Domain Ontologies in Software Engineering:

Use of Protégé with the EON Architecture

Mark A. Musen

Section on Medical Informatics
Stanford University School of Medicine
Stanford, California 94305-5479
U.S.A.

Domain ontologies are formal descriptions of the classes of concepts and the relationships among those concepts that describe an application area. The Protégé software-engineering methodology provides a clear division between domain ontologies and domain-independent problem-solvers that, when mapped to domain ontologies, can solve application tasks. The Protégé approach allows domain ontologies to inform the total software-engineering process, and for ontologies to be shared among a variety of problem-solving components. We illustrate the approach by describing the development of EON, a set of middleware components that automate various aspects of protocol-directed therapy. Our work illustrates the organizing effect that domain ontologies can have on the software-development process. Ontologies, like all formal representations, have limitations in their ability to capture the semantics of application areas. Nevertheless, the capability of ontologies to encode clinical distinctions not usually captured by controlled medical terminologies provides significant advantages for developers and maintainers of clinical software applications.

1 Vocabularies, Ontologies, and Software Engineering

Work in medical informatics entails the creation of software. Although many fundamental contributions in medical informatics are not dependent on computer-based artifacts (e.g., the development of the International Classification of Diseases in no way required the availability of computers), it now is nearly impossible to discuss information management in health care without bringing up the role of information technology. Researchers in medical informatics write computer programs as the primary means of testing theories and of delivering results. Indeed, the evaluation of the performance of specific pieces of software may be the only proxy available for evaluating certain theories. There consequently are compelling reasons for all members of the medical-informatics community to be concerned with questions of software engineering (Blum, 1991).

Work in medical informatics entails the communication of information. Perhaps the most important information involves descriptions of patients—including patients' problems, clinical findings, and treatments. The pressing need to develop repeatable ways of specifying patient descriptions has led to the proliferation of a variety of controlled medical terminologies over the past century. The landscape of medicine is now routinely represented in terms of the International Classification of Diseases (ICD), the Systematized Nomenclature of Human and Veterinary Medicine (SNOMED), Current Procedural Terminology (CPT), and so on. Each one of these terminologies makes different clinical distinctions and is biased by the particular purposes envisioned by that vocabulary's creators and maintainers (Feinstein, 1988). Indeed, one of the primary challenges for current research in medical informatics is to make sense out of these competing schemes for representing clinical information, and to develop more unified approaches to the representation of patient descriptions.

Workers in medical informatics often view controlled terminologies as self-contained lexicons that are independent from specific software artifacts or information technologies. Controlled terminologies are consequently described as circumscribed compilations of codes and terms that seemingly have had an isolated existence. Standard vocabularies such as ICD, for example, had been maintained for decades before their developers had any notion of incorporating them into software systems. Now, however, it is the apparent precision, compactness, and scope of these controlled terminologies that make it only natural to rely on such representations for many computer-based clinical applications. The desire to build computer-based medical-record systems and other complex software systems for health care now make it essential to examine how controlled terminologies function as structures within computer programs. Thus, we must clarify the role of standard vocabularies in the engineering of clinical software applications.

In recent years, as workers in medical informatics have become increasingly concerned with the development and use of controlled terminologies to capture complex patient descriptions, workers in computer science have sought new ways to manage the inherent complexity of large software systems. The practice and theory of software engineering has been evolving, with increasing emphasis on component-based architectures that allow for modularization, encapsulation, and distribution of units of program code (Orfali et al., 1966). One particularly significant trend has been the growing recognition of the importance of the models of application areas that are embedded within computer programs, and of the value of making those models more explicit, editable, and maintainable (Sutcliffe et al., 1996). These models of application areas are called *domain models*. For programs such as word processors, the domain model is one of textual documents, of the components of such documents, and of the changes that users may wish to make to documents. All the tasks that a word-processing program can perform must be expressed in terms of this domain model; it is impossible for the user to express a desire to modify some portion of a document unless that portion can be represented in terms of the underlying domain model.

For programs such as computer-based patient-record systems, the domain model is one of patients, of their medical problems, and of the interventions that health-care workers may make in the management of patients. There is nothing that the user of a computer-based medical-record system can say about a patient that is not in some way a component of the domain model. The creation of precise, well-scoped domain models for clinical information systems is of central importance in medical informatics, and should benefit from software-engineering principles that make the models (and the controlled terminologies from which those models are built) both explicit and manipulable. Although there are several object-oriented analysis approaches that allow software engineers to define comprehensive domain models when software systems are first designed (e.g., Booch, 1994), the models do not usually become permanent constituents of the resulting systems. Moreover, once the computer systems are finally implemented, system maintainers cannot modify the models without worrying about how their changes might affect existing program code that may make assumptions about the initial model formulations.

For more than a decade, our laboratory has been developing a software-engineering methodology known as Protégé (Musen et al., 1995a) that assists software developers in creating and maintaining explicit domain models, and in incorporating those models directly into program code. Protégé allows system builders to construct software systems from modular components, including (1) reusable frameworks for assembling domain models and (2) reusable domain-independent problem-solving methods that implement procedural strategies for solving tasks (Eriksson et al., 1995).

Protégé allows reuse of frameworks for building domain models through its support for declarative *domain ontologies*. An ontology enumerates the concepts that are relevant in an application area, providing a universe of discourse. An ontology defines classes of concepts and relationships among those classes. Typically, however, an ontology does not include information about *instances* of any given class (Guarino and Giaretta, 1995). Thus, an ontology for a computer-based patient-record system might declare that *patients* are treated by *physicians*, but the ontology would not specify the identities of particular patients or physicians. In many ways, an ontology can be viewed as the equivalent of a *schema* in database terms. Whereas controlled terminologies such as ICD provide an enumeration of instances of clinical descriptors—with only implicit representation of an organizing framework for the classes of those descriptors—ontologies emphasize the organizing framework, at the expense of being exhaustive about listing possible instances of the classes of concepts. Although the philosophical distinctions can be a bit enigmatic, for the purposes of this paper, when we combine a domain ontology with an enumeration of the instances intended by that ontology for a particular application, the resulting set of classes and instances is what we call a *domain model*.

In the Protégé methodology, we provide a clear separation between a developer's specification of a domain ontology, her declaration of *instances* of the classes defined by the domain ontology, and her mapping of the domain-specific information to a particular problem-solving method (Eriksson et al., 1995). This distinct division between the target system's domain model and the system's procedural, problem-solving method is reflective of trends in the engineering of both knowledge-based systems and conventional software (1) to clarify a program's domain model and (2) to allow developers to edit and adapt that model. In the Protégé approach, when developers edit the ontology that provides the framework for specification of the complete domain model, they automatically allow generation of a domain-specific, interactive knowledge-acquisition tool that permits users easily to enter the relevant instances of the concepts defined by the ontology (Eriksson et al., 1994). The edited domain ontology also becomes available to the application system that Protégé is used to construct—shaping the interactions of end users with the resultant computer program.

