The Study of Models of Intelligent Interfaces

Angel R. Puerta

Medical Computer Science Group
Knowledge Systems Laboratory
Stanford University
Stanford, CA, 94305-5479, USA
(415) 723-6979
puerta@camis.stanford.edu

ABSTRACT

Researchers in the field of intelligent interfaces have concentrated on building architectures, and have placed little emphasis on defining appropriate models. As a result, this research area is not well defined. L-CID is a model of an intelligent interface that establishes the knowledge requirements for, determines the functionality of, and creates a definition for an intelligent interface. In addition to modeling knowledge-based user interaction, L-CID allows two important functions commonly overlooked in models of intelligent interfaces: (1) self-adaptation, and (2) user-interface management. Examples of implementations of L-CID for each of these functions are presented.

KEYWORDS: Intelligent-interface modeling, self-adaptation, user-interface management, machine learning.

A NEED FOR MODELS

Through the decade of the eighties, and the beginning of the nineties, the areas of artificial intelligence and human–computer interaction have caught the interest of the scientific and business communities. This interest has sparked, among other advancements, the development of expert systems, of advanced graphical interfaces, and of user-interface management systems. Yet, despite all the progress, little has been accomplished in a natural interdisciplinary field: the research and development of intelligent interfaces. Certainly, any field that draws from multiple disciplines is bound to multiply the complexities of each discipline, thereby slowing its own advancement, compared to that of the contributing individual areas, by an order of magnitude. In the case of intelligent interfaces, these complexities include problems such as knowledge representation, proper architectures for knowledge-based systems, and human factors. Nevertheless, there is one important item that has contributed to the slow rate of progress of intelligent interfaces: Researchers have not emphasized studying *models* of intelligent interfaces—a problem that is independent of the multidisciplinary nature of the field.

The development of models of knowledge-based systems is important because proper models establish the system requirements and define the functionality that the system can provide. Models are used extensively in artificial intelligence and in human–computer interaction. For example, the blackboard model [7] has served as a foundation for the construction of numerous knowledge-based systems, and even of expert-system shells [6]. Buchanan's classical model of a learning system [1] is as valid today as it was when it was first proposed. Card's model of human–computer interaction [2] has influenced significantly the development of user interfaces for many years. Furthermore, the features that most of these models share are their simplicity and flexibility, allowing them to be combined to create new models. For instance, Buchanan's model of a learning system can be embodied within the blackboard model.

Unquestionably, there have been attempts at defining the functionality and knowledge requirements of intelligent interfaces. Card's triple-agent model [3] offers a solid base for the implementation of knowledge-based interfaces. It also includes many of the knowledge requirements that Rissland proposed in earlier work [14]. Both of these efforts, however, fail to accommodate complex systems such as self-adaptive interfaces [5]. The bulk of the research to date has been on architectures for intelligent interfaces. Architectures are useful to study specific implementation issues, but they are often completely hardware dependent. It is difficult to apply the results obtained with one architecture to the construction of another architecture that, more often than not, uses a different hardware platform and is intended for a different task or domain.

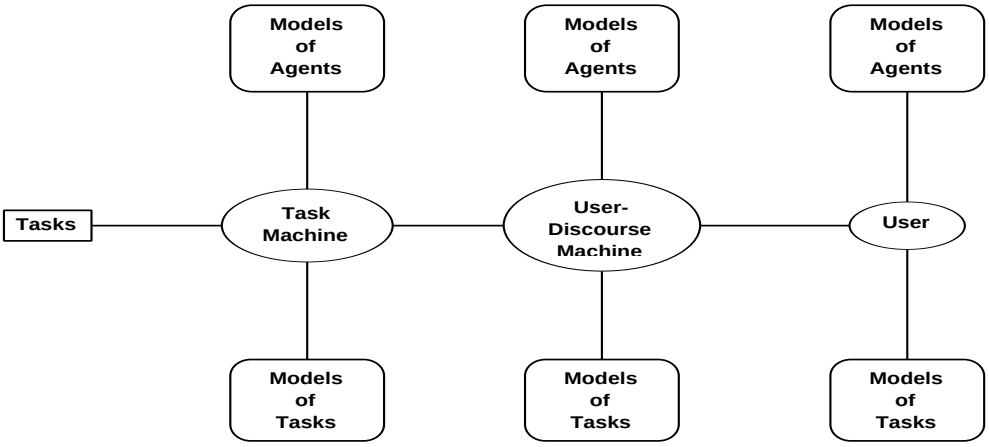

Figure 1. The triple-agent model. Agents—denoted by ovals—cooperate to perform tasks. Each of the agents maintains models of the other agents and of the tasks to be performed.

