Verification and Validation of Knowledge-Based Systems with Formal Specifications

Pedro Meseguer*

Universitat Politècnica de Catalunya

Departament de Llenguatges i Sistemes Informàtics

Barcelona, Spain

Email: meseguer@lsi.upc.es

Alun D. Preece

University of Aberdeen, Computing Science Department

Aberdeen AB9 2UE, Scotland

Phone: +44 1224 272295; FAX: +44 1224 273422

Email: apreece@csd.abdn.ac.uk

*Current address: Institut d'Investigació en Intelligència Artificial (IIIA), Consejo Superior de Investigaciónes Cientificas, 08193 Bellaterra, Spain.

Abstract

This paper examines how formal specification techniques can support the verification and validation (V&V) of knowledge-based systems. Formal specification techniques provide levels of description which support both verification and validation, and V&V techniques feed back to assist the development of the specifications. Developing a formal specification for a system requires the prior construction of a conceptual model for the intended system. Many elements of this conceptual model can be effectively used to support V&V. Using these elements, the V&V process becomes deeper and more elaborate and it produces results of a better quality compared with the V&V activities which can be performed on systems developed without conceptual models. However, we note that there are concerns in using formal specification techniques for V&V, not least being the effort involved in creating the specifications.

Keywords Knowledge-based systems, artificial intelligence, formal specification, verification, validation, life-cycle.

1 Introduction

Unlike most conventional software, knowledge-based systems (KBS) are rarely validated and verified against an explicit specification of user requirements. Probably the main reason for this is the lack of a complete and common understanding of the task which a KBS is intended to perform among end-users, human experts and KBS builders. This lack of understanding is supported by (i) the difficulty to define and describe precisely typical KBS tasks and (ii) the difficulty of communication among them due to the use of different languages. Typically, development proceeds in an exploratory manner—usually via prototyping—until a system is produced which is deemed to embody the implicit requirements of the prospective users. This explains why most of the early efforts to validate KBS involved comparing their performance directly against the performance of human performers, rather than against any specification document (Buchanan and Shortliffe, 1984).

The problems with this approach are obvious: the validation is inherently prone to bias and, even when an "acceptable" system is deemed to have been produced, it is not clear what the system actually does (or may fail to do). Much of the work done in recent years to improve the state of verification and validation (V&V) practice for KBS has held on to the assumption that, because KBS are difficult to specify, practical V&V techniques should not depend upon the existence of detailed specification documents. This assumption is clearly seen in, for example, the work done on checking KBS for domain-independent *anomalies* such as inconsistency and incompleteness (Preece et al., 1992), as well as in the quantitative techniques for comparing KBS with human "experts" (O'Keefe et al., 1987).

Such techniques have been shown to be effective, but to a limited extent only. This paper will examine the limitations of a number of "state-of-the-art" KBS V&V techniques, and will assess how the power of the techniques can be extended when precise specification documents are available for the system. We will focus upon *formal specifications*, because informal and pseudo-formal specifications are too weak to provide a foundation for V&V. (We note, however, that non-formal specifications may be *approximated* by formal specifications and employed for *partial* V&V; this is the role played by the *pseudo-formal* specifications in (Laurent, 1992).)

Therefore, the main aim of this paper is to examine the ways in which formal specification techniques can support the V&V of KBS. In doing so, two related issues naturally come under consideration. Firstly, it becomes apparent that verification and validation techniques can, in turn, support the development of formal specifications. Secondly, it becomes necessary to consider how specification, verification and validation techniques need to be applied within the whole KBS development process. This paper will touch upon these issues, although they will need more detailed consideration in their own right.

Verification and Validation of KBS Before proceeding, it is necessary briefly to define the terms verification and validation for the purposes of this paper. *Verification* is a process aimed at demonstrating whether a system meets its specified requirements; this is often called "building the system right" (O'Keefe et al., 1987), which we take to mean "checking the system against its (preferably explicit and formal) specifications". *Validation* is a process aimed at demonstrating whether a system meets the user's true requirements—often called "building the right system". Verification and validation can be viewed as a set of techniques and an associated process in which the techniques are applied, as part of the whole development process (for example, static verification of the knowledge base, followed by dynamic testing of the whole KBS (Preece, 1990)).

Formal Specification Techniques for KBS Like V&V, formal specification techniques for KBS include a number of techniques (formal specification languages of various kinds) and processes (for example, transformation from a pseudo-formal specification to an implementation through several levels of detail). In software engineering, a *formal specification* for a software system includes (Potter et al., 1991): (i) some specification of the input-output behaviour of the system (establishing the correct relation between data and results) and (ii) a description of how this behaviour can be effected. The first element corresponds to a *black box* view of the system, caring only for an external requirement of I/O behaviour, while the second element corresponds to a *glass box* view, prescribing the internal workings of the required system (Ghezzi et al., 1991).

