

Knowledge Modeling at the Millennium

(The Design and Evolution of Protégé-2000)

William E. Grosso,¹ Henrik Eriksson,² Ray W. Fergerson,¹
John H. Gennari,³ Samson W. Tu,¹ and Mark A. Musen¹

¹Stanford Medical Informatics, Stanford University

² Department of Computer and Information Science, Linköping University

³ Department of Information and Computer Science Department, University of California (Irvine)

ABSTRACT

It has been 13 years since the first version of Protégé was run. The original tool was a small application, aimed mainly at building knowledge-acquisition tools for a few very specialized programs (it grew out of the ONCOCIN project and the subsequent attempts to build expert systems for protocol-based therapy planning). The most recent version, Protégé-2000, incorporates the Open Knowledge Base Connectivity (OKBC) knowledge model, is written to run across a wide variety of platforms, supports customized user-interface extensions, and has been used by over 300 individuals and research groups, most of whom are only peripherally interested in medical informatics. Researchers not directly involved in the project might well wonder how Protégé evolved, what are the reasons for the repeated reimplementations, and how to tell the various versions apart. In this paper, we give an overview of the evolution of Protégé, examining the methodological assumptions underlying the original Protégé system and discussing the ways in which the methodology has changed over time. We conclude with an overview of the latest version of Protégé, Protégé-2000.

1. MOTIVATION AND A TIMELINE

The Protégé applications (hereafter ‘Protégé’) are a set of tools that have been evolving for over a decade, from a simple program which helped construct specialized knowledge-bases to a set of general purpose knowledge-base creation and maintenance tools. While Protégé began as a small application designed for a medical domain (protocol-based therapy planning), it has grown and evolved to become a much more general-purpose set of tools for building knowledge-based systems.

The original goal of Protégé was to reduce the knowledge-acquisition bottleneck (Hayes-Roth et al, 1983) by minimizing the role of the knowledge-engineer in constructing knowledge-bases. In order to do this, Musen (1988, 1989b) posited that knowledge-acquisition proceeds in well-defined stages and that knowledge acquired in one stage could be used to generate and customize knowledge-acquisition tools for subsequent stages. In (Musen, 1988), Protégé was defined as an application that takes advantage of this structured information to simplify the knowledge-acquisition process. The original Protégé was described this way (Musen, 1988):

Protégé is neither an expert system itself nor a program that builds expert systems directly. Instead, Protégé is a tool that helps users build other tools that are custom-tailored to assist with knowledge-acquisition for expert systems in specific application areas.

The original Protégé demonstrated the viability of this approach, and of the use of task-specific knowledge to generate and customize knowledge-acquisition tools. But as with many first-

generation systems, the original Protégé was written with target applications and domains in mind and, as a result, had limited applications.

Over the past 13 years, through 4 distinct releases, the Knowledge Modeling Group at Stanford Medical Informatics (SMI) has worked to turn Protégé into a general purpose set of knowledge-modeling tools. The current system, Protégé-2000, is far more general than the original version while maintaining the original version's focus on the use of meta-knowledge to create usable knowledge-acquisition tools.

Protégé has evolved by building tools and testing them with real knowledge-based systems. This process, wherein each generation of Protégé is tested on real-world problems and used to build knowledge-based systems, is crucial to the project. Just as Protégé embodies a methodology for building knowledge-based systems, the development of Protégé embodies a meta-methodology. Rather than endlessly debate the correct library structure or spend years agonizing over how to formally define constraint-satisfaction, Protégé development proceeds in the following five step cycle:

1. Start with an existing tool
2. Figure out what features are desired in the new version (based on using Protégé to build knowledge-based systems). Usually, this involves relaxing some assumption made in the existing tool
3. Make as many reasonable assumptions about knowledge-acquisition as are needed in order to build a working version of Protégé that has the desired functionality.
4. Build knowledge-bases and knowledge-based systems using the new version of Protégé
5. Return to step 1.

In this paper, we examine the evolution of Protégé. We start by reviewing the state of knowledge-acquisition in the mid 1980's. After a discussion of OPAL, the immediate ancestor of Protégé, we conduct an in-depth examination of the conceptual assumptions that were made in order to build the original Protégé. We then follow Protégé through several reimplementations, discussing the way the underlying methodology and assumptions changed over time before closing with an extended discussion of Protégé-2000, the latest version of the Protégé framework.

In keeping with this historical emphasis, the focus of this paper is on the underlying assumptions and methodology that inform the design of Protégé, rather than on the precise details of any particular feature or formalism; we are more interested in charting the evolutionary arc of Protégé and attempting to place Protégé within a historical context than on explaining any single feature or modeling formalism.

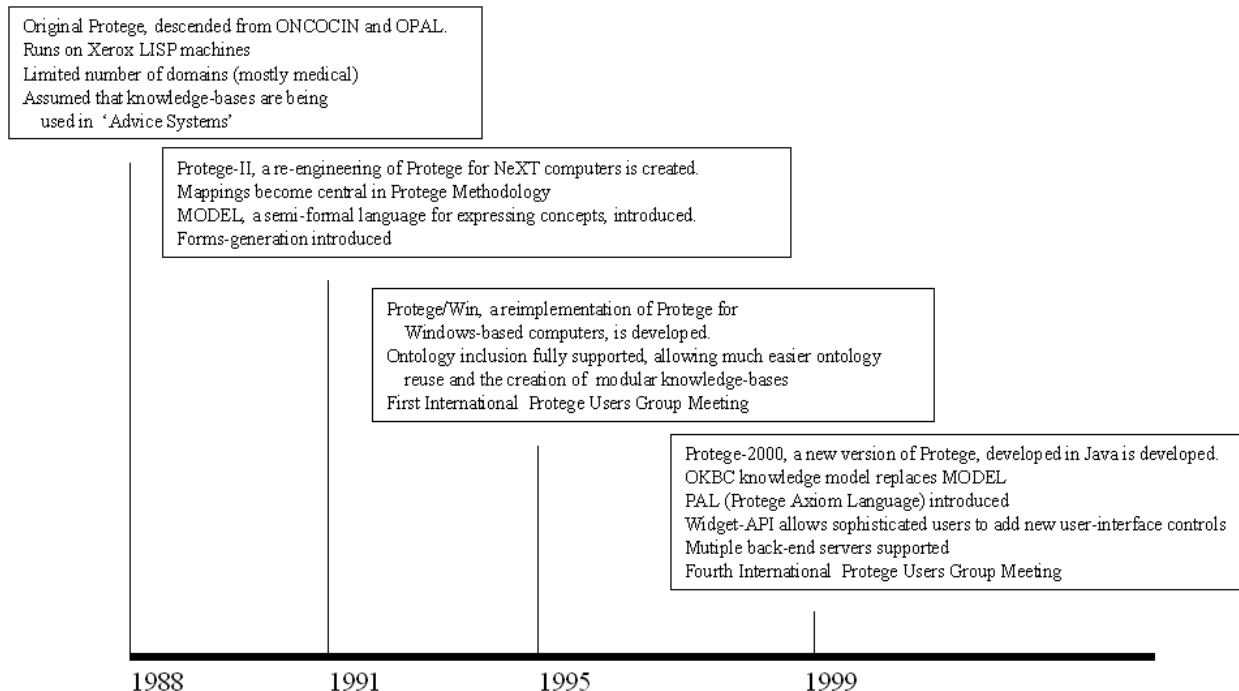


Figure 1. Historical Timeline for Development of Protégé.

2. KNOWLEDGE ACQUISITION IN 1983

The early 1980's were a heady time for Artificial Intelligence. Expert systems research had produced some stunning successes (see, for example, (Buchanan and Shortliffe, 1984), (McDermott , 1980), and (Bachant and McDermott, 1984)). To many people in the field it seemed that AI was on the verge of a dramatic breakthrough. Perhaps the authors of (Hayes-Roth et al, 1983) put it best when they wrote:

Over time, the knowledge engineering field will have an impact on all areas of human activity where knowledge provides the power for solving important problems. We can foresee two beneficial effects. The first and most obvious will be the development of knowledge systems that replicate and autonomously apply human expertise. For these systems, knowledge engineering will provide the technology for converting human knowledge into industrial power. The second benefit may be less obvious. As an inevitable side effect, knowledge engineering will catalyze a global effort to collect, codify, exchange and exploit applicable forms of human knowledge. In this way, knowledge engineering will accelerate the development, clarification, and expansion of human knowledge.

However, even in this volume, in many ways the high-water mark of expert-system optimism, the bulk of the papers discussed difficulties inherent in building expert systems. In particular, (Buchanan et al, 1983) contained a discussion of the difficulties inherent in modeling expert knowledge, beginning with an examination of the role of the knowledge engineer. In their "composite scenario for knowledge acquisition" (section 5.1.2), the knowledge-engineer is involved in all phases of system construction. The knowledge-engineer is required to become familiar with the problem domain, characterize the reasoning tasks necessary to solve the problem, identify the major domain concepts, categorize the type of knowledge necessary to

solve the problem, identify the reasoning strategies used by experts, define an inference structure for the resulting application, and formalize all this in a generic and reusable way.

In this scenario, experts are primarily viewed as a resource for knowledge engineers to draw upon, and as a potential validation mechanism for the finished knowledge-based system. Later in the same paper, the authors propose the following iterative model for the development of knowledge-based systems.

Stage	Description	Performed By	Followed By
1. Identification	Characterize important aspects of problem. Identify participants, problem characteristics, resources, and goals.	Domain experts, knowledge-engineer	2
2. Conceptualization	Make key concepts and relations from Identification Stage explicit.	Knowledge-engineer	3
3. Formalization	Identified concepts are represented in a formal language.	Knowledge-engineer	4
4. Implementation	Knowledge from formalization stage is represented in an expert-system shell.	Knowledge-engineer	5
5. Testing	The completed system is tested on sample cases and weaknesses are identified	Domain experts, knowledge-engineer	6
6. Revision	Redesign and reimplement the system, in light of the results from testing.	Knowledge-engineer	1-4 (Depends on what system modifications are needed)

Table 1. The Classical Model of Knowledge-Based System Development. (after (Buchanan et al, 1983))

3. OPAL AND ONCOCIN

After the early success of Mycin, researchers at Stanford’s Medical Computer Science Laboratory turned to developing a knowledge-based system called ONCOCIN. ONCOCIN¹ was an advice system for protocol-based cancer therapy; information about the history of a specific patient being treated was entered by a domain user (a physician or a nurse) and the program would give advice about further treatment and tests (see (Shortliffe, 1984b) for a discussion of the relationship between MYCIN and ONCOCIN).

The original version of ONCOCIN followed the methodology described in Table 1 above. Knowledge engineers talked to medical specialists and then created the knowledge-bases used in

¹ The name comes from “Oncologic MYCIN”

applications of the ONCOCIN system. This central role of the knowledge-engineer is immediately apparent from the following diagram (reproduced from (Musen, 1988)).

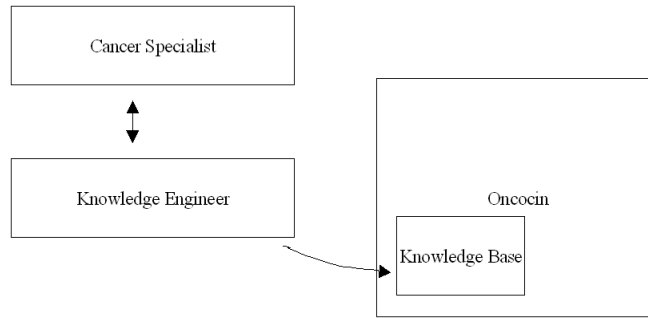


Figure 2. Knowledge-entry into ONCOCIN. In the early versions of ONCOCIN, cancer guidelines were entered into the system by knowledge-engineers who, in consultation with cancer-specialists, programmed if-then rules and other data structures into a knowledge-base containing cancer guidelines (from (Musen, 1988)).

ONCOCIN was quickly augmented by a custom-programmed knowledge-acquisition component named OPAL. The goal of OPAL was to allow the domain-expert to directly enter some forms of domain-specific knowledge (e.g. to remove the knowledge-engineer from stage 4 of table 1). OPAL used domain-specific concepts and ideas in order to present cancer-specialists with carefully designed forms for structured data entry. As (Musen, 1988) put it:

OPAL allows oncology experts to enter cancer protocols in ONCOCIN's knowledge base directly. Instead of typing individual production rules, the physicians themselves describe complete cancer protocols by filling out special-purpose graphical forms .

