Two Aspects of the Validation and Verification of Knowledge-Based Systems

Trevor Bench-Capon, Frans Coenen, Hyacinth Nwana, Ray Paton, and Michael Shave, University of Liverpool

ONE OF OUR PROJECTS DEVELOPED A METHODOLOGY FOR ANALYZING A DOMAIN PRIOR TO CONSTRUCTING A SYSTEM. THE OTHER PROJECT COMPRISES A SET OF TOOLS THAT SUPPORT SYSTEM DEVELOPMENT THROUGH INCREASINGLY FORMAL REPRESENTATIONS.

DEVELOPERS OF KNOWLEDGE-based systems face a number of practical problems when trying to move this technology into routine use. Two research projects at the University of Liverpool have examined problems related to system verification and validation, and the associated issue of maintenance.

One source of problems is that the domain analysis undertaken before actually constructing the knowledge base is often unsatisfactory. Typically, it is carried out at too shallow a level and so fails to reflect the true structure of the domain; or it fails to recognize the boundaries of the embodied expertise, and the bias that may be introduced by concentrating on a particular task within the domain. The Mekas (Methodology for Knowledge Analysis) project has developed a method to give knowledge engineers a thorough characterization of the domain, that is, a full description of the ontology, structure, functions, and theories that underpin the domain. This provides a coherent framework within which a knowledge base can be more readily identified with key aspects of the domain, aiding verification and validation. The framework also allows the addition of new or revised knowledge, thus aiding the maintenance and extension of the knowledge base.

The second project, known as MAKE (Maintenance Assistance for Knowledge Engineers), pursued in collaboration with International Computers Ltd. (ICL) and British Coal, was aimed at the process of constructing and revising the knowledge base. Two things are required: a disciplined and methodical approach to authoring the knowledge base, and tools to support and reinforce this approach. When AI toolkits first appeared, they were often described as "power tools for programmers." Admirable as many of these tools are, they remain directed at programmers, and writing a knowledge base becomes rather like programming. Proper construction of a knowledge base should instead be seen as a process of knowledge representation, which requires a different approach and different tools. Central to our philosophy is that validation and verification should always be carried out at a level higher than the level of executable code; we are engaged in a process of knowledge representation and modeling, not programming.

Mekas

It is essential to capture all pertinent aspects of the expert's view of any real-world problem. This is a nontrivial problem, and can only be achieved by the in-depth analysis that we describe as the characterization stage of the investigation. A structured approach based on analysis of the domain is required at this stage, rather

This is a revised version of an article that appeared on pp. 125-132 of the *Proceedings of EuroVAV '91*, published in 1991 by Logica Cambridge Ltd., Cambridge, England.

than one based on the desired artifact or model of performance.[1] Without such analysis, the software's limitations (implicit in the model that has been constructed) will not be fully understood, and a correct view of the domain's objects and structure will not be achieved. The subsequent design, implementation, and maintenance of the knowledge-based system will remain incomplete. An adequate and comprehensive domain characterization is essential if the products of knowledge acquisition are not to remain shallow and fragmentary.

The Mekas project has sought to address several key issues, including

(1) A theory of the nature of domains and the modeling processes associated with knowledge acquisition. This is needed so that characterization can be clearly understood and documented.
(2) A means by which the knowledge engineer is provided with guiding principles for navigating the domain and making sense of the mass of information obtained. This avoids fudging issues such as dealing with unknowns and gaps in knowledge during the early stages of the process.
(3) An approach that links informal models to the real world, formal models to informal models, and design and implementation concerns to formal models. At each transition, the limitations brought about by abstraction are made explicit so that the scope of the final specification can be mapped back, through formal and informal models, to the real-world characterization. As such, the validity of the informal models is determined with reference to the real world and the expert's models, whereas the validity of the formal models is assessed according to its intrinsic properties.

