The Mecano Project: Enabling User-Task Automation During Interface Development

Angel R. Puerta

Knowledge Systems Laboratory
Section on Medical Informatics
MSOB x215
Stanford, CA 94305-5479
puerta@camis.stanford.edu - http://camis.stanford.edu/people/puerta

Abstract

We propose that automation of user tasks can only be properly addressed as a particular problem of adaptation in human-computer interaction. We further claim that to achieve such automation, the ability to define, edit, and infer from models of user interfaces is needed. We introduce The Mecano Project, a model-based interface development environment that allows precisely such capabilities. Finally, we describe how some traditional approaches to user-task automation, including machine learning techniques and programming by demonstration, can be incorporated into Mecano and have a higher potential of being successful.

Introduction

The issue of user-task automation has been of importance to several artificial intelligence subfields for a number of years. A variety of approaches have been attempted with a relative degree of success. These approaches include, among others, machine learning techniques, demonstrational interfaces, and automated generation of knowledge acquisition tools (Cypher, 1991; Puerta, 1990)

Notwithstanding the potential usefulness shown by many of the techniques mentioned, there is ample evidence of serious limitations with each approach. Such limitations are usually caused by a lack of knowledge about the user task that must be automated, and also about the meaning of the user actions from which inferences must be drawn. Thus, many machine learning algorithms make generalizations that, although justified by the applied learning technique, fail to be effective within the context of the user's task. Furthermore, demonstrational interfaces tend occasionally to automate user tasks for which the user does not want automation, or reach generalizations in such a way that the automation becomes a burden on the user instead of a supporting process. Additionally, in the case of knowledge-acquisition tool generators, the knowledge about the user-task is so incomplete, or even non-existent, that generation can only be accomplished for simple, well-structured tasks that allow very little flexibility to the tool users.

The limitations of current approaches arise from a common source: the lack of, or disregard for, foundation knowledge about the interaction process from which inferences must be made and for which tools are being generated. For example, machine learning techniques may employ sophisticated algorithms to infer generalizations of an user's actions, yet not use much knowledge about what are the goals of those actions, the application domain, or the interaction dialog dictated by the user interface. A similar argument can be made against knowledge-acquisition tool generators that attempt to automate tasks but employ no task-specific knowledge during the generation process (Puerta, 1990).

In this paper, I make two fundamental claims regarding the automation of user tasks in a computer system:

> (1) *Automation* of user tasks really means *adaptation* of the computer system to the user needs. Adaptation can take place at the level of the user task itself, as is the case in programming by demonstration interfaces, or it can take place at the level of the computer system. In the latter case, it is called self-adaptation and it affects the way the system reacts to its own success or failure in adapting to a user. An example of this situation is a machine-learning technique that improves its performance over in recognizing sequences of user actions and automating their execution.
>
> (2) Adaptation of the system to the user is by definition a human-computer interaction problem.

Therefore, any attempts at automating user tasks must take place within the context of this general problem. Knowledge about the characteristics of the interface of a computer system is essential to any approach towards automation. Disregard for such knowledge will result inevitably in techniques which either fail or face serious limitations.

Acceptance of the above principles brings up immediately the question of how do we acquire, manage, and process the interface knowledge that is required for adaptation. In any human-computer interaction situation, we will need explicit knowledge about the user task, the application domain, the user, and the interface dialog and presentation. This knowledge is likely to be available only if we design interfaces based on such knowledge sources by using sophisticated development environments that allow editing of the required knowledge and that provide automated generation facilities. This is the premise of an emerging technology called *model-based interface development*. In this paper, I propose that by using this technology we can produce interfaces where sufficient knowledge about the interaction process is available, thereby providing the *foundation* for any techniques that attempt to automate user tasks. The remaining sections of the paper describe the technology, our own perspective at implementing it with a development environment called Mecano, and a proposal describing how to use techniques, such as demonstrational interfaces, within the context of Mecano-produced interfaces.

The Model-Based Approach to Interface Development

The paradigm of model-based interface development has attracted a high degree of interest in the last few years due to its high potential for producing integrated user interface development environments with support for all phases of interface design and implementation. This type of environments are not available commercially .