In this paper, I introduce L-CID—a model of an intelligent interface—and demonstrate its usefulness with examples of its implementation. L-CID establishes the knowledge requirements of an intelligent interface, defines the functionality of such interfaces, and creates a definition of an intelligent interface. L-CID draws from other models of user interfaces and of knowledge-based systems, augments these models, and corrects some of their inadequacies in the context of intelligent interfaces. In particular, L-CID avoids limiting itself to the definition of knowledge requirements and functionality for the intelligent interaction with users. Although clearly essential, *knowledge-based interaction* is only one of three major functions that an intelligent interface can perform. The other two functions are *self-adaptation* and *automatic generation.* In self-adaptation, the interface can improve its performance over time by observing its actions, localizing faults in its knowledge base, and modifying that knowledge base accordingly. In automatic generation, the interface uses its existing knowledge base to produce (generate) an interface that is particular to the current state of the knowledge base, thus acting as an intelligent user-interface management system.

The rest of this paper is organized as follows. First, I define the proposed model and explain how that model was derived. Next, two examples of instantiations of the L-CID model are presented: (1) an example of the use of the model to build a self-adaptive system, and to test a learning technique for modality selection; (2) an example of automatic generation of interfaces that demonstrates how knowledge-acquisition tools can be derived from task and domain knowledge. To conclude, I argue the benefits of the study of models by intelligent-interface researchers.

THE L-CID MODEL

Figure 1 depicts Card triple-agent model [3] of an intelligent system. The three agents identified are the *user,* the *user-discourse machine,* and the *task machine.* There is a set of tasks that the user can perform with the cooperation of the other two agents. In the performance of a task, the task machine provides the problem-solving knowledge, and the discourse machine conducts any needed dialog with the user. All three agents are capable of creating and maintaining models of the other agents and of the tasks. An intelligent interface in this model, although not represented explicitly, may be part of the task machine and of the discourse machine at the same time.

Using this abstract model, we can identify three primary functions for the intelligent interface: task modeling, user modeling, and translation. *Task modeling* in an interface maintains a satisfactory model of the tasks that captures all the relevant functions and operations. *User modeling* constructs a representation of the user's knowledge of the tasks and the machines. *Translation* converts user intentions into machine actions and, in the opposite direction, machine actions into user identifiable responses. Consequently, these functions define specific knowledge requirements: knowledge of tasks and domains, knowledge of users, and knowledge of interaction modalities. There are, however, additional knowledge requirements and functionality, not distinguished in the triple-agent model, that must be represented in an intelligent-interface model. First, knowledge of the computer, including its available software tools and hardware devices, is needed so that the interface can match tools with users and devices to provide appropriate interaction modes. Second, knowledge of the physical and organizational environment in which the user and machine perform tasks is important. Different types of users place

