# Toward Ontology-Based Frameworks for Knowledge-Acquisition Tools

Angel R. Puerta, Robert Neches*, Henrik Eriksson, Pedro Szekely*, Ping Luo*, Mark A. Musen

Medical Computer Science Group
Knowledge Systems Laboratory
Stanford University
Stanford, CA 94305-5479, USA

*USC/Information Sciences Institute
Admiralty Way
Marina del Rey, CA 90292, USA

Emails:{puerta, eriksson, musen}@camis.stanford.edu—{neches, szekely, ping}@isi.edu

## ABSTRACT

One of the strongest limitations of knowledge-acquisition metatools—tools that automate the development of knowledge-acquisition tools—is that the interface-design knowledge employed by such metatools to produce domain-specific knowledge-acquisition tools is not explicit and cannot be used by other metatools. Therefore, the usefulness of a metatool is restricted to its own implementation environment. A possible solution to this problem is the definition of a shareable ontology of knowledge-acquisition tools that represents explicitly the relevant knowledge of tool design and development. We take the position that the design of knowledge-acquisition tools is similar to the design of user interfaces, and that an ontology of knowledge-acquisition tools is a special case of an interface ontology. We propose that a shareable interface ontology can be defined by analyzing interface ontologies used in model-based user-interface development systems, and by resolving differences and conflicts among the ontologies—a process called *model alignment.* We present the initial results of a model alignment for the interface ontologies of two user-interface development systems: SHELTER and Mecano. We also show how the resulting shareable interface ontology underpins a framework in which each system can be applied to different parts of the process of generating a single, domain-specific knowledge-acquisition tool, thus creating a generation of metatools useful across multiple implementation environments.

## 1. KNOWLEDGE-ACQUISITION TOOLS AS INTERFACES

The traditional view of knowledge-acquisition tools is that of independent systems that form part of the development cycle of a knowledge-based system. From that view, it follows that the design of a knowledge-acquisition tool is normally tied up to the design of a single knowledge-based system, or in the best case, a restricted class of knowledge-based systems. As a consequence, the development of a domain-specific knowledge-acquisition tool is an important resource-consuming task that results in a system that has a very limited applicability and a very short life span.

One successful approach to alleviate the burden of developing knowledge-acquisition tools is the use of *metatools* such as PROTÉGÉ-I [Musen, 1989] and DOTS [Eriksson, 1993b]. Metatools are able to *generate* knowledge-acquisition tools from high-level specifications, thus allowing knowledge engineers to construct tools by manipulating abstract design objects. For example, PROTÉGÉ-I produces automatically a tool from a specification of a problem-solving method (skeletal-plan refinement), and DOTS provides a small number of primitives that can be used to build entire tools.

Although the savings in resources provided by metatools is significant, the usefulness of such tools is restricted to their own development environment. That is, a system like PROTÉGÉ-I provides no help to a developer of a knowledge-acquisition tool with a different problem-solving specification (i.e., different from skeletal-plan refinement). Reuse is stymied even just by moving to a different window system in which the tool must be implemented. This is the direct result of

metatools using *implicit* mappings between the high-level specifications that the knowledge engineer manipulates and the resulting knowledge-acquisition tool. The implicit mapping means that there is no access to the knowledge in the metatools about the process of designing and generating a knowledge-acquisition tool. If that knowledge were accessible in an explicit manner to a knowledge engineer, and if it could be represented in a shareable format, then the design process of a metatool would be useful in other metatools regardless of the underlying system software.

One solution to the problem of representing design and generation knowledge for knowledge-acquisition tools is the definition of a general ontology—a knowledge representation of objects and their interrelationships—for knowledge-acquisition tools. Such an ontology would allow the representation of any tool and the sharing of that representation among metatools. We take the point of view that ontologies for knowledge-acquisition tools are special cases of user-interface ontologies. A knowledge-acquisition tool serves as an interface between a user and a knowledge base by manipulating two different representations of the same knowledge: One representation for the knowledge base and one representation for the visual display of the knowledge to the user. Thus, the key tasks of any knowledge-acquisition tool are *presentation* and *dialogue,* which are the main tasks of any user interface.

In this paper, we analyze the design of knowledge-acquisition tools as user-interface design. We present two systems, SHELTER and Mecano, that employ an emerging technology, called model-based user-interface development, to produce user interfaces. The central piece of both systems is an *interface ontology*, or interface *model*, that drives the interface design process. These interface ontologies can be specialized to represent knowledge-acquisition tool ontologies. Furthermore, by carrying a process of model alignment to resolve the differences and conflict between the SHELTER and Mecano interface ontologies, we can produce a single, shareable interface ontology for knowledge-acquisition tools. The shareable ontology can then be the basis for a framework where SHELTER and Mecano can be utilized during the development of a specific knowledge-acquisition tool regardless of the implementation environment of each system. We present the rationale and initial results of the model alignment process for the ontologies of SHELTER and Mecano and develop the scenario for the joint use of both systems during the development of knowledge-acquisition tools.

The rest of this paper is organized as follows: Section 2 analyzes the task of knowledge acquisition at the software-tool level with the purpose of defining the scope of the design knowledge that must be represented in an ontology of knowledge-acquisition tools. Section 3 describes how SHELTER and Mecano deliver end-product knowledge-acquisition tools that try to address some of the requirements suggested by the analysis in Section 2. In Sections 4 and 5, we describe the interface models used by both systems and discuss how these can be aligned to produce a more general model that can be shared with other implementations.

