Making Knowledge Engineering Technology Work

V. Richard Benjamins*
University of Amsterdam
SWI, The Netherlands
richard@iiia.csic.es

Dieter Fensel
University of Karlsruhe
AIFB, Germany
dfe@aifb.uni-karlsruhe.de

Christine Pierret-Golbreich
University of Paris-Sud
LRI, France
pierret@lri.fr

Enrico Motta
The Open University
KMI, United Kingdom
E.Motta@open.ac.uk

Rudi Studer, AIFB
Bob Wielinga, SWI
Marie-Christine Rousset, LRI

Abstract

The rapid-prototyping approach of the early 1980's failed to deliver high-quality knowledge-based systems. As a reaction, in the early 90's, there has been a large activity in the knowledge engineering community to define methodologies for principled knowledge-based system development. These methodologies succeeded in organising the development process as a set of intermediate models of the system to be built, to end in the final implementation. Feedback from industries that use these methodologies in practice reveals, however, that building a high-quality implemented system remains a difficult and error-prone process. In this paper, we outline an approach for assuring the transition of a conceptual model into a final implementation which satisfies particular quality criteria. The approach is based on identifying and supporting adequate transition paths and activities on models in order to achieve particular quality criteria.

1. Introduction and motivation

Modern knowledge engineering (KE) methodologies such as CommonKADS [20], VITAL [10], MIKE [3], TASK [16] are successful in providing structure to the development process of knowledge-based systems (KBSs) by identifying intermediate models and defining the languages and organisation of these models. These methodologies have been originally set up in response to the industrial need for a rigorous and systematic KBS development methodology, as opposed to the rapid-prototyping practice that was popular in the eighties. Such methodologies have been successful in the sense that conceptual models are now the European industrial standard for describing high-level KBS specifications. However, feedback from industries indicates that moving from a high-level conceptual model to a high-quality implementation is still a difficult and error-prone process. High-quality means that the KBS exhibits certain quality criteria, such as for example, reliability and maintainability. The main reason for these problems is that, although the structure of the intermediate *models* is now well understood, there is not yet adequate provision in the form of methods, guidelines, and tools to support the *transitions* between the models.

The aim of this paper is to outline an approach for assuring the transition from a conceptual model to a final implementation which satisfies particular quality criteria. Rather than inventing a new methodology, we build on existing work, and focus on the issues that remain to be solved. In Section 2, we sketch a typical industrial need, and in Section 3, we outline the approach. Section 4 discusses the kind of tools needed to support the approach, and in Section 5, we present conclusions.

2. A typical industrial need

Modern KBS development methodologies provide a clear structure of the development process by identifying intermediate models, which enables industries to better monitor and control the development process. One of the key notions is that the development process starts with the construction of a conceptual model of the KBS to be built. Based on various European joint projects between universities and industries, several tools have been developed for building conceptual models (e.g., KADSTOOL, OpenKADS, the VITAL workbench, MIKE, VOID, the CommonKADS workbench, PC-Pack, etc.), and some of

*also at Spanish Council for Scientific Research (CSIC), IIIA, Spain.

these are widely in use.

However, industries still have difficulties with applying these methodologies in their full range to develop KBSs. A common problem concerns the transformation of the high-level conceptual model into a particular implementation platform, such as for example an object-oriented environment or rules (e.g., Aion-ds, Nexpert, Clips). The reason for this problem is that the current KE approaches do not provide guidelines or tools to support this transformation, and lack rigour and methodology needed for dealing with KBSs of realistic size. There are several intermediate models (see Section 3.1), and it is not clear in what situation to include a particular model. Moreover, it is difficult to maintain consistency between the models, and between the models and the implementation. However, see [22], for a modest success story of applying such a methodology in its full range.

Knowledge engineering methodologies will only boost industrial productiveness if they (i) support the whole process of KBS development, (ii) tightly relate the different models to each other, (iii) enable optimal reuse of enterprise knowledge, and (iv) deliver high-quality KBSs (e.g., maintainable, correct, etc.). To achieve these goals, we need to provide developers with correct and efficient transition methods and by providing software support for managing the interactions between different models.

3. Approach

The approach is based on three different dimensions: (1) models of the functionality of the KBS; (2) activities on the models; and (3) quality criteria of KBSs.