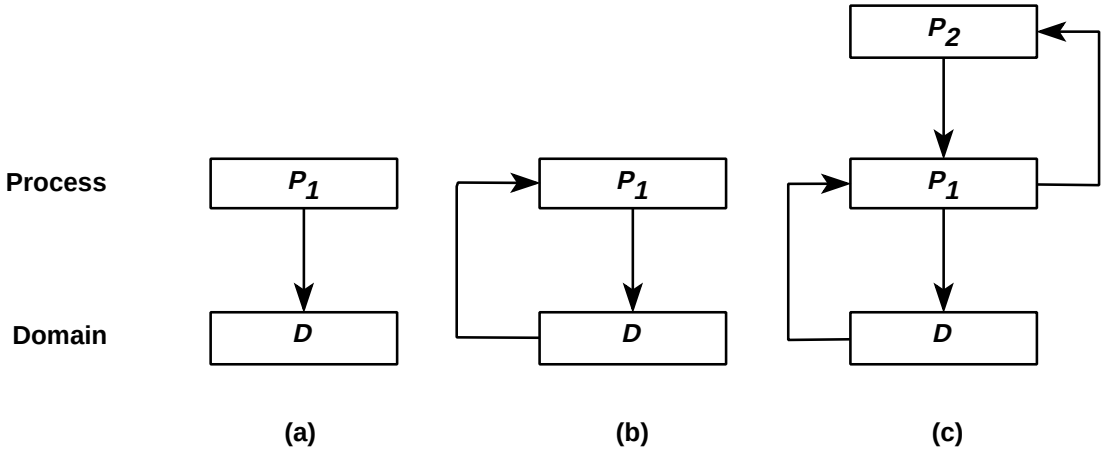

Figure 2. Three types of feedback systems. The system in (a) is nonadaptive because the process does not receive any feedback from the domain. That in (b) is adaptive; P_1 can change how it operates on ***D.*** That in (c) is self-adaptive. P_2 can modify how P_1 operates on ***D***.

different demands on an interface. Furthermore, such demands can vary from environment to environment (e.g., from an office setting to a home setting). Third, the interface must be able to assist the user in performing tasks. Thus, knowledge of when and how to assist, and of how to recognize user intentions and plans, should be disponible.

The knowledge requirements established up to this point address directly the human–computer interaction process. They do not, however, cover an important function that is achievable with a knowledge-based system: self-adaptation. Figure 2 describes self-adaptation from a control perspective [8]. It follows from this figure that, if we denote an intelligent interface as process P_1, then a process P_2, with appropriate feedback from P_1, could improve the performance of P_1 over time. The combination of P_2 and P_1 defines a self-adaptive interface. Knowledge requirements for a self-adaptive intelligent interface are similar to those of a machine-learning system. First, training knowledge is needed to select training instances (in this case, instances of interaction with users), or to generate training instances. Therefore, training knowledge is what defines the learning strategy that the interface follows. Second, evaluation knowledge is required to identify faults in the interface's knowledge base, and to recommend changes to that knowledge base. Finally, implementation knowledge is necessary to translate recommended changes into specific modifications to the interface's knowledge base.

The L-CID model—shown in Figure 3—incorporates the requirements discussed here for a self-adaptive intelligent interface. L-CID is defined within the paradigm of the blackboard model [7] for two main reasons: (1) to represent knowledge requirements explicitly as knowledge sources, and (2) to separate clearly as a *metaprocess* the self-adaptive functionality of the interface by using the multiple-layer capabilities of blackboards. Note that L-CID does not establish necessary conditions for an interface to be qualified as intelligent. Instead, interfaces can be categorized discretely with this model according to how many of the knowledge requirements they fulfill. With L-CID, interfaces can be thought of as exhibiting degrees of intelligence, as opposed to other classifications that may allow only two categories: intelligent and dumb.

A SELF-ADAPTIVE INSTANTIATION OF L-CID

Programming-by-example interfaces, a kind of demonstrational interfaces, are intelligent systems that can infer the user's next action by reasoning about the user's previous actions [4, 9]. Thus, these interfaces can be described as learning-by-example systems that recognize patterns in a sequence of interactions, generalize from these patterns, and establish the next logical step, or sequence of steps, that the user is expected to follow. Their success has been mixed because the learning techniques employed are effective for certain tasks but not for others. This problem is a consequence of presupposing that learning by examples is the most appropriate strategy to learn automatically about user intentions.

