique will depend heavily upon the domain under consideration, the type of knowledge to be extracted and the point the elicitation has reached. For example, the elicitation may commence with the knowledge engineer performing a series of unstructured interviews to extract high level conceptual knowledge. This may then be followed by structured interviews where the relationship of the domain, its structure and more detailed information are obtained. This may then be followed by a series of focused interviews to fill in the low level information of a fine grain size. Several frameworks for the analysis of these techniques have been proposed [Dhalival,90] [Burton,87], including *Cognitive Mapping* and *Knowledge Encoding*, two aspects of Woods's cognitive engineering paradigm [Woods,88] [Roth,89].

The resultant of the elicitation process, depending upon the technique employed, will be, what we have termed the elicited representation. This will, for example be a transcript in the case of an interview or an on-line report. The aim of this stage in the life cycle is to provide a permanent record of the knowledge, in the form in which it was extracted. This will enable the knowledge engineer to follow a knowledge trail later in the process if necessary (e.g., maintenance phase).

The process of eliciting the different knowledge types, perhaps from different sources, with differing Knowledge-Levels, using different techniques at different periods of time means that there will be a set of elicited representations which together form a historical database of elicited knowledge.

3.3.1 The Knowledge Filter

In this section we will attempt to illustrate how the concept of the *specifications as levels* is realized in practice. In order to do this we introduce the software development process we term the *Knowledge Filter*, which is itself a series of subprocesses, each of which take the elicited representation and process it in order to distil a resultant output that reflects the sub-process function. For example, we utilize the sub-process of conversational coherence to obtain an understanding of the *alignment* within the elicited representation [Ragan.83].

The aim of these sub-processes is to act as a series of knowledge filters, each of which enable a different perspective of the elicited representation to be obtained. These can then be utilized in the subsequent system development.

The knowledge filter process can be used to illustrate the notion of specifications as levels. As we have noted, Newell identified four components that define a Knowledge-Level, which can be summarized as follows:

■ A *medium*	to be processed
■ *Components*	that provide primitive processing
■ *Laws of composition*	that permit components to be assembled into *systems*
■ *Laws of behavior*	that determine how the system behavior depends upon a component behavior and the structure of the system

Plus the two constraints:

- ■ The system can be defined autonomously without reference to any other level
- ■ Each level can be reduced to the level below. Each aspect of a level - Medium, Components, Laws of Composition, and Behavior, can be defined in terms of systems at the next level.

Thus, in relation to the knowledge filter described above, we can see that the elicited representation acts as the *medium* to be processed. The sub processes of the knowledge filter act as the *components* that provide the primitive processing. The analysis of the information resulting from the primitive processing results in the meta knowledge model and the intermediate representation, both of which can be defined formally, these act as the *laws of composition* which permit the components to be assembled into systems. The meta knowledge model and the intermediate representation are also bound through formal descriptions of their behavior e.g., what can and can not be added to them, and thus these form Newell's *laws of behavior*. Further each of the representations can be considered in isolation or rigorously transformed into the next representation, thus obeying Newell's constraints.

We can see therefore, that these form one tier of a methodology which when added to the other tiers provides a description of a complete Knowledge-Level development environment.

3.3.2 The Meta Knowledge Model

As we have discussed in the previous section, the knowledge filter acts to isolate the different aspects of information contained within the elicited representation. A resultant of this is the Meta Knowledge Model. The concept of which is illustrated in Figure III:

