

Adoption-Centric Knowledge Engineering

Neil A. Ernst

*Department of Computer Science, University of Victoria
PO Box 3055, STN CSC, Victoria, BC, Canada V8W 3P6
nernst@cs.uvic.ca*

Abstract

Cognitive issues in software engineering are relatively well-documented and well-understood when compared with the domain of knowledge engineering. Current knowledge engineering tools often feature high barriers to the use and re-use of both the tool and its products. An adoption-centric knowledge engineering approach is suggested to deal with these issues. Adoption-centric knowledge engineering is the design of knowledge engineering tools and knowledge engineering processes to ensure the widest possible adoption. In turn, wide adoption will benefit the projects that choose to focus on it by increasing the user-base. One way to make a tool more adoption-centric is to provide increased cognitive support to the potential user.

1. Introduction

The introduction to a well-known project site for adoption-centric software engineering (ACSE) [1] makes the case for ACSE as follows: “Research tools in software engineering often fail to be adopted and deployed in industry.” This is equally true of tools in the discipline of knowledge engineering. User-centered software engineering has seen a wealth of research compared to similar projects in knowledge engineering. This research has produced a body of work which describes theories for how software engineering is practiced, although by no means an exhaustive amount. Knowledge engineering, the design of knowledge-based systems, be they theorem provers, expert systems, or intelligent agents, is not as well documented. I am not referring here to the logical foundations of expert systems, as this is a much-studied area; see [2] or [3] for examples of seminal knowledge-engineering design projects. These papers document in detail the mechanics of designing a knowledge-based system. Unfortunately, the knowledge acquisition field has traditionally ignored the user perspective in these areas. Practitioners have been more concerned with designing a system to solve a problem – say, to diagnose a

specific medical condition – than with the actual methods used to create the system. This is well illustrated in [3]; namely, that few efforts in the field are focused on developing tools for users, being more concerned with knowledge modelling and knowledge elicitation, often at the expense of end-user usability concerns.

We should be concerned with end-user adoption and usability (where the end-user, in this case, is the system designer) because developing good applications is directly related to how simple the chosen tool is to use: the tool should be unobtrusive, a fact shown in certain software engineering studies [4]. Knowledge engineering needs a similar focus. Adoption-centric knowledge engineering (ACKE) would be focused, like its sibling ACSE, on delivering tools that leverage existing user knowledge rather than requiring learning yet another new product; this can provide a significant advantage to developers. In this position paper I first discuss the background of knowledge engineering, particularly with respect to software engineering; section 3 illustrates a typical tool for adding to cognitive understanding of large knowledge bases; finally, section 4 argues that adoption-centric knowledge engineering is of vital importance to many projects.

2. Background

This section describes the fundamental ideas of knowledge engineering and looks at the intersections of both knowledge engineering and software engineering. I have found that empirical studies of cognitive support for knowledge engineering are lacking, and seek to leverage comparable studies in software engineering.

2.1 Knowledge Engineering

There has been an increased focus in recent years on knowledge engineering, particularly in response to the Semantic Web initiative of the World Wide Web Consortium (W3C) [5], [6]. The Semantic Web initiative is concerned with the “abstract representation of data on

the World Wide Web” [7] such that additional, machine-comprehensible metadata might be created. The formation of global standards such as the Resource Description Framework (RDF) [8], the Web Ontology Language (OWL) [9], and XML, combined with the power of distributed application development via the Internet, has led to renewed interest in knowledge-based systems, to perform any number of tasks, such as making inferences on web site metadata to intelligent e-commerce shopping agents [10].

The term knowledge engineering refers to the design and construction of knowledge-based systems, much like software engineering refers to the design and construction of software systems. Traditional knowledge engineering has followed a number of processes, which typically contain some of the following elements [11] (figure 1):

- a) The knowledge base design phase. This is a model of the proposed system (for example, a medical protocol for cancer treatment); this phase is akin to a software modelling phase where requirements are gathered but no actual code is written;
- b) the knowledge acquisition and knowledge elicitation phase, in which domain experts are interviewed by knowledge engineers and data instances are created;
- c) the knowledge entry phase, in which the knowledge engineer enters the newly acquired data into the knowledge base;
- d) the knowledge maintenance phase, where the system is updated to reflect new facts and rules.