The term *user requirements specification* conventionally refers to (i) only, implying that the prospective users do not care how their required system behaviour is achieved. In fact, from the standpoint of traditional software specification, (ii) is perhaps controversial, since it blurs the distinction between specification and design. However, this type of specification is a major feature of current KBS formal specification techniques (Fensel and van Harmelen, 1994; Treur and Wetter, 1993), and can be of great benefit in V&V, so we will not debate its appropriateness here. We note, however, that current formal specification techniques for KBS tend to merge (i) and (ii) into a single description, so that the I/O behaviour must be determined from the specification as a whole.

2 Verification and Validation of KBS

The earliest validation technique in AI was Alan Turing's proposal on how to decide if a program could be considered "intelligent", commonly known as the "Turing test" (Turing, 1950). This is a blind test where the evaluator communicates through a teletype with a person and a program; if the evaluator is unable to differentiate between the person and the program, the program is considered to be intelligent. Although many criticisms have been levelled against the Turing test as a general procedure to characterize intelligent behaviour, the idea of blind testing has remained central in KBS validation from the earliest systems on (see, for instance, the validation of the MYCIN system (Buchanan and Shortliffe, 1984) and blind testing in medical KBS (Chandrasekaran, 1983)).

In addition to testing, KBS developers realized that rule bases could be analyzed for anomalies which are indicative of errors in the construction of the KBS, and which can lead to faulty behaviour at run-time. Commonly considered anomalies included inconsistency, redundancy, subsumption, circularity, unreachable goals, and unfireable rules. Tools to detect such anomalies were called verifiers due to the logical nature of the tests. Early verifiers, performed pair-wise comparison of rules (Suwa et al., 1982); more sophisticated techniques including the effect of rule chaining were used in (Ginsberg, 1988). The use of such verifiers has been widely acknowledged as being complementary to the necessity of testing. Nowadays, validators perform a combination of verification and testing methods in order to obtain maximum evidence as to the correctness of KBS.

It is well known that software validation cannot be delayed until implementation. Otherwise, there is too high a risk that errors will be found late which may be very expensive to correct. This principle, coming from software engineering, also applies to knowledge engineering. However, most of the validation approaches developed for KBS assume to work on an implemented system (in the context of prototyping). Several authors have made proposals to include validation during the early stages of KBS development, but this does not appear to have become common practice, and there is consequently little published evidence of the practical usability of existing techniques early in development.

2.1 Dominant V&V Techniques for KBS

Currently, the dominant techniques for V&V activities can be clustered in four main groups:

- Inspection
- Static verification
- Empirical testing
- Empirical evaluation

Inspection techniques aim at detecting semantically incorrect knowledge in the KB. Inspection is performed manually, by a human who has expertise in the application domain. During development this is usually the same expert who provided the knowledge for the KB, but at some point the KB should be inspected by an expert independent of those involved in the KBS development. (Typically, this technique can be used only infrequently due to the lack of availability of experts.). Inspection is mostly able to identify errors in isolated KB elements: when errors come from the interaction of several KB elements—for instance, chaining of several rules—human inspectors are usually unable to detect it "by eye".

Static verification checks the KB for anomalies. An anomaly is a static pattern in the KB structure which suggest the presence of an error in the encoded knowledge (Meseguer, 1992). Typically, the anomaly pattern is a counterexample of a general property which should hold in the KBS; for example, consistency. Detected anomalies need to be analyzed to determine whether they represent a real error or just a minor defect coming from the encoding process in the selected knowledge representation. Only the most limited verification checks can be performed manually; generally this process requires computational support by automated tools. Depending on the capabilities of the verification, the checks it may perform range from a limited to an exhaustive search for anomalies in the KB (Meseguer and Verdaguer, 1993). It is worth noting that, although the properties to be checked are to a essentially domain-independent, verification tools depend on the specific semantics of the knowledge representation language used. For this reason, verifiers cannot be reused among KBS using different knowledge representation languages.

Empirical testing aims at checking KBS correctness by executing the system on sample data sets. To guarantee complete correctness, testing has to be exhaustive; that is, every potential input should be tested. This is obviously not feasible for real applications, so testing only analyzes a finite set of test data, the *test set*. The selection of the test set is crucial to the effectiveness of the testing process. While in software engineering random testing has been shown to be the most cost-effective technique, in knowledge engineering the combination of structural and functional testing seems to be superior to other techniques (Rushby and Crow, 1990; Zualkernan et al., 1992).

Structural testing aims at executing as many of the KB components as possible; for example, firing as many rules as possible, and instantiating as many object attributes as possible. Functional testing checks the function of the KBS by comparing its observed input-output relationship with that specified in the requirements, without considering internal structure. Real test cases are usually scarce, so test cases have to be synthesized automatically by test case generators. A final difficulty in KBS testing occurs when the application domain is so ill-defined that “correct” behaviour is not well-defined (there is no “gold standard”). In such cases, it is necessary to make some definition as to what is to be considered a “correct” or “acceptable” solution for each test case; usually, the correct solution is approximated by a consensus among the opinions of several human experts.