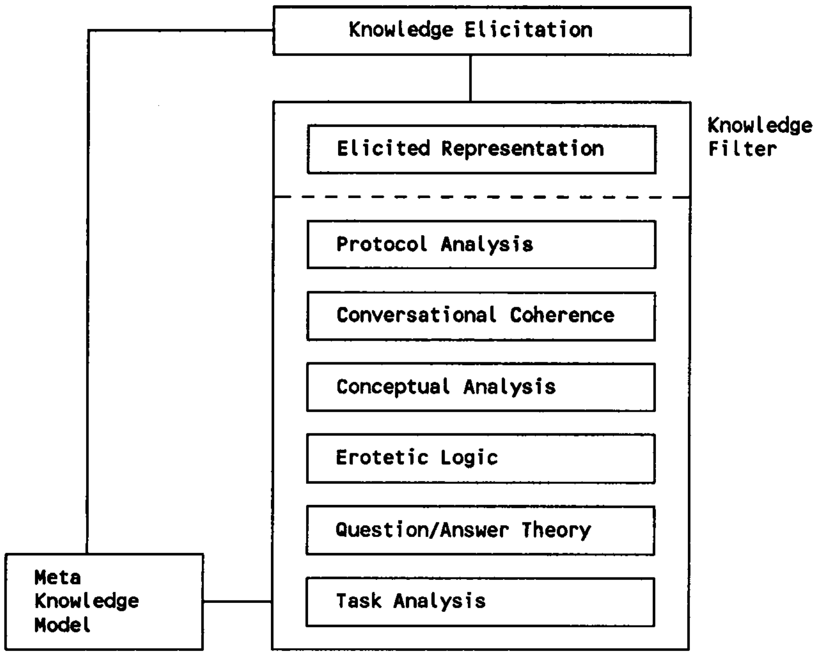

Figure III. Meta Knowledge Filter and Feedback Loop

This shows how the meta knowledge identified during the knowledge filtration process is represented in the Meta Knowledge Model in a formal manner. This knowledge is then fed back into the elicitation process to enable the knowledge engineer to obtain the granularity and scope of knowledge required in conjunction with further meta knowledge. Once the elicitation process has reached a steady state the meta knowledge model is used in conjunction with the intermediate representation to feed into the next level, whereby a more formal domain specification, cognitive engineering specification and representation specification are constructed. The advantage of using the meta knowledge model is that the knowledge engineer is no longer only creating a system based upon static domain information but information from the Meta-Knowledge Level.

3.3.3. The Intermediate Representation

The second output of the knowledge filtration process is the production of the intermediate representation. The primary function of which is to provide a mechanism that is rigorous enough to allow several demanding analyses to take place upon it. One of these ultimately produces a formal specification of the domain knowledge and another acts as the basis for the selection of the high level "classical" representation such as a production system, which ultimately will be used to represent the domain knowledge held in the formal specification.

As stated the aim of this phase is to produce from the elicited knowledge a more rigorous, intermediate representation [Scott,91]. This representation will be structured in form, syntax and semantics, such that the knowledge acquisition necessary to transform the elicited representation will identify the inconsistencies, incompleteness and any incorrectness in the elicited representation. The knowledge engineer will use this intermediate representation to draw together the knowledge from the varying elicited forms: transcripts, repertory grids, questionnaires etc. Intermediate representations are of the form: decision tables, AND/OR graphs, decision trees ; each of which encourage completeness, correctness & consistency; allow for refinement and reduction while having clean yet concise structures.

3.4. Domain Specifications

The creation of the dynamic meta knowledge model and the static intermediate representation allows the knowledge engineer to have a dual perspective in the remainder of the system development. The aim of the intermediate representation is to provide a more rigorous form than the elicited representation with which to reason about the domain, while the meta knowledge representation contributes by allowing the knowledge engineer to understand and be more sensitive to the dynamic aspects of the systems development e.g., the use of meta knowledge allows us to enhance our understanding of the systems explanation capability, as defined in our cognitive engineering specification.

The meta knowledge model and the intermediate representation are used in the creation of three specifications at the next Knowledge-Level: the domain specification, the cognitive engineering specification and the representation specification. The first of these specifications, the domain specification is intended to provide a specification that focuses exclusively upon the domain knowledge, the static knowledge of rules, facts, and heuristics. The aim of this specification is to allow a knowledge-based system to have a repository from which the domain can be considered in isolation. This has several advantages, for example in the course of maintenance or subsequent system updates the domain specification will be the unique location for the domain knowledge to be added, deleted or modified. The knowledge engineer will be able to maintain the correctness, completeness and consistency of the system as far as possible. These changes can then be traced throughout the remainder of the development process. The formalized procedures for updating the domain specification can also be specified for added rigor.

The domain specification therefore will have to have mathematics as its basis and this lead to the adoption of the "Z" notation, a formal specification language. Specifications in "Z" consist of formal text and natural language text. The former provides a precise specification while the latter is used to introduce and explain the formal parts. Specifications are developed via small pieces of mathematics that are built up using the schema language to allow specifications to be structured. This leads to formal specifications that are more readable than a specification presented in mathematics alone.

It is also advantageous to have a domain specification from the perspective of knowledge engineer-user-domain expert communication, as the specification can act as a medium for communication. Thus, it can be seen that the use of a formal language in the development of a knowledge base is very advantageous.