3.1. Models

Methodologies for KBS development are based on multiple models which bridge the gap between an abstract conceptual model and an implementation. The transition process between the models is structure-preserving, which means that content and structure of the knowledge in the conceptual model are preserved throughout the intermediate models to the final implementation. The motivation is that, if two models have a corresponding structure, the transition step between them becomes easier and semi-automatic transformations are possible. This facilitates not only validation and verification of the models, but also their modifications. The nature and structure of the different models have been brought to light in past research in Knowledge Engineering (e.g., CommonKADS [20], VITAL [21], MIKE [3], TASK [16]), and include the following:

A **conceptual model** (CM) is dedicated to capture the expertise in an informal, but structured way. A CM describes the different types and roles of knowledge in reasoning tasks. It facilitates initial KBS specification and human's understanding of the KBS [19]. A **formal model** (FM) encodes knowledge in a symbolic formalism with a mathematically sound basis and a declarative semantics. It allows to eliminate ambiguities and inconsistencies from the CM, and enables formal verification and validation [11, 17]. A **design model** (DM) is concerned with, among others, an efficient realisation of the CM and/or FM on a computer. It specifies, among others, the data structures, the algorithms and the architecture of the target application. A DM also records the rationale for the choices made [15, 23].

3.2. Activities on models

In general, we can perform various activities on the models of the model set mentioned above, as illustrated in Figure 1. Models can be transformed into each other, they can be verified against each other [18], they can be revised and modified, or they can be validated against some external requirements. It is also possible to trace the changes made in one model to an other model.

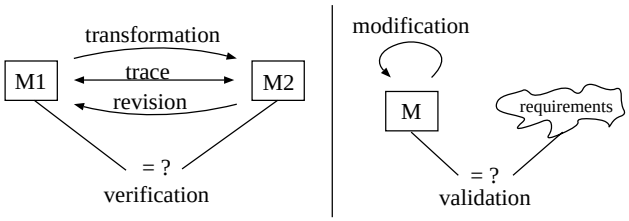

Figure 1. The different activities.

Transformation In the transformation step, the content of one model is transformed into another model, for example, a CM in a DM.

Validation and verification In V&V, we evaluate models. In validation we check whether a model complies with some external (non-formal) requirements (e.g., user needs). Validation works on one model (see right part of Figure 1). Verification needs two models (left part of figure). Its goal is to show that some specification (M2) satisfies its requirements (derived from M1). The output of both activities is an evaluation containing statements about the model. In a multiple model context, V&V can be stated in terms of the properties that the different models must have with respect to each other.

Revision and modification Revision and modification update a model, based on the output of the V&V activities. Modification operates on one model, and modifies it. Revision takes two models and revises the source model (see Figure 1).

Tracing Tracing is performed between several models, and is concerned with finding out the consequences of changes in one model for the other model(s).

Activities are not only in forward direction (e.g., the transformation of a model into a next model or implementation), but also in backward direction (e.g., verification of a model/implementation against another model, or revision of a model/implementation based on the outcome of a verification activity).

3.3. Quality criteria

Quality criteria of a KBS are the properties a KBS should possess. In case of large and complex KBSs, such quality criteria become increasingly important. We distinguish the following quality criteria.

Applicability The knowledge-based system can be applied and behaves as expected.

Maintainability Changes and updates of the KBS can be easily traced through the different models to prevent unexpected and undesired global side effects[1].

Reliability, correctness The KBS delivers reliable and correct solutions.

Consistency The different models of the KBS are consistent with each other, and are internally consistent.

3.4. Transition paths

There are two main motivations underlying our approach in order to make KE technology work in practice. First, we want to identify appropriate transition paths in response to particular quality criteria, and to provide corresponding tool support. A transition path defines the different models to include in the transformation from a conceptual model to a final implementation. We identified four transition paths that make sense in different situations, as illustrated in Figure 2. We can go directly from a conceptual model to an implementation, we can include an intermediate design model or a formal model, or we can include both a formal and design model.

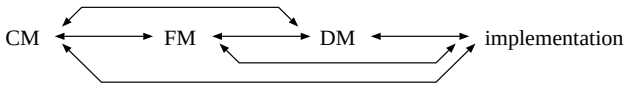

Figure 2. Different transition paths between a conceptual model and an implementation. CM = conceptual model, FM = formal model and DM = design model.

[1]Maintainability is also an important feature of each individual model. This is, however, already included in the existing KE methodologies.

