The Knowledge Engineering Review, Vol. 9:4, 1994, 417–420

ECAI'94 Workshop on Formal Specification Methods for Knowledge-based Systems

DIETER FENSEL

Institut AIFB, University of Karlsruhe, 76128 Karlsruhe, Germany (Email: fensel@aifb.uni-karlsruhe.de)
Dept SWI, University of Amsterdam, Roetostraat 15, 1018 WS Amsterdam, The Netherlands (Email:dieter@swi.psy.uva.nl)

1 Introduction

The Workshop on Formal Specification Methods for Knowledge-based Systems (KBS) took place in Amsterdam on August 8 1994 as part of the workshop program of the 11th European Conference on Artificial Intelligence (ECAI'94). It was the sixth workshop in a series concerned with the development and application of formal and executable specification languages for KBSs. Starting from the first familiarization workshop at GMD in Bonn 1992, where the different research groups met for the first time, further successor workshops were held at the University of Karlsruhe, the University of Amsterdam, and again at GMD in Bonn. Additionally, at ECAI'92 in Vienna, a workshop was held to compare different specification approaches for complex multi-layered KBSs.

Originally, KBSs or expert systems were built by immediately implementing a running system. But soon it become clear that this only works for small programs with short lifetimes. As soon as such programs become bigger or require long-term use, they become neither maintainable nor do they meet the desired functionality as a result of their unstructured development process. Similar to lessons learned in software engineering in the late 1960s, the need for specification methods became a visible research target for the knowledge engineering community. Such a specification should allow for the description of the functionality of a system independently from any implementational details. Such a specification has two main impacts: first, the system and its functionality is described at a conceptual level which improves its comprehensibility; and second, there is no longer any need to unnecessarily fix design and implementation decisions during elicitation and specification of the requirements.

In KBSs, an essential part of the expert knowledge is not only concerned with the functionality of the system but also with the problem-solving process itself. Because of this, specification methods for KBSs must not only aim at describing the desired functionality of a system, but also the (dynamic) behaviour when solving a problem.

In recent years, a number of specification methods for KBSs, like CommonKADS, components of expertise, generic tasks, KADS, MIKE or PROTEGE-II, have been developed and successfully applied. Most of these approaches rely on informal specification techniques only. KBSs are described by natural-language texts organized by semi-formal modelling techniques like the KADS model of expertise. Informal specification techniques have some significant advantages: they are easy to understand and they allow for the description of a system at a higher conceptual level by hiding unnecessary details. However, as is known from experience in software engineering, informal specification techniques also suffer from significant shortcomings: ambiguity, incompleteness, inconsistency and redundancy.

To fill this gap between imprecise specifications and final implementations, a large number of languages have been developed during the last few years in Europe to allow a detailed and precise specification of KBSs which still abstract from their implementation. Languages like DESIRE, FORKADS, GCLA, KARL, KBSSF, $(ML)^2$, Model-K, MODEL/KADS, MoMo, OMOS or QIL can be used to define executable and/or formal specifications of KBSs. A survey on most of these languages can be found in Treur and Wetter (1993) and Fensel and van Harmelen (1994).

Whereas the first workshops on formal specification languages mainly had just the purpose of presenting and comparing the different languages, we are now at the stage where aspects like formal semantics of such specification languages, specific language topics like modularization or methodologies for deriving formal specifications from informal ones are discussed during such workshops.

2 Semantics of static and dynamics

In an introductory talk, Dieter Fensel (University of Karlsruhe) argued that the essence of a formal specification language is its formal semantics. Different types of these semantics can be distinguished:

- Declarative (denotational) semantics define the meaning of an expression in a second language. For example, the semantics of a logical language can be defined in terms of model theory.
- Operational semantics define a procedure which derives the meaning of an expression. Such semantics not only define the meaning of an expression, but also how they can be constructed.
- Mechanized semantics define an algorithm which can compute the meaning of an expression. Operational semantics can in general contain infinitely many inference rules or inference rules working with infinitely many premises. A mechanized semantics should be computable by a Turing machine. To define such a semantics, the expressive power of a language must be restricted to Turing completeness.
- Implemented semantics are defined by a running compiler or interpreter which use a computer to compute the meaning of an expression. Often, efficiency or storage aspects leads to additional restrictions or modifications.

Besides the development of such semantics for a specification language, the proofs of their (partial) correspondence are important tasks. As specification of KBSs must include the specification of the (dynamic) behaviour, these semantics must additionally include the specification of static and dynamic aspects. Dynamic and temporal logic are two promising candidates for this purpose.

Temporal logic with partial models has been chosen by the DESIRE group from the Free University of Amsterdam to define a declarative semantics for the dynamic reasoning pattern of a KBS. A current state in time is represented by a partial model which assigns true, false or unknown as truth values. The reasoning process is now modelled as a function between such partial models, each of them representing one point in time. By assigning a time point to these partial models a description of the complete reasoning is achieved and can be used to reason about its properties. Logical expressions in restricted temporal logic can be used to constrain possible control flows (i.e. reasoning patterns).

A different way in which to represent dynamic control was chosen by $(ML)^2$ (from the University of Amsterdam) and KARL (from the University of Karlsruhe). Both use dynamic logic, which is a well-established logical framework for the procedural representation of control flow. Dynamic logic was originally developed by the program verification community to reason about programs written in procedural languages, and therefore allow the explicit representation of procedural knowledge like sequence, branch and loops of operations. As these control flow statements are embedded in a logical framework, declarative description and reasoning about such a control flow become possible.