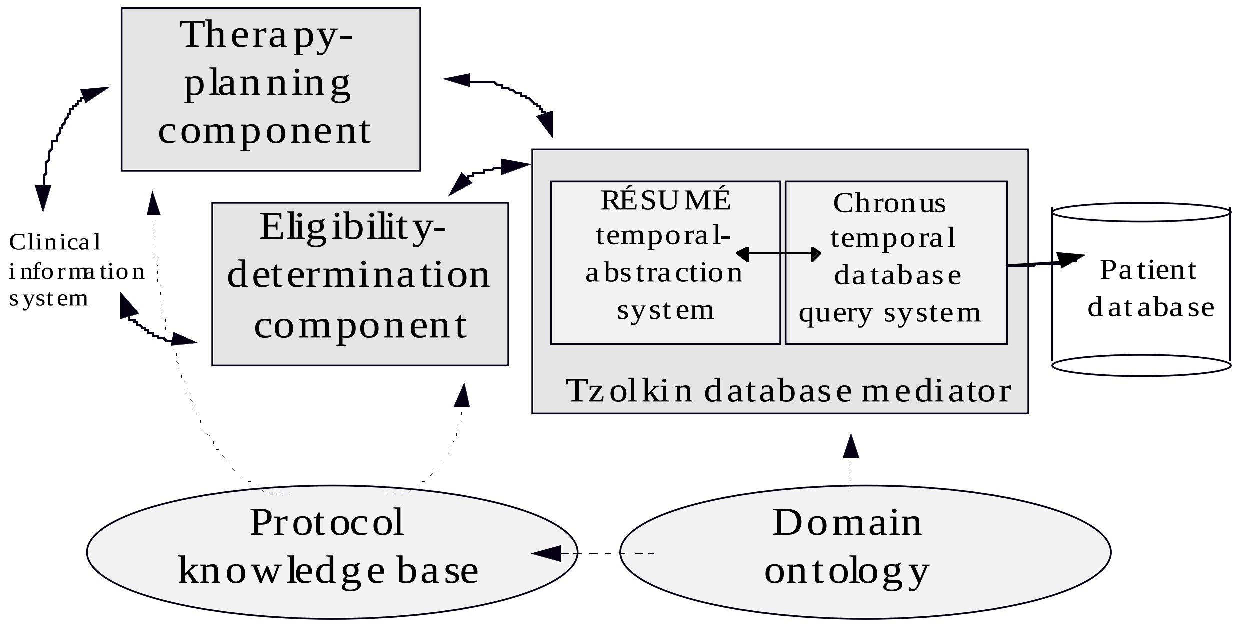

Therapy-planning component
Eligibility-determination component
Clinical information system
RÉSUMÉ temporal-abstraction system
Chronus temporal database query system
Tzolkin database mediator
Patient database
Protocol knowledge base
Domain ontology

Figure 1: *The EON archiтecure. EON comprises a number of problem-solving components (e.g., programs that plan protocol-based therapy and that determine whether a patient potentially is eligible for protocols) that share a common knowledge base of protocol descriptions. The protocol domain ontolology, created with the Protégé system, defines the format of the protocol knowledge base. The same ontology also defines the schema for the Tzolkin database mediator, a system that channels the flow of patient data between the problem-solving components and an archival relational database. The entire architecture is embedded within a clinical information system that can use the domain ontology to fashion its user interface.*

In this paper, we discuss how we have used Protégé to build a computer-based patient-record system that assists providers in the application of clinical practice guidelines and protocols. We show how the approach allows (1) principled construction of a domain ontology, (2) instantiation of that ontology with domain facts, and (3) generation of an application program that developers can easily maintain and modify over time. We emphasize the relationship between standard, controlled terminologies and the domain ontologies that form the foundation of building application systems using Protégé. Our work demonstrates the use of explicit domain ontologies in shaping all aspects of the software process, and suggests how the incorporation of those ontologies directly into the resulting software systems can lead to enhanced development and to improved system maintenance.

2 The EON Architecture for Protocol-Based Care

We will ground our discussion in an example: the use of the EON architecture to automate protocol-based care. Although the Protégé methodology (Musen et al., 1995; Eriksson et al., 1995) and the use of Protégé to construct decision-support systems based on the EON architecture (Tu et al., 1995; Musen et al., 1996) have been presented previously, it is helpful to review here how our group has used Protégé to create a family of application systems that provide assistance with protocol-based patient care. In the example that follows, we emphasize the role of the domain ontology both in driving the software-engineering process and in providing a structure for the data on which the ultimate application program operates.[*]

EON is an application system that comprises a number of modular components (Figure 1). The architecture is intended to be embedded within a clinical information system, processing clinical data to offer decision support concerning various aspects of protocol-based care. The architecture contains three classes of components:

EON contains **problem-solving modules** that perform the computations necessary to automate specific tasks associated with guideline-directed therapy. One such module takes as input a standard clinical protocol description and relevant patient data, and generates as output a situation-specific treatment plan for the

[*] Previous descriptions of Protégé that have appeared in the literature have depicted a version of the system that runs under the NeXTSTEP operating system; the version of Protégé depicted in this paper runs under Windows NT and Windows 95. At the same time, the particular components of the EON architecture have evolved significantly since the most recent published description of EON (Musen et al., 1996).

current patient encounter (Tu and Musen, 1996). Another module takes as input a standard protocol description and relevant clinical data, and generates as output the qualitative likelihood that a patient is eligible for the given protocol (Tu et al., 1993). The EON architecture is extensible, so that other problem-solving modules easily can be added. For example, although we have not constructed such a module, the architecture readily would support incorporation of a component that might review provider decision making retrospectively and determine episodes of therapy that may have been noncompliant with specific guidelines.

EON contains a **database mediator** that provides the conduit between the problem-solving modules and the clinical data stored in an archival database. This mediator, known as Tzolkin (Das et al., 1994), itself has two components: (1) the Chronus temporal data query system (Das and Musen, 1994) and (2) the RÉSUMÉ temporal-abstraction system (Shahar and Musen, 1996). When a problem-solving module, such as the therapy-planning component (shown in the upper left of Figure 1) wishes to resolve a question such as “Was there a past episode of moderate anemia that lasted for more than two weeks?”, the problem-solving module passes the query to Tzolkin. Tzolkin (1) determines that *anemia* is not a primitive datum stored in the archival patient database, but rather is an abstraction of *hemoglobin*, values of which are indeed stored in the database, (2) uses Chronus to query the patient database for the hemoglobin values, and (3) uses the RÉSUMÉ abstraction system to determine whether there are sequences of low hemoglobin values that constitute intervals of moderate anemia. Because all the temporal database management and all data abstraction operations are performed transparently by the Tzolkin mediator, none of the problem-solving modules in EON needs to duplicate these basic functionalities.

EON contains a declarative **knowledge base** of protocol specifications, defining the sequences of interventions required by the each protocol or guideline, and the criteria that define whether patients are eligible for that protocol. These guideline specifications are essential run-time inputs to the problem-solving components that reason about protocol-directed patient care.