Evaluation addresses the relation between the operational KBS and the final user. Typical evaluation issues are technical performance, acceptability, inclusion in the organization, responsibility issues, and so on. Empirical evaluation is performed by using the operational KBS either in a controlled environment (laboratory evaluation) or in the working environment (field evaluation). KBS evaluation is a human activity which is highly application-dependent.

Of these four groups, inspection and empirical evaluation methods are clearly application-dependent and they are not candidates for potential reuse across different KBS. On the other hand, verification and testing *methods* can be reused to a great extent on different KBS, even though the computational tools supporting the methods are usually bound to specific knowledge representation languages. However, even in the reusable methods, the role of human experts remains significant, because they are needed to evaluate verification and testing outputs.

2.2 Limitations of Current Approaches

While success has been achieved using the above techniques, there are still causes for concern. The presence of requirements which are hard to formalize or express without ambiguity induces different weaknesses in the V&V process. This kind of requirement is difficult to validate, and so one never knows to what extent such requirements are fulfilled. This problem becomes more difficult when different requirements coming from different users are to be integrated. Here there is an extra issue: to guarantee the internal consistency of the set of requirements. In any case, there is no rigorous way to verify the correspondence between a set of requirements and its final implementation.

The absence of formal specifications limits the capability of KBS verification, which remains constrained to assuring that some domain-independent properties hold in the system. An example of a domain-independent property is *consistency*, which means that from a consistent input the KBS cannot produce a contradictory output. Domain-independent properties appear as prerequisites for adequate functioning of a KBS, and they should be tested. However, although they are necessary they are not sufficient because they say little about the level of actual KBS *correctness*. Consider a (somewhat exaggerated) analogy comparing knowledge engineering with numerical programming: checking a rule base for consistency is analogous to checking in a numerical program that no computation arrives to overflow or underflow. Obviously, it is something useful to know, but it has nothing to do with the correctness of the computation: one can have a consistent rule base (or a neither-overflowing-nor-underflowing numerical program) that performs totally incorrect computations, which nevertheless are consistent!

To evaluate correctness, a number of domain-dependent properties should be tested;

these contain constraints that the KBS output must satisfy to comply with the domain knowledge. Some examples of domain-dependent properties in the medical domain are the following: (i) in diagnosis, a KBS cannot ignore a cause which may put the patient's life in danger (for example, a dangerous bacteria) even if the chance of it being the actual cause of the illness is small, and (ii) in treatment, a KBS cannot prescribe a drug dose which causes a risk to the patient's life.

Results from testing indicate that the system performs adequately on a set of cases, and it is reasonable to expect that the system will behave in the same way on similar cases. If the test set is representative of the set of possible inputs (which in complex KBS rarely occurs), this may represent enough evidence for non-critical tasks. However, when a new case appears, we cannot be sure that the system will behave properly on it (except for trivial cases, or very specific applications where advanced testing techniques can be applied).

In addition to the weaknesses induced by the presence of ambiguous or hard to formalize requirements, another important limitation of current V&V techniques comes from the fact that they are only applicable to implemented KBS. As we have stated previously, validation cannot be delayed until implementation; otherwise the cost of correcting errors can be very high. Ignoring validation until implementation is finished has also an important impact on KBS design: typically, KBS are constructed to be executed but not to be validated (nor to be maintained, etc.). This makes the validation task more difficult, and as a consequence, in many occasions it can only provide indirect evidence of the properties to be tested and of the quality of KBS parts.

In summary, the main weakness of current V&V approaches with respect to implemented KBS lies in the fact that they do not really bring confidence in the quality of the system as a whole and of the system parts. V&V methods should provide adequate answers to issues such as correctness, completeness, robustness, precision, safety, and so forth. Currently, many of these questions are answered only partially or by indirect evidence only. To overcome these defects V&V methods have to employ more precise information about the *task* the KBS is required to perform. Formal specifications can play a fundamental role in accomplishing this goal.

3 Formal Specifications for V&V

Strictly speaking, no verification or validation is possible without specifications, by the definition of verification (Section 1). While some of the existing verification techniques may *appear* to operate in the absence of any specifications, in reality they make reference to *implicit* specifications; for example, the techniques for verifying the consistency of knowledge bases make reference to some model of consistency, which can be considered a domain-independent requirement (that the KBS be free from inconsistency) (Preece et al., 1992). Similarly, early validation efforts comparing a KBS against human performers made implicit reference to a requirement that the KBS should emulate such human performers (Buchanan and Shortliffe, 1984).

Once the necessity for specifications is clear, it must be said that the more precise and detailed are the specifications, the more thorough will be the V&V process. At the least-desirable extreme we have the case where the sole requirement is implicit emulation of human performers. If we have informal or semi-formal statements-of-requirements for the system, then we can devise tests to determine if the system complies with these requirements; however, because the requirements are not stated formally:

- there is no way to be sure if we have tested them completely (or to what extent we have tested them);
- they may be ambiguous, incomplete and incorrect.

The most desirable case, then, is to have requirements that are specified formally (with a well-defined syntax and semantics). Such specifications are produced by formal methods and, thus, such methods clearly have a role in the V&V process for KBS.