It would be advantageous to determine what learning techniques are most effective for given tasks before committing to a full implementation of the intelligent interface. L-CID can be used to build a rapid-prototyping system that allows developers to test learning techniques and to localize deficiencies in the techniques by simulating the interaction process for the task of interest. A full description of this implementation of L-CID has been

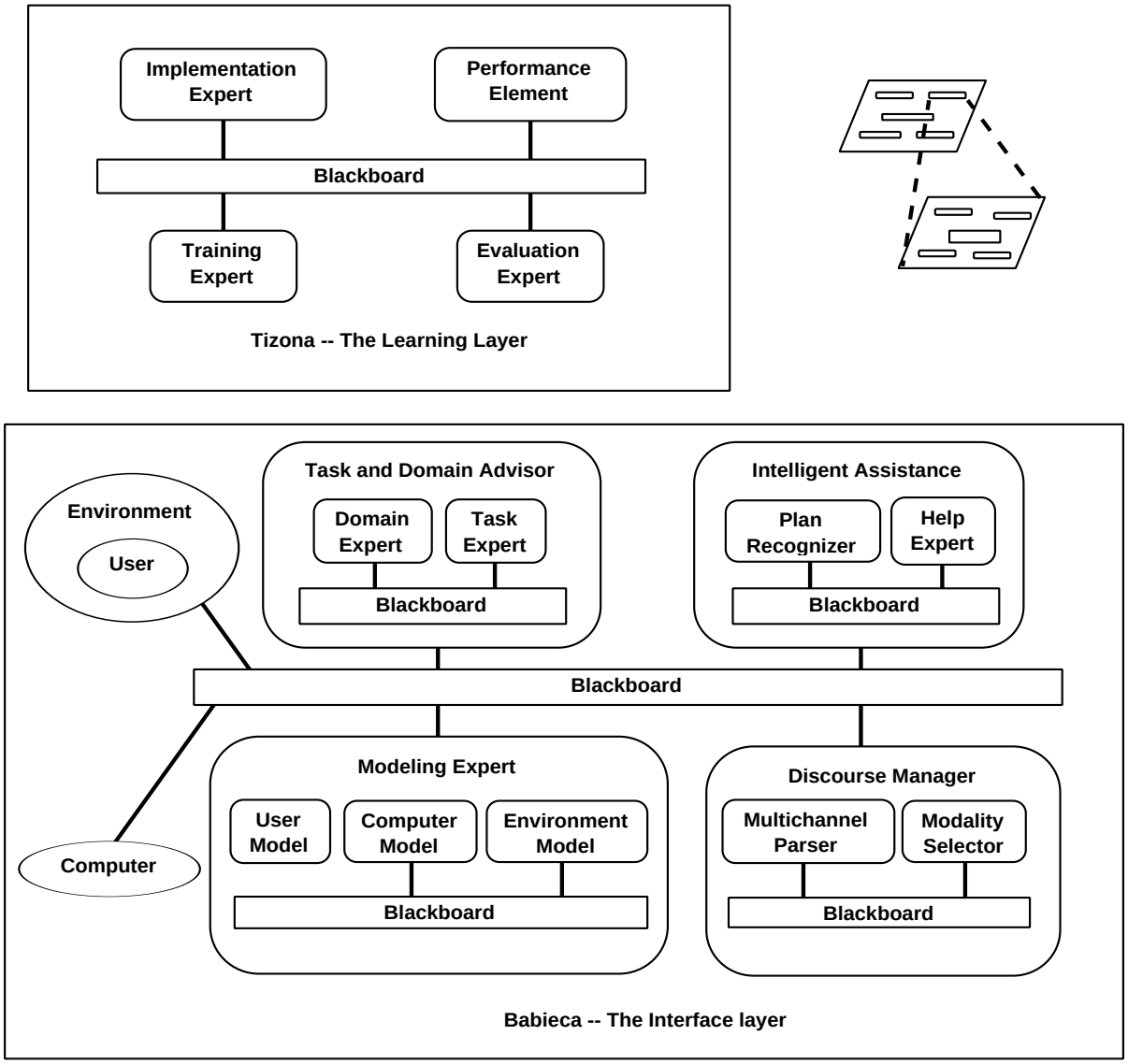

Figure 3. The L-CID model of a self-adaptive intelligent interface. The knowledge requirements are represented explicitly as knowledge sources in a blackboard model. The model is multilayer, with the performance element of the learning layer being the lower, interface layer.

reported elsewhere [11, 12] and is beyond the scope of this paper. Its principles, however, are described here to illustrate how the application of a general model can yield significant benefits to intelligent-interface developers.