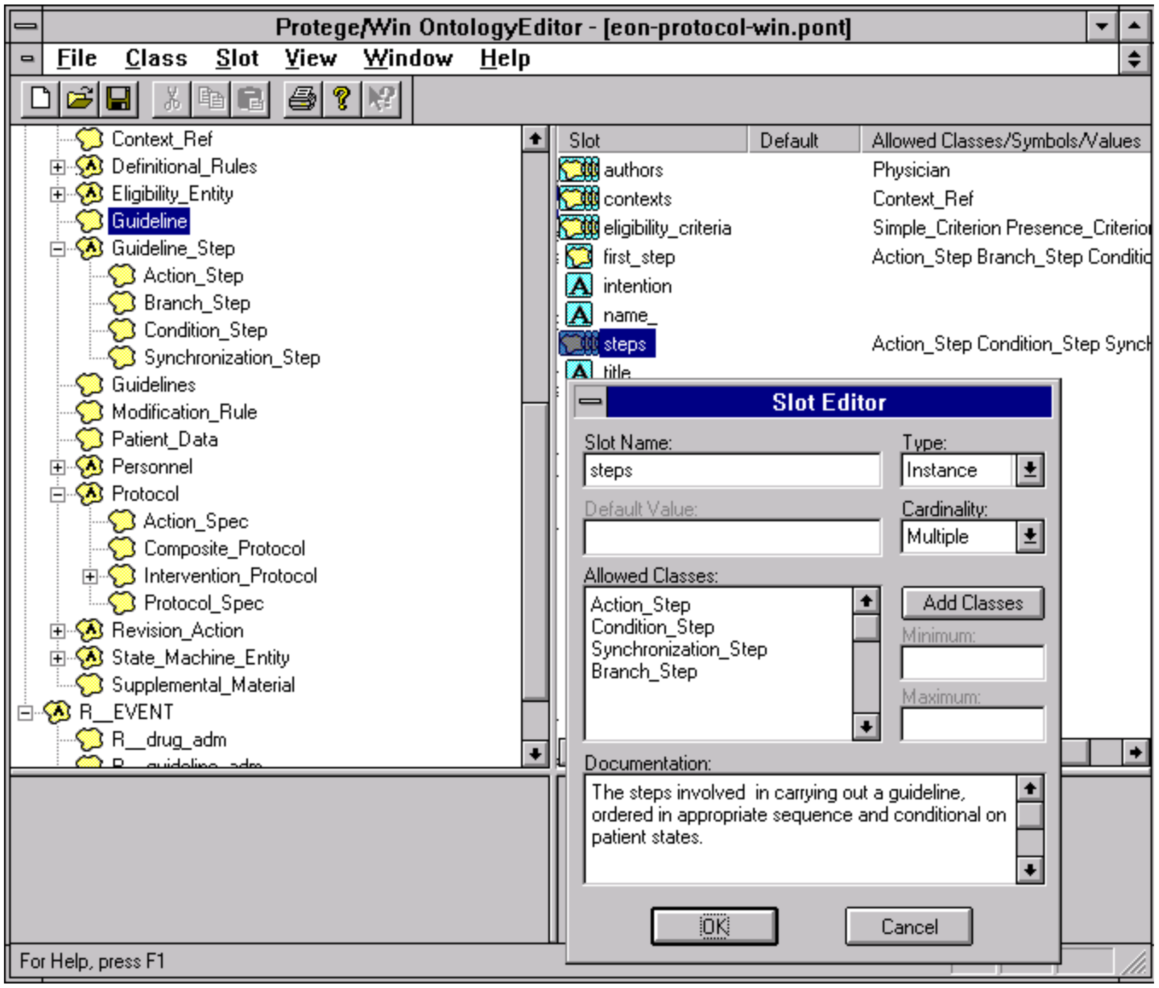
3 The Domain Ontology in EON

All the software components in the EON architecture benefit directly from a shared, computer-based representation of the domain ontology that defines the form of all the data on which elements of the system may operate. There are no data interchanged among any of the EON components that do not emanate explicitly from the data definitions contained in the ontology. The domain ontology defines a precise and consistent data model for all modules in the EON architecture. If developers should modify that ontology, then the changes will affect every component that accesses the corresponding data.
Figure 2 shows a portion of the domain ontology for EON represented in the Protégé Ontology Editor. The ontology comprises two different elements:

The **guideline ontology** consists of a set of class definitions that describe concepts related to clinical protocols. This ontology indicates that guidelines have a set of authors, a set of clinical contexts in which the guidelines apply, and a set of eligibility criteria. A guideline has an intention (i.e., a purpose), and a sequence of steps that dictate the clinical actions that serve to achieve that intention.

The **medical-specialty ontology** defines the particular clinical interventions that are typical for a given area of medicine, and the types of patient findings that are most commonly reported in a given medical discipline. By making the medical-specialty ontology an explicit component, we acknowledge that different classes of health-care providers tend to make different classes of observations about their patients and perform different kinds of patient-care activities. Oncologists, for example, may be particularly concerned with tumor histology and cytotoxic chemotherapy; physicians who care for people with AIDS, on the other hand, pay extreme attention to T-cell subsets and to the dosing of antiretroviral drugs. By editing the medical-specialty ontology, developers can tailor the data processed by EON to reflect the distinct requirements of particular clinical disciplines.

The guideline ontology is generic, and independent of any medical specialty. To create a version of EON that is suitable for a particular medical specialty, the developer first must create an ontology that defines the concepts relevant for that specialty. The developer then must integrate the new medical-specialty ontology into the guideline ontology by defining subclass relationships that denote how the generic notions of (1) clinical findings, (2) patient problems, and (3) medical interventions in the guideline ontology relate to the detailed classes of concepts defined in the medical-specialty ontology. To date, our group has demonstrated this approach by constructing ontologies for protocol-based care in AIDS and breast cancer (Musen et al., 1996). Whereas our AIDS ontology includes medical interventions such as the administration of drug regimens and the scheduling of follow-up visits, the breast-cancer ontology adds several additional classes of interventions, such as surgery, radiotherapy, and home visits. The distinctions made by each ontology reflect elements that medical specialists with whom we have collaborated have found important for their respective practices.



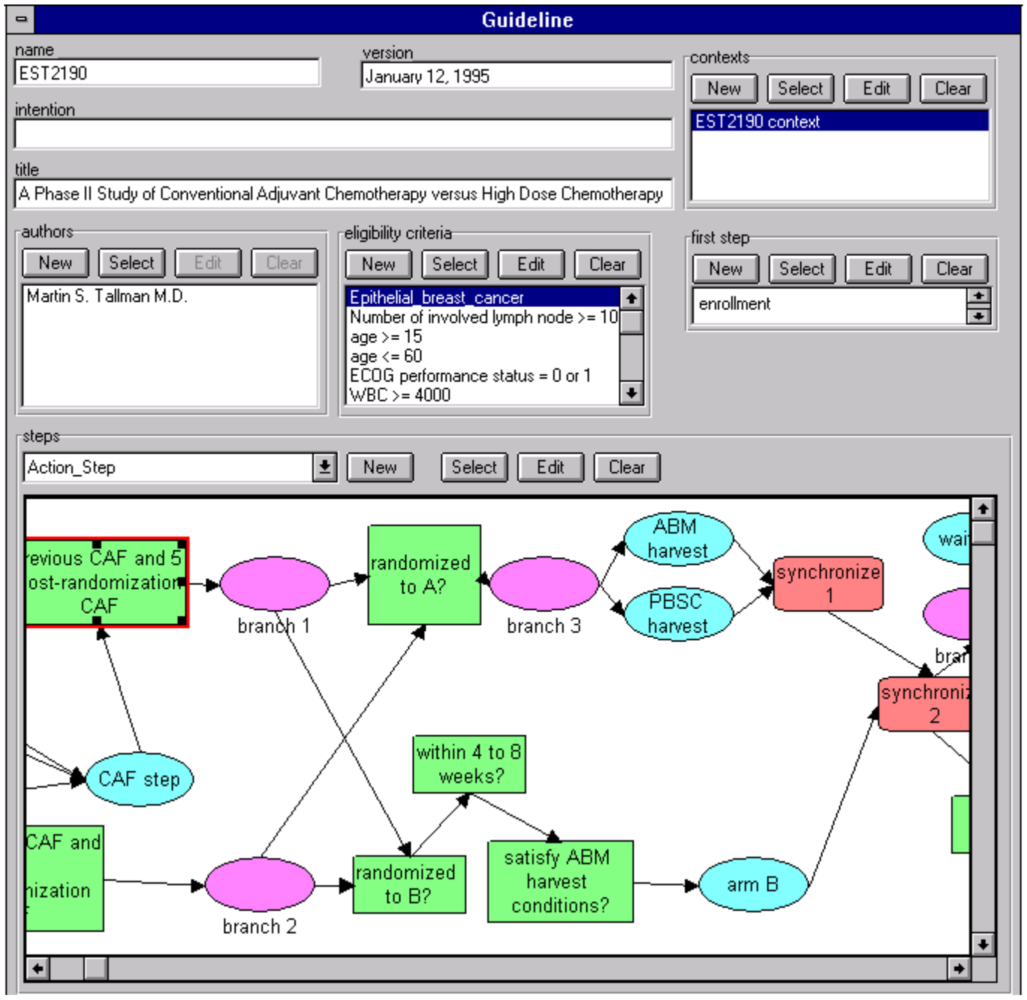
Slot	Default	Allowed Classes/Symbols/Values
authors		Physician
contexts		Context_Ref
eligibility_criteria		Simple_Criterion Presence_Criterio
first_step		Action_Step Branch_Step Conditio
intention		
name_		
steps		Action_Step Condition_Step Synch
title		

Slot Name: steps
Type: Instance
Default Value:
Cardinality: Multiple
Allowed Classes: Action_Step Condition_Step Synchronization_Step Branch_Step
Minimum:
Maximum:
Documentation: The steps involved in carrying out a guideline, ordered in appropriate sequence and conditional on patient states.