## 2. TASK ANALYSIS OF KNOWLEDGE-ACQUISITION TOOLS

An ontology of knowledge-acquisition tools allows a developer direct access to explicit representations of the design knowledge required for such tools. To understand the scope of design knowledge that must be captured in the ontology—and how such knowledge corresponds to general user-interface design knowledge—it is necessary to conduct a task analysis of knowledge-acquisition tools such as the one presented in this section.

In a discussion of cognitive issues arising at various phases of the knowledge-acquisition process, Neches pointed out a number of problems that a knowledge-acquisition system must be designed to address [Neches, 1992]. These include:

- *identifying the domain knowledge that is to be formally encoded*
- *mapping that knowledge into the encoding utilized by the computer*;

- *validating that the encoding accurately reflects what the domain expert was trying to express*;
- *validating that what was expressed is indeed the appropriate knowledge to apply.*

Addressing all four of these needs requires that a comprehensive knowledge-acquisition system must support a number of enabling capabilities. In turn, a knowledge-acquisition metatool must enable one to build the knowledge-acquisition systems that support those capabilities.

Among the necessary capabilities are the following:

- *Visualization and browsing of existing knowledge*. The knowledge engineer and domain expert must have some means of seeing what knowledge has currently been entered into the system, both concepts and instances. Needs include the ability to see various hierarchical structures of the knowledge, *IS-A* and *part-of* hierarchies being the most common examples. Also necessary is the ability to see structure of individual concepts (e.g., attributes and values), as well as the need to see clusters of related concepts.

- *Editing and review of knowledge*. Mechanisms are needed to allow the concept definitions to be altered, to create or change relationships among concepts, and to change the problem-solving methods utilized.

- *Entry of new knowledge.* As the skeleton of a knowledge base takes shape, knowledge engineers and domain experts also need the ability to populate those knowledge bases with instances of the concepts that have been defined. Ideally, the interactions that allow them to do so make it possible to enter large numbers of relatively similar items quickly and painlessly, so that this work can be performed with minimal demands on the time of expensive, skilled personnel.

- *Search and retrieval of specific parts of the knowledge base.* Like Rome, large knowledge bases are not built in a day. Knowledge acquisition systems must allow users to provide descriptions of knowledge so that they can return to work on portions of the knowledge base. Also, as will be discussed further below, the ability to describe and access portions of a knowledge base gives a user access to rich sources of examples from existing knowledge that can stimulate the description of new knowledge.

- *Sequencing of knowledge entry.* As the organization of a particular knowledge base becomes increasingly better understood during its evolution, it becomes possible to impose more order on the way information is acquired to populate the knowledge base. By providing for control over acquisition dialogues, it becomes possible to request related information at the same time. This facilitates completeness and consistency, as well as making it possible to structure the interaction to minimize demands upon users' time and energy.

- *Annotation.* In the early stages of defining a knowledge base, formal representation may not be feasible. Knowledge engineers and domain experts may not yet have figured out exactly how to express what they have in mind, or they may be unsatisfied with what they have expressed. In such cases, a means is needed for informally recording notes that provide place holders for information to be provided in the future or that record concerns about existing entries in the knowledge base. (Indeed, some researchers argue that formal representations can be dispensed with entirely when the knowledge base is intended only for human consumption.)

## 3. THE SHELTER AND MECANO APPROACHES

The design and implementation of the functionality outlined in Section 2 is a difficult research area in knowledge acquisition. We follow the generative production paradigm common to knowledge-acquisition metatools. Our systems, SHELTER (being developed at USC/ISI) and Mecano (being developed at Stanford), are ontology-based frameworks that allow developers to manipulate tool-

design knowledge in an explicit manner to create knowledge-acquisition tools, as special cases of user-interface design. Both employ an emerging technology called model-based user-interface development but emphasize different aspects of the lifecycle of interface development. SHELTER concentrates on the ontology-editing phase whereas Mecano focuses on the tool-instantiation phase. Therefore, it is possible that given a shareable ontology of knowledge-acquisition tools, SHELTER and Mecano could form a comprehensive framework for tool design across implementation environments. In this section, we introduce the concept of model-based user-interface development, present both systems within the context of that technology, and detailed the application of the technology to the design and implementation of knowledge-acquisition tools.

### 3.1. Model-Based User-Interface Development

SHELTER and Mecano are examples of an emerging technology called model-based user-interface development. In this section, we present the basic architecture of model-based systems and describe the particular implementations of each environment.

Currently, building a user interface involves creating a large procedural program. Model-based programming provides an alternative new paradigm. In the model-based paradigm, system builders create a declarative model—an instance of a generic interface ontology—that describes the tasks that users are expected to accomplish with a system, the functional capabilities of a system, the style and requirements of the interface, the characteristics and preferences of the end users and the I/O techniques supported by the delivery platform. Based on the model, a much smaller procedural program then determines the behavior of the system.

This has several advantages. The declarative model is a common representation that tools can reason about, enabling the construction of tools that automate various aspects of interface design, that assist system builders in the creation of the model, that automatically provide context-sensitive help and other run-time assistance to users. The common model also allows the tools that operate on it to cooperate. Because all components of the system share the model, this promotes interface consistency within and across systems and reusability in the construction of new interfaces. The declarative nature of the model helps system builders understand and extend systems. For these reasons, the paradigm is an important next step in computer science's ever-continuing quest to handle greater complexity through increasingly high-level specifications.