Second, we want to identify the simplest, but still adequate, transition paths as possible for different situations. The reason for this is that industries are not willing to spend energy and time on activities that are not essential. For example, consider the inclusion of a formal model. Formal models are notoriously difficult to build [9], but have the advantage that they can guarantee the correctness of the future system [7]. Therefore a company will only be ready to spend money on a formal model if the application cannot do without it (e.g., if faults have disastrous consequences).

3.5. Quality criteria, transition paths and models

In our approach, we relate quality criteria to *activities*, which, on their turn, are related to certain `transition paths`. Table 1 summarises this.

Applicability If we have to guarantee the applicability of the KBS, it is important to carry out the *validation* activity on the final implementation. Applicability is a kind of minimum requirement for any KBS. The resulting transition path is a direct `transition from the conceptual model into the implementation`. This does, however, not mean that design decisions are unimportant here. What it does mean is that for such applications the design decisions are not likely to change over time, and that therefore one can rely on a fixed design environment, that need not be elaborated on in the transition process. Sometimes this is referred to as "explorative prototyping", because it is possible to have a quick idea of the power of the knowledge represented in the conceptual model.

Safety-critical If we are dealing with safety-critical applications, such as airplanes, chemical process industries, space missions, nuclear power plants, etc., reliability is an essential features, as well as correctness and consistency. To guarantee that the final KBS exhibits those quality criteria, formal *verification* is required [18] and a specific `step through an intermediate formal model appears in the transition path`.

Evolutionary systems KBSs that evolve considerably over time need to be easily maintainable. This means that the consequences of *modifications* and *revisions* of one model need to be clear for the other models. For example, when modifications are made in the conceptual model due to changing requirements, it should be clear how this change propagates to the design model and the implementation. On the other hand, when the implementation

or the design model is changed, for instance due to repairing a bug, the implications for the conceptual model should be clear (in user-understandable terms). Therefore an important activity is *tracing* between the different models. Experience shows that such evolutionary systems benefit much from an `explicit design model in the transition path` that captures the design decisions taken. If for some reason a design decision has to be revised, it is clear what else has to be updated.

Quality criteria can be combined (as in the last row of Table 1), which implies that their corresponding models have to be included in the transition path.

Notice that there can be different types of each model, and the type to include in the transition path depends on the specific aim. For instance, if we want to use a formal model for disambiguating the conceptual model, we require a FM with high *expressiveness* such as $(ML)^2$ [25], otherwise there is the risk that we cannot represent all knowledge in the conceptual model. On the other hand, if our aim is validation, then an *operational* FM such as KARL [12], would be more appropriate (limiting the expressive power of the FM). See also [13, 17, 11]. Analogously, we can have different types of DMs [23, 15].

4. Supporting the transition paths

Past experience has shown that good methodologies will not work in practice if they are not properly supported with software tools and guidelines. This holds also for current KE methodologies. Performing all transition steps through the intermediate models until the final implementation is a complex process, and it is practically impossible to manage a complex transition path without dedicated tool support.

A good KE methodology supports guidance and good documentation facilities and leads to reduction of effort in building KBSs. In our approach, these goals are achieved through the following[2]:

Guidelines During the transition process of one model into another, many decision have to be taken, in particular, because the models become increasingly more precise and they have a one-to-many relation to each other. If the source and destination model have an isomorphic structure (i.e., structure-preserving transitions) then concrete guidelines can be given. This is done for the transition between the conceptual and the formal model in the context of the CommonKADS methodology [2][3], where a set of 70 concrete guidelines is provided. Guidelines for the transition from a conceptual model into an implementation are described in [23].

Automatic translators It is also possible to build semi-automatic translators for the transition of one model into another. Such translators can be considered as computerised versions of the guidelines mentioned above. [24] describe a semi-automatic translator for two CommonKADS models: CML (conceptual) and $(ML)^2$ (formal). The VOID tool[4] contains several automatic translators between different representational languages (CML, Ontolingua, Express).

Reusable libraries Reuse of ready-made components is a promising way to facilitate the transition process. Instead of specifying or generating a model fragment from scratch, existing components can be retrieved from libraries. Since our approach involves different models (conceptual, formal, design) and executable code, libraries for each of these should be considered. Several libraries exist to (automatically) support the initial construction of the conceptual model [4, 8, 5]. A library of general formal fragments can be helpful in specifying the formal model [1]. In [6], we show how this formal library helps to formalise a diagnostic reasoner. Needless to say that there exist several libraries of executable code in various languages that can also serve our approach.