Key aspects of domains. To satisfy these (and other) issues, we had to recognize key aspects of domain knowledge. Human knowledge about domains is so complex that it is essential to identify the domain's fundamental characteristics before developing the knowledge-based system. The purpose of Mekas is to give structure to a knowledge analysis stage, which characterizes a domain by organizing the information acquired from a variety of sources into a coherent and unambiguous whole. This is a difficult problem and a major hurdle to the development of knowledge-based systems. The purpose of knowledge analysis is to probe the underlying nature of the domain and investigate how experts think about it. In our approach, called SAAGS (for Specification, Anticipation, Acquisition, Generation, Specification), the products of analysis capture the domain's conceptual richness through a series of iterative analysis-modeling cycles.

Using this approach, we can characterize domains in terms of certain broad (nondisjoint) sets of features:

AN ADEQUATE AND COMPREHENSIVE DOMAIN CHARACTERIZATION IS ESSENTIAL IF THE PRODUCTS OF KNOWLEDGE ACQUISITION ARE NOT TO REMAIN SHALLOW AND FRAGMENTARY.

- Theory: the conceptual framework used to construct and maintain a domain.
- Purpose: the problems addressed by the domain in terms of their solution.
- Metaphor: the language used to maintain a domain, especially in global terms.
- Metatheoretical constraints: fundamental concepts related to time, causality, category, and ontology.
- Relations to other domains.
- History of the domain.
- Structure: the parts, relations, and organization of the domain.

These seven features provide a top-level objective, which forms the basis of all subsequent analysis. This yields analytical outcomes with both breadth and depth. The consequence of a broad description is that knowledge engineers come to share a wider perspective of the domain with the experts; the depth captures the deeper knowledge, in terms of the ontology, structure, and theory of the domain, necessary for building second-generation knowledge-based systems.

SAAGS. SAAGS is a four-stage cyclical modeling process. It receives a loose specification from a client as input and ultimately produces a comprehensive specification that provides the domain's characterization. The overall goal is to detail the domain's structure and purpose in the context of its evolving theory. The principal stages are

(1) *Specification* leading ultimately to the production of a domain's characterization. The seven top-level features must be steadily accounted for in the successive iterations of this stage, which must also include relevant models and descriptive details such as the epistemic (domain) boundaries yielded by the analysis.
(2) *Anticipation* of the nature of the domain. The top-level features allow the knowledge engineer some anticipation of the breadth of the characteristics of the domain. Anticipations provide a way of critically testing the knowledge engineer's evolving understanding. The anticipation of top-level characteristics will guide the domain-based knowledge acquisition process.
(3) *Acquisition* of knowledge. This includes knowledge elicitation from experts and other sources (such as books and manuals). The outcomes from the anticipation stage are used to structure acquisition in a way that relates to the emerging nature of the domain.
(4) *Generation* of models, including the synthesis of outputs from the acquisition stage into a collection of models. These outputs drive the analysis forward and provide the explicit means for confirming, refuting, or elaborating anticipations and thus prepare for an improvement of the specification.

The cycle now repeats with a new specification stage, in which all the modeling information is accounted for, together with details of the emerging (hypothetical) epistemic boundaries of the domain. Cycles will continue until both the domain expert and the knowledge engineer are satisfied that the specification adequately characterizes the domain. This methodology, with its cyclical nature and emphasis on producing a set of interrelated models describing the domain, provides an essential analysis against which systems subsequently developed in the domain can be validated and verified.

Mekas and validation. The application of the Mekas approach is pertinent to the issues of verification, validation, and maintenance for three reasons. First, the process of domain analysis and modeling is structured so that gaps and inconsistencies in the knowledge are made explicit. Also, SAAGS requires a comprehensive documentation of the domain in its specification, which helps reduce the costs of system maintenance, extensibility, and renovation. Finally, the relationships between models can be made explicit; that is, real world $\rightarrow$ informal (referential) models $\rightarrow$ formal (denotational) model $\rightarrow$ artifact. As such, the epistemic boundaries associated with each, in relation to the real world, domain, and artifact, are clarified.