Until now, it has not been at all clear whether dynamic or temporal logic is the better candidate for the purpose of specification languages. Criteria for judging these and other candidates are their suitability for modelling KBSs and their appropriateness for formal reasoning about these models.

3 Modularization of specifications

Modularization and hierarchical decomposition are important means to tackle large and therefore complex specifications. These techniques improve the comprehensibility of a specification as well

as the efficiency of its execution and of the final system. DESIRE defines a compositional framework where the entire system can be specified as a set of interacting components. In addition, they introduced hierarchical refinement in their framework, which implies that such a component can again be a structured set of simpler ones.

The knowledge acquisition environment CERISE (University of Paris) uses abstract data types to specify KBSs in a formal and modular manner. This well known technique from software engineering is extended in such a way that it can be used to specify domain knowledge and problem-solving methods in a modular manner. The domain layer is described by algebraic specifications, and the inference layer is described by generic specifications with formal parameters. Their connection is reached by parameterization.

The three-layered architecture of the Multi-Context System MS (from IRST in Trento, implemented in the GETFOL system) can be used to declaratively specify KBSs. The object level provides a modular specification of the knowledge and reasoning steps of a particular application domain. One module can be specified by expressions of first-order logic. So-called "bridge rules" can be used to specify the interaction between modules. The second level, a Problem Solving Context, allows the specification of meta-level knowledge about the problem domain by inference and reflection rules. The final level presents heuristic reasoning strategies applying the rules specified in a Problem Solving context. Therefore, MS allows the uniform description of an entire KBS in an integrated logic-oriented manner.

4 Operationalization of specifications

GCLA (pronounced "Gisela" from SICS in Sweden) is a logic programming language based on partial inductive definitions, which extends Prolog. Even though GCLA was not developed as a specification language for KBSs, the group could show how easily the KADS model of expertise with its distinction between domain, inference and task layer fits quite well to the distinction between axioms, inference rules and inference strategies in GCLA. Therefore, a language like GCLA can be viewed as an executable specification language, or even as an implementation language which enables structure-preserving design and implementation of the KBS.

KBSSF (pronounced KSF) is a formal specification language developed by the Dutch PTT in Groningen in the VITAL project. During the workshop, an executor for KBSSF was shown which enables the evaluation of KBSSF specifications by a running prototype. As prototyping or testing has been proven to be a very valuable tool for evaluating specifications or programs, this was indeed a significant improvement of their approach. They use C++ for realizing the dynamic parts of the specification and Prolog for the static parts.

Mapcar is a reflective production rule language for executable specification of KBSs enabling prototyping at the knowledge level. Domain knowledge is represented by production rules, and inference and control knowledge is specified by meta rules. Thus, the different types of knowledge can be represented in a uniform and easy to understand representation formalism. As this representation is executable, it enables the evaluation by a running prototype.

5 Relationships to formal approaches in the field of requirements engineering

In an invited talk, Robert Darimont discussed the KAOS project. The project is a methodology for requirements engineering containing a formal language for their representation. The actual requirements are specified by instantiating the meta-model of KAOS. It contains objects, actions, agents, goals, constraints and scenarios. With this framework not only functional requirements by objects and actions (what-questions) but also why questions (goals), who questions (agents), and when-question (events and triggers) can be specified. For the latter, a restricted temporal logic is provided. KAOS provides heuristics and strategies for acquiring such requirements. For example, the goal reduction rules are very close to ideas of task decomposition in knowledge engineering.

The discussion with people from software engineering, requirements engineering and information system development will become an important issue for further activities. First, significant parts of the tackled problems in the different communities are similar—specifying a KBS means specifying a special software system—and knowledge engineering can learn a lot from 30 years of experience in software engineering. Second, KBSs will normally not be used as stand-alone systems, but as integrated parts of larger systems, which enforces the combination of their development methods with traditional ones.

6 Methodologies

Two further contributions to the workshop tackled the problem of developing a formal specification of a KBS. Until now, the research work has concentrated on developing formal languages for specifying KBSs, but less methodological support has been provided in working with these languages. The first paper was concerned with the transition of first-order specifications into implementation-level design languages, and the second paper presented a case study to investigate the usability of the formal specification language $(ML)^2$. In particular, the close relationship between informal and formal specification of this KADS-based language arose as a significant feature, which allows common problems in the use of formal specifications to be by-passed. The discussion during the session mainly tackled problems related with formal specifications, like: What is the right abstraction level from implementation details; Does the additional effort of writing a formal specification buy anything; and is there no standard specification language which should be used for a given application?

7 Future work

After some years of common research activities, there exist now a large number of executable or formal specification languages, and it no longer seems an important research goal just to present new languages. Instead, two important tasks will be the use and improvement of the existing languages. The former requires the development of methodological frameworks like refinement calculi which support the construction process of formal specifications. The latter means the development of a reasoning calculus which allows the validation of formal specifications by proving properties of such specifications. For this, deeper studies on the underlying semantics of the languages must be done.

The next workshop on formal specification languages will be held in Amsterdam in January 1995. A further workshop in June 1995 (in conjunction with EUROVAV, the European symposium on validation and verification) will discuss the usefulness of formal specifications for validation and verification of KBSs, as well as the use of these techniques for evaluating such specifications.

References

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

Fensel, D and van Harmelen, F, 1994, "A comparison of languages which operationalize and formalize KADS models of expertise." *The Knowledge Engineering Review* **9**(2) 105–146.