OPAL then translated the expert's input into ONCOCIN's internal representation. Figure 3 shows the resulting system architecture.

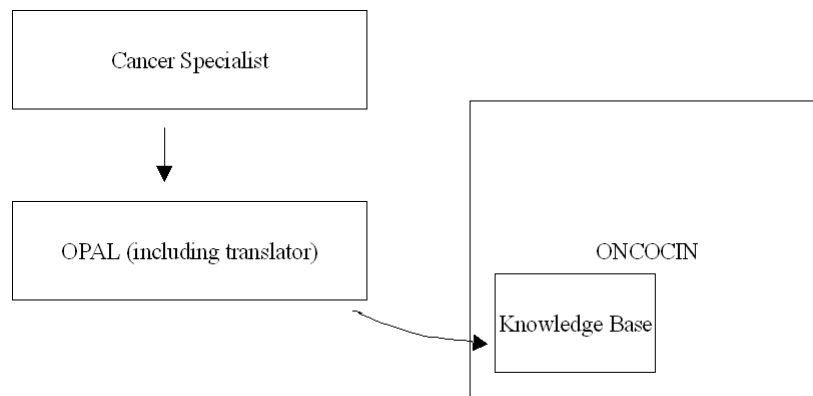


Figure 3. Knowledge Entry into OPAL. OPAL was a computer-based knowledge-acquisition tool that allowed physicians to enter new oncology protocols directly. OPAL automatically constructs the rules and other data structures that are required to run ONCOCIN consultations (from (Musen, 1988)).

Researchers quickly observed that the OPAL architecture implicitly asserts a qualitative division among the types of information a knowledge-based system requires. Figure 4 is a modification of Figure 3 to show this division; it also indicates the roles the different types of information enter the system.

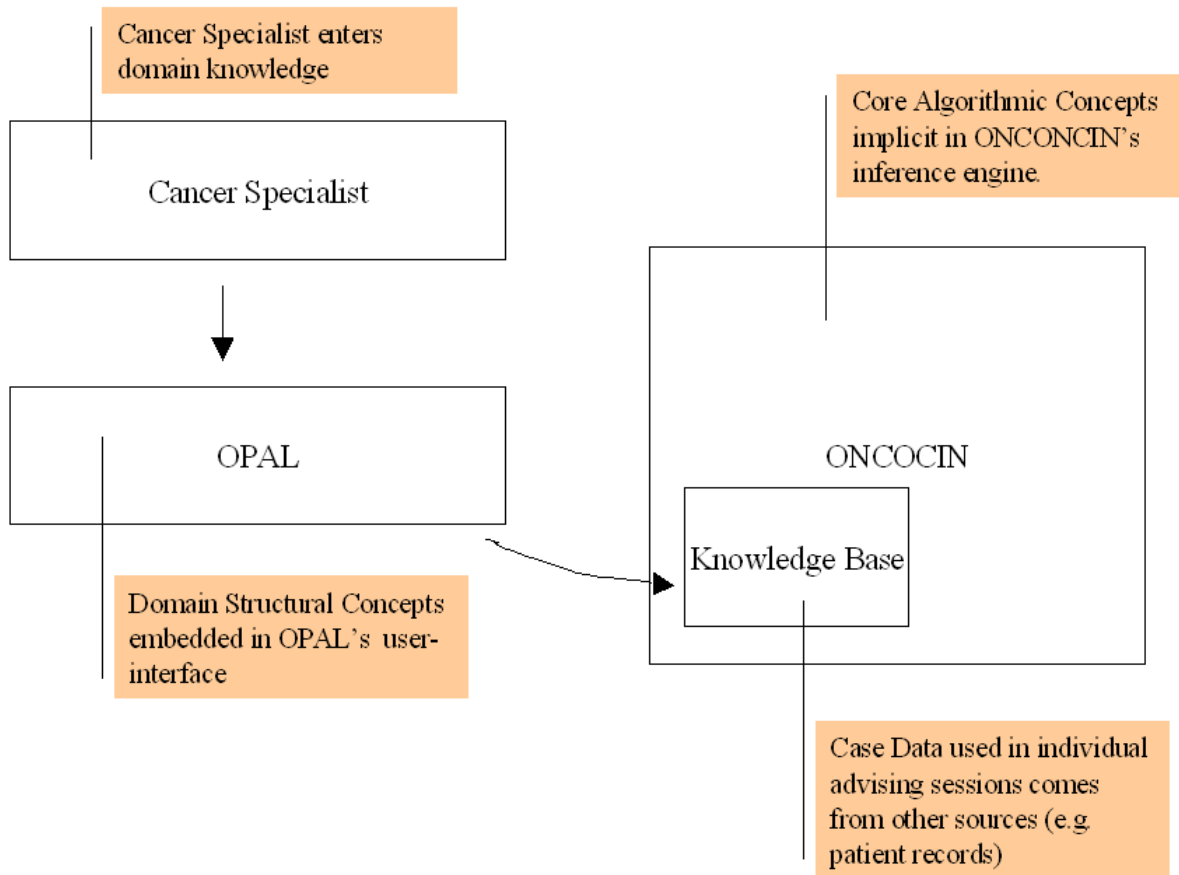


Figure 4. The 4 types of information, as they are implicit in the OPAL architecture. Core algorithmic concepts and domain structural concepts are “hard-wired” into the architecture.

The assertion that information can be cleanly partitioned in this way is crucial to the Protégé methodology. Indeed, it is the only major assumption made across all versions of Protégé and will be referred to in the rest of this paper as *the information-partitioning hypothesis*. Table 2 discusses the information types in more detail.

Information Type	Level of Genericity	Prototypical Examples	In Modern Parlance
Core Algorithmic Concepts	Reusable across classes of systems that share same general inference structure (e.g. <i>Heuristic Classifiers</i> or <i>Advice Systems</i>).		No clear equivalent. This would be a small part of an “upper ontology” and a number of rules that structure the resulting application

Domain Structural Concepts	Domain specific. Useful within a system, and across knowledge-bases intended to work on the same domain (using the same inference structure)	The structural common to all health-care guidelines	Roughly equivalent to mid-level ontology and distinguished instances. Also includes rules.
Domain Knowledge	Instantiations of domain specific concepts.	A particular health-care guideline	Lower-level ontological distinctions and instances acquired from experts
Case Data		Patient data	

Table 2. The Four Types of Information.

4. THE FIRST PROTÉGÉ

OPAL and ONCOCIN, by adopting the information-partitioning hypothesis and then building so many core concepts into the actual tools, modified the classical model of knowledge-based system development to that shown in Table 3.

Stage	Description	Performed By	Followed By
1. Identification	Characterize important aspects of problem. Identify participants, problem characteristics, resources, and goals.	Tools developers (not done for any specific system)	2
2. Conceptualization	Make key concepts and relations from Identification Stage explicit.	Tools developer (not done for any specific system)	3
3. Formalization	Identified concepts are represented in a formal language.	Knowledge-engineer, Domain experts.	4
4. Implementation	Knowledge from formalization stage is represented in an expert-system shell.	Domain experts	5
5. Testing	The completed system is tested on sample cases and weaknesses are identified	Domain experts, knowledge-engineer	6
6. Revision	Redesign and reimplement the system, in light of the results from testing.	Knowledge-engineers, domain experts	3 or 4 (Depends on what system modifications are needed)

Table 3. The OPAL/ONCOCIN model of Knowledge-Based System Development. Knowledge-engineers have been wholly or partially replaced in many phases of system development.

Protégé began as an attempt to generalize the OPAL/ONCOCIN architecture by reclaiming stage 2 as a part of system development. Instead of hard-wiring the domain structural concepts, Protégé included tools for defining domain structural concepts and then automatically generating an OPAL-like knowledge-acquisition tool for acquiring domain-knowledge.

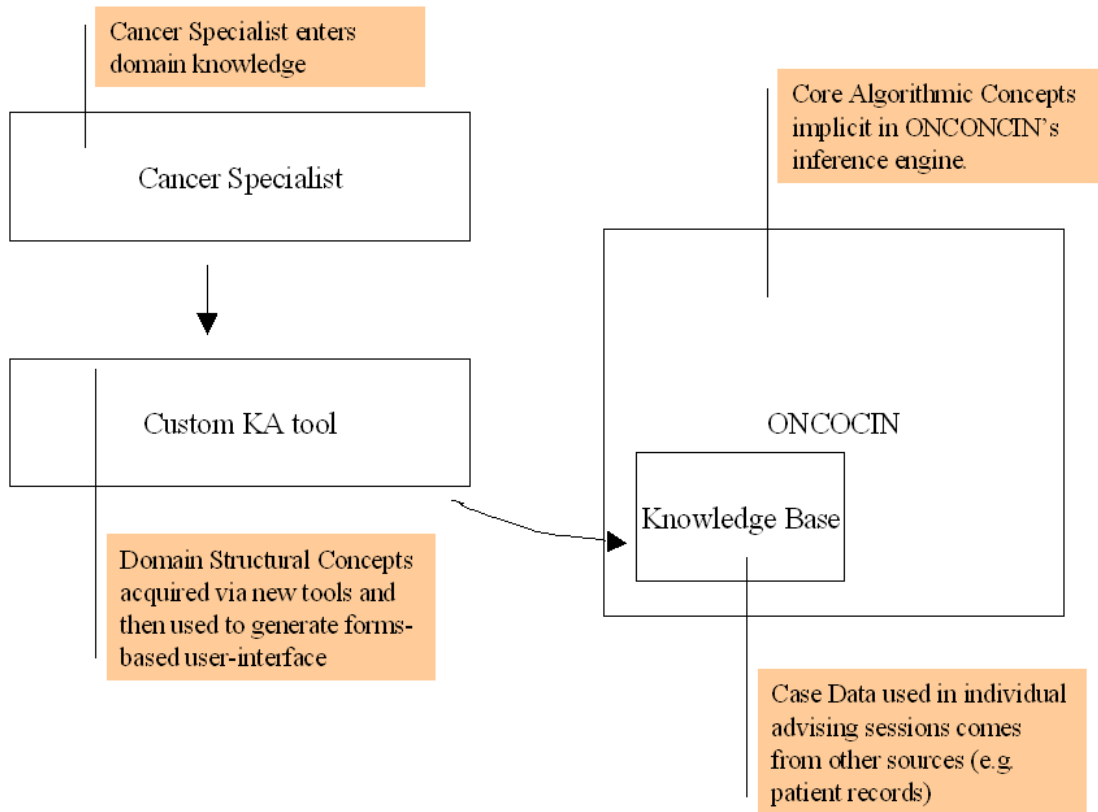


Figure 5. The Architecture of the original Protégé. OPAL has been replaced by a customizable, knowledge-acquisition tool automatically generated from previously-acquired domain structural concepts (see also (Musen, 1989b))

4.1. ASSUMPTIONS IN THE ORIGINAL PROTÉGÉ

In order to accomplish this, and in order to build a system in which the main hypotheses could be quickly tested, (Musen, 1988) made several very strong methodological assumptions which limited the generality of the system. One such assumption has already been discussed: the information-partitioning hypothesis constitutes a major assertion about the nature of knowledge. In the rest of this section, we discuss the other major assumptions made in the design of the original Protégé (while we treat them as distinct assumptions, they're not really independent. It might be more accurate to think of them as aspects of a single "Protégé Assumption")

4.1.1. KNOWLEDGE-BASES ARE PROBLEM-SPECIFIC ARTIFACTS

As has already been discussed, Protégé grew out of ONCOCIN, assumed that the knowledge-bases being constructed were for use with the ONCOCIN inference engine and algorithm (episodic skeletal-plan refinement; see (Tu et al, 1989) for details), and explicitly adopted the information-partitioning hypothesis.

This helped to define a Protégé methodology—knowledge is acquired in stages and the information acquired in each stage is meta-knowledge for the subsequent stages that helps lower the barriers to knowledge-acquisition. Acquiring the domain structural concepts is simplified by the presence of the core structural concepts. And the domain structural concepts are used to automatically generate user-interface forms which make it easier to acquire domain knowledge.

Because it assumes knowledge about the inference process in the knowledge-acquisition process, and because the knowledge-acquisition process is so highly structured, Protégé knowledge bases tended to record only conceptual distinctions used by the inference algorithm (this is mostly a pragmatic decision: people don't often deliberately acquire useless information). Knowledge-bases thus tended to contain only the information required for a specific application that solved a particular task.