Figure 2: *The domain ontology for guideline-based care in the Protégé Ontology Editor. The panel in the upper left of the screen shows the hierarchy of classes in the ontology. The user has selected the class guideline, which, as we see in the panel in the upper right, contains an attribute called steps. The user has invoked the slot editor for this attribute, which appears at the lower left of the screen. The information in the slot editor indicates that the steps of a guideline are denoted by instances of more specialized classes in the ontology, such as instances of action_step, condition_step, and so on.*

3.1 Knowledge Acquisition

The Protégé system uses a domain ontology to generate automatically a custom-tailored tool that developers use to enter the instances of the concepts entailed by the ontology (Eriksson et al., 1994). This knowledge-acquisition tool allows system builders to complete the specification of a domain model by adding to the ontology specific propositional statements that instantiate that ontology for a given application area. Thus, the EON guideline ontology, when merged with a medical-specialty ontology for breast-cancer therapy, is used by Protégé to create a knowledge-acquisition tool that allows developers to describe instances of breast-cancer guidelines (Figure 3). When comparing the knowledge-acquisition tool with the domain ontology in Figure 2, we can see how the *classes* in the ontology define expectations for the *instances* to be acquired by the knowledge-acquisition tool. The data type and expected cardinality of each attribute of each class in the ontology determines the corresponding graphical widgets that appear in the knowledge-acquisition tool and that are used to acquire the instance information. Thus, guidelines have an attribute called *name* that has a data type of *string* and a cardinality of *single* (see Figure 2); the knowledge-acquisition tool accordingly acquires the name of the guideline using a single text box (see Figure 3). Guidelines also have an attribute called *steps* whose data type consists of instances of other classes (namely, *action_step*, *condition_step*, *synchronization_step*, and *branch_step*) and whose cardinality is multiple; the knowledge-acquisition tool acquires the steps of a guideline using a graph, where the nodes in the graph may be multiple instances of the different allowed classes. The automated generation of the knowledge-acquisition tool for entry of breast-cancer guidelines is driven directly from the domain ontology. The resulting tool uses the ontology to



Guideline

name	EST2190
version	January 12, 1995
intention	
title	A Phase II Study of Conventional Adjuvant Chemotherapy versus High Dose Chemotherapy
contexts	EST2190 context
authors	Martin S. Tallman M.D.
eligibility criteria	Epithelial_breast_cancer Number of involved lymph node >= 10 age >= 15 age <= 60 ECOG performance status = 0 or 1 WBC >= 4000
first step	enrollment
steps	Action_Step

Figure 3: *A knowledge-acquisition tool for entry of breast-cancer protocol specifications. This tool is generated automatically from an ontology that combines the generic guideline ontology (see Figure 2) and an ontology that defines the classes of clinical findings, problems, and interventions associated with the care breast-cancer patients. Protégé constructed this particular window from the guideline class of the ontology. The protocol depicted in the figure specifies the knowledge required to carry out a clinical trial that compares the effects of conventional adjuvant chemotherapy with those of high-dose chemotherapy followed by bone-marrow transplantation.*

provide a framework for communicating with its user, where the semantics of each entry into the knowledge-acquisition tool are determined by the class definitions in the domain ontology.

3.2 Problem Solving

The electronic knowledge base that instantiates the domain ontology (the domain model), which is generated from a user's entries into the knowledge-acquisition tool (e.g., the description of Protocol EST2190 in Figure 3), becomes a shared component of the EON problem solving system (Figure 1). All the problem-solving modules in EON (i.e., the therapy planner, the eligibility-determination component, and any other module that may be developed in the future) use this common representation of protocol knowledge to perform their particular reasoning functions.

The problem-solving modules are able to access the protocol knowledge base appropriately because system builders relate the input–output requirements of each problem solver to the classes and attributes in the domain ontology that provide the structure for the protocol knowledge base. Thus, the developer declares that the *plans* processed by the therapy planner correspond to *guidelines* in the domain ontology. For every datum processed by a problem solver, the developer must specify the appropriate mapping to the domain ontology in an explicit mapping relation (Gennari et al., 1994).

The Protégé approach is quite different from that taken in traditional object-oriented programming, where both the domain knowledge (slots of objects and the values associated with particular slots) and the problem solvers (methods associated with specific objects) are bundled together. In traditional object-oriented programming, program execution is controlled by sending messages from one object to another, where each object encapsulates both data and the methods that operate on those data (Booch, 1994). In the Protégé approach, however, domain models (data) are kept completely separate from the problem-solving methods (programs) that may be applied to those models; the problem-solving methods are first-class entities that have formal parameters that must be mapped to the appropriate referents in the domain knowledge. The separation of problem-solving methods from the domain knowledge on which those methods operate is essential for component reuse. The architecture allows the same domain knowledge to be used by different problem-solving methods, possibly to solve completely different tasks (e.g., the tasks of planning therapy and of determining eligibility for protocols). Alternatively, developers can reuse a given problem-solving method with new domain knowledge to solve the same task in a new domain (e.g., to perform therapy planning either for AIDS or for breast cancer).

The use of a shared, explicit domain model simplifies the system maintenance task considerably. If developers should need to update a protocol knowledge base, then no reprogramming of the problem-solving modules is ever necessary. The developers need only to substitute the new knowledge base for the old one. If the domain ontology that structures the knowledge base should change, then programmers may need to define new mapping relations, however. Once those new mappings are defined, then all new knowledge bases that are created using the revised ontology are readily incorporated into the architecture.

EON's separation of the knowledge base from the problem solvers that operate on it is reminiscent of the situation in first-generation expert systems in which developers made a distinction between the knowledge base (typically a collection of rules or frames) and the *inference engine* that would operate on that knowledge base (Waterman et al., 1983). In first generation expert systems, however, the knowledge base did not consist of an explicit domain model that could serve as input to a variety of problem solvers. Instead, the knowledge base was always a set of representations that were dependent on one specific inference engine (e.g., the production-rule interpreter in a system such as EMYCIN). In first-generation expert systems, the knowledge base could be used to solve only one specific task. In EON, however, each task related to protocol-based care is addressed by a different problem-solving component that accesses the common protocol domain model. A problem-solver such as the therapy-planning component is a software module that operates on a model of clinical protocols—not an inference engine that operates on arbitrary rules or frames. All protocol models on which the EON problem solvers may operate are direct extensions of the corresponding domain ontology created using Protégé.

3.3 Data Definition and Manipulation

Just as all the problem solvers in EON share a common static knowledge base of protocol descriptions, all the problem solvers share a common gateway to the patient data needed for real-time decision support. In the current version on EON, all problem-solving components use the Tzolkin database mediator (see Figure 1) as the common access path both to primary patient data (e.g., laboratory-test results, patient findings) and to abstractions of those data (e.g., intervals of possible drug toxicity or of disease progression). Tzolkin contains two modules: (1) the Chronus temporal data query system and (2) the RÉSUMÉ temporal-abstraction system. Both Tzolkin modules access the same domain ontology used by the other components of the EON architecture (see Figure 1).