The basic premise of model-based technology is that interface development can be fully supported by a generic, declarative model of all characteristics of a user interface, such as its presentation, dialog, and associated domain, user, and user task characteristics. As depicted in Figure 1, with such model at hand, suites of tools that support editing and automated manipulation of the model can be built so that comprehensive support of design and implementation is possible. Typically, users of model-based environments refine the given generic model into an application-specific interface model using the tools available within the environment. A runtime system then executes the refined model as a running interface.

The benefits of model-based development are manifold. By centralizing interface information, model-based systems offer support within a single environment for high-level design as well as for low-level implementation details. Global changes, design visualization, prototyping, consistency of resulting interfaces, and software engineering principles in general are much improved over currently available tools, such as interface builders, which offer only partial and localized development support. Over the past few years, several model-based systems (Foley at al., 1991; Johnson, Wilson and Johnson, 1994; Puerta et al., 1994; Szekely, Luo and Neches, 1993; Vanderdonckt and Bodart, 1993; Wiecha et al., 1990) have demonstrated the feasibility of the model-based approach.

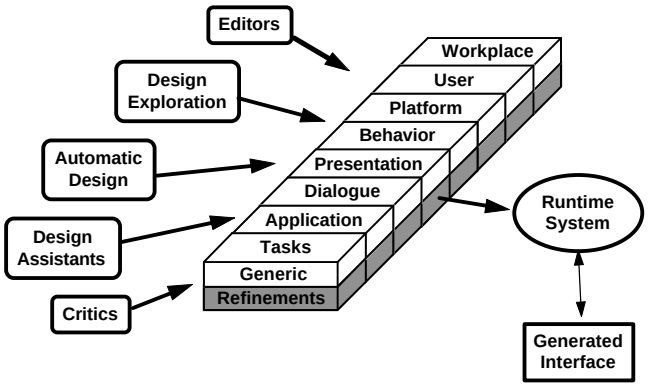

Figure 1. The model-based paradigm. Design tools operate on a generic interface model to produce an application-specific refined model that is then executed by a runtime system.

Despite all the potential shown, model-based technology is struggling to find its way out of the laboratories. This is due mainly to the absence of one of the key elements needed by the technology to truly prosper. There are two central ingredients for success in model-based systems: (1) A declarative, complete, and versatile interface model that can express a wide variety of interface designs, and (2) a sufficiently ample supply of interface *primitives*, elements such as push-buttons, windows, dialog boxes, and similar that a model-based system can treat as black boxes. The need for the first ingredient is clear: without a vocabulary rich enough to express most interface designs, the technology is useless. The second ingredient is also critical because model-based approaches fail if developers are required to model too low-level details of interface

elements—a problem painfully demonstrated by the erroneous modeling abstraction levels of some model-based systems, especially early ones.

Whereas there is little question that good sets of interface primitives are available in most platforms, researchers have fallen short of producing effective interface models. The problems with current interface models can be summarized as follows:

- *Partial models*. Models constructed up-to-date deal only with a portion of the spectrum of interface characteristics. Thus, there are interface models that emphasize user tasks (Johnson, Wilson and Johnson, 1994), target domains (Puerta et al., 1994), presentation guidelines (Vanderdonckt and Bodart, 1993), or application features (Szekely, Luo and Neches, 1993). These models generally fail when an interface design puts demands on the model beyond the respective emphasis areas.
- *Insufficient underlying model*. Several model-based systems use modeling paradigms proven successful in other application areas, but that come up short for interface development. The Entity-Relationship model, highly effective in data modeling, has been applied with limited success in interface modeling (Foley et al., 1991; Vanderdonckt and Bodart, 1993). These underlying models typically result in partial interface models of restricted expressiveness.
- *System-dependent models*. Many interface models are non-declarative and are embedded implicitly into their associated model-based system, sometimes at the code level. The models are tied to the interface generation schema of their system, and are therefore unusable in any other environment.
- *Inflexible models*. Experience with model-based systems suggests that interface developers many times wish to change, modify, or expand the interface model associated with a particular model-based environment. However, model-based systems do not offer facilities for such modifications, nor the interface models in question are defined in a way that modifications can be easily accomplished.
- *Private models*. Interested developers or researchers wishing to obtain a generic interface model from one of the currently available model-based systems, quickly find that there is no version of an interface model that is publicly available, or even obtainable via a licensing agreement. The inability to produce an interface model fit for distribution to third parties is one of the major shortcomings of model-based technology.