4.1.2. FORMAL SEMANTICS ARE IRRELEVANT

In the first Protégé, very little attention was paid to specifying a formal knowledge model. Aside from a very high level description of the idea of a frame (“frames are a common form of knowledge representation in AI”) and a relational database schema given in an appendix, (Musen, 1988) does not specify a modeling language, or provide any sort of model-theoretic semantics (ala (Hayes, 1979) or (Chaudhri et al, 1998) for the acquired “knowledge.” Indeed, given the existence of a unique inference algorithm, it made little sense to do so and Protégé therefore explicitly adopted a operational semantics, relying on the structure derived from the ONCOCIN inference engine to give meaning to the knowledge.

4.1.3. ATOMIC KNOWLEDGE BASES

The final major assumption made by the developers of the first Protégé was that knowledge-bases were atomic. That is, the acquired knowledge-bases were entirely self contained (they didn't reference any other knowledge-bases or knowledge-sources at all).

4.2. HISTORICAL COUNTERPARTS

Protégé was not the only knowledge-acquisition tool built in the mid-1980's that attempted to use knowledge about the target inferencing algorithm to simplify knowledge acquisition. Systems such as MOLE (Eshelman et al, 1987) and SALT (Marcus and McDermott, 1989) made similar assumptions, also with the goal of simplifying knowledge acquisition.

However SALT and MOLE (and their ideological descendants, such as Expect (Gil and Melz, 1996) used this knowledge in a very different way from Protégé. Consider the following statement from (Eshelman et al, 1987).

MOLE understands that experts are not very good at providing such information and so does not require that the expert provide a fully specified network of associations. Instead, MOLE relies on its expectations about the world and how experts enter data in order to mold

an under-specified network of associations into a consistent and unambiguous knowledge-base.

MOLE uses its meta-knowledge about the world, and about the inference algorithm, to interpret, correct, and complete knowledge entered by an expert.

This is in marked contrast to the Protégé approach. Where systems like MOLE use meta-knowledge to try and validate a knowledge base, Protégé uses meta-knowledge to make it easier for a domain-expert to understand the contents of a knowledge-base.

4.3. SUMMARY OF THE ORIGINAL PROTÉGÉ

Protégé succeeded in substantially lowering the barriers to knowledge acquisition for medical advice systems. But the knowledge-bases in these systems weren't very reusable—it would be amazing if a task-specific knowledge base which only makes the distinctions that are important to a specific algorithm and has no formal semantics *wasn't* difficult to reuse.

Moreover, the assumptions made during system development had the effect of limiting Protégé to small, already-structured, domains where the knowledge and reasoning processes were well-understood by experts. All of which meant that the early versions of Protégé, while successful in lowering the barriers to knowledge-acquisition, had, in some sense, simply moved the bottleneck (from knowledge-acquisition to knowledge reuse).

The state of Protégé at the end of this first cycle of development is summarized in table 4

System	Goals	Assumptions	Historical Counterparts
Protégé	Generalize OPAL/ONCOCIN architecture by automatically generating OPAL-like interfaces from domain structural concepts	Information-partitioning hypothesis, problem/task-specific knowledge-bases, informal semantics sufficient, atomic knowledge bases	MOLE, SALT

Table 4. Summary of the Original Protégé.

5. PROTÉGÉ-II: PROBLEM-SOLVING METHODS AND THE DOWNHILL-FLOW ASSUMPTION

Early experiments in reuse attempted to reuse general-purpose inference engines (van Melle et al, 1984). This experience, and later experience in adapting the MYCIN knowledge-base to teach medical students diagnosis (Clancey, 1979), helped researchers realize that the production rules in an expert system can be divided into several levels. There are basic rules that concern domain concepts and domain knowledge. But there is also meta-knowledge: there are rules about other rules, embodying control-knowledge (when certain rules are applicable) and meta-knowledge that can be used for explanations and justifications.

At about the same time as the development of the original Protégé, researchers in Artificial Intelligence, building upon this realization, and the ideas in (Clancey, 1985), had begun talking about, and formalizing, the idea of libraries of inference algorithms which could be easily reused across a number of different applications (see (McDermott, 1989), (Wielinga et al, 1992)).

Rather than include a set of rules, and the basic concepts related to them, in the knowledge base, researchers began to explore the idea of putting the inference mechanism in an entirely separate component, which could be developed independently from the knowledge-base.

Another way to understand this distinction is to notice that, in early versions of Protégé, knowledge-bases are constructed for a specific task-oriented inference structure. By explicitly separating domain modeling from the requirements of a specific inference structure (rechristened as a *problem-solving method*), researchers hoped to be able to reuse both more easily.

A basic example of this style of reuse is given in (Gennari, Altman, and Musen, 1994). A simple constraint satisfaction algorithm, Propose-and-Revise, was implemented as part of the Sisyphus-II project on elevator configuration (Rothenfluh et al, 1993) and then reused in applications dealing with ribosomal configurations. However, in order to achieve this capability, we needed to make a number of changes and additions to the original Protégé system.

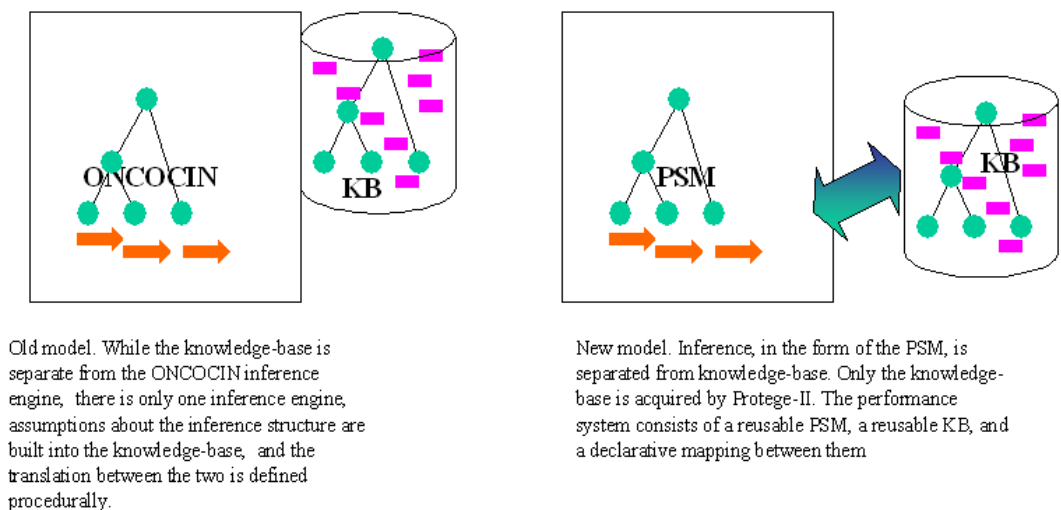


Figure 6. Old and New Models of System Development

5.1. CHANGES IN PROTÉGÉ AND THE PROTÉGÉ METHODOLOGY

Protégé -II therefore began a process of explicitly separating the problem-solving method from the domain knowledge. This adoption of the idea of a library of problem-solving methods as the way to build systems brought about several major changes in the Protégé methodology. The system development task (outlined in Table 1) was divided into three separate tasks (building a problem-solving method, building a knowledge base, and building an application). Moreover, separating the problem-solving method from the domain knowledge introduces translation task: the domain knowledge needs to be somehow put into a form which the problem-solving method can use. This led to the development of a more formal frame language (MODEL) for domain knowledge and the idea of mappings which define the relationship between the domain knowledge and the problem-solving method.

In the remainder of this section, we discuss these methodological changes more thoroughly and the changes they caused in the design of Protégé-II.

5.1.1. MODIFYING THE SYSTEM DEVELOPMENT PROCESS

The 6-step knowledge-acquisition process outlined in Table 1 reflects a conflation of system-building with knowledge-base construction. The Protégé-II methodology replaced this single process by three activities, two of which aimed at building reusable components. These new stages are: building a problem-solving method, building a knowledge-base, and building a knowledge-based system.

5.1.1.1. Building a Problem-Solving Method

This Protégé -II model of system construction draws a lot of its appeal from the idea that inference algorithms are highly reusable and stable. While domain knowledge can quickly become out of date and can require substantial modifications, inference algorithms such as Propose-And-Revise or Cover-and-Differentiate seem well-understood and unlikely to change. Thus, this stage, where a well-known inference algorithm is analyzed and put into reusable form, is a one-time expense for each algorithm. As more problem-solving methods are built, this phase will require asymptotically less effort

Stage	Description	Performed By	Followed By
1. Definition of core algorithmic concepts and knowledge requirements	Fundamental structural concepts belonging to generic problem-solving method (for example, skeletal planning) conceptualized and represented	Protégé-developers	2
2. PSM Formalization	Identified concepts are represented in a formal language.	Protégé-developers	3
3. PSM Implementation	Underlying inference algorithm is written in a programming language.	Protégé-developers	4
4. Testing	The completed PSM is tested on sample cases and weaknesses are identified	Protégé-developers	5
5. Revision	Redesign and reimplement the PSM, in light of the results from testing.	Protégé-developers	1-4

Table 5. The Protégé Model of PSM Development.

5.1.1.2. Building a Knowledge-Base

Traditionally, Protégé knowledge-bases are built when they are needed, by knowledge-engineers and subject matter experts. However, once such a knowledge-base exists, if a related knowledge-base is needed (a knowledge-base containing similar information), it can be built either as an extension of the first knowledge-base or independently. In either case, a process of merging and aligning (much like that discussed in (Noy and Musen, 1999)) eventually results in a generic and reusable knowledge-base which has been tested by several applications.

Most of the tools produced in the Protégé project deal with this phase of the system-development process.

Stage	Description	Performed By	Followed By
1. Conceptualization	Make knowledge explicit.	Domain experts	2
2. Formalization	Identified concepts are represented in a formal language.	Domain experts	3
3. Data Entry	Data is entered	Domain experts and domain-users.	4
4. Validation and Verification	The knowledge-base is verified by domain experts, both for correctness and completeness.	Domain experts	1-3, 5
5. Maintenance	Update the knowledge-base over time, in response to changes in the world (or our conception of it)	Domain experts and domain-users	1-5

Table 6. The Protégé Model of Knowledge-Base Development.

5.1.1.3. Building a Knowledge-Based System

Finally, after the constituent components have been created (and, quite possibly, tested independently), the actual performance system must be assembled. One compelling goal is the partial-automation of this phase: given a psm-library, a knowledge-base, and a description of the task, it ought to be possible to, given a description of the task, generate the mappings. This boils down to two related problems: figuring out which PSM to use and automatically generating the mappings. The first of these has been the focus of a large amount of recent research in the knowledge-acquisition community (see (Fensel et al, 1999) for an overview and an extensive bibliography).

Stage	Description	Performed By	Followed By
1. Define problem	Characterize important aspects of problem. Identify participants, problem characteristics, resources, and goals.	Domain experts	2
2. Obtain knowledge-base	Make knowledge explicit.	Domain experts	3
3. Choose PSM		Domain experts	4
4. Connect components	Build mappings	Domain experts assisted by knowledge-engineer	5
5. Testing	The completed system is tested on sample cases and weaknesses are identified	Domain experts and domain-users, knowledge-engineer	8

6. Revision	Redesign and reimplement the system, in light of the results from testing.	Domain experts assisted by knowledge-engineer	2 - 5 (Depends on what system modifications are needed)
-------------	--	---	---

Table 7. The Protégé-II Model of System Development.

5.1.2. PROBLEM-SOLVING METHODS AND MAPPINGS

In this new model of system construction, there must be a way to connect the problem solving method to the knowledge-base. In Protégé-II, problem-solving methods were defined to have a *method ontology* to facilitate this task. That is, a problem-solving method consisted of an algorithm and a set of class definitions which serve to define the both the input and output data formats for the algorithm.

A mapping, in the Protégé-II methodology, consists of a declaratively specified transformation, from the domain structural concepts to the method ontology. A slight extension of the information-partitioning hypothesis implies that this mapping will automatically induce a transformation on domain knowledge and data, turning the instances in the knowledge base into instances which the PSM understands.