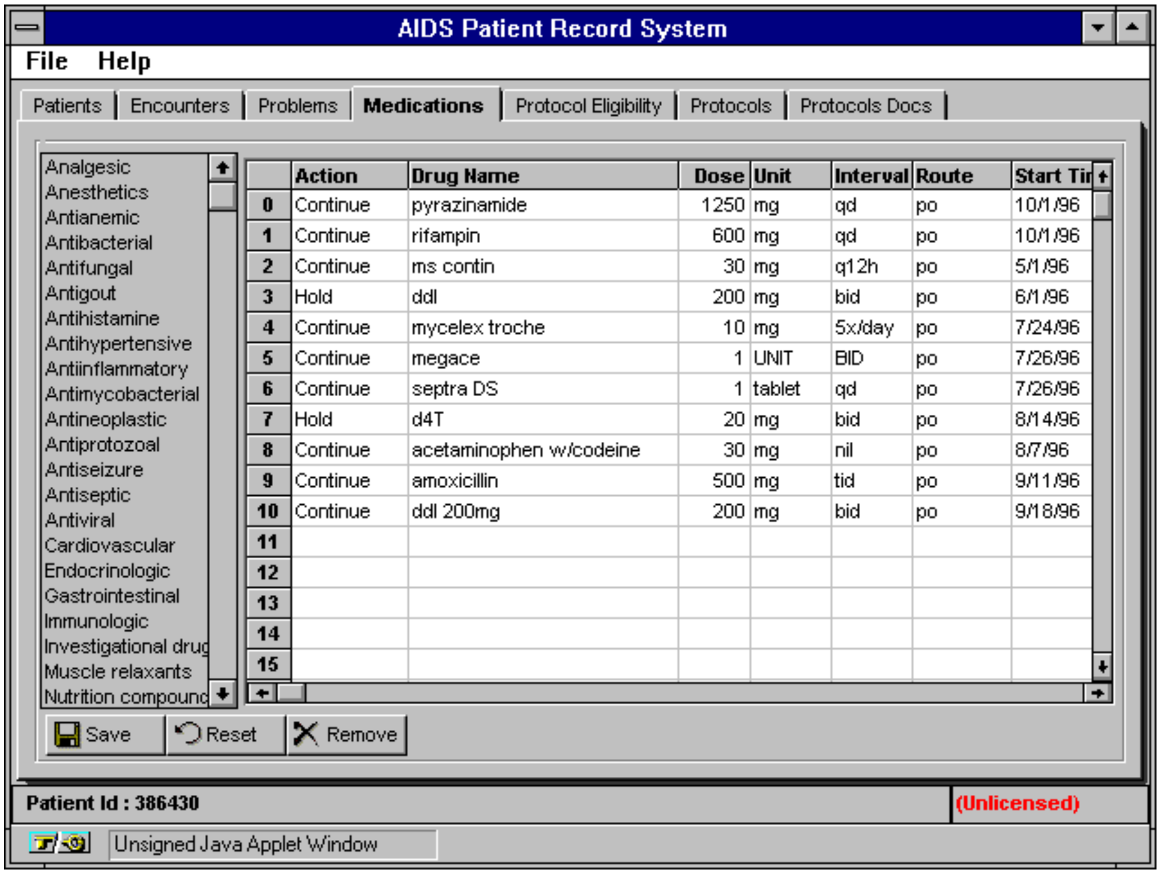
3.3.1 Chronus

Chronus (Das and Musen, 1994) serves as both a preprocessor and a postprocessor to the data query and manipulation system of a standard relational database. The system adds to every relation in the database that describes patient-specific data an explicit start time and end time. Chronus implements a temporal relational algebra that confers special status to the time stamps, ensuring that any set of data-manipulation operations on a time-stamped relation returns a new relation with appropriate start and end times.

Chronus uses the domain ontology to establish a schema for all the patient-specific data that will be stored in an external relational database. The medical-specialty portion of the domain ontology informs Chronus of the classes of clinical observations, patient problems, and therapeutic interventions that health-care providers will want to record in their computer-based patient record and that the EON problem solvers will need when executing their decision-support tasks. Chronus does not need to access the domain ontology dynamically during run time; Chronus examines the ontology only when the patient database is created initially. The classes of concepts defined in the domain ontology serve as input to a module that generates appropriate statements in the data-definition language of the relational database that Chronus accesses.

3.3.2 RÉSUMÉ

Often, problem solvers will issue queries that request information that is not stored directly in the database, but rather that represents an abstraction of primitive data. Tzolkin handles such queries transparently, invoking the RÉSUMÉ system automatically whenever a query requires an abstraction to be made.


AIDS Patient Record System

File Help

Patients | Encounters | Problems | **Medications** | Protocol Eligibility | Protocols | Protocols Docs

Analgesic
Anesthetics
Antianemic
Antibacterial
Antifungal
Antigout
Antihistamine
Antihypertensive
Antiinflammatory
Antimycobacterial
Antineoplastic
Antiprotozoal
Antiseizure
Antiseptic
Antiviral
Cardiovascular
Endocrinologic
Gastrointestinal
Immunologic
Investigational drug
Muscle relaxants
Nutrition compound

	Action	Drug Name	Dose	Unit	Interval	Route	Start Tir
0	Continue	pyrazinamide	1250	mg	qd	po	10/1/96
1	Continue	rifampin	600	mg	qd	po	10/1/96
2	Continue	ms contin	30	mg	q12h	po	5/1/96
3	Hold	ddI	200	mg	bid	po	6/1/96
4	Continue	mycelex troche	10	mg	5x/day	po	7/24/96
5	Continue	megace	1	UNIT	BID	po	7/26/96
6	Continue	septra DS	1	tablet	qd	po	7/26/96
7	Hold	d4T	20	mg	bid	po	8/14/96
8	Continue	acetaminophen w/codeine	30	mg	nil	po	8/7/96
9	Continue	amoxicillin	500	mg	tid	po	9/11/96
10	Continue	ddI 200mg	200	mg	bid	po	9/18/96
11							
12							
13							
14							
15							

Save Reset Remove

Patient Id : 386430 (Unlicensed)

Unsigned Java Applet Window

Figure 4: *A Java-based clinical interface for EON. Although EON is designed to be embedded within a variety of clinical information systems, we have experimented with the EON components in our laboratory using a user interface accessible via the World-Wide Web. The portion of the interface represented here allows clinicians to enter and revise a patient's current drug presciptions. The domain ontology (see Figure 5) includes a taxonomy of drugs, a portion of which is represented in the left-hand side of the clinical-interface display.*

As described in detail elsewhere (Shahar, in press), the RÉSUMÉ system requires specific, well defined knowledge about the primary data that it abstracts, and, in turn, about the resulting abstractions—which RÉSUMÉ can use to generate further abstractions. For each class of data or abstraction, RÉSUMÉ needs to know certain temporal-semantic properties. For example, these properties include whether two contiguous temporal intervals that describe the same kind of event (e.g., two episodes of *anemia* or two episodes of *seizures*) may be described as one longer interval. (The episodes of *anemia* may be combined into a longer interval, but the episodes of *seizures* should be viewed separately.) Other properties include whether *subintervals* of the event necessarily hold true (e.g., *anemia* during a given interval implies *anemia* during any portion of that interval, whereas *labile hypertension* during an interval does not imply *labile hypertension* during every subinterval), and mathematical functions that define the probability that, when equal measurements of some clinical parameter are made at two points separated in time, the parameter's value is the same throughout the intervening interval. All these properties of clinical parameters are defined declaratively in a parameter ontology, where each semantic property is represented explicitly as a specific attribute of each class. Although this parameter ontology was represented as a separate entity in the original version of RÉSUMÉ (Shahar and Musen, 1996), the Tzolkin mediator uses an enhanced version of RÉSUMÉ that accesses the necessary temporal-semantic knowledge directly from the domain ontology that informs the operation of all the other EON components.

The medical-specialty portion of the EON ontology includes the enumeration of laboratory tests, clinical signs and symptoms, and therapeutic interventions that may be applied to individual patients who may be the subjects of the clinical protocols about which the EON components can reason. Unlike a controlled medical terminology, which merely will list individual vocabulary elements (and, often, their associated codes), the EON domain ontology provides a rich structure that defines the semantic properties of each element as required by the RÉSUMÉ temporal-abstraction system. The additional semantic information about the elements of the ontology can be accessed by all the problem-solving elements of EON. Thus, although probabilistic information regarding the expected persistence of data values is included in the domain ontology specifically for the benefit of the RÉSUMÉ system, such descriptors can be used by other components—for example, the clinical information system that embeds the EON architecture, which may need to make decisions regarding the *display* of data values that have a temporal dimension.

3.4 Clinical Information System

The EON architecture comprises a set of CORBA-compliant modules that we have designed to be embedded within a variety of clinical information systems. An earlier version of EON (Musen et al., 1996), for example, was implemented within the T-HELPER system for protocol-based care of patients with AIDS (Musen et al., 1992). In our current work, we have created a Java-based clinical-information-system interface that replicates much of the functionality of the T-HELPER interface, but which is accessible in a distributed manner via the World-Wide Web (Figure 4).