Figure 4 shows an abstract view of a blackboard implementation of L-CID that tests a learning technique for modality selection. The knowledge sources—as defined in the blackboard paradigm—for the task, user, computer, and environment models contain facts about the features of each of these elements. The goal of the learning technique is to determine what interaction mode is indicated for each valid combination of feature values for the four models. The mapping of combinations to modes is stored in a separate knowledge source, called *map of modes*, that, in the abstract L-CID model, is included in the *discourse manager* knowledge source. The learning layer of the implementation contains the correct mapping information to simulate acceptance and rejection by the user of the choice of modes made by the interface. Based on whether a choice is accepted or rejected, the learning layer changes the map-of-modes knowledge source in the interface layer. Using this simulation, interface developers can run different tests to establish how sensitive the learning technique is to changes in the feature values, to establish learning curves for the

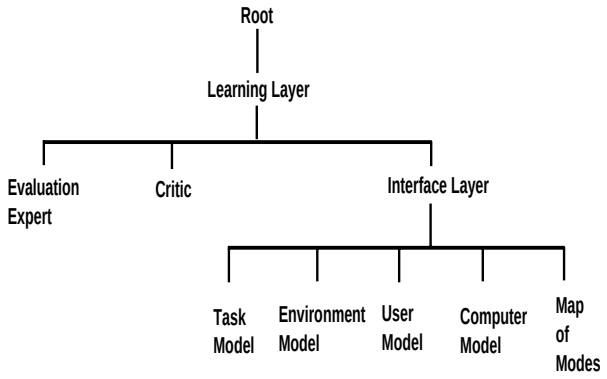

Figure 4. Abstract view of a blackboard implementation of L-CID that tests learning techniques for modality selection. The *model* knowledge sources contain facts about features of users, tasks, environments, and computers. The goal of the learning layer is to map an interaction mode to every possible combination of feature values in the *model* knowledge sources by observing examples of interactions with users.

technique, and to identify the limits of the technique with respect to the maximum number of feature combinations whose mappings can be effectively learned. Developers can then use the test results to modify the technique in order to eliminate its weak areas [11].

By applying a model such as L-CID, we can study what are the strengths and weaknesses of programming-by-example interfaces, what are the tasks for which these interfaces are most useful, and what are the learning strategies that are most appropriate to infer user's actions for tasks that are outside the scope of this type of intelligent interfaces.

USER-INTERFACE MANAGEMENT WITH L-CID

One function that researchers commonly overlook when discussing intelligent interfaces is the automatic generation of interfaces from the current state of the knowledge base. It should be possible to examine knowledge stored in the knowledge base and to derive interface requirements for, say, window layout or dialog sequence, that are not defined explicitly in the knowledge base. L-CID allows for this type of inference. One way to achieve such functionality is to assume a learning-by-deduction strategy in the learning layer of L-CID. Unlike the learning-by-examples strategy of the previous example, learning-by-deduction does not observe the interactions with the user. Instead, the strategy consists of examining the current state of the knowledge sources of the interface layer, and augmenting them by deducing additional facts and knowledge from the present state. In this manner, the learning layer can deduce, for example, the unknown parameters of a window layout from the existing knowledge about the user.

To illustrate this feature of L-CID, we shall describe an example of the generation of a graphical interface from domain knowledge. In our case, the domain is that of clinical trials for AIDS patients. In these trials, physicians administer new drugs to patients according to a treatment plan specified by documents called *protocols.* The documents are written by committees of physicians who are experts in the field of AIDS therapy. Normally, the procedural part of a protocol—the sequence of actions that the administering physicians must follow—is drawn as a flowchart. The symbols allowed in protocol flowcharts are those of relevance to the particular area of treatment. For AIDS therapy, these symbols include terms such as *medication, regimen* (the administration of one or more medications), and *laboratory test.*