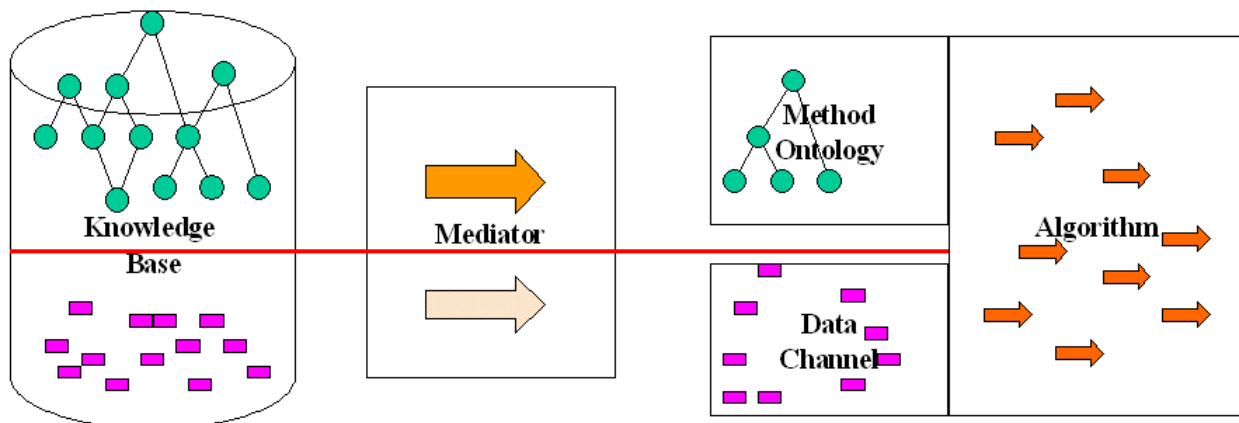


Figure 7. Mappings rely on the information-partitioning hypothesis (the center line). The map is defined from structural concepts (from the knowledge base's ontology to the method ontology) .

5.1.3. MODEL: ADOPTING A SEMI-FORMAL REPRESENTATION LANGUAGE

Another consequence of allowing alternative problem-solving methods (moving the problem-solving methods out of the knowledge-base and attempting to build reusable knowledge-bases) is that the implicit procedural semantics of the original Protégé system, based on an intended use and a specific algorithm, no longer exist. Since reusable knowledge-bases must be recognizable as knowledge-bases and have a defined and discernable meaning, it seems necessary to mimic mathematical logic and supply both a syntactical definition (the analog of logic's wffs) and a model-theoretic semantics (see, for example, (Hayes, 1974) and (Hayes, 1979)). Protégé-II took an important first step in this direction with the introduction of the MODEL representation language (Gennari, 1993).

MODEL was a frame-based extension of the language used in the CLIPS 5.0 production system. As such, it supplied a syntax for knowledge-bases as standalone entities. MODEL also had a partial procedural semantics (interpretations were supplied both by the implied map into CLIPS and from the ways in which Protégé-II manipulated the frame-structures).

Even though it was mostly a syntactical specification, the use of MODEL entailed four significant benefits:

1. It enabled the separation of problem-solving methods from knowledge bases by defining a common data format.
2. It clarified the conflation between classes/instances and the knowledge/data distinction (which was, until then, mostly a background assumption)
3. It enabled the definition of the MARBLE tool and an associated language for declarative specification of mappings ((Gennari, Tu, Rothenfluh and Musen, 1994). This work was later extended in (Park et al, 1997)).
4. It enabled early experiments in knowledge-base reuse across knowledge-modeling frameworks. For example, the Ontolingua system (see (Gruber, 1991) and (Gruber, 1993)) included a Protégé translator. While this was mostly a curiosity, it nonetheless underscored the need for a richer semantic model and helped pave the way for Protégé-2000's adoption of the OKBC knowledge model.

5.2. NEW ASSUMPTIONS IN PROTÉGÉ-II

5.2.1. “DOWNHILL FLOW” DURING KNOWLEDGE ACQUISITION

MODEL was based on classes and instances and implicitly cordons off the classes entirely-- classes only refer² to classes (and primitive types) and instances, once their class-type is specified, only refer to instances (and primitive types). This language feature, while not originally part of the theoretical foundation of Protégé, was quickly absorbed into the Protégé methodology as part of a general “downhill flow” model of information. That is, the following three aspects of the Protégé-II methodology were tentatively identified as three ways of making the same distinction:

1. Domain structural concepts affect the forms that are generated and therefore can change (e.g. invalidate or alter) domain knowledge.
2. Changes to a class definition affect the instances of that class in a knowledge base, but changes to an instance do not affect the class definition at all.
3. It is very easy to associate a form to a class and then “acquiring an instance” is simply “filling out the blanks on the form.”

Consequently the Protégé-II methodology associated domain structural concepts with ontologies (e.g. sets of classes), each concrete class had an associated form, and acquiring domain

² E.g. “have a pointer to” or “have a slot pointing to”

knowledge was synonymous with creating instances of classes (which was done by filling out the form associated with the class).

This lends a great deal of structure to the knowledge base, and to the knowledge-acquisition process. But it also rules out a fair number of modeling paradigms, including some of the basic examples from the frame-based literature. Consider, for example, the first three "major intuitions" stated in section 1 of (Bobrow and Winograd, 1977)

Knowledge should be organized around conceptual entities with associated descriptions and procedures.

A description must be able to represent partial knowledge about an entity and accommodate multiple descriptors which can describe the associated entity from multiple viewpoints

An important method of description is comparison with a known entity, with further specification of the described instance with respect to the prototype.

An approach which features partial information, multiple views on the same entity, and a reliance on prototypes, is not easily accommodated within Protégé's strict segregation of domain structural concepts and domain knowledge (and the subsequent conflation of these with classes and instances).

5.3. HISTORICAL COUNTERPARTS

PROTÉGÉ -II was being developed at the same time that the KADS methodology was being popularized in Europe (Wielinga et al., 1992). Whereas KADS was a very large effort that attempted to address the entire knowledge-engineering enterprise, PROTÉGÉ -II concentrated primarily on development of what KADS referred to as "the model of expertise." Domain ontologies in PROTÉGÉ -II corresponded to the "domain layer" in KADS; PSMs in PROTÉGÉ -II reflected knowledge modeled at both the "inference layer" and the "task layer" in KADS. KADS and PROTÉGÉ -II were very compatible in philosophy. A principal difference, however, was that in PROTÉGÉ -II the model of expertise also served as a design model. The emphasis on using PROTÉGÉ -II to build operational knowledge-based systems directly required that the building blocks for the model of expertise (domain ontologies and PSMs) translate into actual program code that could implement the corresponding system. PROTÉGÉ -II consequently made pragmatic choices regarding the structuring of conceptual models, and offered developers fewer distinctions for fashioning the model of expertise than were available in the KADS methodology.

At around the same time, Steels (1992) was implementing the KREST workbench, based on his view of "components of expertise" (Steels, 1990). KREST had similarities to PROTÉGÉ -II, including the use of domain ontologies (models) and PSMs as the basic building blocks. Although PROTÉGÉ -II shielded the developer from having to understand the programming details of domain ontologies and PSMs, KREST required system builders ultimately to understand how the building block were implemented in common LISP. The abstract model of

PSMs implemented in KREST later would become an important element of the common KADS methodology.

In the United States, workers at the University of Southern California's Information Sciences institute began to experiment with the incorporation of discrete PSMs within systems based on description logic (Gil and Melz, 1996) The PSMs included within the EXPECT architecture tended to be more domain-specific and of smaller grain size than those used within PROTÉGÉ - II, but the approach corroborated the utility of using PSMs as an abstraction for control knowledge in large knowledge-based systems. EXPECT also demonstrated the value of description logic in facilitating the acquisition of knowledge-base instances.

5.4. SUMMARY OF PROTÉGÉ-II

Protégé-II was an attempt to make knowledge-bases more reusable by removing knowledge about the problem-solving method from the knowledge base. The approach taken involved formally modeling problem-solving methods and then use the method ontologies to define mappings. This approach has remained an active area of research in the knowledge-acquisition community (Fensel et al, 1999).

An interesting aspect of this phase of Protégé development is that, in retrospect, all the major changes involved moving from an informal model to a more formal one. The central change, the removal of algorithm dependencies from the knowledge-base, requires a more formal knowledge model, leads to a deeper understanding of the knowledge-base structure required by the problem-solving methods, and spurs the development of mappings.

The state of Protégé at the end of this second cycle of development is summarized in table 8

System	Goals	Assumptions	What can be reused when developing new systems ?	Historical Counterparts
Protégé	Generalize OPAL/ONCOCIN architecture by automatically generating OPAL-like interfaces from domain structural concepts	Information-partitioning hypothesis, problem/task-specific knowledge-bases, informal semantics sufficient, atomic knowledge bases	User skills and conceptual models	MOLE, SALT
Protégé-II	Make knowledge-bases more reusable. Generalize Protégé by removing knowledge-base dependencies on a particular problem-solving method.	Information-partitioning hypothesis, atomic knowledge bases, downhill-flow model of knowledge acquisition	Problem-solving methods and knowledge-bases (within a Protégé-centered development environment)	Components of Expertise, KADS and EXPECT

Table 8. Summary of the Original Protégé and Protégé-II.

6. PROTÉGÉ/WIN: MAKING KNOWLEDGE BASES INTO COMPONENTS

Protégé-II removed task and problem-solving method dependencies from domain knowledge-bases, thereby taking a major step towards reuse of both domain knowledge-bases and problem-solving methods. The next step in the evolution of Protégé combined a pragmatic change, moving from the near-total isolation of a software system that only ran on NeXT workstations³ to a Windows-based system, with a number of changes aimed at making knowledge-bases more modular.

The need for modular knowledge-bases was driven, as most changes in Protégé have been, by the desire to reuse the artifacts created by knowledge-engineers. Many of the medical knowledge-bases built over the first 10 years of Protégé shared a large percentage of their domain structural concepts (the second type of information in table 2). Most medical knowledge bases, whether they be for use within EON (Musen et al, 1995) or RESUME (Shahar and Musen, 1993) or some other knowledge-based medical system, share notions such as *patient*, *disease*, and *drug*.

6.1. CHANGES IN PROTÉGÉ AND THE PROTÉGÉ METHODOLOGY

6.1.1. INCLUDABLE ONTOLOGIES

This observation, that there is a common set of abstractions that many related knowledge-bases have in common is, of course, hardly original to the Protégé team. Unfortunately, it is such an obvious observation that it immediately leads into theoretical difficulties.

On the one hand, partisans of large knowledge-bases (such as (Guha and Lenat, 1990), (Lenat, 1995)) make a convincing argument that large knowledge-bases are essential for truly flexible and robust applications. On the other hand, many seemingly insurmountable difficulties stand in the way of creating such generic knowledge (the objections range from situated-cognition style arguments ala (Clancey, 1997) to the discussion of the Interaction Problem in (Bylander and Chandrasekharan, 1988).

Stanford's Knowledge Modeling Group, unable to conclusively adjudicate the theoretical claims, decided to make domain structural concepts (the second type of information in table 2), contained in an ontology of class definitions, into components that could be included in another ontology (and hence, in another knowledge base). This solution had many, admittedly pragmatic, virtues:

- It enabled conceptual reuse in the domains where reuse was immediately seen to be useful.
- It opened the door to large knowledge-bases built using mediators and “gluing” small knowledge-bases along their common ontologies (along the lines of the vision articulated in (Wiederhold, 1994)).
- The current state of artificial intelligence⁴ strongly implies that if we want knowledge-bases to be computationally amenable, they must be rather small

³ Total sales over a 7 year period: under 50,000. Currently for sale on e-Bay: 3

⁴ A fast microprocessor doesn't help when algorithms are exponentially slow.

- The current state of experts strongly implies that if we want experts to be able to maintain and understand the contents of knowledge-bases, we shouldn't overwhelm them with abstractions unrelated to their domain of expertise.

An example of this style of conceptual reuse is the GuideLine Interchange Format (GLIF) (Ohno-Machado et al, 1998). GLIF is an ontology of concepts used in medical guidelines⁵. It includes such generic concepts as *Guideline*, *Guideline Step*, and *Criterion*.