Because the domain ontology is stored in a declarative manner that makes it easily accessible, any information system that embeds the EON components can also retrieve concept descriptions from the ontology. Many distinctions made by the medical-specialty portion of the ontology are relevant to the clinical information system. For example, the taxonomy of drugs that is relevant to the given medical specialty can be used by the clinical information system to guide the user's entry of prescriptions. The categorization of drugs that appears in the clinical user interface in Figure 4 is thus directly informed by the portion of the domain ontology shown in Figure 5. The same domain ontology is used by the problem-solving components (e.g., by the eligibility-determination module, when it uses the domain ontology to determine the individual drugs that may belong to a given class to decide whether a patient's current drug treatment excludes him from a particular protocol). Other elements of the medical-specialty portion of the ontology that may be beneficial to the clinical user interface include the classes of patient signs and symptoms that may be used to guide entry of clinician progress notes (Musen et al., 1995b), and the classes of laboratory tests that may be used both for order entry and results reporting. Thus, the shared ontology, which is essential for execution of the various decision-support modules in EON, also can be useful in fashioning the user interface accessed by health-care workers who may rely on EON's decision making elements for rendering protocol-based care.

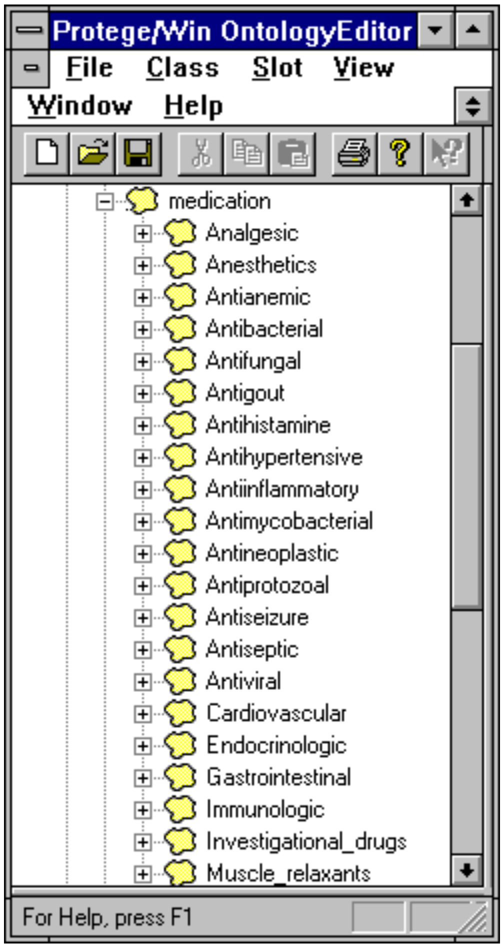


Figure 5: *The drug taxonomy from the AIDS medical-specialty ontology. Every medical-speciality component of an EON domain ontology includes a specialty-specific classification of commonly prescribed drugs. This component of the domain ontololgy not only is available to the decision-support modules in EON, but also can inform the clinical user interface (see Figure 4).*

4 Discussion

Our laboratory developed the Protégé methodology primarily as a general approach for the construction of knowledge-based systems. The use of explicit, reusable and sharable ontologies, domain models, and problem-solving components has broad applicability in general software engineering, however. Our work demonstrates that, once developers fashion an ontology that captures the basic concepts and relationships among concepts in a given application area, that ontology can guide the execution of large numbers of general software components that contribute to a final system. In the case of EON, for example, we have seen that the original domain ontology defines the semantics of the knowledge-acquisition tool that allows developers to enter individual protocol descriptions (i.e., the instantiated domain models that provide static inputs to the EON components at run time). Furthermore, the domain ontology defines the database schema used by the Chronus component of the Tzolkin mediator, as well as the properties of the clinical parameters processed by the RÉSUMÉ component. Although EON has been designed as a collection of CORBA-compliant *middleware* components, any clinical information system that embeds EON also can take advantage of the domain ontology, as demonstrated by our Java-based EON front end (see Figure 4). Thus, all the components of the EON system interoperate with a single, shared representation of the static knowledge that defines the application area.

4.1 Ontologies and Software Engineering

The reuse of the domain ontology throughout the EON system has important advantages. The most apparent benefit is the ease of system maintenance that results from having only a single locus of information that developers must update when the application area evolves. In a given medical discipline for which EON might be used to assist protocol-based care, system maintainers can respond to the advent of new classes of laboratory tests, clinical interventions, or patient descriptions simply by updating the domain ontology (see Figure 2), and then allowing the changes to propagate throughout the system components. This pattern of maintenance clearly is much simpler than requiring developers to examine every module in a large system for possible references to the modified domain knowledge and to reprogram each component as necessary.

Each problem-solving component in the EON system has a clearly identified source for the domain knowledge that the component operates on—namely, the protocol knowledge base. The task that each component performs is defined independently of the entries either in the protocol domain knowledge bases or in the domain ontology. The relationships among all the EON components are well delineated, and debugging of the overall system is correspondingly enhanced. Indeed, assuming that the reusable problem-solving components such as the therapy planner and the eligibility-determination system are fully tested and reliable, when run-time errors occur, it is easy to trace the failure either to a faulty proposition in the protocol domain knowledge base, or to a misconceptualization in the domain ontology.

Given that the EON components may operate in a variety of clinical application areas, and that application areas themselves evolve over time, there may be several sets of domain ontologies (and, correspondingly, a large number of different protocol domain models derived from those ontologies) with which the EON components may need to interact in a particular installation. Despite potential problems in version management, all domain ontologies and protocol models can be stored in a compartmentalized fashion, separate from the problem-solving components that operate on them. As a consequence of this modularization, the principal requirement at run time simply is to assure that the relevant ontology and protocol domain models are being used to inform the other EON components. Our research group, like several other teams of investigators (Gruber, 1993; van Heijst et al., 1995), continues to explore new approaches for archiving, indexing, and retrieving appropriate selections from on-line libraries of domain ontologies.

Investigators in the area of software engineering increasingly emphasize the importance of making ontologies and domain models explicit when building computer programs (Sutcliffe, et al., 1996; Regoczei and Plantinga, 1987; Hayes-Roth, 1994). For example, the Domain-Specific Software Architectures (DSSA) program recently supported by the United States Defense Advanced Research Projects Agency has had as its centerpiece the notion of building software artifacts by first defining an appropriate domain ontology and domain model (Hayes-Roth, 1994). The DSSA philosophy is that software engineers should construct an explicit, semiformal domain description as one of the first steps in designing a system. The domain model (which primarily defines concepts and relationships, and which thus is more like an ontology) provides an exhaustive catalogue of the domain concepts about which the software will need to be concerned. In the DSSA approach, the domain model (ontology) is an external source of documentation to which developers can refer when implementing their code. As new pieces of program code are written, the programmers can refer to the domain model to ensure that their software's references to elements in the application area are internally consistent, and that no new domain concepts get written into the software without first documenting those concepts in the domain model.

The Protégé approach takes the DSSA methodology one step further, by asking software engineers to represent the domain ontology in a machine-readable format. The list of concepts and relationships is available not only to provide system documentation, but also to inform the knowledge-acquisition system, the database system, and the problem solvers that operate on the domain knowledge in a direct and transparent fashion. Of course, a consequence of this additional functionality is that developers must represent the domain ontology in a restricted, formal language.

The Protégé Ontology Editor enforces considerable structure on the way in which developers define domain concepts. We believe that this degree of rigidity is quite justified, given the benefits that result from the availability of domain ontologies and domain models in machine-processable form. Other recent methodologies for development of knowledge-based systems, such as commonKADS (Schreiber et al., 1994) and GAMES II (van Heijst et al., 1995) have adopted similar views on the role of machine-readable domain ontologies in the system-development process.