3.5. The Cognitive Engineering Specification

The cognitive engineering specification, as we have already noted, is composed of many aspects, including: cognitive task analysis, knowledge-encoding, competence and performance modelling. The combined effect of utilizing these cognitive components is very powerful, and can be considered as a chief factor in maintaining the semantic correctness of the system as a whole, filling the gaps in the decision making process. This can be seen as aspects of differing phases feed into each other. For example, the knowledge contained in the domain specification can be considered in light of the knowledge-encoding techniques and this has an impact upon the choice of representation in the representation specification, which in turn will determine the systems ability to manipulate domain knowledge.

The cognitive engineering specification also provides two models:

- The Competence Model: that provides a model of the required competence expected from the model in the domain. [Roth,89]

- The Performance Model: that describes the knowledge and strategies that characterize good and poor performance in the domain. [Roth,89]

The adoption of the cognitive engineering specification in these two roles can then act as the basis of a quality assurance mechanism, for competence and performance.

3.6. The Representation Specification

The next step in the development methodology is to identify which (if any) classical or hybrid representation is the most suitable form, around which to base the representation specification, where the classical representations are *"frames", "production systems", "semantic networks"* etc. In order to find the most suitable form, several influencing factors have to be taken into account:

- Information obtained from performing knowledge acquisition upon the intermediate representation.
- Information pertaining to representation selection that can be obtained from considering the composition of the domain specification.
- Information resulting from the cognitive engineering processes.

Each of these information sources provide valuable insights on which representation would provide the best basis for the domain under consideration. The analysis of the intermediate representation will allow a coarse analysis of the underlying domain structure to be obtained. This is refined by considering the composition of the domain specification in terms of its knowledge and data types, their inter-relationships, and structures. This is then enhanced by the cognitive mapping drawn from the cognitive engineering processes.

In order for a suitable match, the characteristics looked for in the intermediate representation, the domain specification, and the cognitive models have to be re-engineered in the examination of the representation schemes themselves (rules, frames, etc). Once this is achieved the match can then be made. The chosen representation is formally specified in terms of its semantics and syntax, this forms the representation specification [Craig,91].

3.7. The Concrete Specification

Having created the domain, cognitive and representation specifications, we are now at a point at which these specifications can be combined into a form that will allow us to move towards implementation. This stage is known as the *concrete specification.*

The creation of the concrete specification is in stages, firstly the domain knowledge is transformed from its "Z" specification into the form advocated by the representation specification, and secondly a formal specification of the control architecture that is associated with the representation is created.

It should be noted that this is not the implementation as the representation is a hybrid between a high level version of what is to be implemented and a formal specification in the style suggested by the syntax and semantics of the representation specification (e.g., pseudo code). The aim being to produce an implementation independent representation of the system. This will allow the knowledge engineer to have a simplified version (minus the complex syntax) with which to reason about the implementation later in the systems life cycle e.g., maintenance.

3.8. Coding

Having created the concrete specification this is then used as the basis of the system implementation. The interface issues being resolved by referral to the cognitive engineering and man-machine interface specifications.

The implementation of the system should be the most straight forward of all the stages, due to the high degree of structuring and refinement that has been performed upon the system in the previous phases.

The mechanism through which the system is implemented is left open to the knowledge engineer as this is considered a trivial exercise once the specifications have been developed.

3.9. Validation and Verification

The methodology we have presented does not have an explicit validation and verification stage, as the approach has aimed at presenting a philosophy of total quality development. We can qualify this by equating verification to the Knowledge-Level. Verification has been defined by IEEE (no. 729-1983) as:

> "The process of determining whether or not the products of a given phase of software development meet all the requirements established during the previous phase."

This we can equate to Newell's second Knowledge-Level constraint (cited earlier in the paper), where by each Knowledge-Level can be shown to be reduced to the level below. Thus, by following a refinement process such as we have advocated, the verification of the system will naturally follow.

3.10. Testing and Integration

Having completed the development of the knowledge-based component the developer can now consider the integration of the system into any other system. An aim of this methodology is to minimize the amount of overhead involved in the process of embedding the knowledge-based component. This is the reason for the use of system wide development standards, historical databases, metrics, formal methods and an adherence to integration throughout the system/process life-cycle.