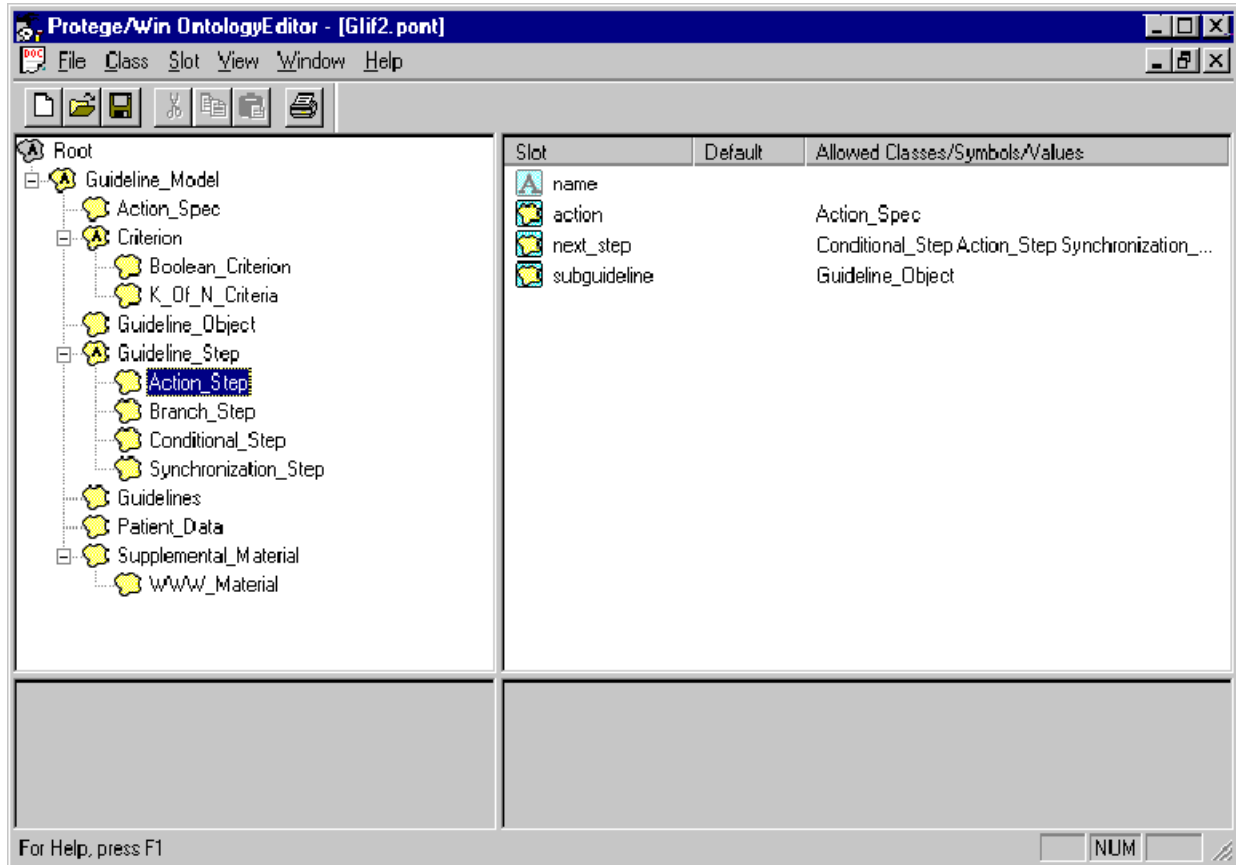


Figure 8. The GLIF Ontology, in Protégé/Win.

Domain experts working on a specific class of guidelines (such as the work on breast cancer protocols described in (Fridsma et al, 1996)) include the GLIF ontology and specialize it by adding more precise and specialized concepts, such as *Toxicity_Check* and *Radio_Therapy*.

⁵ A medical guideline is a plan for patient treatment, including conditional tests and courses of action.

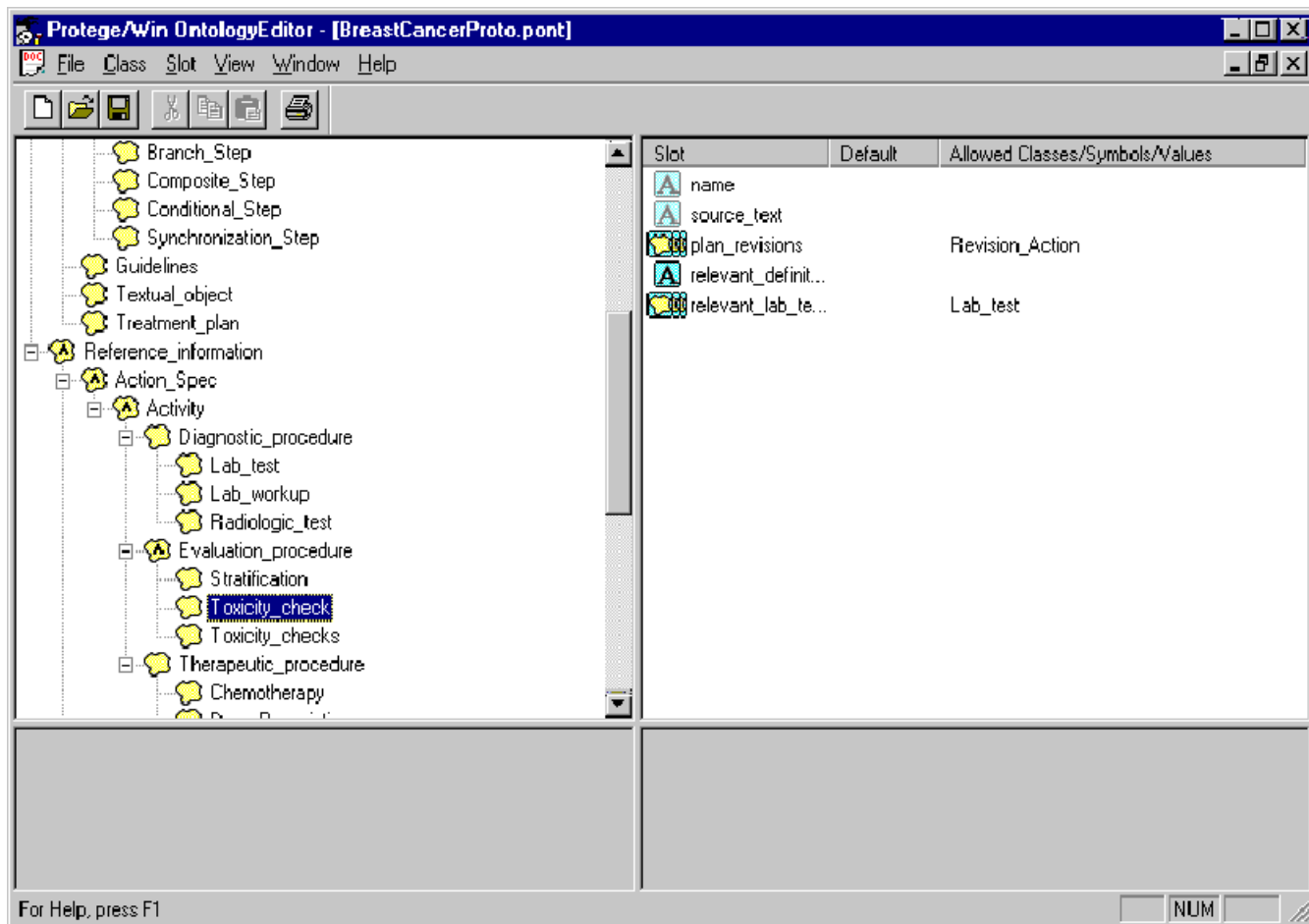


Figure 9. The Breast Cancer Ontology, in Protégé/Win. The Breast Cancer Ontology is approximately 3 times the size of the core GLIF ontology.

6.1.2. THE KA-TOOL: REIFYING THE NOTION OF FORMS

Protégé-II removed assumptions about problem-solving methods, and the associated algorithmic notions, from domain knowledge-bases. However, Protégé-II knowledge bases still contained a large amount of Protégé-specific information. In particular, the Protégé-II knowledge-model contained a number of idiosyncratic facets that aided knowledge-acquisition. For example, the *ka-subwindow* facet (Gennari, 1993) is a template-facet that allows the user-interface generation tool to break the form for acquiring an instance into multiple forms (this can be useful when a class has a large number of slots) and the *ka-specification* facet is used to ensure that certain slots don't appear on forms.

As long as knowledge-bases were atomic, and as long as knowledge-bases were reused as a consistent whole for a fairly restricted purpose, knowledge-acquisition facets are a reasonable, if inelegant, way to specify layout information. In particular, they are consistent with the downhill-flow assumption—if you associate a form to each class, then you need to associate form layout information to each class as well. The ontology is a logical place to store such information.

However, reusable ontologies make knowledge-acquisition facets problematic. If an ontology is going to be used in more than one knowledge base, the associated forms might be different in the different knowledge bases (the user interface is more specialized than the concept definitions). This implies that knowledge-acquisition information should not be stored in the ontology, as facet values.

Accordingly, Protégé/Win makes a distinction between the knowledge in a knowledge-base (the instances) and the implicit project (which consists of the instances, the ontology, the inclusion and namespace information, and the presentation information) and supplies a completely separate editor, called the forms-layout editor, for presentation information⁶. The forms-layout editor was based on the following principles:

1. Every concrete class in a knowledge-base has an associated a form
2. Every slot on a class is associated with a widget on the form (the widget displays the slot values and allows slot values to be edited).
3. Customization consists of two operations: associating widget to slots (there is a predefined list of appropriate widget choices for each slot type) and arranging the widgets on the form (placing and sizing the widgets)
4. If the user doesn't define a form, one will automatically be generated using general-purpose heuristics (both default widget-associations and rules of the form "all slots of a given type are grouped together").

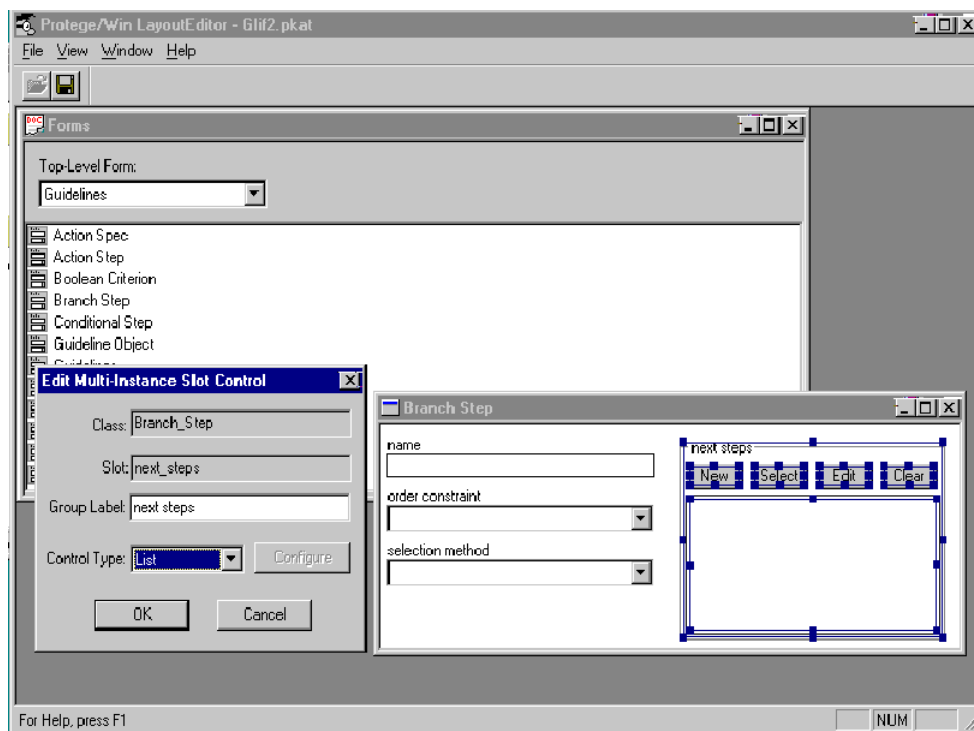


Figure 10. Editing the Form Associated to the Branch_Step class in the GLIF Ontology (in Protégé/Win).

⁶ A version of this, named DASH and based on ka-facets, actually existed in Protege-II.

These principles introduced another implicit assumption: that forms which are constructed out of a small set of pre-enumerated widgets and rely on structured data entry suffice for knowledge base creation and visualization are sufficient for knowledge-acquisition. This assumption was, however, immediately recognized as a limitation of the software (which would be corrected in Protégé-2000).