The development of informal models is crucial in characterizing the domain because they provide a way to bridge from the complexities of the real-world domain to the necessary simplifications of formal models. No model can fully account for the real world (if it did, it would not be a model), so any model will have gaps. If we do not make these gaps explicit, we will always have problems that will undermine attempts to verify and validate the formal models. The methodology just described seeks to generate the required models in a structured way, driven by analysis of the domain. This ensures that the models are sensitive to the underlying nature of the domain, rather than determined by a particular design method, predetermined representation formalism, or predefined task library.

MAKE

The MAKE project addressed the later stage in knowledge-based system development, which centers on the actual construction of the knowledge base. Its prime focus was the construction of maintainable systems, particularly in domains based on regulations that are particularly vulnerable to change. Maintenance issues cannot, however, be considered in isolation from validation and verification: Maintenance involves the detection of flaws in the knowledge base, and any changes that are made must themselves be validated and verified. Thus many of the techniques developed for maintenance apply equally to validation and verification—and vice versa—and the maintenance environment developed on the MAKE project can also be used to validate and verify a system under construction.

Validation, verification, and maintenance should not be considered only when a system has been produced; rather, they must inform the whole development process. This is supported by the methodologies for designing and building conventional systems. These methodologies have as their primary aim the construction of better systems, that is, systems in which we can have a high degree of confidence, and systems that can be more readily maintained.

A SERIES OF MODELS MOVE FROM THE ORIGINAL SOURCE MATERIAL THROUGH INCREASINGLY FORMAL MODELS, UNTIL AN EXECUTABLE REPRESENTATION IS REACHED.

The MAKE project therefore prescribes a development methodology that we believe will produce systems that are more capable of validation, verification, and maintenance. The starting point for development should be a characterized domain such as that produced by Mekas. Development takes place in the MAKE Authoring and Development Environment (Maude), which supports system production via a series of models that move from the original source material through increasingly formal models, until an executable representation is reached. The output from each stage of the development cycle should reflect the structure of the material from the previous phase. Maude provides a suite of authoring tools, including textual and graphical browsers; a way to link individual rules to the source material through the different development stages; and tools to aid the verification and validation of the knowledge-based system during development, on completion of development, and after a maintenance session. Verification and validation is performed statically by comparing the various models, and dynamically by animating the executable representation so that consequences of rules and assertions can be explored.

Maude. The design of Maude revolves around three base windows. The Kant window is used to form and link various levels of structures. The Maude window is the interface to the compiler and the various MAKE tools. The Mappe (MAKE Application Environment) window provides the user view of the application. This allows the developer to execute the representation so that test cases can be run and the consequences of particular assertions explored.

The methodology uses a development environment based on Kant (Knowledge Analysis Tool), a hypertext-like tool built to help develop a knowledge-based decision support system for assessing Social Security benefit claims.[2]

System development using Maude consists of using Kant to construct four levels of structure. At the first level, source documents are imported into the Kant system in their original format, ready for analysis. In MAKE's target application (support for processing insurance claims made on British Coal), these sources comprised the legislation and other supporting material. In a domain that lacks this rich textual basis, a Mekas characterization would form the source documents. The role of Kant is to support the copying and pasting of words, phrases, and so on, from these documents into other structures, and the definition and maintenance of links between these sources and the new structures. The links are vital: They enable the direct comparison of the various models with one another.

Freestyle Kant structures, comprising the second level, can best be described as structured English notes. Kant readily supports the structuring of such notes (called nodes in Kant) through the use of child and sibling relations, and through separable fields within a node. The author is free to construct any number of such structures, with a variety of contents. The aim of this stage is to select and organize the material on which the proposed knowledge-based system will be based: In particular, it is necessary here to identify the entities relevant to the domain, and for each entity the attributes it can possess and the values these attributes can take. This will form the vocabulary to be used in the formal model of the domain. Here too, relations between these features are identified, thus forming the basis of the rules of the final system.

MIR (MAKE Intermediate Representation) structures are the third level. These form the intermediate representation, which uses yet more Kant structures, but are constructed according to a defined syntax.[3] MIR structures use this formal representation language to specify the entities, their attributes, and allowable values; the rules governing the relationships between them; and the topics and questions to be used in dialogues with the user. The role of this intermediate representation is crucial: It is formal enough to allow for the unambiguous representation of knowledge, and so can permit manipulation to support the various MAKE tools, but it remains uninfluenced by concerns specific to a target executable representation language.