The key to our work is a model-based approach to tools for designing, prototyping, and maintaining user interfaces. Model-based systems can be thought of as a marriage of object-oriented programming and declarative specifications of semantics. A model-based system maintains a hierarchy describing the conceptual objects in its topic area, their behavior, and the relationships between objects. Ideally, in this approach, descriptions of the behavior the system is to evince are associated with concepts at as high a level in the abstraction hierarchy as is possible and are inherited down to lower level concepts. This is similar to object-oriented programming, but model-driven programming seeks to specify behavior in a fashion that reduces the burden of handling control explicitly.

The model-driven approach's focus on abstraction hierarchies supports a customization-oriented style of system-building in which core systems are built for a class of applications, rather than for a single application. Knowledge is added at lower (i.e., more specialized) levels in the hierarchies in order to extend the core system for a particular application. This heavily promotes reusability of software (and thereby reduces maintenance and development costs that come with building custom systems from scratch). Furthermore, the formally analyzable nature of declarative representations utilized in model-driven approaches makes it possible to provide much more powerful and sophisticated aids to system builders and end users than would otherwise be possible.

### 3.2. SHELTER

The primary goal of SHELTER is to provide extensive interface-ontology editing facilities. The system facilitates sharing and reuse by helping developers find candidate material to reuse and ensuring that it is used properly. There are two key ideas in SHELTER. One is an interaction paradigm that encourages reuse of specifications, embodied in a set of browsing and retrieval tools. The other is a set of methods for helping ontology builders to record design-rationale metaknowledge using structured forms that the system can interpret to assist developers in ensuring the appropriateness of later modifications. These ideas are realized through the integration of three tools:

- *BACKBORD:* A knowledge base browser that helps create queries and other specifications. In SHELTER, the primary use of BACKBORD is as a mechanism to promote reuse of representations (such as interface ontologies). BACKBORD allows developers to create descriptions incrementally. It is intended to make it easy for developers to discover representational work that they can reuse, and to make it easy to incorporate that work in the model they are building via copying or modifying.

- *TINT:* An intelligent notes mechanism for capturing and manipulating information that cannot be fully anticipated at design-time. TINT is intended for two kinds of use within SHELTER. First, it can be used to capture knowledge that is not yet understood well enough to formalize. In that usage, it can be thought of as recording knowledge-acquisition goals: that is, topics or concepts where further work is planned. Second, TINT can be used to annotate formally-represented knowledge in order to provide information about the model of potential significance to others considering making use of that knowledge.

- *Scenarios/Agendas:* a language and planning system for scheduling and controlling the activities of multiple agents dealing with multiple, extended, interleaved problem solving tasks. Like notes, scenarios can be used as stubs for knowledge which has not yet been fully formalized. While notes deal with declarative, structural knowledge, scenarios deal with problem solving and procedural knowledge. In this mode of use, scenarios are used to define, in effect, the outline of a problem-solving process and end users are asked to fill in the details of such process by performing specific problem-solving steps not yet implemented in the system.

These tools, and the various views that they support constitute the base set from which any application-specific knowledge acquisition environment is constructed. Instances of interface ontologies defined with SHELTER are implemented with the HUMANOID user interface management system [Szekely, Luo, Neches 1992].

### 3.3. Mecano

Whereas SHELTER emphasizes the ontology definition phase of knowledge-acquisition tool development, the Mecano environment emphasizes the phases of ontology instantiation and tool implementation. In Mecano, the design of knowledge-acquisition tools, when visualized as user interfaces, has three main concerns: Definition of presentation and dialogue, translation between knowledge representations, and maintenance of iterative tool designs. In this section, we examine each one of these three activities.

The central design problem for a knowledge-acquisition tool is the definition of a *directed dialogue* with a user (knowledge engineer or domain expert). This dialogue translates into the entry of new knowledge into a knowledge base and into the refinement of existing knowledge from that knowledge base. The purpose of directing the dialogue is to ensure that the knowledge to be acquired, and revised, forms a complete and consistent unit. In essence, a directed dialogue establishes domain-specific limitations to the functionality available in a knowledge-acquisition tool. A simple example would be guiding a medical domain expert through all the windows and

entry fields that must be completed in order to define a component of a treatment plan, such as a chemotherapy. A directed dialogue has two levels that must be addressed separately. High-level dialogue design is concerned with the definition of windows, grouping of knowledge items for each window, and construction of a navigation tree to let the user move from one window to another. Low-level dialogue design determines what type of dialogue element (widget) will represent each type of knowledge item in the tool.

In addition to a dialogue, the knowledge-acquisition tool must have a *presentation* that defines the relative location and the appearance of each dialogue element in the tool. The presentation of a tool is driven by the dialogue and by the domain-specific characteristics of the knowledge to be presented. Both high-level and low-level dialogue design must take place before presentation design can proceed.

Furthermore, the knowledge-acquisition tool may require a translation process between the format used to present knowledge to a user, and the format used to permanently store the knowledge in a knowledge base. This translation is by no means trivial because the knowledge representation in the knowledge base can change according to the needs of the underlying knowledge-based system. Any such change implies a corresponding change to the dialogue and presentation of the tool.

Finally, the design of the dialogue and presentation of a knowledge-acquisition tool is an iterative process. That is, a number of revisions and enhancements to the tool must be made before reaching a stable design. Almost every change to the tool design impacts dialogue, presentation, and translation decisions made prior to that change. In some instances, changes to a part of the design may introduce incompatibilities with other parts of the current design. Changes to the knowledge representation of the tool can cause translation difficulties to and from the representation in the knowledge base. Thus, maintenance issues are as much a design focus in the construction of knowledge-acquisition tools as the definition of dialogues and presentations.