6.2. HISTORICAL COUNTERPARTS

At the time that Protégé /Win was developed, common KADS and VITAL were highly visible projects in Europe. The common KADS notions of domain ontologies, PSMs, and mappings resonated very well with the same concepts in Protégé /Win. Like the original KADS approach, common KADS has been essentially a paper-and-pencil methodology--one that Protégé /Win could in principle support in an automated fashion. Common KADS made certain distinctions about PSMs that, for pragmatic reasons, were a bit blurred in Protégé /Win; because the model of expertise is equivalent to the design model in Protégé /Win, this pragmatism seemed quite appropriate.

Concomitantly, the VITAL project added the novel notion of a "generalized directive model" (GDM) to provide a grammar for composing knowledge-based system building blocks. The VITAL project did develop a usable knowledge-acquisition workbench for implementing real systems. That workbench, however, has now been supplanted by a commercial product that currently does not include the GDM library that distinguished the academic research on VITAL.

6.3. SUMMARY OF PROTÉGÉ/WIN

The state of Protégé at the end of this third cycle of development is summarized in table 9

System	Goals	Assumptions	What can be reused when developing new systems ?	Historical Counterparts
Protégé-II	Make knowledge-bases more reusable. Generalize Protégé by removing knowledge-base dependencies on a particular problem-solving method.	Information-partitioning hypothesis, atomic knowledge bases, downhill-flow model of knowledge acquisition	Problem-solving methods and knowledge-bases (within a Protégé-centered development environment)	Components of Expertise, KADS and EXPECT
Protégé/Win	Make knowledge bases more reusable and maintainable by making them into modular components that can be included in one another. Make software tools more usable by porting them to a standard platform	Information-partitioning hypothesis, downhill-flow model of knowledge acquisition, small-widget set assumption	The above, plus domain structural concepts in the form of includable ontologies.	COMMONKADS, VITAL

Table 9. Summary of Protégé at the end of Protégé/Win

7. PROTÉGÉ-2000: THE CURRENT PLATFORM

Protégé-Win proved to be a useful tool for building models of small domains and experimenting with various types of knowledge-based systems (typical examples include (Shahar and Musen, 1993) and (Stein et al, 1996)). The assumptions that remained from the original Protégé (the information-partitioning hypothesis and the downhill-flow assumption) weren't particularly limiting as long as Protégé was used within a problem-oriented context (e.g. when building knowledge-bases for a specific purpose). In fact, at the end of 1998, Protégé/Win had a user base of approximately 300 researchers doing just that, in applications that ranged from the traditional (Johnson et al, 1999) to the newfangled (Fensel et al, 1999).

But along with a burgeoning user base come new requirements, and a new definition of what a knowledge-modeling tool ought to be able to do. In particular, Protégé/Win suffered from three very significant limitations:

1. The standard set of user-interface widgets is too limited for many envisioned uses of Protégé. The standard widgets are generic widgets, usable across a wide variety of domains. They are useful, and often suffice, but experience (for example, experience gained in the RESUME system (Shahar, 1997)) has indicated that domain-specific widgets might be useful as well.
2. Interoperability with other modeling frameworks was limited by the difficulty of translating knowledge-bases in and out of MODEL
3. MODEL wasn't flexible enough for many domains. In particular, it lacked support for own slots on classes, and support for terminological representations (ala the formalisms used in systems like GALEN (Rector et al, 1995)).

Moreover, as the focus of the knowledge-modeling group changes from building small task-oriented knowledge-bases towards confronting the problems inherent in building and maintaining larger knowledge-bases, these problems become more significant.

7.1. CHANGES IN PROTÉGÉ AND THE PROTÉGÉ METHODOLOGY

7.1.1. ADOPTING THE OKBC KNOWLEDGE MODEL

In the 1980's the knowledge-representation community, building on the idea of design-tradeoffs (a classic discussion is contained in (Levesque and Brachman, 1984), thoroughly explored a large number of different knowledge-representation languages.

In the early 1990's, a series of papers (foremost among them (Neches et al, 1991) and (Karp, 1993)) pointed out that researchers had also, and just as thoroughly, fragmented the knowledge that was being acquired. The barriers to reusing a knowledge-base in a system with a different representation language were formidable, even if both representation languages were "frame-based." And, similarly, tools to edit and manipulate knowledge were dependent on particular representation systems and specific APIs.

OKBC ((Fikes and Farquhar, 1997), (Chaudhri et al, 1998)) is an attempt to partially overcome these difficulties by providing both a knowledge model, based upon a mapping of typical frame-based notions into first-order logic (similar to the approach used in (Hayes, 1979) and a rich and expressive API for programs and components to interact with knowledge-base servers. By adopting the OKBC knowledge model⁷, Protégé gains three major advantages

1. *Greater Expressivity.* While MODEL contains some second-order predicates and facets not supported by OKBC (such as the *role* facet described in (Grosso et al, 1998)), the OKBC Knowledge model allows a much wider range of modeling conventions and idioms. Features such as reified slots and explicit support for meta-classes have been enthusiastically received by many of the more experienced Protégé users.
2. *Clean Model-Theoretic Semantics.* The OKBC knowledge-model is defined using mathematical logic. While it is possible to supply such a mapping for MODEL ((Grosso et al, 1998) does this by defining MODEL as a layer on top of OKBC), the OKBC knowledge model builds upon years of knowledge-representation experience across a wide range of systems.
3. *The Possibility of Reuse.* The vision articulated in (Fikes and Farquhar, 1997) of distributed ontology servers is persuasive. While the set of publicly available Ontolingua ontologies is still fairly small, experiments in reuse (such as (Uschold et al, 1997) and work done for DARPA's High Performance Knowledge Bases project⁸) are promising.

7.1.2. SUPPORT FOR COMPONENTS

One of the major goals of Protégé-2000 is to make it easier for modelers to customize and extend Protégé in task and domain specific ways. Moreover any such extensions should, themselves, be easily reused and extended— in principle there is very little difference between the following two widgets:

- **Annotated X-Ray Widget:** Displays an image, associated to an X-Ray and allows users to circle regions and annotate them with instances of classes in the Oncology ontology. Later users can, by clicking on a circled region, view the instances.
- **Roman Temple Widget:** Displays images of various temples and, when the user clicks on a part of the temple, brings up an instance from the knowledge base describing the item that was clicked on.

If the process for extending Protégé is structured enough, and well-documented, then it becomes possible to establish a repository of such extensions, that all users of Protégé-2000 could use (both directly and as models for further extensions).

⁷ Actually a superset of it. Protégé-2000's modeling language also contains some second-order predicates and a full-fledged constraint language ([Grosso et al, 1999]).

⁸ See <http://www.teknowledge.com/HPKB/meetings/meet040799/Musen1/index.htm> for a preliminary report

In order to make this possible, Protégé-2000 is built as a system of 3 layers. Each layer communicates with the layers beneath it via well-defined APIs, and the explicit adoption of the OKBC knowledge model gives the APIs a strong semantic basis.

Layer	Primary Purpose	Default Configuration	How Extensions are Possible	Standard Extensions
Widget Layer	User-interface components that allow a small slice of the knowledge-base to be viewed and edited	Protégé-2000 comes with a standard set of widgets, roughly equivalent to those in Protégé/Win	Knowledge-engineers can easily create new widgets, either by subclassing <i>AbstractWidget</i> or writing an object that implements the <i>Widget API</i> .	PAL, the Protégé Axiom Language, is implemented as a tab widget. As are a variety of simple examples to demonstrate use of the API.
Control Layer	The “plumbing” for the system. Handles standard actions (things typically found on menus) and handles the connections between widgets and underlying knowledge-base as well as things like value-caching (for user interface performance)		None. This layer is neither customizable nor replaceable.	
Knowledge-Base Server	A wrapper around the actual knowledge-base server. There are a set of well-defined model interfaces that specify what must be implemented	The in-memory knowledge-base server is a single-user server that performs most operations in memory and uses a custom-file format for persistence.	Implement the IKB interface, translating IKB functions into the underlying server calls. Note that the adoption of the OKBC knowledge model makes a wide variety of back-end servers accessible	Protégé-2000 also has a simple wrapper around a relational database (This is a proof of concept, and a demonstration of how to integrate a knowledge-base server, into Protégé-2000, and does not take advantage of the full power of modern relational systems)..

Table 10. The Layers of Protégé-2000

Protégé-2000 is built, from the ground up, out of replaceable, and interchangeable, components. Researchers and knowledge-engineers can add new widgets, change knowledge-base servers, reuse parts of the Protégé framework in custom applications, or even alter the underlying knowledge model with very little difficulty. Indeed, at this point, it might be more accurate to think of Protégé-2000 as a set of knowledge-base components loosely joined together in a default configuration.

Adopting a layered approach, and building Protégé-2000 as a loose aggregate of components, enables Protégé-2000 to realize 5 distinct types of benefits:

1. *User-interface benefits.* Custom user-interface widgets can be built quickly and are easy to add to a knowledge base. This makes it easier to achieve the original Protégé goal of removing knowledge-engineers from the knowledge-base construction and maintenance loop.
2. *Interoperability and scalability benefits.* Protégé-2000 can be used as the “front-tier” of a client-server system, using a knowledge-base server residing on another machine. This opens a wide-range of possibilities, from simultaneous editing of a knowledge-base to the use of legacy servers (via OKBC).
3. *Maximizing reusable user skills.* The original motivation for GFP (Karp, 1995) and one of the primary motivations for the adoption of OKBC, is that OKBC allows applications to work across multiple knowledge-base servers. Protégé-2000 shows similar promise for people, in that, to the extent that knowledge-engineers write adapters to the various knowledge-base servers, Protégé-2000 will become a universal user interface.
4. *Embedding knowledge-base technology inside other applications.* This is an interesting dual possibility to (2). Protégé knowledge-bases and software components can be embedded as part of a larger application which makes little or no use of the Protégé interface components.
5. *Scientific experiments.* This is perhaps the most exciting new possibility that Protégé-2000’s component architecture opens up. The ability to add and remove components from a knowledge-base system opens up the possibility of fine-grained testing. This holds both for ideas about how users interact with knowledge-based systems and for more general claims about knowledge-modeling and software-engineering.

One of the great untested claims of the knowledge-modeling literature is that building large “upper ontologies” and enabling knowledge-engineers to reuse them in domain specific knowledge-bases them, will greatly reduce system development time and result in more robust and flexible applications. To the extent that we lower the barriers to knowledge-reuse, this claim might actually become testable.

7.1.3. EXTENDING THE IDEA OF KNOWLEDGE BASES

In Protégé/Win, in order to construct a knowledge-base, a knowledge-engineer had to build (or otherwise acquire) three things: An ontology (which could include other ontologies), a set of instances, and layout information for the forms used in knowledge-acquisition.

Protégé-2000 introduces the explicit notion of a *project*. Projects contain a knowledge-base and configuration information. Because there is no longer a clear-cut distinction between classes and instances, the knowledge-base is simply a collection of frames(it also contains things like reified slots, facets, and axioms). And they also contain layout information for the forms used in knowledge-acquisition. The configuration information contains descriptions of all the widgets

that have been added to the project, information about the knowledge-base server being used, and a list of all the projects that are included by the current project.

This last item, inclusion of projects rather than of ontologies, is interesting for a number of reasons. It was forced into Protégé-2000 by the new knowledge-model—when the distinction between classes and instances has been blurred, it's not entirely obvious what “including an ontology” even means. And it clearly has pragmatic benefits—instances which are domain knowledge (the third type of information in Table 2) and forms-layout information are things that people want to reuse.

But it also underscores the need for careful experimentation. While including projects instead of ontologies seems like a small change to make, and an obviously desirable one, some knowledge-engineers report that the way in which they build knowledge-bases has changed. In particular, that they are more likely to create concepts and instances that aren't immediately useful but probably will be needed later on. If this is true, it raises interesting questions about how the choice of a modeling tool affects the final knowledge-base. For example:

- Are Protégé-2000 knowledge bases really larger over the long run?
- Where is the additional information? Are the knowledge-bases somehow less cohesive?
- Will this impact the ability of domain-experts to validate and maintain knowledge-bases? If so, how do we minimize⁹ the impact?
- Are these new knowledge-bases better than ones created using Protégé/Win?

7.1.4. MODIFYING THE “DOWNHILL FLOW” ASSUMPTION

The downhill-flow assumption was an integral part of the Protégé-II and Protégé/Win methodologies. It built on the strong distinction made between classes and instances in MODEL and attempted to structure the process of building a knowledge base.