Compiled MIR, the final level, is an executable representation of the MIR. The MAKE inference engine interprets the compiled MIR and forms the dynamic, instantiated user application. In the MAKE project we used a clausal form of compiled MIR, and translation between it and the MAKE inference engine is completely automatic. Alternative compilers could be written to target different formalisms, if the application were to be executed in, for example, Nexpert. A key idea is to confine authoring and maintenance tasks to the intermediate representation represented by the MIR structures and the preceding structures and sources, so that Maude can be used as a generalized development environment irrespective of the ultimate executable formalism.

The passage through these levels of formality is an iterative process, both because requirements will change over time and so source material will be superseded, and because the transitions from one stage to the next will not be right the first time. What is important, however, is that every item in a derived structure can be identified with the relevant item or items in the previous structure, so that its validity can be determined.

The MAKE tools

Some of the MAKE tools are particularly relevant to validation and verification.[4-5] The central tool is the rule map, which provides a graphical view of the intermediate representation and access to several other MAKE tools. It is a left-to-right directed graph showing the relationship between attributes and rules in the rule base. The user can scroll up and down the rule base from the root attributes to the leaf attributes, inspecting any desired attribute or rule along the way. This gives the maintenance engineer a clear view of the rules in the knowledge base and the attributes they use. By following a path through the rule map, it is possible to determine the leaf attributes and propositions into which a root attribute ultimately unfolds, and vice versa.

EVERY ITEM IN A DERIVED STRUCTURE CAN BE IDENTIFIED WITH THE RELEVANT ITEM OR ITEMS IN THE PREVIOUS STRUCTURE, SO THAT ITS VALIDITY CAN BE DETERMINED.

Options are provided to allow the user to interrogate the rule map to display a rule and the clauses in which an attribute appears. Clicking on a rule node displays the structures from which the rule was derived, so that the links through the various layers of model can be followed and faulty transitions detected.

The project also specifies a proposition/clause rule map to give an alternative view of the rules, showing in fine detail how values of attributes affect one another, providing a fine-grain perspective on the rules.

The rule map also gives users a number of structural tools that test for the correct formation of the rule base. Here we can identify redundant rules, which make no contribution to the overall goal; dead-end rules, which fail to lead to any assertable leaf proposition; and subsumed rules, whose effect is duplicated by another broader rule. An inconsistency tool identifies groups of leaf attributes that would, if asserted simultaneously, lead to a contradiction. The non-cotenable propositions can then be examined to see whether they are in fact incapable of holding together.

Provenance and jeopardy tools. While the rule map provides excellent access to the intermediate representation (a level of abstraction suitable for the system developer), we may need to go further back through our models. This in turn requires tools that exploit the links between structures created during development. Two tools support these transitions.

The provenance tool. The provenance tool essentially follows the links back from executable representation through the intermediate representation and the freestyle structures to the source. When running test cases, we may find that the system behaves in some unexpected manner. Having identified the particular executable rule that is the source of the problem, we can use the provenance tool to find the structures from which it derived. If the transitions are correct, however, the fault must lie in the original source; thus, there was a mischaracterization in the analysis phase. If this happens, the provenance tool can be used to identify all the parts of the source material from which the problem clause derived. These can then be shown to the domain expert who, given these specific items and the identified problem, should be able to detect what is wrong, and the source can then be corrected accordingly. The provenance tool also records the history of the item: when it was created, who created it, and any comments. This can provide valuable help in assessing the correctness of a transition.

The problem at this point is that changing the source may have invalidated other aspects of the representation. It is therefore necessary to identify all the parts of the other structures that have been jeopardized by the change. This identification is carried out using the jeopardy tool.