The Mecano environment is integrated into the PROTÉGÉ-II environment for knowledge-based system development [Puerta et al., 1992; Puerta et al., 1993]. PROTÉGÉ-II has been designed to remove limitations in PROTÉGÉ-I that constrained knowledge-based development to a single problem-solving method. Mecano started as the component of PROTÉGÉ-II used in generating and running knowledge-acquisition tools and is being generalized into a user-interface development environment. The use of Mecano, however, is still focused on the construction of knowledge-acquisition tools for PROTÉGÉ-II knowledge-based systems. Figure 1 shows the ontology-based framework of Mecano. Based on ontologies for a domain and a problem-solving method supplied by PROTÉGÉ-II, an intelligent mapping process partially instantiates an interface ontology for a knowledge-acquisition tool. The interface ontology may be edited and custom tailored by a developer, and once ready for use it is implemented as a tool by a run-time system.

The main assumption in Mecano, as it relates to generating tools for PROTÉGÉ-II, is that the users of these knowledge-acquisition tools are domain experts (possibly naive computer users) as opposed to knowledge engineers. Thus, the aspect of tool design most emphasized by Mecano is dialogue. Both the process of ontology mapping and the run-time system work to generate tools that have very specific window-navigation trees and dialogue operations. Users of the tools are guided in the knowledge-editing process so that the knowledge they provide is complete and consistent while at the same time users are steered away from operations that contextually have no meaning at a given stage of knowledge editing.

Presentation, on the other hand, is assumed to be a custom-tailoring operation and little effort is made to automate it. The layout of dialogue elements in windows is subject to so many variables, not the least important one being user preferences, that automatic layout is a very complex, and perhaps futile endeavor [Szekely et al., 1992]. Given that the tools produced by Mecano are to be used by a few at most, many times a single domain expert, it is reasonable to layout manually the

dialogue elements using system-provided interface builders. In many instances, the end user (domain expert) can participate directly in the layout design assisting the developer.

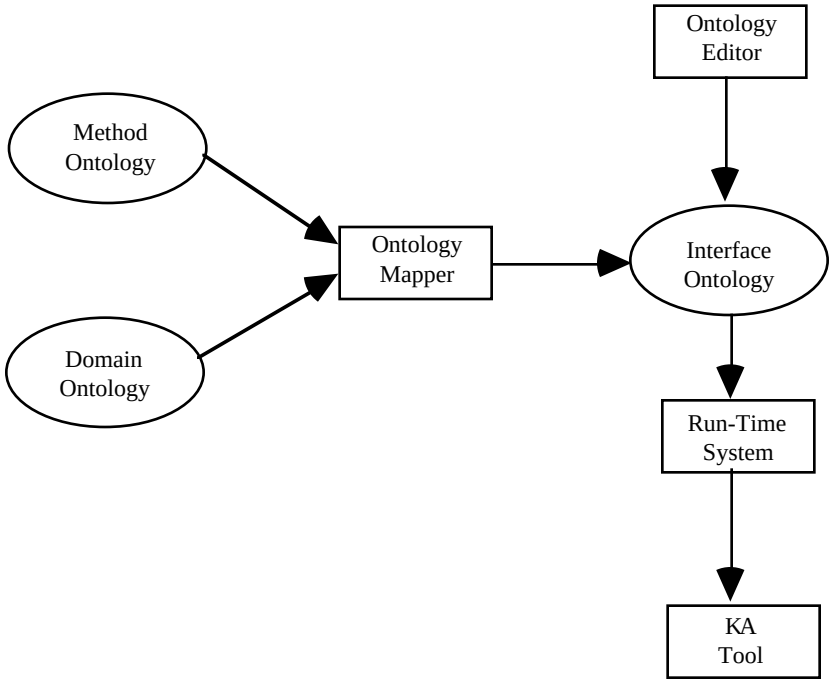


**Figure 1.** Conceptual framework of Mecano. An intelligent mapper creates an instance of an interface ontology based on given domain and problem-solving method ontologies obtained using PROTÉGÉ-II. After possible user editing, the interface ontology is implemented as a knowledge-acquisition tool by a run-time system.

The translation mechanism in Mecano is driven by the format of the knowledge bases in PROTÉGÉ-II systems. This format is determined by the domain ontology, which is shared between Mecano and PROTÉGÉ-II. Although sharing the domain ontology facilitates the communication between Mecano and PROTÉGÉ-II, it also places a heavy maintenance burden on Mecano because any changes made to a domain ontology outside of Mecano result in necessary changes to the design (dialogue and presentation) of the knowledge-acquisition tool corresponding to that domain ontology. Therefore, the issues of translation and maintenance are intertwined in Mecano. Currently, Mecano provides little support for automatic management of changes to the knowledge representations of the tool and of the knowledge base. We are, however, researching possible solutions for the most common cases of domain ontology augmentation and revision that affect the translation between knowledge representations.

The major architectural components of Mecano are shown in Figure 2. Mecano utilizes three different ontologies: A domain ontology and a method ontology defined using PROTÉGÉ-II, and an interface ontology that models knowledge-acquisition tools as interfaces. User-interface development is supported by three design tools: An ontology editor (Maître) [Gennari, 1993], an automated dialogue designer (DASH) [Eriksson et al., 1993a], and a graphical, direct manipulation interface-layout tool (Interface Builder, supplied by the NeXT environment). User-interface implementation is supported by a run-time system called Mart (Mecano at run-time).