4. Institutionalization

An aspect of development that we have not yet addressed is that of institutionalization. This has been identified by Liebowitz [Liebowitz,91] and others as being the critical factor affecting system acceptance, usage, and ultimate success. The process of institutionalization can be broken down into three fundamental aspects: implementation, transitioning and maintenance, all three of which have historically been weak when considered with respect to the case of expert system development. Liebowitz identifies four areas vital to the institutionalizationprocess: i) An awareness of expert systems for managers, ii) User training strategies, iii) User support service strategies, iv) maintenance. All of these areas incorporate what Badiru [Badiru,88] terms Triple C - Communication, Co-Operation & Co-Ordination, important management aspects to the creation of an expert system. Thus, we see the process of institutionalization as the connecting link between the management and technical perspectives of system development [Plant.93b].

The institutionalization of system development can be considered as the *holistic* approach to development, where all levels of personnel, from users to managers are involved in the development process, what Leonard-Barton terms "integrative innovation" [Leonard-Barton,87]. The model we have proposed here attempts to overcome these problems of institutionalization through the participation of management and instilling an awareness of the technology involved to them. Further, the model aims to actively involve the user/client in all aspects of the development. This is vital to the institutionalizationprocess in that the technology transfer is greatly eased. This is apparent in many ways; the user becomes more understanding of the technology involved, the developer is relieved of the total obligation for system correctness as this is now shared with other members of the development team, and the probability of system success is increased if there is a continual involvement and thus continual feedback from all members and levels of the organization. Liebowitz has also identified other influencing factors that effect the institutionalization process, including the following:

- System Migration
- Standards
- Configuration Management
- Testing
- User Support Services
- Maintenance
- User Training

We will now briefly touch upon some of these areas as they relate to our model. However, for a fuller treatment the reader is referred to Liebowitz [Liebowitz,91].

The ability for the developers to perform system migrations is greatly eased by having a set of specifications from which to work. This facilitates the changes in platform that may occur over the systems life cycle. These specifications and the adoption of a rigorous development strategy also allow for close adherence to standards and subsequent changes in those standards.

It has been our aim to maximize the systems correctness, however a by-product of the representation refinement approach is to allow for, and support maintenance. By partitioning the specifications into their functional areas, the knowledge engineer can integrate any maintenance needs into the existing specifications in such a way as to assess the impact this will have upon the existing specification - thus maintaining integrity and correctness. As stated by Liebowitz "maintenanceis a key issue in institutionalizingexpert systems" [Liebowitz,91], and hence we have placed a heavy emphasis in designing our methodology to support this function.

5. Summary & Conclusions

This paper has attempted to illustrate several points. Firstly, that when a development lifecycle of representation refinement is utilized, that follows the principles of Newell's Knowledge-Level, then the system will become self validating.

The second issue addressed was the use of a meta Knowledge-Level that allowed for the construction of a meta knowledge model. This model allows for a dynamic perspective of the system to be obtained in conjunction with the static aspects found in the intermediate representation. The use of the knowledge filter to produce a better meta knowledge model and intermediate representation was introduced along with the meta knowledge feedback loop to induce the elicitation of further meta knowledge.

The paper therefore is intended to show that the ability to validate knowledge based systems is not one that can be seen only from the static testing of the knowledge base but must emanate from a holistic Knowledge-Level development method.

In conclusion the paper has shown a methodology for the creation of knowledge-based systems that has attempted to utilize the rigor of software engineering and cognitive engineering towards the obtainment of a better understandingof the knowledge engineering process. The use of formalized techniques in conjunctionwith a rigorous Knowledge-Level development style will enable better degrees of software quality to be obtained. The future of knowledge based software development methodologies lies in the utilization of formal methods in conjunction with software metrics. The use of metrics is vital for us to obtain a better understanding of our methodologies, transformationprocesses and development techniques. Through the use of metrics we will be able to better establish the correct or most appropriate representation to use for a given domain segment, or assist the knowledge engineer to better understand the interface issues. Thus, the next stage in the research is to develop a model that utilized these techniques to better understand the Knowledge-Level development process.

References

Badiru, A.B., (1988), Successful Initiation of Expert Systems Projects", IEEE Transactions on Engineering Management, Vol. 35, No. 3, IEEE, August 1988.

Bellman, K., & Landauer, C. (1991), Testing and Evaluating Knowledge-based Systems, AAAI workshop on Validation and Verification, August 1991, Boston MA.