⁹ Or maximize. It's entirely possible that the new knowledge-bases are easier for domain experts to comprehend.

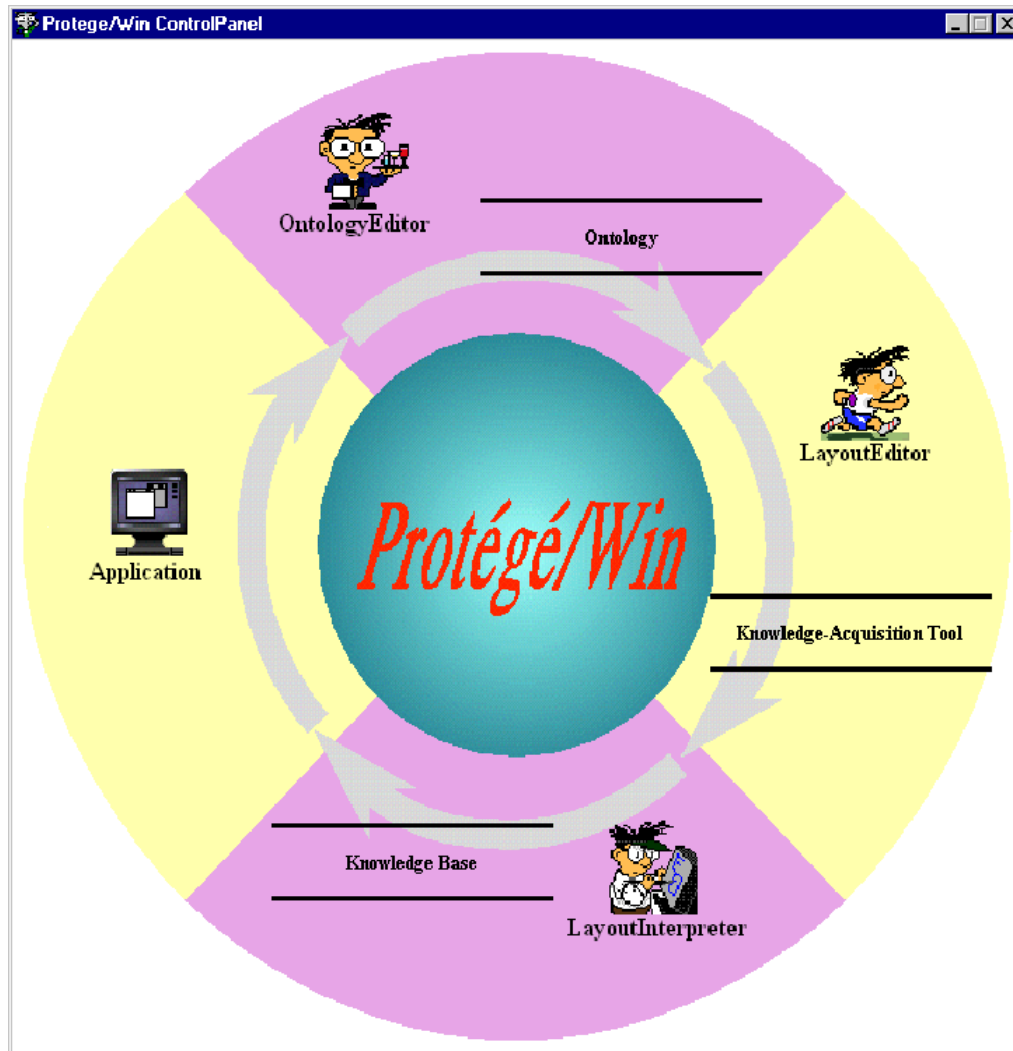


Figure 11. The Protégé/Win Control Panel, used in system demonstrations. Implicit here is the downhill-flow assumption: First an ontology is created, then forms are laid out, then instances are acquired.

However, the downhill-flow assumption is gradually being relaxed. The newer knowledge model, the utility of canonical instances in modeling complex domains, the reification of axiom frames¹⁰ in the Protégé Axiom Language (PAL), the ability to include entire knowledge-bases when designing a new knowledge base, and the increasing complexity of widgets all seem to indicate that, while the downhill-flow assumption is a useful assumption to make in certain situations, and it greatly eases knowledge-acquisition, it may not be feasible to maintain the strict separation between structure and domain-knowledge in some cases.

In all cases, a central tenet of the original Protégé methodology does remain: the utility of meta-knowledge in acquiring information. Consider, for example, meta-classes. Before a modeler can acquire, or define anything in Protégé -2000, the appropriate meta-knowledge must first be in place. The ability to define meta-classes is a new feature in Protégé -2000 and involved defining a canonical meta-class (:CLASS) with 3 template slots (:Role, :Documentation, and

¹⁰ Axioms are instances of :AXIOM

:Constraints). In ordinary knowledge-bases, all classes are instances of :CLASS. That is, defining a class is simply acquiring an instance of :CLASS.

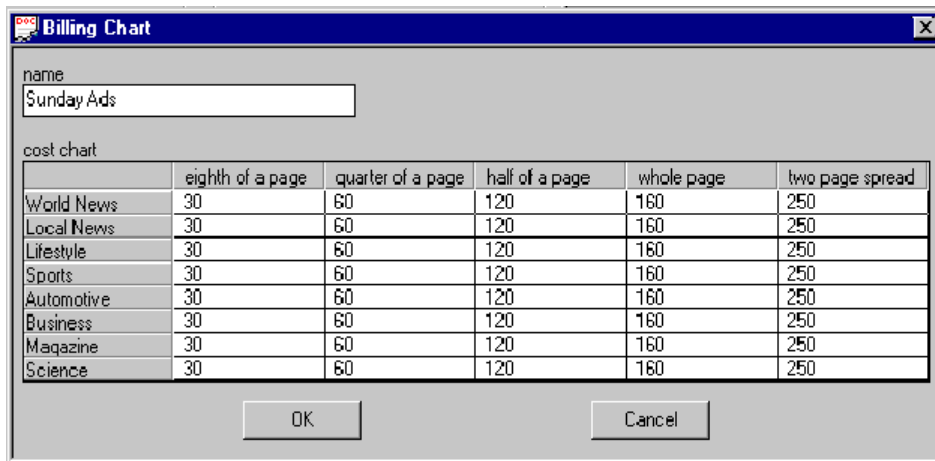
If a domain model requires a different meta-class, a different meta-class can be defined. Once it is defined, instances of it (which are classes) can easily be acquired.

The end-result is that Protégé -2000 manages to maintain its predecessor's focus on the importance of meta-knowledge in aiding the definition, construction, and maintenance of knowledge-bases while relaxing the assumptions that held them back.

EXAMPLE: TABLES AND BAYESIAN NET

S

As an example of the increasing complexity of widgets, consider the notion of Tables and Bayesian networks. The Protégé/Win table is a widget designed for slots which can have more than one value. The slot definition doesn't imply that the slot values are acquired using a table, but only that the slot can have multiple values. In the knowledge base, the slot values are stored as a list. Which implies that the "tableness" is entirely in the user interface.



```
([instance_00076] of Billing_Chart
  (cost_chart 30 60 120
    160 250 30 60 120 160 250 30 60 120 160
    250 30 60 120 160 250 30 60 120 160 250
    30 60 120 160 250 30 60 120 160 250 30
    60 120 160 250)
  (name_ "Sunday Ads"))
```

Figure 12. The Table of information is stored as a list of values for a slot

Tables have turned out to be a useful user-interface component. But in most situations in which they have been used, they actually reflect domain structure: a table is a visual representation of a function from a set of indexes to a value. This information, which is domain knowledge, should be in the knowledge-base, instead of just embedded in a user-interface. Thus, Protégé-2000 makes the following distinctions for information in a table:

- *Structural Information.* Specification of the domain and range of the associated function. Dimension of the domain (the range is implicitly 1 dimensional), types and restrictions for each coordinate.
- *Layout information.* How the function is visualized and edited. Which domain coordinate is the horizontal axis. Which domain coordinate is the vertical axis. How to display other domain coordinates (if there are other coordinates) and editing characteristics for values (for example: can the user do “in-place” editing).
- *Actual values.* Essentially tuples consisting of a set of indices and a value.

Adopting the downhill-flow assumption for tables seems fairly reasonable and natural. But now consider Bayesian networks. A Bayesian network has nodes, each of which has an associated conditional probability table. In most models of Bayesian networks, the nodes of the network, and the arcs between nodes are instances, acquired during knowledge-acquisition.

Which leads to a conundrum. When a new arc (a new causal dependency between nodes) is acquired, the dimension and the index definitions of the associated conditional probability table are altered. That is, structural information is being altered as a consequence of acquiring an instance. Which is a severe violation of the downhill-flow assumption.

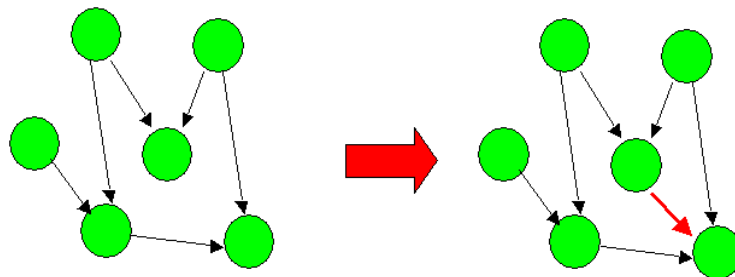


Figure 13. Knowledge-acquisition and Bayesian networks. Acquiring a new causal dependency alters structural information (either of the node or of the associated conditional probability table).

7.2. WHAT IS PROTÉGÉ-2000 ?

The current state of Protégé is summarized in table 10

System	Goals	Assumptions	What can be reused when developing new systems ?	Historical Counterparts

Protégé/Win	Make knowledge bases more reusable and maintainable by making ontologies into modular components that can be included in one another. Make software tools more usable by porting them to a standard platform	Information-partitioning hypothesis, downhill-flow model of knowledge acquisition, small-widget set assumption	Ontologies, Problem-solving methods, knowledge-bases that don't need structural modifications	
Protégé-2000	Make Protégé knowledge-bases reusable across modeling frameworks by adopting standard representation languages. Lay groundwork for addressing scalability issues in knowledge-engineering	Weakened form of the downhill-flow assumption	Everything	PowerLoom and Ontolingua

Table 10. Summary of Protégé -2000

Acknowledgements

Parts of this work were funded by the High Performance Knowledge Base Project of the Defense Advanced Research Projects Agency (contract N660001-97-C-8549) and the Space and Naval Warfare Systems Center (contract N6001-94-D-6052) .

References

- Bachant, J. and McDermott, J. D. G., R1 Revisited: Four Years in the Trenches. *AI Magazine*, Fall 1984, pages 21 –32
- Bobrow D. G., and Winograd, T. (1977). An Overview of KRL, a Knowledge Representation Language. *Readings in Knowledge Representation*, Morgan Kauffman, pp. 264-285.
- Buchanan B. and Shortliffte, E.H. (Eds) (1984). *Rule-Based Expert System. The Mycin Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley.
- Buchanan B. et al , (1983). *Constructing and Expert System*. In Hayes-Roth, F., Waterman, D., and Lenat, D. (Eds) (1983). *Building Expert Systems*. Addison-Wesley, pp. 127-169.
- Chandrasekaran, B. (1986). *Generic Tasks in knowledge-base Reasoning: high-level building blocks for expert system design*. Reprinted in Buchanan and Wilkins (Eds.), *Readings in Knowledge Acquisition and Learning*, Morgan Kauffman, pp. 170-177.
- Chaudhri, V., Farquhar, A., Fikes, R., Karp, P., and Rice, J. (1997). *The Generic Frame Protocol 2.0* [Online] Available <http://www.ai.sri.com/~gfp/spec.html>.
- Clancey, W. (1979). *Tutoring rules for guiding a case method dialogue*. *International Journal of Man-Machine Studies*. **11**, pages 25-49.
- Clancey, W. (1985). *Heuristic Classification*. *Artificial Intelligence* **27**(3), pages 289-350.
- Clancey, W. (1997). *Situated Cognition: On Human Knowledge and Computer Representations*. Cambridge University Press.
- Eriksson, H., Puerta, A. R., Gennari, J.H., Rothenfluh, T.E., Tu, S.W., and Musen, M.A. (1994). *Custom-Tailored Development Tools for Knowledge-Based Systems*. Stanford SMI Technical Report 94-551.
- Eshelman, L., Ehret, D., McDermott, J., and Tan, M., H.(1987). *MOLE: a tenacious knowledge-acuisition tool*. In Buchanan and Wilkins (Eds.), *Readings in Knowledge Acquisition and Learning*, Morgan Kauffman, pp. 253-263.
- Fensel, D., Motta, E., Benjamins, V. R., Decker, S., Gaspari, M., Groenboom, R., Grosso, W.E., Harmelen, F., Plaza, E., Schreiber, G., Studer, R., and Wielinga, B. (1999). *The Unified Problem-solving Method description Language UPML*. In ESPRIT Projekt 27169 IBROW3 : *An Intelligent Brokering Service for Knowledge-Component Reuse on the World-Wide Web*, Deliverable 1.1, Chapter 1.
- Fikes, R. and Farquhar, A. (1997) *Distributed Repositories of Highly Expressive Reusable Knowledge*. Stanford KSL Technical Report 97-02.
- Friedman, B. (1997). *Human Values and the Design of Computer Technology*. CSLI Publications / Cambridge University Press.
- Fridsma, D.B., Gennari, J.H., and Musen, M.A. (1996). *Making Generic Guidelines Site-Specific*. Stanford SMI Technical Report 96-0618.
- Genesereth, M., Fikes, R., et al (1992). *Knowledge Interchange Format 3.0* [Online] Available <http://logic.stanford.edu/kif/kif.html>.