The jeopardy tool. The jeopardy tool works by following links forward. Depending on the level at which the change was made, different aspects of the representation can be identified for checking:

- If the change in the source material necessitates a revision of the type hierarchy, the jeopardy tool is used to identify those classes, attributes, and possible values that were founded on the changed source material. Then the appropriate modifications can be made.
- If a change is made to the type hierarchy defining the vocabulary of the domain, certain rules may become invalidated. Thus if an attribute is removed, or the possible values that it can take are modified, the rules that use that attribute must be identified, examined, and if necessary, modified.
- If a class is removed from the type hierarchy, any attributes introduced into the type hierarchy through that class will no longer be available. These attributes must be identified and reintroduced into the hierarchy, either through the super class of the removed class or through one or more of the subclasses, as appropriate.
- Sometimes the source change does not alter the type hierarchy. Here the jeopardy tool is used to identify all the rules in the intermediate representation that are linked to an altered source so that they can be reevaluated.
- A change to a rule may threaten those rules that call or are called by the modified rule. The jeopardy tool identifies the subset of the rules that must be reexamined after a rule has been modified.

Both the jeopardy and provenance tools use the links that the knowledge engineer created during system development. This is an essential part of the Maude methodology.

Dynamic tools. Although we believe that inspection and modification should be carried out at a level above that of the executable code, we must still be able to execute the representation to ensure that the behavior is correct. Facilities to execute representations are integrated into the Maude environment, and supplemented by tools that help to determine the source of any unexpected behavior.

The rule base animation tool. Test cases are typically executed as a whole. Where there is a well-founded set of prototypical cases associated with a set of expected results, this can provide good confidence in the rule base. Sometimes, however, we may wish to test the rule base in a more exploratory way. For this reason the Maude environment includes a tool for constructing test cases interactively.[6]

The display of this tool is similar to the rule map, but we can also mouse-click on nodes to assert values for the attributes represented by those nodes. Once one or more values have been asserted, inference can be invoked and the effect of the assertions shown on the display. This in turn can suggest what other assertions will form an interesting test case. The interactive, menu-driven nature of this process supports the rapid identification of critical combinations of facts. If the behavior is not what we expected, we can find the cause using the next three tools described.

THE PRIMARY FOCUS OF VERIFICATION AND VALIDATION MUST BE ON THE MODELS, AND THE TRANSITIONS BETWEEN THEM, RATHER THAN SIMPLY ON THE EXECUTABLE REPRESENTATION.

The justification and consequence browsers. During execution of a case, the inference engine keeps track of how each object/slot obtained its value. It does this by maintaining justifications in terms of the rules, user input, or predefined sources of values. The justification browser will then allow the maintenance engineer to examine the justification links for any given object/slot. The consequence browser is similar to the justification browser but allows the user to examine the consequences of a particular object/slot value.

The "why not" tool. The why not tool lets the maintenance engineer determine why an attribute of an instance has not assumed an expected value. Of course, the tool can also be used in reverse, to ask why an attribute for a specific instance has assumed a certain value. The tool requires the user to select an attribute of an instance and then offers a list of rules affecting this attribute. The user can then select a rule that he/she thinks should have led to the expected value, and an instance for each variable used by the selected rule. Maude will then test each tail proposition in each clause of the rule and write a report stating which is true and which is false. Inspection of the list allows the user to determine "why not" in terms of either missing conditions to the rule or an error in the rule. In the first case, the why not tool may be used again to discover why the input is missing.

TOGETHER, THESE PROJECTS reflect our view that validation and verification cannot be addressed in isolation. Rather, these issues must influence every stage of system development and construction. Moreover, Mekas and MAKE reflect our conception of knowledge-based-system development as a process of modeling and knowledge representation: The primary focus of verification and validation must be on the models, and the transitions between them, rather than simply on the executable representation.

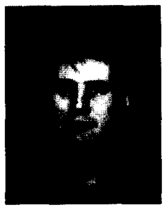
Acknowledgments

Mekas was sponsored by Shell Research (Thornton) and Unilever Research (Port Sunlight). The MAKE project was supported by the Information Engineering Directorate of the United Kingdom Department of Trade and Industry and the United Kingdom Science and Engineering Research Council. The project collaborators are ICL, the University of Liverpool, and British Coal. The views expressed in this article are those of the authors and may not necessarily be shared by the other project collaborators.