3.1 Formal Specifications for KBS

A growing number of formal specification languages are available—and have already been used—to specify KBS. We can categorise them roughly according to the intent of their developers:

- *General purpose* specification languages, developed in the context of conventional software engineering, for the specification of any type of software system. For example, Z (Plant and Gold, 1990) and VDM (Haugh, 1988) have been used to specify KBS.
- *Special purpose* specification languages, predominately European in origin, developed for the purpose of specifying complex knowledge-based reasoning systems. Among the best known are DESIRE, KARL and $(ML)^2$, surveyed in (Fensel and van Harmelen, 1994; Treur and Wetter, 1993).

Although some work has been done applying general purpose specification languages to KBS (Krause et al., 1990), for our purposes the second category is more attractive

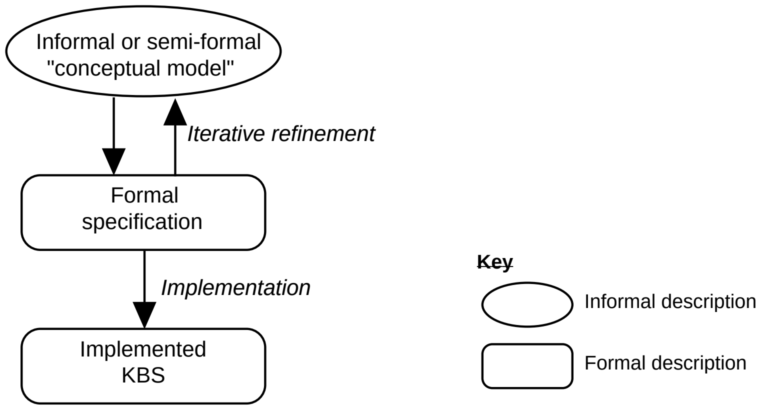

Figure 1: Using a formal specification in KBS development.

because the facilities of the languages are well-suited to specifying KBS and these languages are better-suited to the practical needs of KBS developers. Using special purpose specification languages a specification can be developed gradually, as a refinement of informal and semi-formal descriptions of the system: see Figure 1. Here, the informal description—typically called the "conceptual model" in various methodologies—is iteratively refined to create a more precise formal specification, with *both* descriptions undergoing gradual modification during the process. The formal specification keeps the structure and vocabulary of the conceptual model. It is kept to make easier the communication with domain experts in the development and validation process. The formal description will form the basis for the implementation: later we discuss the various ways in which this can be done.

In fact, the possibilities of using formal specifications are richer than shown in Figure 1. We may want to have several formal specifications, providing different formal descriptions of the system, for example:

- At different levels of detail; for example, a *black box* specification of the system, showing the input-output relations only, and a complementary *glass box* description of the system, showing internal aspects of its required behaviour. These two views of a KBS specification have been referred to as the *problem specification* and the *solution specification*—or as the “contract” and “blueprint”—respectively (Batarekh et al., 1991).

- Of different “models” of the system; for example, the *cooperation model* may describe how the system interacts with its users, while a separate *task model* may describe the tasks the system performs on its own (Wielinga et al., 1992).

3.2 Formal Specifications for V&V

From an idealistic point-of-view, the goal in performing verification and validation is to deliver a KBS that is as reliable as possible; more pragmatically, the goal is to deliver a KBS that is as reliable as *necessary*—given the users’ needs (Miller, 1990). The most important decision is that of *what* to verify and validate.

When formal specifications are available, they provide more opportunities for doing V&V than are available otherwise. Figure 2(a) shows what V&V can be done when the only available descriptions of the system are an informal statement of requirements, and the implementation itself.

The implementation can be verified for internal consistency and apparent completeness, and validated for (approximate) compliance with the informal requirements, using the methods described in Section 2, but that is all. The weaknesses of V&V conducted to this extent were highlighted at the end of the previous section. These weaknesses can be repaired by the use of formal specifications in the following way:

- Issues caused by the presence of ambiguous requirements are now eliminated because V&V is performed using formal specifications, which are not ambiguous. Obviously, there exists the problem of constructing such formal specifications from the informal requirements. This issue can be considered separately from the verification process, which can be performed in a much clean way.

- V&V activities are no longer delayed until implementation. Formal specifications provide statements which are amenable to analysis and verified in their own form (something that was impossible for the informally stated requirements). In this sense, formal specifications allow V&V activities to be included in the early stages of KBS development.

In addition, when formal specifications are available many more V&V opportunities exist, as shown in Figure 2(b). Here, there are the two levels of formal specification—black box and glass box—in addition to the implementation. Note that, in this context, validation is the whole process, which is composed by the first step between the informal conceptual model and the black box specification, plus the subsequent verification steps through to implementation. Hence, validation is performed by verification at several levels of abstraction.

The availability of the multiple formal descriptions provides support for additional:

- *Intra-model V&V* (V&V of a single description of the KBS in and of itself), because the formal descriptions of the KBS can be verified and validated in and of themselves, for properties such as consistency and completeness (appropriately defined); these descriptions can also be validated manually for correctness, because the specifications are unambiguous.