- Gennari, J. H. (1993). A Brief Guide to Maitre and MODEL: An Ontology Editor and Frame-Based Knowledge Representation Language. Stanford SMI Technical Report 93-486.
- Gennari, J. H., Altman, R.B., and Musen, M.A. (1994). Reuse with Protégé-II: From Elevators to Ribosomes. Stanford SMI Technical Report 94-549.
- Gennari, J. H., Tu, S. W., Rothenfluh, T. E., and Musen, M. A. (1994). Mapping Domains to Methods in Support of Reuse. *International Journal of Human-Computer Studies*, **41**, pages 399–424.
- Gennari, J. H., Grosso, W. E., Musen, M. A. (1998). A Method-Description Language: An Initial Ontology with Examples. Proceedings of the Eleventh Banff Knowledge Acquisition for Knowledge-Bases Systems Workshop. Banff, Canada
- Gil, Y. and Melz, E. (1996). Explicit Representations of Problem-Solving Strategies to Support Knowledge Acquisition. Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96). Portland, Oregon.
- Grosso, W., Ferguson, R. W., Gennari, J.H., Musen, M. A. (1998). When Knowledge Models Collide (How it Happens and What to Do). Proceedings of the Eleventh Banff Knowledge Acquisition for Knowledge-Bases Systems Workshop. Banff, Canada
- Grosso, W., Ferguson, R. W., Musen, M. A. (1999). Knowledge Models and Axiom Languages: The Formal Underpinnings of Protégé. SMI Technical Report, in progress.
- Gruber, T.R. (1991). A translation approach to portable ontology specifications. *Knowledge Acquisition*, **5**, 199-220.
- Gruber, T.R. (1993). Ontolingua: A mechanism to support portable ontologies. Stanford KSL Technical Report 91-66.
- Guha, R. V. and Lenat, D. B. (1990). Cyc: A Midterm Report. *AI Magazine* (Fall 1990).
- Hayes, P. (1974). Some Problems and Non-Problems in Representation Theory. In Brachman and Levesque (Eds.), *Readings in Knowledge Representation*, Morgan Kaufman, pages 3-22.
- Hayes, P. (1979). The Logic of Frames. In Brachman and Levesque (Eds.), *Readings in Knowledge Representation*, Morgan Kaufman, pages 287-295.
- Hayes-Roth, F., Waterman, D., and Lenat, D. (Eds) (1983). *Building Expert Systems*. Addison-Wesley.
- Johnson, P.D., Tu, S. W., Booth, N., Sugden, B., and Purves, I.N. (1999). A Guideline Model for Chronic Disease Management in Primary Care. Submitted to AMIA 1999 Fall Symposium
- Karp, P. (1993) The design space of frame knowledge representation systems. SRI AI Center Technical Note #520.
- Karp, P, Myers, K., and Gruber, T. (1995) The generic frame protocol. Proceedings of the 1995 International Joint Conference on Artificial Intelligence, pp. 768 – 774.
- Kramer, M. Knowledge-Management becomes catch phrase but eludes easy definition. *PC Week Labs, PC Week*. December 6, 1998. Also available at <http://www.zdnet.com/products/stories/reviews/0,4161,374814,00.html>
- Lenat, D. B. (1995). Cyc: A Large-Scale Investment in Knowledge Infrastructure. *Communications of the ACM* **38** (11): 33-38

- Levesque, H., and Brachman, R. (1984). A fundamental tradeoff in knowledge representation and reasoning (Revised Version). In Brachman and Levesque (Eds.), *Readings in Knowledge Representation*, Morgan Kauffman, pp. 41-71.
- Marcus, S. and McDermott, J. (1989). A Knowledge Acquisition Language for Propose and-Revise. In Buchanan and Wilkins (Eds.), *Readings in Knowledge Acquisition and Learning*, Morgan Kauffman, pp. 263-281.
- McDermott, J. (1980). R1: An Expert in the Computer Systems Domain. In *Proceedings of AAAI-80, National Conference on Artificial Intelligence*, pages 269-271.
- Musen, M.A., (1988). Generation of Model-Based Knowledge Acquisition Tools for Clinical-Trial Advice Systems. PhD thesis, Stanford University.
- Musen, M.A., (1989). Automated Support for Building and Extending Expert Models. *Machine Learning* **4**, pages 347-376
- Musen, M.A., (1989b). An Editor for the Conceptual Models of Interactive Knowledge-Acquisition Tools. *International Journal of Man-Machine Studies* **31**, pages 673-698.
- Musen, M.A., Tu, S.W., Shahar, Y. (1992). A Problem-Solving Model for Protocol-Based Care: From e-ONCOCIN to EON. Stanford SMI Technical Report 92-393.
- Musen, M.A., Tu, S.W. (1993). Problem-solving models for generation of task-specific knowledge-acquisition tools. In J. Cuenca (Ed.), *Knowledge-Oriented Software*, Elsevier, pp. 23-50.
- Musen, M.A., Tu, S.W., Eriksson, H., Gennari, J.H., Puerta, A.R. (1993). Protégé-II: An Environment for Reusable Problem-Solving Methods and Domain Ontologies. Stanford SMI Technical Report 93-491.
- Musen, M.A., Tu, S.W., Das, A. K., Shahar, Y. (1995). A Component-Based Architecture for Automation of Protocol-Directed Therapy. Stanford SMI Technical Report 95-566.
- Neches, Fikes, Finin, Gruber, Patil, Senator, and Swartout (1991). Enabling technology for knowledge sharing. *AI Magazine* **13**(3): 37-56.
- Noy, N.F., and Musen, M.A. (1999). SMART: Automated Support for Ontology Merging and Alignment. Submitted to the Twelfth Workshop on Knowledge Acquisition, Modeling, and Management, 1999. Banff, Canada.
- Ohno-Machado, L., Gennari, J. H., Murphy, S., Jain, N. L., Tu, S. W., Oliver, D. E., Pattison-Gordon, E., Greenes, R. A., Shortliffe, E. H., and Barnett, G.O. (1998), The GuideLine Interchange Format: A Model for Representing Guidelines. *Journal of the American Medical Informatics Association* **5** (4) : 375-372
- Park, J. Y. Gennari, J. H. and Musen, M. A. (1997). Mappings for Reuse in Knowledge-based Systems. SMI Technical Report 97-0697.
- Puerta, A.R., Egar, J., Tu, S.W., and Musen, M.A. (1992). A Multiple-Method Knowledge-Acquisition Shell for the Automatic Generation of Knowledge-Acquisition Tools. Stanford SMI Technical Report 92-367
- Puerta, A.R., Tu, S.W., and Musen, M.A. (1993). The New World of Mechanisms. Stanford SMI Technical Report 93-444

- Puerta, A.R., Neches, R., Eriksson, H., Szekely, P., Luo, P., Tu, and Musen, M.A. (1994). Towards Ontology-Based Frameworks for Knowledge-Acquisition Tools. Stanford SMI Technical Report 94-501
- Rector A.L., Glowinski A.G., Nowlan W.A., and Rossi-Mori A. (1995). Medical concept models and medical records: An approach based on GALEN and PEN&PAD. *Journal of the American Medical Informatics Association* 2(1): pages 19-35.
- Rothenfluh, T.E., Gennari, J.H., Eriksson, H., Puerta, A.R., Tu, S.W., and Musen, M.A. (1993). Reusable Ontologies, Knowledge-Acquisition Tools, and Performance Systems: Protégé-II Solutions to Sisyphus-2. Stanford SMI Technical Report 93-502
- Shahar, Y. and Musen, M.A. (1993) RESUME: A Temporal-Abstraction System for Patient Monitoring. Stanford SMI Technical Report 93-457.
- Shahar, Y. (1997) A Framework for knowledge-based temporal abstraction. *Artificial Intelligence* 90: pages 79-133.
- Shortliffe, E.H. (1984). Studies to Evaluate the ONCOCIN System: Six Abstracts. Stanford SMI Technical Report 84-094.
- Shortliffe, E.H. (1984b). On the Mycin System. Stanford SMI Technical Report 84-095.
- Stein, A., Musen, M.A., and Shahar, Y. (1996) Knowledge-Acquisition for Temporal Abstraction. Stanford SMI Technical Report 96-614.
- Studer, R., Eriksson, H., Gennari, J.H., Tu, S.W., Fensel, D., and Musen, M.A. (1996). Ontologies and the Configuration of Problem-Solving Methods. Stanford SMI Technical Report 96-648.
- Tu, S.W., Kahn, M.G., Musen, M.A., Ferguson, J.C., Shortliffe, E.H., and Fagan, L.M. (1989). Episodic Skeletal-Plan Refinement Based on Temporal Data. *Communications of the ACM* 32 (12): 1439-1455
- Tu, S.W., Eriksson, H., Gennari, J.H., Shahar, Y., and Musen, M.A. (1994). Ontology-Based Configuration of Problem-Solving Methods and Generation of Knowledge-Acquisition Tools: Application of Protégé-II to Protocol-Based Decision Support.. Stanford SMI Technical Report 94-520.
- Tu, S.W., Eriksson, H., Gennari, J.H., Shahar, Y., and Musen, M.A. (1994). Ontology-Based Configuration of Problem-Solving Methods and Generation of Knowledge-Acquisition Tools: Application of Protégé-II to Protocol-Based Decision Support.. Stanford SMI Technical Report 94-520.
- Uschold, M., and Clark, P. (1997). Ontologies Ontologies Everywhere: Who Knows What to Think. Submitted to IJCAI 99 Workshop on Ontologies and Problem-solving Methods.
- Uschold, M., Clark, P., Healy, M., Williamson, K., and Woods, S. (1998). An Experiment in Ontology Reuse. Proceedings of the Eleventh Banff Knowledge Acquisition for Knowledge-Bases Systems Workshop. Banff, Canada
- Uschold, M., and Jasper, R. (1999). Ontologies Ontologies Everywhere: Who Knows What to Think. Submitted to IJCAI 99 Workshop on Ontologies and Problem-solving Methods.
- van Melle, W., Shortliffe, E.H., and Buchanan, B.G. (1984). EMYCIN: A Knowledge Engineer's Tool for Constructing Rule-Based Expert Systems. In Buchanan and Shortliffe (Eds.), Rule-

Based ExpertSystems: The Mycin Experiments of the Stanford Heuristic Programming Project, Addison-Wesley, pages 302-313.

Wielinga, B. J., Schreiber, A. T., and Breuker, J.A. (1992). KADS: A Modeling Approach to Knowledge-Engineering. Reprinted in Buchanan and Wilkins (Eds.), Readings in Knowledge Acquisition and Learning, Morgan Kauffman, pp. 93-116.

Wiederhold, G. (1994). Interoperation, Mediation, and Ontologies. Proceedings International Symposium on Fifth Generation Computer Systems (FGCSOB94), Workshop on Heterogeneous Cooperative Knowledge-Bases, Vol.W3, pages 33-48