The language for expressing ontologies in Protégé is a frame-based representation system in which classes have slots of defined cardinality and data type. Slots may have data types such as integer, float, string, or Boolean, or may take on values that represent instances of other classes in the ontology (e.g., when class called *prescription* has a slot called *drug-prescribed* that takes on as values instances of another class called *drug*). When the data type of a slot is *instance*, the ontology-definition language allows the developer to set explicit constraints on the classes whose instances are allowed as values for that slot. When the data type of a slot is *string*, the language allows the user optionally to specify a grammar that restricts the kinds of strings that may be used as values for that slot. Some ontology-definition systems such as Ontolingua (Gruber, 1993) allow developers to enter arbitrary logical expressions (axioms) to place further constraints on slot values (e.g., an axiom such as a *prescription* may not specify *intravenous* administration of a *drug* if the form of the drug is *tablet*). Protégé currently does not have the facility to support general axioms, specifically because Protégé has no way of enforcing such arbitrary constraints (e.g., by preventing a user from entering a prescription for oral administration of an injectable drug). In Ontolingua, such axioms provide the developer with documentation of semantic relationships, but are not enforced by the computer system. We believe that axioms should provide more than documentation, however, and that knowledge-acquisition tools should be able to use such axioms to check for semantic inconsistencies in the information that users enter. We currently are investigating the possibility of incorporating into our ontology-representation system a constraint language that is more restrictive than the one available in Ontolingua (which assumes full first-order logic), so that constraints on elements of the ontology can be verified by the computer in a tractable manner. The sanctioning rules in GRAIL (Rector et al., 1993) constitute one such restricted constraint language, although constraints in GRAIL are tied to the semantics of particular kinds of medical concepts. For our Protégé work, we are seeking a constraint language that can put restrictions on the classes and instances of an arbitrary ontology, but that will not require the use of complex theorem proving to evaluate expressions.

4.2 Maintaining the Semantics

The Protégé methodology requires developers to construct domain ontologies as organizing frameworks for their software-development activities. Our approach does not prescribe a means for developers to build those ontologies in the first place. We view ontology development fundamentally as a creative, modeling activity, and recognize that the distinctions that developers will make about a given application area typically are ad hoc and motivated by the kinds of tasks that the developers envision information technology will address (Musen, 1992). For example, our empirical experience when creating an ontology of clinical concepts for a computer-based progress-note system confirms that a physician can construe patient-specific descriptors in highly idiosyncratic ways, based on how the physician might foresee using those descriptors in practice (Musen et al., 1995b). The result is that there is no canonical model for how ontologies for any particular application area should be defined. The ontological distinctions that system builders make necessarily reflect the distinctions that they anticipate end users will need to make in their work.

Computers are vehicles for communication. When developers construct a domain ontology, they are defining the basis of communication between the developers and the computer system's end users. Winograd and Flores (1986) refer to the terms that constitute such an ontology as a *systematic domain*, a collection of symbols with precise, agreed-upon semantics shared by all people who may interact with the computer—developers, end users, and maintainers. In this sense, the terms that form the basis for Protégé's domain ontologies constitute systematic domains. The ontologies add to the terms of these systematic domains explicit representations of some of the salient semantic relationships among concepts, including taxonomic and partonomic relationships, and constraints on cardinality and data type. Ultimately, however, the semantics of the concepts themselves (e.g., "What is a *prescription*?", "What is a *drug*?") are not declared in the ontology, but are part of the assumed background knowledge that allows both developers and users to interpret the contents of the ontology in the first place. Even when an ontology includes axioms about the necessary and sufficient conditions required to classify each concept (e.g., as in a description logic; Campbell et al., 1996), the symbols used to define those conditions typically lack explicit semantics themselves, and ultimately it is the shared background of the developers and users that allows these individuals to interpret the full meaning of each concept in the ontology.

This reliance on shared background to interpret the meanings of terms in a systematic domain is not an arcane issue: This phenomenon is reflected in every interaction that every user may have with every computer system. As Norman (1986) cogently describes, the developer of a computer system has a mental model of what she wants to communicate to the end user who interacts with the finished piece of software. (In the case of EON, the developers have a mental model of how the different problem solvers will assist with aspects of protocol-based care.) End users of the software perceive the behavior of the computer, and from that behavior infer their own mental model of what it is that the original software developer intended to communicate. For example, the end user of a clinical information system containing the EON components may be informed that certain patients are *potentially eligible* for a given protocol. Because the term *potentially eligible* has a precise meaning defined in terms of specific logical operations on protocol eligibility criteria and patient data, the end user must develop his own ©1mental model of what *potentially eligible* means in the context of interacting with the system that contains EON. In general, for end users to be fully effective in interacting with the computer, they must be able to develop mental models of protocol-eligibility determination that match the mental models of those developers (1) who created the ontology of protocol-eligibility terms and (2) who encoded descriptions of the eligibility criteria of specific protocol in terms of that ontology.

All interactions with the decision-support elements of the EON system require end users to develop mental models of the meanings of the terms used to communicate protocol-specific advice. These meanings are the same as those ascribed to the concepts in the domain ontologies that guide the whole software-engineering process. Whereas a domain ontology developed using Protégé serves the important functions of cataloging the relevant domain concepts and of defining a subset of relationships among those concepts, much of that ontology's semantics unfortunately must remain implicit in the mental model of the developer. It is the developer's common background with his user community, however, that allows the users to apprehend those implicit semantics by observing the computer's behavior and thus constructing their own mental models of the developer's intentions.

The notion that so much of an ontology's semantics can be underspecified seems at odds with the formal representation systems used in Protégé and other approaches to construct domain ontologies in the first place. Clearly, there is an assumption that, by representing the ontology in logic, developers can assure well-specified semantics. Unfortunately, however, no representation system can overcome these semantic difficulties. There is an allure when using systems such as Ontolingua to enhance the representation of ontologies by specifying multiple axioms for each class to define logical relationships that are not captured directly in the frame hierarchy. Even if it were possible to specify semantic relationships completely, however, those declarative semantics become undone when the domain ontology is interpreted by a problem-solving method, such as EON's therapy planner or eligibility-determination module. Because problem-solving methods are procedural components, they apply operational semantics to the data structures that they process—disregarding any declarative specifications that the methods otherwise cannot interpret.

4.3 Ontologies and Terminologies

Standard controlled terminologies such as the ICD lack the explicit structure and declarative relationships found in the ontologies processed by Protégé. Controlled terminologies may specify hierarchical relationships among terms, but they do not attempt to define concepts in terms of attributes and other relationships. Some workers are attempting to impose additional structure on controlled terminologies by various means. The Medical Entities Dictionary (MED; Cimino et al., 1994), for example, makes explicit the hierarchical relationships that exist in a number of standard terminologies in use at Columbia-Presbyterian Medical Center. Current work in the Kaiser-Permanente Medical Group attempts to add additional distinctions to an expanded version of SNOMED using description logic (Campbell et al., 1996). These efforts improve on the underlying controlled terminologies significantly by clarifying relationships among concepts and by facilitating both manual and semi-automated maintenance over time. They also provide a common database that applications programs at a given site can query to identify appropriate elements of the terminologies. In the EON architecture (see Figure 1), all components have access to the domain ontology precisely for the purpose of making these kinds of queries.

There is a difference in the philosophy of our work, however, and that of these other investigators. When system builders create a central database of clinical concepts that can be queried by application programs built independently of that database, the application programs may make a host of assumptions about the clinical domain that are independent from the terminology stored in the database. Although it is advantageous to separate out from these programs as much of the domain-specific terminology as possible, the programs still must make many domain-specific assumptions. A results-reporting system that uses the Medical Entities Dictionary to provide standard names for laboratory tests, for example, may still have buried within it a model of clinical specimens (involving concepts such as time of collection, time of accession, normal ranges, and so on) that exists only in program code and that is not immediately inspectable or editable; an order-entry system that uses a central database of controlled terms for clinical interventions still may make many assumptions about the practice patterns of health-care workers that are inaccessible to direct inspection. Our goal in the EON project is to maximize the knowledge about the entire clinical enterprise represented within our domain ontologies, so that there is a central, declarative representation of all the domain knowledge that any software module in the system might want to access. We seek not only an explicit enumeration of terms or clinical concepts, but also a shared resource that defines a comprehensive model of the clinical enterprise—allowing us to automate clinical tasks by using generic "shell" programs that access this shared resource to obtain essentially all their domain-specific knowledge.

Our approach requires us to use controlled terminologies such as SNOMED as building blocks for our domain ontologies. To create an ontology for a particular area of medicine, we would not necessarily want to import all the concepts used in a given controlled terminology, but rather we would attempt to select a subset of terms relevant for the domain at hand. We thus vie w controlled terminologies—and computer-based terminology servers such as the GALEN (Rector et al., 1993) and the MED (Cimino et al., 1994)—not only as resources to be used primarily by application programs, but also as essential inputs to help software engineers in the construction of the more comprehensive domain ontologies needed to build component-based systems such as EON.