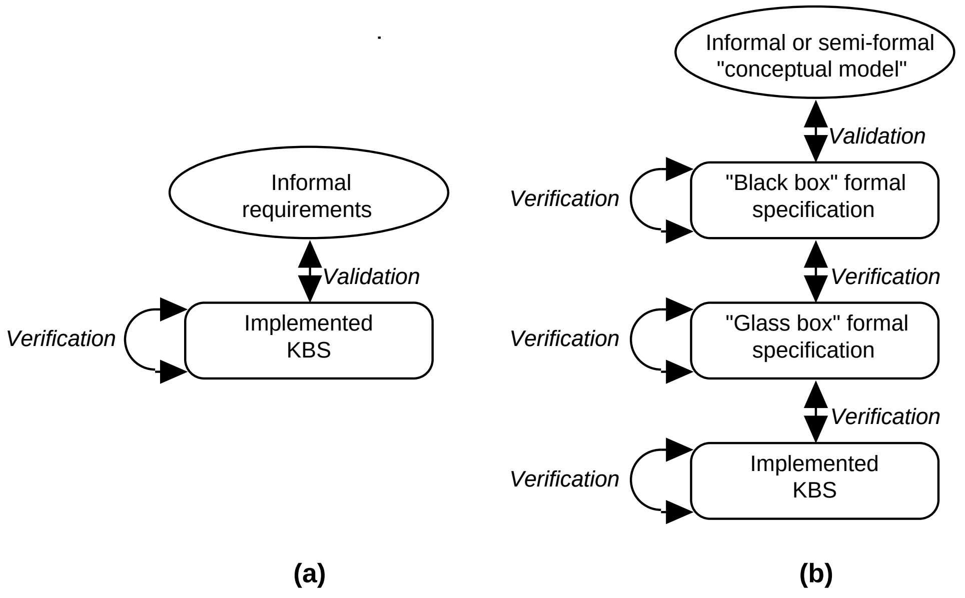

Figure 2: V&V opportunities: (a) without any formal specification; (b) with two levels of formal specification.

- *Inter-model V&V* (V&V between different descriptions of the system), because different descriptions of the system can be verified with respect to one another; this supports validation because one level of description can be the user's requirements, made explicit.

Since the formal specifications are descriptions of the system, V&V techniques may be used to ensure the validity of the specifications themselves in the above contexts.

As described in Section 1, a formal specification provides (i) a description of the intended I/O behaviour, and (ii) a description of how this behaviour can be constructed. Some of the potential uses of a formal specification in V&V are the following:

- To give *structure* to the domain knowledge, providing a vocabulary of knowledge elements which can be used in V&V.

- To allow verification of the intended I/O behaviour in a "black box" manner.

- To allow verification of the way a solution is constructed by the system, in a "glass box" manner. The structure of the domain knowledge plus the specification of how the solution should be constructed must be sufficient to devise V&V techniques specific to the task performed by the system, to provide the necessary level of confidence in the reliability of the system.

- Some languages, such as DESIRE (Treur and Wetter, 1993) and KARL (Fensel and van Harmelen, 1994) produce executable specifications. They can be used to test the system against the ideal behaviour provided by the specifications.

Further to the last point above, it is worth noting that languages producing executable specifications are typically less expressive than non-executable ones in the sense that the latter can deal with infinite domains, while executable specifications are restricted to finite domains. On the other hand, executable specifications can be executed, which is doubly valuable for V&V purposes: (i) it shows the feasibility of the specification with respect to an implementation, and (ii) it demonstrates an ideal behaviour of the system, which can be used to test these specifications against a particular implementation. To use non-executable specifications for these purposes, symbolic execution techniques are required. Typically they produce very large lists of logical predicates as output, which are difficult to manage in practice (Kemmerer, 1985).

The various V&V techniques described in Section 2 can—and in some cases must—be redefined in this context. The question of verifying the specifications themselves is still open, as few of the existing special-purpose KBS specification languages have well-developed proof techniques at present. If general-purpose specification languages are used, then their existing proof techniques apply, but these have been found to entail a great deal of labour when KBS are specified (Haugh, 1988). However, extended versions of the existing V&V techniques for KBS become possible: we consider each of the four groups of techniques described in Section 2.

Inspection In intra-model V&V, at the specification level, it is often argued that the unambiguous nature of formal specifications permits easier direct validation via inspection by customers and—in the case of KBS—domain experts. This is made difficult in practice when unfamiliar and unintuitive notations are employed for the formal descriptions, although this difficulty may be alleviated to some extent because the declarative nature of much of the specification (for example, logical rules) is more comprehensible to non-programmers than procedural descriptions. In addition, some graphical interfaces are being developed in association with formal specification languages; these graphical interfaces can make the communication with domain experts easier. Although all these facilities can alleviate the problem of understanding formal specifications for customers and domain experts, we believe however, that it will usually be more practical to employ the semi-formal conceptual model in this capacity as a more intelligible (though not unambiguous in and of itself) version of the specification.