Apart from reusing model fragments, our approach aims also at identifying reusable components in the form of packages. Such a package consists of a successful path from a part of a generic conceptual model to its corresponding generic part in the implementation, through possible intermediate models (formal model, design model).

Verification support Proving that a formal model has certain properties is a time consuming and error-prone process. Therefore, we advocate the use of verification tools to automate as much as possible, and to bother the user only with the difficult issues. An interesting tool for such support is the Karlsruhe Interactive Verifier, KIV[5] [14] that enables the reuse of partial proofs.

Change-management systems When dealing with multiple models that are closely related to each other, it becomes very hard to keep track of the consequences of modifications in one model for the other models. In practice, many decisions that are taken during the transition between models are subject to revision in a later stage. For instance, in the design phase, conflicts between requirements may be detected which did not surface earlier. In case of a revision,

[2] Most current KE tools already provide good documentation facilities, therefore we will not elaborate on this.

[3] See http://www.swi.psy.uva.nl/usr/manfred/abstracts/Aben:92c.html.

[4] See http://www.swi.psy.uva.nl/projects/void/roadmap.html.

[5] See http://i11www.ira.uka.de/~kiv/.

Quality criterion	Activity	Transition path
applicability	validation	CM $\rightarrow$ implementation
maintainability	tracing	CM $\rightarrow$ DM $\rightarrow$ implementation
reliability, correctness, consistency	verification	CM $\rightarrow$ FM $\rightarrow$ implementation
reliability, maintainability	verification, tracing	CM $\rightarrow$ FM $\rightarrow$ DM $\rightarrow$ implementation

Table 1. The relations between quality criteria, activities, and transition paths.

the resulting modifications often must not only be made in a more specific model, but must also be reflected in earlier models, and must then be propagated in forward direction in order to avoid models getting out of sync. Furthermore, the decisions taken in the development process build on each other. Thus, if one decision will be revised later on, several others will also be invalidated, while others are not affected by a particular revision [15]. Additionally, we will address how decisions that are not affected by a revision can be "replayed" (maybe in a modified context) after the revision has taken place (a kind of reuse).

5. Conclusions

In this paper we argued that, although there exist good knowledge engineering methodologies for developing knowledge-based systems, this is not enough to make these methodologies work in practice. Feedback from industries tells us that it is still difficult to produce high-quality KBSs. In our research, we have become aware that this is caused by the lack of concrete guidelines and software support to apply the methodologies. The basic problem is that, although the structure of the KBS development process is well understood (as a set of intermediate models), the transition process between the models is less clear. The approach outlined in this paper can be seen a proposal to *complete* the current KE methodologies such as CommonKADS, VITAL, MIKE, and TASK by focusing on the transitions, and by identifying and developing appropriate tool support.

Ideally, we would want to combine the different existing KE methodologies into a coherent environment that takes the strong points of each of them. Seeds of such an integrated environment already exist at the various research institutes involved. A first step towards such an integrated environment could be achieved by making the different components (e.g., the various specification languages for the different models, the reusable libraries, the translators and the change-management systems) available through the WWW[6]. In fact, several efforts are currently undertaken to do so. However, their real integration is far from trivial.

[6] See http://www.swi.psy.uva.nl/, http://www.aifb.uni-karlsruhe.de/WBS/index.engl.html, http://www.lri.fr/equipes/iasi/, http://kmi.open.ac.uk/kmi.html.

Acknowledgement

This research was partially supported by the Netherlands Computer Science Research Foundation with financial support from the Netherlands Organisation for Scientific Research (NWO), and by the European Commission through a Marie Curie Research Grant (TMR).

References

[1] M. Aben. Formally specifying re-usable knowledge model components. *Knowledge Acquisition*, 5:119–141, 1993.

[2] M. Aben. *Formal Methods in Knowledge Engineering*. PhD thesis, University of Amsterdam, Amsterdam, 1995.

[3] J. Angele, D. Fensel, D. Landes, S. Neubert, and R. Studer. Model-based and incremental knowledge engineering: the MIKE approach. In J. Cuena, editor, *Knowledge Oriented Software Design, IFIP Transactions A-27*, Amsterdam, 1993. Elsevier.

[4] V. R. Benjamins. *Problem Solving Methods for Diagnosis*. PhD thesis, University of Amsterdam, Amsterdam, The Netherlands, 1993.

[5] V. R. Benjamins. Problem-solving methods for diagnosis and their role in knowledge acquisition. *International Journal of Expert Systems: Research and Applications*, 8(2):93–120, 1995.