PROTÉGÉ-II and Mecano ontologies share the same representation language and can be edited with the same editor. The ontologies are represented as hierarchies of classes and the ontology editor, Maître [Gennari, 1993], is a graphical browser that allows the visualization and revision of ontologies. The role of ontology mapper shown in Figure 1 is carried out by DASH [Eriksson et al., 1993a]. DASH acts as an interface-dialogue designer by inferring parts of the interface ontology for a knowledge-acquisition tool from the ontologies of a domain and of a problem-solving method. During the ontology mapping process, DASH prepares a default presentation for the dialogue components of the knowledge-acquisition tool. This presentation, however, is only preliminary—not usable in most applications—and must be custom tailored. A final layout can be accomplished utilizing Interface Builder, a graphical tool available in the NeXT environment that allows users of the tool to place dialogue elements in windows by direct manipulation.

The interface ontology instantiated by DASH is implemented as a running knowledge-acquisition tool by Mart as a knowledge-acquisition tool. Mart embeds the roles of application-state monitoring and of translation to and from a PROTÉGÉ-II knowledge base. Mart provides support for many types of dialogue elements from simple ones such as menus, and text fields, to complex ones such as text editors and graphical editors.

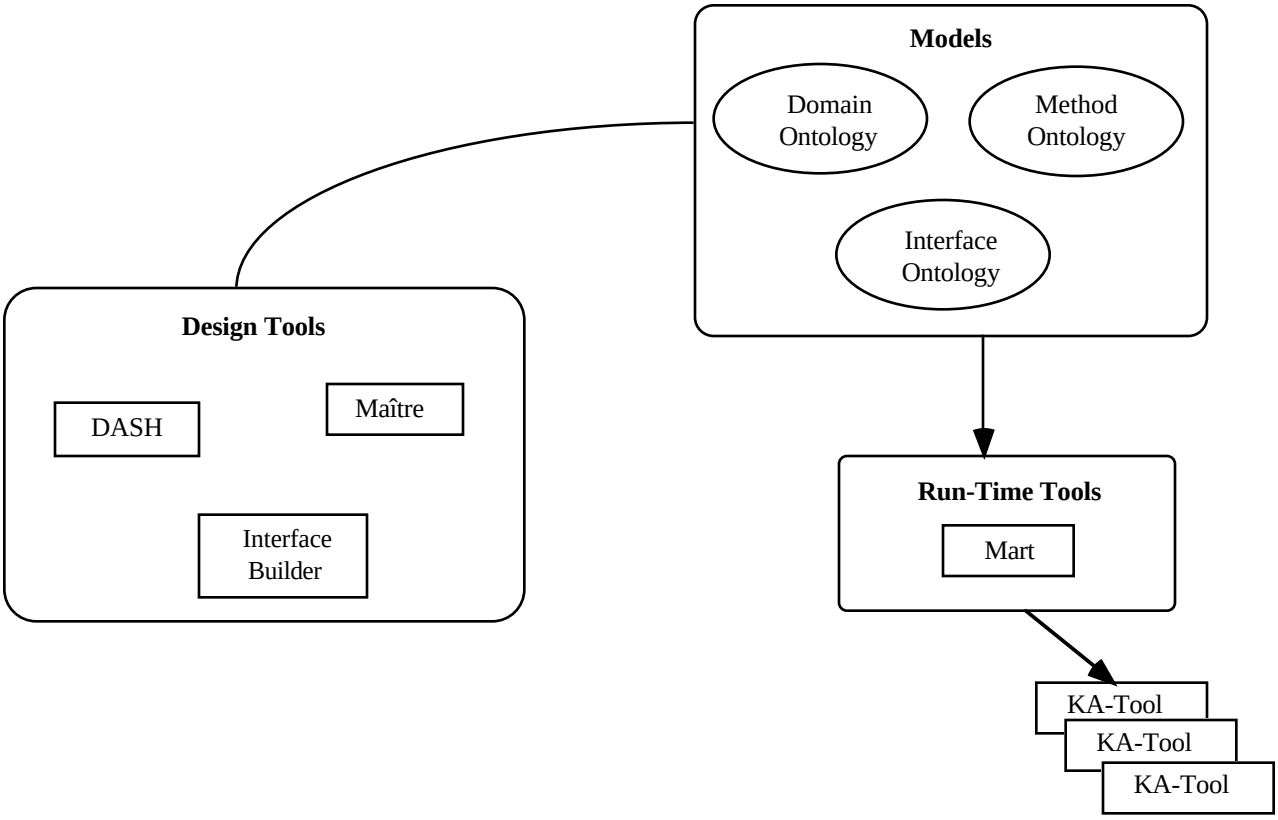


**Figure 2.** The main architectural components of Mecano that implement the conceptual framework of Figure 1. The environment includes an ontology editor (Maître), an intelligent dialogue designer (DASH) and an interface layout tool (Interface Builder, supplied by the NeXT environment). Interface ontologies are implemented as running knowledge-acquisition tools by Mart, the run-time system.

## 4. MODEL ALIGNMENT

Although SHELTER and Mecano are built around similar conceptual frameworks, they differ on the central piece of that framework: The interface ontology. Each system defines and represents the interface with models that have different expressiveness and views of the interface-design process. The ontologies, however, share many characteristics, and through a process of comparison and analysis, it is feasible to attempt to merge both ontologies into a shareable one. A merger of ontologies would more solidly support the claim that the ontology is independent of implementation. The exercise of merging the models can be viewed as a process of *model alignment.* This process is more likely to succeed if, as is the case here, the ontologies to be aligned contain many similarities among them.