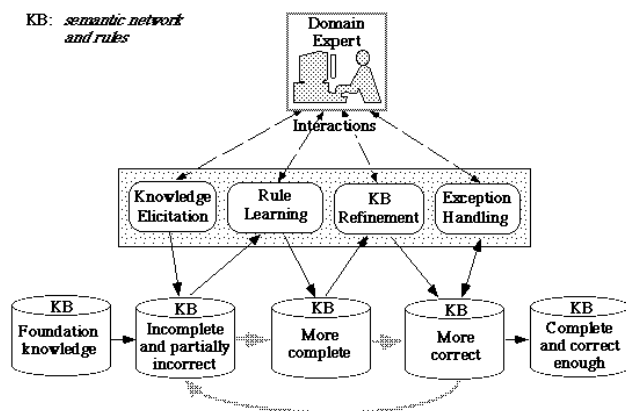


Figure 1 - The knowledge acquisition process ([12])

This process is somewhat non-linear, as indicated by the grey arrows. Knowledge engineering, like software engineering, can be very iterative; as the system is tested and faults are found, the model may be changed or the data instances modified.

In knowledge engineering there is a third party involved in a significant way: the domain experts are a

crucial and significant part of the knowledge engineering process. Since the domains are so complicated, and the goals so high-level, a second expert is often needed to explain this, particularly where the knowledge engineer has no domain knowledge. Furthermore, the modelling of knowledge-based systems often takes place at the knowledge level [13], rather than the symbol level; this higher degree of abstraction leads to problems in deciding exactly what the system is capturing and modelling.

Knowledge engineering tools are systems designed to automate some aspect of this complex process. They range from tools which help with the design of knowledge-based systems [14] and the elicitation of knowledge [15], to tools which provide a mechanism to maintain and upgrade the knowledge base ([16], [17]). There are two broad classes of users for these tools. One is an end-user, for instance, the doctor or engineer accessing a knowledge-base for decision support with some task. The second category, with which my research is most concerned, is the system engineer, either the knowledge engineer who created the system, or a maintenance engineer who seeks to ensure accuracy, speed, and other performance measures as defined by the application specification. This class of user has obvious and natural equivalencies with his software engineering counterpart, for example, a maintenance programmer. Both users require cognitive support for understanding the model for which they are responsible. To further narrow the types of use-case we seek to model, I have wilfully ignored the cases where we seek to enter knowledge (knowledge acquisition), focusing instead on maintaining knowledge-bases. This is akin to the software maintenance and program comprehension tasks. One of the big difficulties in this area is the lack of comprehensive research in the field. While there is a fair amount of work in the areas of knowledge engineering methodology, such as ensuring accuracy and performance, comparatively little has been done on knowledge base maintenance, and there are no theories on cognitive support for knowledge base engineering, unlike some work in software engineering [18].

2.2 Knowledge Engineering and Software Engineering: Perspectives

What are the differences and similarities between knowledge engineering and software engineering? I believe there are two perspectives to take on this relationship; one is to examine knowledge-based software engineering, the other to consider software-centred knowledge engineering. Software engineering can often be said to be knowledge-centric, in that it seeks to construct a knowledge-based model of a particular application domain. This would apply to projects which leverage the knowledge of end-users to support decisions

they might make at a later date. Such systems often contain a large degree of knowledge, such as the forest stand lifecycle data of the Sortie project [19], within data structures of the program, whatever the programming language.

Clearly, the greatest overlap occurs in languages such as Prolog, which is typically used by software engineers to construct knowledge-based systems. We describe these systems as software-centered, because their functionality arises from a software-based tool or solution. In this perspective, software tools are used to construct a symbol-level representation of the knowledge-level model.