References

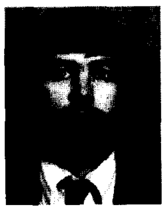
1. R.C. Paton et al., "From Real-World Problems to Domain Characterisations," *Proc. Fifth European Knowledge Acquisition Workshop*, GMD-Studien Nr. 211, GMD, Sankt-Augustin, Germany, 1992, pp. 235-236.
2. G.E. Storrs and C.P. Burton, "Kant, A Knowledge Analysis Tool," *ICL Technical J.*, Vol. 6, No. 3, May 1989, pp. 572-581.
3. T.J.M. Bench-Capon and J.M. Forder, "Knowledge Representation for Legal Applications," in *Knowledge-Based Systems and Legal Applications*, T.J.M. Bench-Capon, ed., Academic Press, London, 1991, pp. 245-264.
4. T.J.M. Bench-Capon and F.P. Coenen, "Practical Application of KBS to Law: The Crucial Role of Maintenance," in *Legal Knowledge-Based Systems, Aims for Research and Development*, C. van Noortwijk, A.H.J. Schmidt, and R.G.F. Winkels, eds., Koninklijke Vermande BV, Lelystadt, Netherlands, 1991, pp. 5-17.
5. F. Coenen and T.B.C. Bench-Capon, "A Graphical Interactive Tool for KBS Maintenance," in *Database and Expert Systems Applications (DEXA '91)*, D. Karagiannis, ed., Springer-Verlag, Vienna, 1991, pp. 166-171.
6. F. Coenen and T.J.M. Bench-Capon, 1992, "KBS Maintenance Validation Using Simulation," in *Application of Artificial Intelligence in Engineering VII*, D.E. Grierson, G. Rzevski, and R.A. Adey, eds., Computational Mechanics Publications/Elsevier Applied Science, New York, 1992, pp. 215-228.

Trevor Bench-Capon is a senior lecturer in computer science at the University of Liverpool. His chief research interests are in knowledge representation and knowledge-based systems, particularly as applied to law. He has a PhD in philosophy from St. John's College, Oxford.

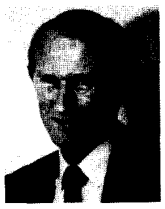


Frans Coenen is a research associate at the University of Liverpool. His research interests include the maintenance, verification, and validation of rule bases, computes-supported cooperative work, and the application of geographic information systems and AI technology (knowledge-based systems and neural nets) to the maritime industry. His current work focuses on techniques to reduce the overheads associated with group working. He has a PhD in computer science from the Liverpool John Moores University.

Hyacinth S. Nwana is a lecturer in computer science at the University of Keele. When the research for this article was performed, he was a postdoctoral research fellow in computer science at the University of Liverpool. His main interests are in intelligent tutoring/expert systems, knowledge acquisition, and integrating connectionist and symbolic architectures. He holds a PhD in artificial intelligence from Aston University. He is a member of the Society of Arti-ficial Intelligence and Simulation of Behaviour.



Ray Paton is a lecturer in computer science at the University of Liverpool. His research interests include domain analysis, computer-based learning, philosophy of science, and biologically motivated computing. He is vice-chair and coeditor of the *ACM SIGBio*, and a member of ACM, the Institute of Biology, and the European Society for the Study of Cognitive Systems. He has a BEd in biology and education from the University of Liverpool and a PhD in cognitive science from the University of Leeds.



Michael Shave was appointed the first professor of computer science at the University of Liverpool in 1982. His research has centered on databases, knowledge analysis, and knowledge representation. Currently he is involved with a project developing methods and tools for intelligent communications software. He is honorary secretary of the Conference of Professors of Computer Science.

Bench-Capon, Coenen, Paton, and Shave can be reached at the Dept. of Computer Science, University of Liverpool, Chadwick Building, Liverpool L69 3BX, United Kingdom; e-mail, tbc@compsci.liverpool.ac.uk

Nwana can be reached at the Dept. of Computer Science, University of Keele, Staffordshire, England; e-mail, nwanahs@cs.keele.ac.uk