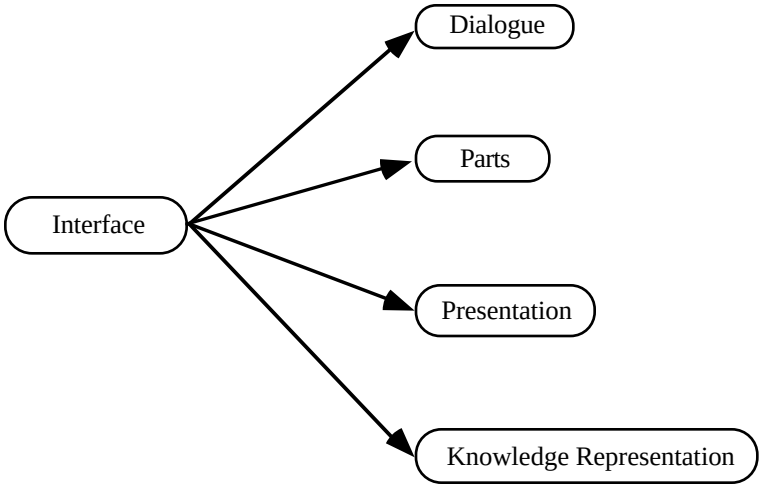


**Figure 3.** Top classes in the *part-of* view of the interface ontology. The dialogue class models the user–tool discourse while the parts class models the dialogue elements such as windows and menus. The presentation class represents the appearance of the dialogue elements and the knowledge representation class models the knowledge types and elements displayed via the dialogue elements.

The purpose of model alignment is not to build a *consensus* ontology. Rather, it is intended to build a shareable ontology, one that can express the interfaces generated by both systems, but that is not necessarily implemented fully by either system. Naturally, model alignment causes each model to be re-analyzed and revised. Inconsistencies must be resolved and syntactic similarities identified during the process so that the resulting ontology is concise but at least as expressive as any of the individual ontologies being aligned. In this section, we present our current progress toward an interface ontology that aligns the SHELTER and Mecano models.

Figure 3 shows the top hierarchy of the interface ontology (*part-of* relationship view). The *dialogue* class models all the dialogue operations, and their effects, that can take place in an interface discourse with a user. The *parts* class defines the characteristics of all the dialogue elements, while the *presentation* class models the appearance characteristics of each dialogue element. Finally, the *knowledge-representation* class defines the knowledge classes that the interface presents to the user. The implementation-independent components of these four classes are depicted in Figure 4.

*The Dialogue Class* . This class models the user *commands* in a knowledge-acquisition tool. The central subclass is the *structure* class. The user–tool dialogue consists of one or more dialogue *structures* that determine the possible ways in which a user can navigate through the dialogue elements (e.g., windows, browsers, text fields) present in a knowledge-acquisition tool. Dialogue