Other systems exist outside these perspectives. There are software projects, such as an operating system, which make less use of specific knowledge from end-users. Requirements in this type of project are largely functional and quantitative. The Linux project, for example, is not as concerned with capturing user-specific information as it is with providing certain functionality, such as USB support. The inverse of this is the knowledge base which uses no or few software engineering techniques to design the tool. We might characterize knowledge bases which use domain-specific tools in this manner. An example of this is a controlled terminology which is used to standardize the vocabulary of a particular domain. On their own, these terminologies use no specific software engineering approaches; however, they are often combined with other tools, such as Internet application servers, to deliver the product.

3. Our Approach

Our research group is developing a tool called Jambalaya [17]. Jambalaya is the integration of a software understanding tool, SHriMP, with a knowledge engineering tool, Protégé [20], to attempt to provide some cognitive support for developers of software-centric knowledge bases. We have recently completed a user questionnaire on how people might use the visualizations of Jambalaya for enhancing their comprehension of the (often-complex) knowledge structures in Protégé, in order to provide a better understanding of these issues: a paper describing these results is in progress. Amongst the things that emerged were:

- the large number of domains being worked on;
- the different sizes of ontologies which users manage;
- the relative lack of usable visual representations of the knowledge structures.

These points seem to indicate that a major challenge in both designing tools for knowledge engineering and, perhaps more importantly, increasing the adoption of those tools, will be in bridging the number of domains and approaches which exist. The reason better cognitive

support is needed remains relatively unclear except on an ad-hoc level, and needs to be addressed through user evaluations and interviews. Some preliminary work, however, seems to illustrate the need fairly clearly. Blythe et al. [21], for example, identifies some typical concerns that users may have when adding new knowledge to an intelligent system:

- Users do not know where to start and where to go next;
- Users do not know if they are adding the right things;
- Users often get lost as it takes several steps to add new knowledge.

Our goal is to attempt to address some of these concerns by providing enhanced cognitive support for developers in understanding the nature of the knowledge-base they are maintaining. For example, Jambalaya provides a series of different graph layout algorithms to allow users to maintain different perspectives on the model. One area of research of great interest is on what techniques knowledge engineers use to understand the model. For example, research in software engineering indicates several strategies for program comprehension, such as top-down, bottom-up, knowledge-based, as-needed, and integrative [22]. We are interested in exploring whether such techniques translate readily to the knowledge engineering domain.

4. The Need for Adoption-Centric Knowledge Engineering

Our tool is currently integrated with a reasonably popular knowledge engineering framework. Protégé has fairly widespread support, but is by no means a universal tool. Much like the popular software development environment Eclipse (eclipse.org), Protégé has a number of adoption-centric advantages, including an extensible plug-in architecture, an open-source licence, and a lengthy history in the community. Nevertheless, Protégé has limitations in specific areas. For example, Protégé uses a frame-based knowledge representation, which is only one of many formalisms, each of which has its own advantages. The frame-based representation is somewhat similar to object-oriented software engineering paradigms, and Protégé uses one version of it, much like Eclipse offers Java integration. Some large applications, such as the U.S. National Cancer Institute's controlled terminology, prefer instead to use different representations, for a variety of reasons, just as other projects may use pure first-order logic or variants thereof. The particular formalism a tool uses may affect the type of cognitive assistance required: for example, in a frame-based system like Protégé, a hierarchical object-centric view may be appropriate; in a first-order logic rule-based

expert system, a visualization of how the rules were fired may be of most interest. ACKE tools should provide support for the engineer, and not the formalism chosen (just as ACSE should not differentiate between programming languages). In other words, ACKE tools should be flexible enough to support different and varied environments and formalisms, as the user may require. This may not be possible in all domains and areas of interest, naturally; rather, what is being proposed is a user-centered approach to the design of these tools, as much as possible. Many current tools focus only on the semantics of the formalism they are attempting to implement – that is, does the tool fulfil the formal model and syntax specification of the formalism – and not on how usable the tool may be for developing different applications.