Like a formal specification, the conceptual model provides an implementation-independent description of the intended task, and therefore can be inspected by domain experts without the burden of implementation details. At this level, inspection aims at evaluating the knowledge quality and completeness with respect to the intended task. If necessary—due to ill-definition or ambiguity—formal specifications can complement the conceptual model. In this case, domain experts would likely require assistance from the knowledge engineers to deal with formal languages.

In inter-model V&V, the high degree of structure provided by a formal specification can be used manually to check for the detection of corresponding structures in the implemented system. The conceptual model may play a role here also: at the implementation stage, one can check the correspondence between the conceptual model and the final implementation of the system.

Static Verification Static verification is possibly the group of V&V techniques most enhanced by the introduction of formal specifications. Here, the specifications allow verification (i) to be performed on the same specification, and (ii) to check either the specification or the implementation for domain-dependent properties. Both lead to significant advances over the current state of static verification.

In intra-model V&V at the specification level, if a formal specification is to be verified according to the properties for static verification (consistency, redundancy, etc), then these properties need to be defined in relation to the formal specification language (see (van Harmelen and Aben, 1995) for an initial approach in this direction). Once this is done, however, the reusability of this approach is high, because the same anomalies can be checked for in any specification written using the language. Furthermore, formal specification languages provide additional opportunities for verification by some of the techniques, as a benefit of their features. For example, the modular architecture and declaration of hierarchies of types (sorts) provided by languages like DESIRE, KARL and $(ML)^2$ permit additional properties to be checked in static verification (for example, violation of modularity, and type mis-matches).

In addition to checking the specification for domain-independent properties, it can be checked for domain-dependent ones. This can be performed in an inter-model V&V

among different levels of specifications. Thus, in Figure 2 (b), a "glass box" specification can be verified against a "black box" specification. The necessity to verify specifications themselves has already been acknowledged in software engineering.

Static verification can also be employed to detect structural nonconformances between specification levels and implementation. This provides an automatic (and hence more reliable) version of the inter-model inspectional verification discussed above. Recall that, in addition to the I/O behaviour, a formal specification describes how this behaviour can be achieved. This implies that a formal specification provides, to some extent, elements of the KBS structure. After implementation, these elements can be checked to verify whether their functionality meets their specification. This is an important step towards constructing correct and reliable KBS because (i) verification is no longer limited to general purpose properties; it can check domain-dependent properties, and (ii) verification can be made of specific parts of the KBS structure, providing direct evidence of correctness for these parts.

Empirical Testing The first benefit that empirical testing obtains from formal specifications is that testing can be performed on the formal specification itself. In this way knowledge engineers can assess whether they are specifying the intended system. This view has been already considered in software engineering (Kemmerer, 1985), in order to prevent the development of costly formal specification proofs which are at the end unachievable in practice. In general, this may require symbolic execution of formal specifications. However, executable specification languages like DESIRE (Treur and Wetter, 1993) and KARL (Fensel and van Harmelen, 1994) are directly testable can be of great help in early stages of KBS development. Testing specifications directly can take the conventional form of running test cases on an executable specification, or can take the form of proving properties (such properties corresponding to “test cases”) of the specification. In order to do this, proof techniques must have been defined for the specification language.

Once testing has been performed on the formal specification of the system, an idealised view of the development process would suggest that empirical testing on the implemented system is unnecessary, because the implementation should be shown to be compliant with the previous specifications. Although the presence of formal specifications may suggest that testing is unnecessary, this view is currently too extreme and impractical, and is likely to remain so for the foreseeable future. The translation from specifications to implementation is largely performed manually, and some errors can be introduced. Therefore, testing is still necessary to obtain empirical evidence that the implemented system behaves properly on a set of cases. Testing can be performed in its classical variants, structural and functional, and formal specifications provide extra support for both approaches. A formal specification provides the intended I/O behaviour, so it contains all the information needed to perform functional testing. On the other hand, the level of structure in formal specifications can be used to support structural testing. In addition, specifications can help greatly in the selection of the test set, or if no test cases are available, to support its automatic generation.

Finally, executable specifications may simplify the testing process of implemented systems. The specification execution acts as the gold pattern which the implemented system must follow, so it is easy to identify behaviour differences and mismatches between the execution traces of both systems. This brings new perspectives on testing techniques, which were unforseen with past approaches.

Empirical Evalation This aspect of V&V would seem at first to benefit little from the use of formal specifications. However, a number of possibilities exist. For example, it is possible to use a formalised model of the required cooperation between KBS and users to check for potential system integration problems in a more principled way—and at an earlier stage in development—than would otherwise be possible. Secondly, subjective requirements such as "user friendliness" can be approximately expressed by means of pseudo-formal specifications, and then become amenable to V&V methods other than experimental evaluation (Laurent, 1992).