Breuker, J. & Wielinga, B., (1987), Use of Models in the Interpretation of Verbal Data. In, *Knowledge*

Acquisition for Expert Systems: A Practical Handbook, Kidd, A.L. (Ed), New York Plenum

Burton, A.M., Shadbolt, N.R., Hedgecock, A.P., & Rugg, G. (1987), A Formal Evaluation of Knowledge Elicitation techniques for Expert Systems: domain 1., In, *Research and Development in Expert Systems IV*, Editor: D.S. Moralee., BCS Series, Cambridge University Press. Cambridge, England.

Craig, I.D., (1991), *Formal Specification of Advanced AI Architectures*, Chichester, England, Ellis Horwood.

Culbert, C, (1990), Verification and Validation of Knowledge-Based Systems, Special Issue: Expert Systems with Applications, Vol. 1., No. 3. Permagon Press, New York.

Dhaliwal, J.S, & Benbasat, I. (1990), A Framework for the Comparative Evaluation of Knowledge Acquisition Tools and Techniques. Knowledge Acquisition, 2, 145-166.

Hesketh, P., Barett, T. (1990). An Introduction to the KADS methodology. Esprit Project P1098, Deliverable M1, EEC ESPRIT Project, Brussels, Belgium.

Jacob, R.J.K. (1983), Using Formal Specifications in the Design of a Human-Computer Interface. Communications of the ACM, April 1983, Vol. 26, No. 4.

Jones, C. (1980), Software Development a Rigorous Approach, Prentice Hall.

Leomard-Barton, D. (1987), The Case for Integrative Innovation: An Expert System at Digital, Sloan Management Review, MIT, Cambridge, MA.

Liebowitz, J. (1991), *Institutionalizing Expert Systems: A Handbook for Managers*, Englewood Cliffs, Prentice Hall

Liebowitz, J. (1986), Useful Approach for Evaluating Expert Systems, Journal of Expert Systems. 3(2):86-96.

Miller, L. (1990), A Realistic Industrial Strength Life Cycle Model for Knowledge-Based System Development and Testing. AAAI Workshop Notes: Validation and Verification, August 31st, Boston Mass.

Morell, L.J., (1989), Use of Metaknowledge in the Verification of Knowledge-Based Systems, NASA Contractor Report 181821, Langley research Center, Virginia.

Newell, A. (1982), The Knowledge-Level, Artificial Intelligence 18, pp 87-127, North Holland.

O'Leary, D.E. (1987), Validation of Expert Systems- with Applications to Auditing and Accounting Expert Systems. Decision Sciences, Vol 18. pp 468-486.

O'Leary, D.E. (1993), *Collected Papers 1989-92:AAAI Workshops on Validation & Verification*. (In Preparation)

Plant, R.T. (1993), A Rigorous Development Methodology for Knowledge-Based Systems. Communications of the ACM (To Appear)

Preece, A.D., Grogono, P., & Shinghal, R. (1993), Assessing the Capability of Knowledge-Based System Developers, IEEE CAIA Workshop on Validation and Verification, Orlando, Fl, March 2nd 1993.

Ragan, S.L. Aligment and Conversational Coherence, In, Conversational Coherence: Form, Structure, and Strategy. Edited by Robert T. Craig & Karen Tracy, Sage.

Roth, E.M, & Woods, D.D., (1989), Cognitive Task Analysis: An Approach to Knowledge Acquisition for Intelligent System Design., In: *Topics in Expert System Design*, G. Guida & C. Tasso (Editors), pp 233-264, Elsevier Science Publishers.

Royce, W.W. (1970), Managing the development of large software systems. Proc. WESTCON, Ca. USA.

Rushby, J. (1988), Quality Measures and Assurance for AI Software. NASA Contractor Report 4187.

Scott, C. (1991), A Practical Guide to Knowledge Acquisition. Reading MA, Addison Wesley.

Spivey, J.M. (1990), The Z Notation. Prentice Hall

Sufrin, B.A. & He, J. (1990), Specification, Analysis and Refinement of Interactive Processes. In, *Formal Methods in Human-Computer Interaction*. Editors, Harrison, M., & Thimbleby, H. pp 153-200., Cambridge University Press.

Welbank, M. (1983), A Review of Knowledge Acquisition Techniques for Expert Systems. British Telecom Research Labs. Martlesham Consultancy Services.

Woods, D.D., & Hollnagel, E. (1987), Mapping Cognitive Demands in Complex Problem Solving Worlds. International Journal of Man-Machine Studies, 26, 257-275.

Woods, D.D, & Roth, E.M., (1988), Cognitive Systems Engineering. In M.Helander (Ed.), Handbook of Human-Computer Interaction. New York: North Holland.