In particular, the Semantic Web initiative defines only representation mechanisms and not tools to perform knowledge engineering operations. While Protégé seems likely to be a part of Semantic Web application development, it is unreasonable to assume that it would be the only tool used, which illustrates the need for ACKE tools. If a new platform for editing Semantic Web services becomes widespread, we should not force users who have a need for the enhanced cognitive support that Jambalaya may offer to adopt Protégé as well. Instead, we should focus on adapting our tool to the user requirements, rather than vice-versa. The ACSE approach suggests that the best method for providing developers of knowledge-based systems with tool support is to focus on developing either simple add-ins to commonly used platforms, or providing interfaces to those platforms. One approach might be to embed the tool in another product with a larger base of users, much as we have done with Protégé and SHriMP, but another example might be to offer SHriMP-like functionality in an SVG-based web tool. This would allow users to continue using their web user-agent, such as Internet Explorer, with whose functions they are very familiar, while also accessing more complex functionality to support the creation and browsing of sophisticated knowledge-based applications. An interface approach might suggest developing a standard import/export mechanism, allowing for interoperability between tools. A relatively recent example of this is the ability of Protégé to export XMI serializations of its models, allowing Protégé users to access a wider range of products, such as Rational Rose [23]. The challenge for the developers of cognitive aids, such as the SHriMP team, is to reduce the feature set and user interface challenges of the tool to a point where the essential features are preserved, yet the cognitive overhead of learning the tool is still low enough to encourage adoption. It is not sufficient to merely embed the tool in a knowledge-engineering platform; it must still be compelling and intuitive – in other words, it must rapidly answer the user's question, What does this do for

me? I believe previous knowledge engineering tools, while their formal *utility* may have been high, nevertheless required a great deal of learning before they met simple *usability* criteria, forcing users to learn not only new user interfaces (Protégé, for example, defines its own look and feel, and is one of the simpler and better-designed UIs in the area), but also to understand and leverage new syntax and semantics. The hurdles imposed by the RDF syntax, for example, are high enough without forcing users to learn a complex tool as well.

5. Conclusions

A primary goal of adoption-centric software engineering is to increase the number of users of software engineering tools, to increase awareness of the potential benefits these tools offer. My position is that such a focus is equally important for knowledge engineering tools, particularly with the emerging focus on the development of knowledge-aware applications on the Internet. Recent trends suggest that the divisions between software engineering and knowledge engineering may be blurring, and the demand for more powerful application design and construction tools, such as Eclipse and Protégé, is growing, whether the application is knowledge-centric or software-centric.

If the vision of the Semantic Web is to be realized, it will likely arise in distributed fashion, much like its forefather the World Wide Web has done. To leverage the true capabilities of the Semantic Web, we will see increasing returns with more and more providers making information – knowledge – available to other applications. The vision has many other hurdles, among them privacy and provenance, but the lack of easy to use tools is one which should be straightforward to overcome. I propose making ACKE a focus for new knowledge engineering tools to support this vision.

7. References

- [1] A. Weber, *Adoption-Centric Software Engineering*, available at: <http://www.acse.cs.uvic.ca/>, Department of Computer Science, University of Victoria: 2003
- [2] B. G. Buchanan and E. H. Shortliffe, *Rule-Based Expert Systems: The MYCIN experiments of the Stanford Heuristic Programming Project*. Reading, MA: Addison-Wesley, 1984.
- [3] N. F. Noy, R. W. Fergerson, and M. A. Musen, "The knowledge model of Protege-2000: Combining interoperability and flexibility," in *Proceedings of the 2nd International Conference on Knowledge Engineering and Knowledge Management (EKAW'2000)*, Juan-les-Pins, France, 2000.
- [4] M.-A. D. Storey, K. Wong, F. Fracchia, and H. Mueller, "On Integrating Visualization Techniques for Effective Software