Regarding practical experiences of the use of formal specifications in the context of V&V, to our knowledge no such experiences exist so far. The development of conceptual models for knowledge engineering and their representation in formal languages is quite a new approach, and their potential use for V&V is a topic of current investigation, of which this paper is an example. Given the benefits that formal specifications may offer, it is foreseeable that the use of these techniques will support more principled KBS V&V in the near future.

3.3 Outstanding Issues

While formal specification techniques offer clear benefits to V&V, there are a number of unclear issues at present.

Nature of the Development Process One question concerns the nature of the development process incorporating formal specification with V&V. Should a transformation

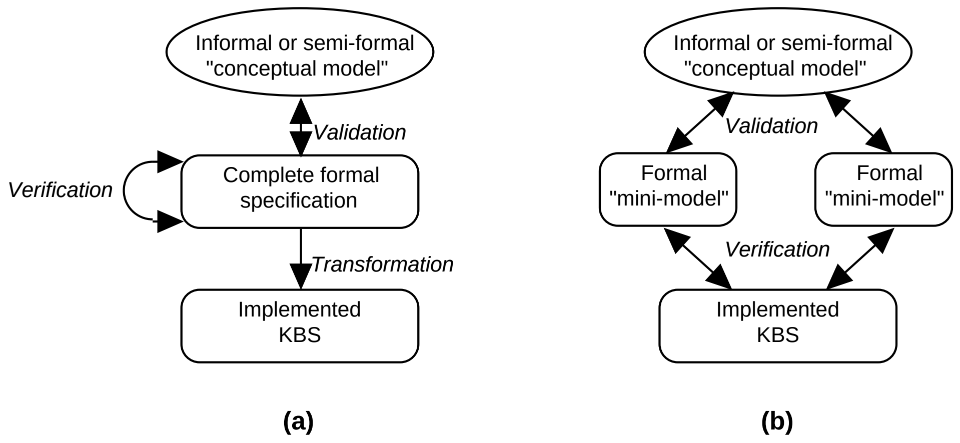

Figure 3: Using formal specifications for V&V: (a) transformation from complete formal specification; (b) use of "mini-models".

approach be adopted, wherein the specification is gradually refined into an implementation, as shown in Figure 3(a)? This is attractive from the V&V point-of-view because it is easier to promote and control validity throughout the process: all verification is performed directly upon the formal specification (benefitting from its well-defined syntax and semantics), and correctness is then assured by the transformation process. However, taking the transformational approach requires a significant effort for developers, who must construct several levels of refined specification until an acceptable implementation is achieved. (The question of effort involved in creating formal specifications is addressed by the companion papers in this special issue.)

The transformational approach contrasts with the more conventional approach shown previously in Figure 2(b), where an implementation is crafted with the intention that it will comply with the specification, without actually being derived from it. V&V techniques are used to establish the compliance between the descriptions.

Completeness of Formal Specifications Another important question concerns the completeness of the specifications; that is, whether the specification should (and, in complex cases, whether it *can*) describe every aspect of the system. One approach to development is to aim towards building a formal specification which is a complete description of the KBS. This can then be transformed (as in Figure 3(a)) or constructed into an implementation.

An alternative approach is to specify different aspects of the system as independent "mini-models", against which the implemented system can be verified (Bellman, 1990)—see Figure 3(b). There may be a strong practical reason for choosing the latter approach: when the KBS is complex and ill-structured, requiring a great deal of knowledge aquisition, analysis and refinement (typically supported by exploratory prototyping) before a reasonably complete version exists. However, transformational implementation is no longer possible because the specification is not complete.

4 Conclusions

In knowledge engineering, the V&V activity is a kind of "watching eye" aimed at detecting deviations between what is intended to be built and the artifact being built. To perform this task, V&V requires information about the intended KBS. The more information that is available, and the more precise that information, the better the validation process will be. This will have a direct impact in the quality of the final KBS.

Conceptual models of the intended task appear to be essential in KBS construction. These models play a fundamental role in the validation process, because they act as the reference to compare against at any stage of KBS development. Conceptual models allow one to structure the domain knowledge, identifying domain-dependent properties which can be effectively tested. These properties can provide direct evidence of the different dimensions to be evaluated in V&V. Importantly, this checking can be made at any stage in the development process, so the difficulties of including V&V activities during KBS construction are removed by this approach.

Conceptual models can support the development of well-structured formal specifications. The presence of formal specifications eliminates one of the main weaknesses in V&V process: the use of vague or ambiguous requirements. Formal specifications can be used to verify the system, and to define precisely its boundaries (one of the classical problems in KBS validation). In addition to describing the I/O behaviour, formal specifications define how this behaviour can be achieved. This brings more information for V&V, because a formal specification provides, to some extent, elements of the KBS structure. These elements can be effectively used in the V&V process, which is no longer forced to be a "black box" testing process.

The construction of conceptual models and their translation into formal specifications is neither an easy nor currently a complete task. Some typical KBS tasks remain elusive to formal specification, and the specification construction can be a costly process. Topics for immediately-necessary future work include:

- making concrete the proposals for using formal specifications with the various V&V techniques, and evaluating the effectiveness of doing so;
- investigating the utility and practicality of different approaches to formal specification (e.g. "mini-models" versus complete specification);

- development of development processes in which all of the techniques can be applied and managed effectively;

These issues remain open in knowledge engineering, but they should not be obstacles to the development and use of formal specifications for KBS construction. Their benefits will improve the quality of KBS in the near future.