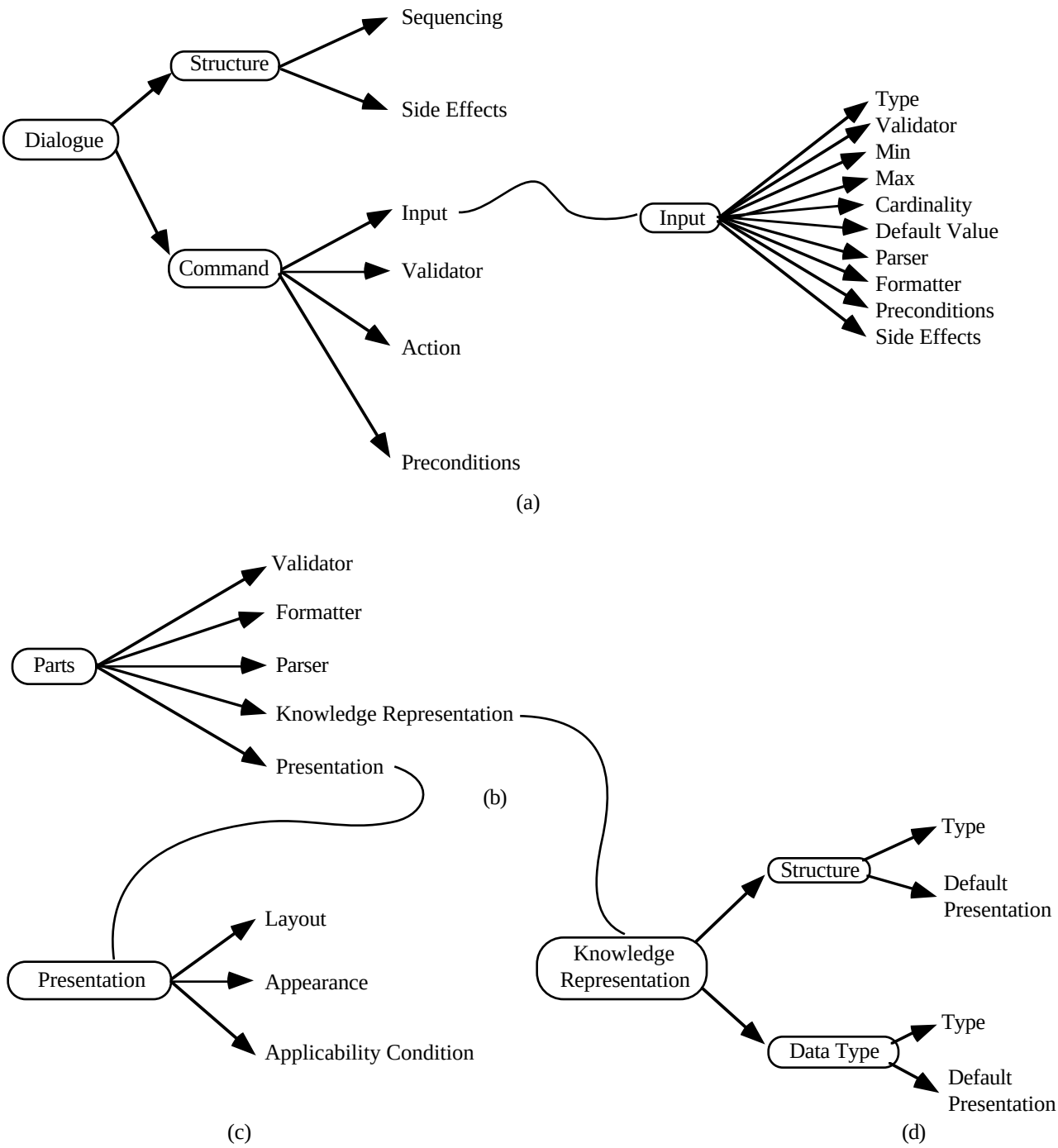


**Figure 4.** Top classes and slots of a shareable interface ontology for knowledge-acquisition tools.

structures include fixed and variable sequences of commands; each command can have *side effects.* For example, entering a numeric value in a field may cause a separate field to be disabled for input. Each command comprises one or more *inputs* and is available to a user according to *preconditions.* Execution of a command implies carrying out one or more *actions*. Throughout the ontology, there are a number of *validators*—objects that declare valid a command, input, or other object if it satisfies a predicate, and that output an error message if it does not.

*The Parts Class*. Each dialogue element (e.g., a menu) in a knowledge-acquisition tool constitutes a *part* of that tool. Any given part allows the visualization of an element of the *knowledge representation* of the tool and has a specific *presentation*. Because a part must communicate with the knowledge base of the underlying application, it must know how to translate between its knowledge representation and that of the knowledge base. Such is the purpose of the *parser* associated with each part. In addition, the knowledge to be displayed has to be molded into the structure required by the dialogue element using a *formatter*.

*The Presentation Class*. The presentation of each interface part is defined through its *layout* and its *appearance*, and it is applicable in a given situation depending on one or more *applicability conditions*. Most of the slots of the layout and appearance objects are implementation dependent. In Mecano, for instance, layout consists of *height* and *width*, and *x* and *y* coordinates while the appearance object has slots such as color, and font.

*The Knowledge Representation Class*. This class is used to define the knowledge types and elements that are visualized and edited in the knowledge-acquisition tool. There are two types of objects under this class: *Data types* and *knowledge structures*. Data types include primitive types such as *integer* and *string*. Knowledge structures are much more complex and can include such objects as *logical expressions* and *frames*. Each data type and knowledge structure has a *default part* (or dialogue element). For example, a Boolean data type may default to a check-box dialogue element. In Mecano, procedural type knowledge defaults to a graphical editor.

## 5. DISCUSSION

The ARPA-sponsored Knowledge Sharing Effort [Neches et al., 1991] has as its main goal the development of knowledge-based systems from reusable components. Key to the ability to share and reuse knowledge from one system to another one is the existence of an ontology common to both systems. For example, the basis for sharing knowledge among two medical diagnosis expert systems may be the presence of a common domain ontology that represents the parts of the medical domain relevant to diagnosis. An important application area for the idea of common ontologies is the domain of user-interface development. Although the design of a user-interface typically consumes over half of the development time of a system, there is very little automated assistance for such a task. Moreover, due to the domain-specificity of user-interfaces, reuse and sharing of designs is almost non-existent. The key reason for such problems is the lack of a declarative model of interfaces, and their design, that developers can manipulate abstractly to build interfaces.

As a consequence, there is considerable interest in developing an interface ontology as part of the Knowledge Sharing Effort. A shareable interface ontology would mean not only that the interfaces in two applications could be built as instantiations of the ontology, but it would also entail that the interface ontology would use the same representation language and level of abstraction as other ontologies used in the application (e.g., domain ontology, problem-solving ontology). The latter fact implies that it would be much easier to automate the development of parts of an interface by inferring, for example, some parameters of a specific interface design directly from the domain ontology.

Apart from the general interest that for knowledge-based systems development the availability of an interface ontology may have, there is an important special case of the use of an interface ontology: that of the design and development of knowledge-acquisition tools. The main purpose of a knowledge-acquisition tool is to allow its user to review, revise, and augment a knowledge base. Thus, there is an implicit interfacing role in a knowledge-acquisition tool, a role that entails establishing communication between two knowledge representations: One for the knowledge base where the knowledge is stored, and another one for the visual display where the knowledge is presented to the user for editing. As a consequence, taking the view of knowledge-acquisition

tools as user-interfaces allows us to have an ontology for knowledge-acquisition tools as a corollary of an interface ontology.

The benefits of defining a knowledge-acquisition ontology are similar to those embodied in the Knowledge Sharing Effort. Building a knowledge-acquisition tool is one of the more resource-consuming facets of knowledge-based system development. An interface ontology allows knowledge engineers declarative access to the design knowledge of a specific interface. This in turn provides a basis for the building of software tools that can assist, and automate, the design process, therefore realizing savings in developer time-and-effort resources. Furthermore, designing knowledge-acquisition tools through an interface ontology facilitates considerably the maintenance tasks due to iteration in the development of the application. In knowledge-based systems development, it is common to make multiple revisions to the data model (or domain ontology, in the case of an ontology-based development framework) that underlies the system. Each such change may necessitate a corresponding change to the knowledge-acquisition tool of that system. Propagation of such changes from knowledge-based system to knowledge-acquisition tool can be greatly facilitated by the presence of a knowledge-acquisition tool ontology. If the ontology is available, then an automatic mapping of the changes is possible from the domain ontology (or data model) of the system to the knowledge-acquisition tool ontology. This is the case in the systems developed with PROTÉGÉ-II and the knowledge-acquisition tools developed with Mecano. The DASH design tool [Eriksson et al., 1993] is able to regenerate a new knowledge-acquisition tool design automatically after any changes to the domain ontology defined with PROTÉGÉ-II for a target knowledge-based system.