Domain knowledge can be represented as a structured collection of terms, and of their interrelationships; this collection is called an *ontology.* Figure 5 shows the domain ontology for our example. If we assume that the ontology of Figure 5 is stored in the interface layer of L-CID, we can infer from such domain knowledge several characteristics of an interface for constructing protocols, including the composition and behavior of the interface. The terms defined by the ontology prescribe the legal

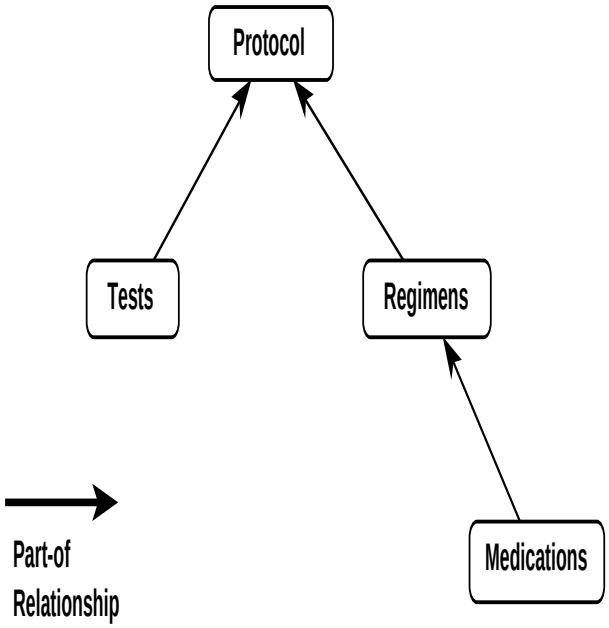

Figure 5. A domain ontology for AIDS therapy. The ontology structures the terms of relevance and describes the relationships among terms.