- Exploration," in *Proceedings of the InfoVis '97*, Phoenix, AZ, 1997.
- [5] T. Berners-Lee, M. Fischetti, and M. Dertouzos, *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by its Inventor*. San Francisco: Harper, 1999.
- [6] N. F. Noy, M. Sintek, S. Decker, M. Crubézy, R. W. Ferguson, and M. A. Musen, "Creating Semantic Web Contents with Protégé-2000," *IEEE Intelligent Systems*, pp. 60-72, 2001.
- [7] E. Miller, R. Swick, D. Brickley, B. McBride, J. Hendler, and G. Schreiber, *W3C Semantic Web*, available at: <http://www.w3.org/2001/sw/>, World Wide Web Consortium: 2003
- [8] F. Manola and E. Miller, *RDF Primer Working Draft*, available at: <http://www.w3.org/TR/rdf-primer>, World Wide Web Consortium: 2002
- [9] M. K. Smith, D. McGuinness, R. Volz, and C. Welty, *Web Ontology Language (OWL) Guide version 1.0*, available at: <http://www.w3.org/TR/owl-guide>, World Wide Web Consortium: 2002
- [10] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," in *Scientific American*, 2001.
- [11] S. Russell and P. Norvig, *Artificial Intelligence: A modern approach*. New Jersey: Prentice-Hall, Inc., 1995.
- [12] G. Tecuci, *Constructing and Refining Knowledge Bases: A Collaborative Apprenticeship Multistrategy Learning Approach*, available at: <http://lalab.gmu.edu/Projects/HPKB/boston-briefing/HPKB-Boston-index.htm#Index>, Learning Agents Laboratory, Computer Science Department, George Mason University: 1997
- [13] A. Newell, "The Knowledge Level," *Journal of Artificial Intelligence*, vol. 18, 1982.
- [14] Protege-2000, *The Protege-2000 website*, available at: <http://protege.stanford.edu>, Stanford Medical Informatics: 2003
- [15] P. Clark, J. Thompson, K. Barker, B. Porter, V. Chaudhri, A. Rodriguez, J. Thomere, S. Mishra, Y. Gil, P. Hayes, and T. Reichherzer, "Knowledge Entry as the Graphical Assembly of Components," in *Proceedings of the 1st International Conference on Knowledge Capture (K-Cap '01)*, 2001.
- [16] I. Horrocks, "FaCT and iFaCT," in *Proceedings of the International Workshop on Description Logics (DL'99)*, P. Lambrix, A. Borgida, M. Lenzerini, R. Möller, and P. Patel-Schneider, Eds., 1999, pp. 133-135.
- [17] M.-A. D. Storey, M. A. Musen, J. Silva, C. Best, N. Ernst, R. Ferguson, and N. F. Noy, "Jambalaya: Interactive visualization to enhance ontology authoring and knowledge acquisition in Protege," in *Proceedings of the Workshop on Interactive Tools for Knowledge Capture, K-CAP-2001*, Victoria, B.C. Canada, 2001.
- [18] A. Walenstein, *Cognitive Support in Software Engineering Tools: A Distributed Cognition Framework*, Unpublished Ph.D. thesis, Computer Science, Simon Fraser University
- [19] M.-A. D. Storey, S. E. Sim, and K. Wong, "A collaborative demonstration of reverse engineering tools," *ACM SIGAPP Applied Computing Review*, vol. 10, pp. 18 - 25, 2002.
- [20] N. F. Noy, R. W. Ferguson, and M. A. Musen, "The knowledge model of Protégé-2000: combining interoperability and flexibility," SMI Technical Paper.
- [21] J. Blythe, J. Kim, S. Ramachandran, and Y. Gil, "An integrated environment for knowledge acquisition," in *Proceedings of the Int. Conf. on Intelligent User Interfaces*, 2001.
- [22] M.-A. D. Storey, F. D. Fracchia, and H. A. Müller, "Cognitive Design Elements to support the Construction of a Mental Model During Software Exploration," *Journal of Software Systems, special issue on Program Comprehension*, vol. 44, pp. 171-185, 1999.
- [23] H. Knublauch, *XMI Backend: Storing Protégé ontologies in XMI*, available at: <http://protege.stanford.edu/plugins/xmi/>, Stanford Medical Informatics: 2003