The goals of saving developer resources and of facilitating design iteration are the same goals behind the idea of knowledge-acquisition metatools—systems that generate knowledge-acquisition tools from a high-level specification. The shortcoming of metatools has been that in general, they have not made explicit the design knowledge employed in tool generation. Therefore, the usefulness of a metatool is restricted to the environment where it is implemented. An ontology-based framework for knowledge-acquisition tools, however, may implement the actual working tools in a given environment, but it is based on design knowledge that can be shared across implementations.

Given the conviction that a common ontology for knowledge-acquisition tools is useful and necessary, the question remains as to how to achieve it. Our own approach has been bottom-up. Our groups at the Information Sciences Institute and at Stanford University have been working on user-interface development environments that are model based, that is, built around the concept of an interface model that can be manipulated to generate user interfaces. Our systems, SHELTER and Mecano, respectively, have interface models built independently of each other, but that upon examination, share a number of definitions. Therefore, we are proposing that a shareable interface ontology can be built by *aligning* these interface models. Model alignment consists in carrying out a number of comparisons, revisions, redefinitions, and deletions in both models to achieve a single one that is useful in both environments. In this paper, we are presenting the first results of the model alignment between SHELTER and Mecano, that is, the output of merging the models and eliminating syntactic differences. This is a first and necessary precondition to reach a shareable ontology. Clearly, we must now apply the resulting model in other model-based environments to test its usefulness and limitations. There is a strong likelihood that further alignment with yet more interface models will be necessary.

Ontology-based frameworks for knowledge-acquisition tools suffer the same constraints that any other ontology-based system does. The ability to generate knowledge-acquisition tools is always limited by the expressiveness of the ontology and by the ability of the underlying software to implement the ontology definitions and relationships. With SHELTER and Mecano we have demonstrated that ontology-based frameworks for knowledge-acquisition tools are feasible and that these frameworks produce important savings in time-and-effort resources. At the same time, we

have taken a first step towards a knowledge-acquisition tool ontology that can be shared by other environments and that brings knowledge reuse across implementations one step closer to reality.

## ACKNOWLEDGMENTS

The work of Puerta, Eriksson, and Musen has been supported in part by grants LM05157 and LM05305 from the National Library of Medicine, and by a gift from Digital Equipment Corporation and by NSF Award IRI-9257578.

The work of Neches, Szekely, and Luo has been supported in part by contracts from the Advanced Research Projects Agency of the Department of Defense.

We thank John Egar for his insightful comments and discussions, which have helped us to develop the concept and design of Mecano. We thank John Gennari for his work in developing Maître.

## REFERENCES

Eriksson, H., Puerta, A.R., and Musen, M.A. (1993a). *Generation of Knowledge-Acquisition Tools from Domain Ontologies. In B.R. Gaines (Ed.), Proceedings of the Annual Workshop on Knowledge Acquisition,* Banff, Canada, February 1994.

Eriksson, H. (1993b). Specification and generation of custom-tailored knowledge-acquisition tools. In *Proceedings of IJCAI'93*, August 1993, Chambéry, France, pp. 510–515.

Gennari, J.H. (1993). *A Brief Guide to MAÎTRE and MODEL: An Ontology Editor and a Frame-Based Representation Language*. Knowledge Systems Laboratory Report KSL-93-46, Stanford University, Stanford, California, June 1993.

Musen, M.A. (1989). *Automated Generation of Model-Based Knowledge-Acquisition Tools*. London: Pitman.

Neches, R. (1992). Cognitive Issues in the SHELTER Knowledge Base Development Environment. *In Proceedings of the AAAI Spring Symposium on Cognitive Issues in Knowledge Acquisition*, Stanford, CA, March, 1992.

Neches, R. (1991) Acquisition of Knowledge for Sharing and Reuse. *In B.R. Gaines (Ed.), Proceedings of the Annual Workshop on Knowledge Acquisition*, Banff, Canada, October 6-11, 1991.

Neches, R. Foley, J.D., Szekely, P., Sukaviriya, P., Luo, P., Kovacevic, S., Hudson, S. (1993) Knowledgeable Development Environments Using Shared Design Models. *In Proceedings of the ACM/AAAI International Workshop on Intelligent User Interfaces*, Orlando, FL, January, 1993.

Neches, R., Fikes, R., Finin, T., Gruber, T., Patil R., Senator, T., and Swartout, W.R. (1991). Enabling technology for knowledge sharing. *AI Magazine*, **12**(3), Fall 1991, pp. 36–56.

Puerta, A.R., Egar, J.W., Tu, S.W., and Musen, M.A. (1992). A multiple-method knowledge-acquisition shell for the automatic generation of knowledge-acquisition tools. *Knowledge Acquisition*, **4**(2), pp. 171–196.

Puerta, A.R., Tu, S.W., and Musen, M.A. (1993). Modeling tasks with mechanisms. *International Journal of Intelligent Systems*, **8**(1), pp. 129–152.

Szekely, P., Luo, P., and Neches, R. (1992). Facilitating the exploration of interface design alternatives: The HUMANOID model of interface design. In *Proceedings of CHI'92*, May 1992, Monterey, California, pp. 507–515.