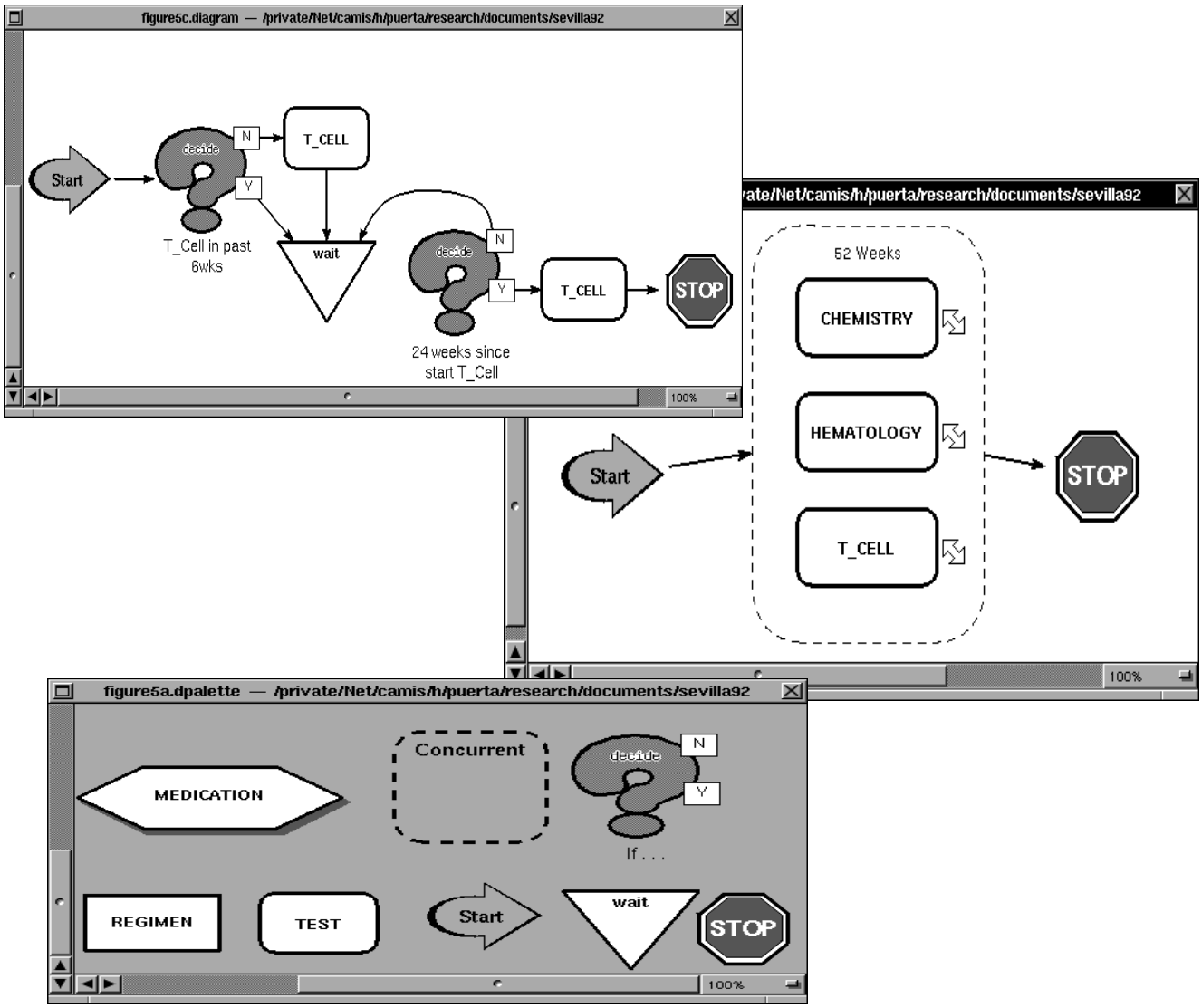

Figure 6. **A**n interface that allows an expert physician to draw AIDS protocols as flowcharts. Physicians are presented with a palette of graphical elements (lower window) derived from the domain ontology (with the addition of control primitives). The upper window is a subwindow that expands the *test* box (for T-Cell counts) of the middle window.

symbols for the flowcharts of protocols. The part-of relationships constrain the possible combinations of symbols that can be used in the flowchart. For example, in this context, a laboratory test cannot include the administration of a medication.

Figure 6 shows an interface for the drawing of protocols in the AIDS-therapy domain. The symbols declared in the domain ontology are represented as elements of a palette for a graphical editor. The constraints derived from the ontology result in connectivity limitations of the palette elements. For instance, note that the upper window in the figure expands the *test* box (T-Cell) of the main protocol window to detail the latter's internal procedure. As inferred from the domain ontology in our example, dragging a *medication* box from the palette onto the drawing area of the upper window is an illegal operation in this interface.

Interfaces like the one shown in Figure 6 are generated within PROTÉGÉ-II [10, 13]—a knowledge-acquisition shell that produces automatically knowledge-acquisition tools that domain experts can employ to edit an expert system's knowledge base. Although, in this case, we showed only how a generic graphical editor can be custom-tailored to a given domain, other types of knowledge can be used to make interface-design decisions. For example, knowledge about interaction modalities, and the fact that protocols are procedural in nature, can determine that the most appropriate interaction

style for the interface to be generated is a graphical editor.

DISCUSSION

Defining what an intelligent interface is remains a difficult problem. Categorizing an interface as intelligent or dumb seems to be just as hard. In any given case, every researcher appears to have a different answer. Is an interface intelligent because it interacts with a knowledge-based system, even though it does not use knowledge to do so? Or is knowledge use *sine qua non?* What if knowledge was used to create the interface, but not to run it? Other branches of artificial intelligence present less of a challenge when it comes to definition and classification of their products. Machine-learning systems, tutoring systems, and expert systems, are examples of systems that are normally less difficult to categorize. This peculiarity of intelligent interfaces is not the result of intrinsic complexity. Rather, it is the consequence of the emphasis of researchers in studying the *use* of intelligent interfaces, as opposed to studying their *definition.* When use of a system is researched, the end product is a series of architectures. But, when definition of a system is the focus of the research, the end product is a series of models.