As demonstrated by the EON architecture, separation of the domain knowledge from the problem solvers leads to a new kind of software engineering. Software engineers consolidate their domain modeling as a separate development activity, and map generic, reusable problem-solving components to their domain models in explicit ways. We already have evidence that the use of domain ontologies to generate domain-specific knowledge-acquisition tools facilitates domain modeling; our expectation is that the Protégé methodology will have measurable affects on system maintenance as well. Our component-based approach to software engineering also is readily supported by emerging standards for distributed object systems, such as CORBA.

The current Protégé Ontology Editor does not offer the developer very much support for merging several ontologies together, or for incorporating into ontologies concepts that might be derived from some vocabulary server. Future work in our group will concentrate on principled ways to bring two or more ontologies together, and to import into our ontologies lists of concepts derived from standardized sources. Our ultimate goal is to develop the technology necessary to manage libraries of ontologies that might be downloaded from distributed servers on the Internet, and to aid software engineers in adapting those ontologies to the requirements of the applications that ultimately they wish to develop.

Acknowledgments

This work has been supported in part by grants LM05708 and LM05304 from the National Library of Medicine, by a grant from the National Action Plan on Breast Cancer, and by contract N66001-94-D-6052 supported by the Defense Advanced Research Projects Agency. Dr. Musen is the recipient of National Science Foundation Young Investigator Award IRI-9257578. Samson Tu, John Gennari, and Aneel Advani participated in many valuable discussions on issues of vocabulary and ontology in Protégé and EON. Ray Fergerson and Zaki Hasan have provided superb programming support for the Protégé and EON projects, respectively.

References

Blum, B.I. (1991). The software process for medical applications. In: Timmers, T. and Blum, B.I (eds). *Software Engineering in Medical Informatics*. pp. 3–26. Amsterdam: North-Holland.

Booch, G. (1994). *Object-Oriented Analysis and Design with Applications*. Second edition. Reading, MA: Addison–Wesley.

Campbell, K.E., Cohn, S.P., Chute, C.G., Rennels, G.D., and Shortliffe, E.H. (1996). Gálapagos: Computer-based support for evolution of a convergent medical terminology. In: *Proceedings of the AMIA Fall Symposium*. American Medical Informatics Association. Washington, DC, October, pp. 269–273.

Cimino, J.J., Clayton, P.D., Hripcsak, G., and Johnson, S.B. (1994). Knowledge-based approaches to the maintenance of a large controlled medical terminology. *Journal of the American Medical Informatics Association* **1**:35–50.

Clancey, W.J. (1983). The epistemology of a rule-based expert system: A framework for explanation. *Artificial Intelligence* **20**:215–251.

Das, A.K., Shahar, Y., Tu, S.W., and Musen, M.A. (1994). A temporal-abstraction mediator for protocol-based decision-support systems. *Proceedings of the Eighteenth Annual Symposium on Computer Applications in Medical Care,* Washington, DC, November, pp. 320–324.

Das, A.K. and Musen, M.A. (1994). A temporal query system for protocol-directed decision support. *Methods of Information in Medicine,* **33**:358–370.

Eriksson, H., Puerta, A.R., and Musen, M.A. (1994). Generation of knowledge-acquisition tools from domain ontologies. International Journal of Human–Computer Studies, 41:425–453.

Eriksson, H., Shahar, Y., Tu, S.W., Puerta, A.R., and Musen, M.A. (1995). Task modeling with reusable problem-solving methods. *Artificial Intelligence* **79**:293–326.

Feinstein, A.R. (1988). ICD, POR, and DRG. Unsolved scientific problems in the nosology of clinical medicine. *Archives of Internal Medicine* **148**:2269–2274.

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**:399–424.

Gruber, T.R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5, 199-220.

Guarino, N., and Giaretta, P. (1995). Ontologies and knowledge bases: Toward a terminological clarification. In N.J.I. Mars (ed.), *Towards Very Large Knowledge Bases*, IOS Press, pp. 25-32.

Hayes-Roth, F. (1994). Architecture-based acquisition and development of software: Guidelines and recommendations from the ARPA Domain-Specific Software Architecture (DSSA) program. Technical report. Teknowledge Federal Systems, Palo Alto, CA.

Musen, M.A. (1992). Dimensions of knowledge sharing and reuse. *Computers and Biomedical Research* **25**:435–467.

Musen, M.A., Carlson, C.W., Fagan, L.M., Deresinski, S.C., and Shortliffe, E.H. (1992). T-HELPER: Automated support for community-based clinical research. In: *Proceedings of the Sixteenth Annual Symposium on Computer Applications in Medical Care,* pp. 719–723, Baltimore, MD.

Musen, M.A., Gennari, J.H., Eriksson, H., Tu, S.W., and Puerta, A.R. (1995a). PROTÉGÉ-II: Computer support for development of intelligent systems from libraries of components. In: *Proceedings of MEDINFO '95*, Eighth World Congress on Medical Informatics, pp. 766–770, Vancouver BC.

Musen, M.A., Wieckert, K.E., Miller, E.T., Campbell, K.E., and Fagan, LM. (1995b) Development of a controlled medical terminology: Knowledge acquisition and knowledge representation. *Methods of Information in Medicine* **34**:85–95.

Musen, M.A., Tu, S.W., Das, A.K., and Shahar, Y. (1996). EON: A component-based approach to automation of protocol-directed therapy. *Journal of the American Medical Informatics Association* **3**:367–388.

Norman, D.A. (1986). Cognitive engineering. In: Norman, D.A. and Draper, S.W. *User Centered System Design: New Perspectives on Human–Computer Interaction*. pp. 31–61. Hillsdale, NJ: Lawrence Erlbaum Associates.

Orfali, R., Harkey, D., and Edwards, J. (1996). *The Essential Distributed Objects Survival Guide*. New York: John Wiley & Sons.

Rector, A.L., Nowlan, W.A., and Glowinski, A. (1993). Goals for concept representation in the GALEN project. In: *Proceedings of the Seventeenth Annual Symposium on Computer Applications in Medical Care*, pp. 414–418, Washington D.C.: McGraw-Hill.

Regoczei, S., and Plantinga, E.P.O. (1987). Creating the domain of discourse: Ontology and inventory. *International Journal of Man-Machine Studies* **27**:235–250.

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

Shahar, Y. and Musen, M.A. (1996). Knowledge-based temporal abstraction in clinical domains. *Artificial Intelligence in Medicine*, **8**:267–298.

Shahar, Y. (in press). Knowledge-based temporal abstraction. *Artificial Intelligence*.

Sutcliffe, A.G., Benyon, B., and van Assche, F. (eds) (1996). *Domain Knowledge for Interactive System Design*, Proceedings of the IFIP TC8/WG8.2 Conference on Domain Knowledge in Interactive System Design, Switzerland, May 1996. London: Chapman and Hall.

Tu, S.W., Eriksson, H., Gennari, J.H., Shahar, Y., and Musen, M.A. (1995). Ontology-based configuration of problem-solving methods and generation of knowledge-acquisition tools: Application of PROTÉGÉ-II to protocol-based decision support. *Artificial Intelligence in Medicine*. 1995; **7**:257–289.

Tu, S.W. and Musen, M.A. (1996). The EON model of intervention protocols and guidelines. *Proceedings of the AMIA Fall Symposium*. American Medical Informatics Association. Washington, DC, October, pp. 587–591.

Tu, S.W., Kemper, C.A., Lane, N.M., Carlson, R.W., and Musen, M.A. (1993). A methodology for determining patients' eligibility for clinical trials. *Methods of Information in Medicine* **32**:317–325.

van Heijst, G., Falasconi, A., Abu-Hanna, G., Schreiber, G., and Stefanelli, M. (1995). A case study in ontology library construction. *Artificial Intelligence in Medicine* **7**:227–255.

Waterman, D.A., Hayes-Roth, F., and Lenat, D.B. (eds). (1983). *Building Expert Systems.* Reading, MA: Addison–Wesley.

Winograd, T., and Flores, F. (1986). *Understanding Computers and Cognition: A New Foundation for Design.* Reading MA: Addison–Wesley.