[6] V. R. Benjamins and M. Aben. Structure-preserving KBS development through reusable libraries: a case-study in diagnosis. *International Journal of Human-Computer Studies*, to appear, 1997.

[7] J. P. Bowen and M. G. Hinchey. Ten commands of formal methods. *IEEE Computer*, 28(4):56–63, 1995.

[8] J. Breuker and W. van de Velde, editors. *CommonKADS Library for Expertise Modeling*. IOS Press, Amsterdam, The Netherlands, 1994.

[9] D. Craigen, S. Gerhart, and T. Ralston. An international survey of industrial applications of formal methods. Technical report, U.S. Department of Commerce, Technology administration, National Institute of Standards and Technology, Computer Systems Laboratory, Gaithersburg, MD 20899, USA, Mar. 1993.

[10] J. Domingue, E. Motta, and S. Watt. The emerging VITAL workbench. In A. et al., editor, *EKAW'93 Knowledge Acquisition for Knowledge-Based Systems. Lecture Notes in Artificial Intelligence, LNCS 723*, Berlin, Germany, 1993. Springer-Verlag.

[11] D. Fensel. Formal specification languages in knowledge and software engineering. *The Knowledge Engineering Review*, 10(4), 1995.

[12] D. Fensel. *The Knowledge-Based Acquisition and Representation Language KARL*. Kluwer Academic Pubblisher, 1995.

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

[14] M. Heisel, W. Reif, and W. Stephan. Formal software development in the kiv system. In M. Lowry and R. McCartney, editors, *Proc. of IJCAI Workshop on Automating Software Design*, pages 115–124, Palo Alto, USA, 1991.

[15] D. Landes. DesignKARL - a language for the design of knowledge-based systems. In *Proceedings 6th Int. Conference on Software Engineering and Knowledge Engineering SEKE'94*, pages 78–85, Jurmala, Latvia, 1994.

[16] C. Pierret-Golbreich. TASK model: a framework for the design of models of expertise and their operationalization. In B. R. Gaines and M. A. Musen, editors, *Proceedings of the 8th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, pages 37.1–37.22. SRDG Publications, University of Calgary, 1994.

[17] C. Pierret-Golbreich and X. Talon. Tfl: algebraic language to specify the dynamic behaviour of knowledge-based systems. *The Knowledge Engineering Review*, 11(2), 1996.

[18] J. Rushby. Formal methods and their role in the certification of critical systems. Technical Report CSL-95-1, SRI, 1995.

[19] A. T. Schreiber, B. J. Wielinga, J. M. Akkermans, W. Van de Velde, and A. Anjewierden. CML: The CommonKADS conceptual modelling language. In L. Steels, A. T. Schreiber, and W. Van de Velde, editors, *A Future for Knowledge Acquisition. Proceedings of the 8th European Knowledge Acquisition Workshop EKAW'94*, pages 1–25, Berlin/Heidelberg, 1994. Springer-Verlag.

[20] A. T. Schreiber, B. J. Wielinga, R. de Hoog, J. M. Akkermans, and W. Van de Velde. CommonKADS: A comprehensive methodology for KBS development. *IEEE Expert*, 9(6):28–37, December 1994.

[21] N. Shadbolt, E. Motta, and A. Rouge. Constructing knowledge-based systems. *IEEE Software*, 10(6):34–39, Nov. 1993.

[22] P.-H. Speel and M. Aben. Applying a library of problem solving methods on a real-life task. In B. R. Gaines and M. A. Musen, editors, *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, pages 10.1–10.21, Alberta, Canada, 1996. SRDG Publications, University of Calgary. http://ksi.cpsc.ucalgary.ca:80/KAW/KAW96/KAW96Proc.html.

[23] A. Stutt and E. Motta. Recording the design decisions of a knowledge engineering community to facilitate re-use of design models. In *Proceedings of the 1995 Knowledge Acquisition Workshop, Banff*, Canada, 1995.

[24] F. van Harmelen and M. Aben. Structure preserving specification languages for knowledge-based systems. *International Journal of Human-Computer Studies*, 44(2):187–212, 1996.

[25] F. van Harmelen and J. R. Balder. $(ML)^2$: a formal language for KADS models of expertise. *Knowledge Acquisition*, 4(1), 1992. Special issue: 'The KADS approach to knowledge engineering', reprinted in *KADS: A Principled Approach to Knowledge-Based System Development*, 1993, Schreiber, A.Th. *et al.* (Eds.).