There are three key benefits that a model of an intelligent interface offers: (1) it establishes the knowledge requirements of the interface, (2) it prescribes the functionality provided by the system, and (3) it defines the concept of an intelligent interface. The L-CID model presented in this paper identifies the knowledge requirements for an intelligent interface and explicitly represents them as knowledge sources within a blackboard model. In addition, it helps researchers and developers to visualize how functions such as machine learning, self-adaptation, and user-interface management can be performed. Furthermore, L-CID forms a basis for categorizing interfaces as intelligent, to a discrete degree, according to how many of the knowledge requirements are fulfilled. On the other hand, architectures are useful to study specific implementation issues. Problems such as the appropriateness of knowledge representations, effectiveness of learning techniques, or efficiency of control strategies can be solved at only the architecture level. But unless architectures are abstracted into general models, they cannot provide any of the benefits that I have enumerated. In the ideal situation, however, research is conducted along both lines—that is, models and architectures are used synergistically. For example, when architectures are derived from models, the difficulties encountered implementing the architecture can help researchers pinpoint faults in the conceptual model. Conversely, when problems arise with an architecture, comparing it with general models of the system may identify shortcomings in the design of the architecture.

By giving more attention to models, researchers of intelligent interfaces should be able to understand and define more precisely the conceptual issues that affect the development of these systems. L-CID serves as a basis for the study of those issues, as well as for the development of new architectures. By coordinating the definition of models and the implementation of architectures, researchers can finally provide the impulse that the field of intelligent interfaces needs to become an important area for both artificial intelligence and human–computer interaction.

ACKNOWLEDGMENTS

I would like to thank Ronald Bonnell, John Egar, Henrik Eriksson, and Mark Musen for their insightful comments and discussions that have helped build L-CID and its implementations. I am also grateful to Lyn Dupré for her editing of a previous version of this paper

REFERENCES

1. Buchanan, B., Mitchell, T.M., Smith, R.G., and Johnson, C.R. Models of learning systems. In *Encyclopedia of Computer Science and Technology,* Belzer, J., Holzman, A.G., and Kent, A., editors. Marcel Dekker, New York, 1977, Volume 11, pp. 24–51.

2. Card, S.K., Moran, T.P., and Newell, A. *The Psychology of Human–Computer Interaction.* LEA Associates, Hillsdale, New Jersey, 1983.

3. Card, S.K. Human factors and artificial intelligence. In *Intelligent Interfaces Theory, Research, and Design,* Hancock, P.A. and Chignell, M.H., editors. North Holland, New York, 1989, pp. 27–41.

4. Cypher, A. EAGER: Programming repetitive tasks by example. In *Proceedings of CHI'91,* Robertson, S.P., Olson, G.M., and Olson, J.S., editors, pp. 33–40. New Orleans, Louisiana, May 1991.

5. Edmonds, E.A. The man–computer interface: a note on concepts and design. *International Journal of Man–Machine Studies,* **16**, (1982), 231–236.

6. Hayes-Roth, B. A blackboard architecture for control. *Artificial Intelligence,* **26**, (1985), 251–321.

7. Nii, H.P. Blackboard systems: The blackboard model of problem solving and the evolution of blackboard architectures. *AI Magazine,* **7**(2), (1986), 38–53.

8. Martin, F.A. Control models in computer assisted learning. *Expert Systems,* **5**, (1988), 316–326.

9. Maulsby, D.L., Witten, I.H., Kittlitz, K.A., and Franceschin, V.G. Inferring graphical procedures: The compleat metamouse. *Human–Computer Interaction,*

7, (1992), 47–90.

10. Musen, M.A. A*utomated Generation of Model-Based Knowledge-Acquisition Tools.* London: Pitman, London, England, 1989.

11. Puerta, A.R. L-CID*: A Blackboard Framework to Experiment with Self-Adaptation in Intelligent Interfaces*, Doctoral Dissertation, USCMI Report Number 90–ESL–6, Center for Machine Intelligence, University of South Carolina, Columbia, South Carolina, July 1990.

12. Puerta, A.R. Rapid prototyping of self-adaptive interfaces with the L-CID model. In *Proceedings of the Intelligent Multimedia Interfaces Workshop*, AAAI-91, pp. 37–46. Anaheim, California, July 1991.

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