References

Batarekh, A., Preece, A. D., Bennett, A., and Grogono, P. (1991). Specifying an expert system. *Expert Systems with Applications*, 2(4):285–303.

Bellman, K. L. (1990). The modeling issues inherent in testing and evaluating knowledge-based systems. *Expert Systems with Applications*, 1(3):199–215.

Buchanan, B. G. and Shortliffe, E. H. (1984). The problem of evaluation. In Buchanan, B. G. and Shortliffe, E. H., editors, *Rule-Based Expert Systems: the MYCIN Experiments of the Stanford Heuristic Programming Project*, chapter 30, pages 571–588. Addison-Wesley, Reading MA.

Chandrasekaran, B. (1983). On evaluating AI systems for medical diagnosis. *AI Magazine*, 4(2):34–37.

Fensel, D. and van Harmelen, F. (1994). A comparison of languages which operationalise and formalise kads models of expertise. *Knowledge Engineering Review*, 9(2):105–146.

Ghezzi, C., Jazayeri, M., and Mandrioli, D. (1991). *Fundamentals of Software Engineering*. Prentice Hall, New York.

Ginsberg, A. (1988). Knowledge-base reduction: A new approach to checking knowledge bases for inconsistency & redundancy. In *Proc. 7th National Conference on Artificial Intelligence (AAAI 88)*, volume 2, pages 585–589.

Haugh, J. (1988). The application of formal specification techniques to knowledge-based system development. In *UK IT 88 Conference Publication*, pages 95–98, London. University College, Information Engineering Directorate.

Kemmerer, R. A. (1985). Testing formal specifications to detect design errors. *IEEE Transactions on Software Engineering*, 11(1):32–43.

Krause, P. J., Byers, P., Hajnal, S., and Fox, J. (1990). The use of object-oriented process specification for the verification and validation of decision support systems. In Laurent, J.-P. and Ayel, M., editors, *Verification, Validation and Test of KBS*. John Wiley & Sons, New York.

Laurent, J.-P. (1992). Proposals for a valid terminology in kbs validation. In Neumann, B., editor, *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI 92)*, pages 829–834, New York. John Wiley & Sons.

Meseguer, P. (1992). Incremental verification of rule-based expert systems. In Neumann, B., editor, *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI 92)*, New York. John Wiley & Sons.

Meseguer, P. and Verdaguer, A. (1993). Verification of multi-level rule-based expert systems: Theory and practice. *International Journal of Expert Systems: Research and Applications*, 6(2):163–192.

Miller, L. A. (1990). Dynamic testing of knowledge bases using the heuristic testing approach. *Expert Systems with Applications*, 1(3):249–269.

O'Keefe, R. M., Balci, O., and Smith, E. P. (1987). Validating expert system performance. *IEEE Expert*, 2(4):81–90.

Plant, R. T. and Gold, D. (1990). Increasing expert system reliability through the use of a formal specification. In Culbert, C., editor, *AAAI-90 Workshop on Knowledge Based Systems Verification, Validation and Testing*. AAAI.

Potter, B., Sinclair, J., and Till, D. (1991). *An Introduction to Formal Specification and Z*. Prentice-Hall, New York.

Preece, A. D. (1990). Towards a methodology for evaluating expert systems. *Expert Systems*, 7(4):215–223.

Preece, A. D., Shinghal, R., and Batarekh, A. (1992). Principles and practice in verifying rule-based systems. *Knowledge Engineering Review*, 7(2):115–141.

Rushby, J. and Crow, J. (1990). Evaluation of an expert system for fault detection, isolation, and recovery in the manned maneuvering unit. NASA Contractor Report CR-187466, SRI International, Menlo Park CA.

Suwa, M., Scott, A. C., and Shortliffe, E. H. (1982). An approach to verifying completeness and consistency in a rule-based expert system. *AI Magazine*, 3(4):16–21.

Treur, J. and Wetter, T., editors (1993). *Formal Specification of Complex Reasoning Systems.* Ellis-Horwood, Chichester.

Turing, A. (1950). Computing machinery and intelligence. *Mind*, 59:236–248.

van Harmelen, F. and Aben, M. (1995). Applying rule-base anomalies to KADS inference structures. In Gamble, R. and Landauer, C., editors, *Working Notes from IJCAI-95 Workshop on Verification and Validation of Knowledge-Based Systems.*

Wielinga, B. J., Schreiber, A. T., and Breuker, J. A. (1992). KADS: a modelling approach to knowledge engineering. *Knowledge Acquisition*, 4(1):5–54.

Zualkernan, I. A., Tsai, W.-T., and Kirani, S. (1992). Testing expert systems using conventional techniques. In *Proceedings of 16th Annual Computer Software and Applications Conference*, pages 320–325.