The Mecano Project

To address the limitations outlined above, we started at the end of 1994 *The Mecano Project*. This project draws from our own experience building Mecano (Puerta et al., 1994)—a model-based system where interface generation is driven by a model of an application domain—and from our examination of several model-based systems built in the past few years. The project encompasses two phases: (1) Development of a comprehensive interface model, and (2) Implementation of a model-based environment based on the model obtained in (1).

Phase one: The interface model.

In this phase, we define a generic interface model with a high degree of completeness, portability, and independence from a corresponding model-based system. The interface model is available as a resource to the HCI community.

The requirements of completeness, flexibility, and system independence of an interface model are very difficult to achieve within a monolithic structure for interface modeling, as is the case with current model-based systems. Even the most elaborate interface model will run into difficulties if changes or extensions are needed. Furthermore, the idea that a single generic interface model that can express most interfaces can be defined is debatable at best, and certainly contrary to experience gathered with the use of model-based systems.

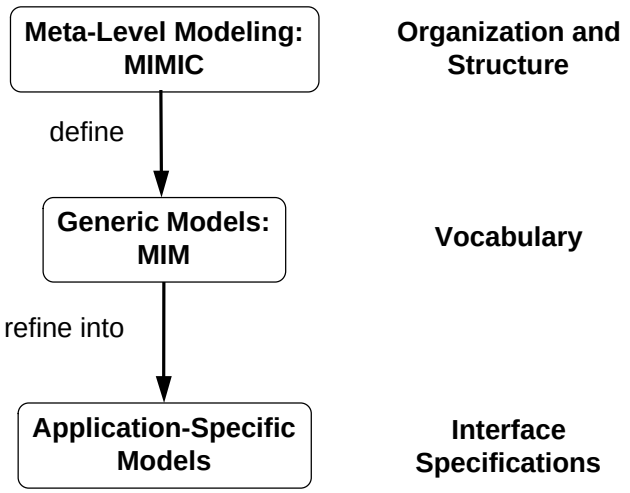

Figure 2. A multilevel approach to interface modeling.

The key reasons why interface models lack flexibility are first that they were never designed with the intention of being changed once implemented, and second, but perhaps more importantly, that they lack a description of the organization and structure of the model. Without such description, it is difficult to understand the role played in an interface design by the different interface elements being modeled, and it is also hard to visualize the relationships among those elements. As a consequence, tools cannot be built to support the model expansion process, and manual changes are coding exercises usually only accessible to the original designers of the interface model.

In The Mecano Project, we overcome the various limitations of current interface models by means of a modeling approach at multiple levels of abstraction, as shown in Figure 2. The result is an interface modeling language, called MIMIC, that can be used to express both generic and application-specific interface models. We also provide one generic model called the Mecano Interface Model, or MIM. The MIMIC language follows the following principles:

- *Explicit representation of organization and structure of interface models*. MIMIC provides a metalevel for modeling that assigns specific roles to each interface element, and that provides the constructs to relate interface elements among themselves. There is no fixed way to relate elements, so developers are free to build their own schema (e.g., building a Petri Net of dialog elements).
- *No single generic model*. We have discarded the idea that a single, all-encompassing generic interface model can be built successfully as previously assumed. Instead, MIMIC supports the definition of generic interface *models*. We provide one such generic model in MIM and our model-based system will support that generic model. However, we envision that developers, and the HCI community in general, will produce a number of such generic models, or extensions of generic models, that are suited for specific user tasks, application domains, or given platforms.
- *Explicit interface design representation*. Interface models written with MIMIC will define not only interface elements, but also characteristics of the design process for the modeled interface. This is a feature lacking in all previous schema for interface modeling, but it is a crucial one if we are to give developers access to and control of the automated processes of interface generation in model-based systems.

Phase two: The Mecano model-based environment.

In this phase, we implement a model-based environment, called Mobi-D (Model-Based Interface Designer) that supports interface generation based on the phase-one interface model. The main components of Mobi-D can be seen in Figure 3. The system has three main features:

- *User-centered interface development in an integrated and comprehensive environment*. Developers build interfaces manipulating abstract objects such as user tasks and domain objects. The production of presentation styles and dialogs is automated in most part by the environment.
- *Transparent modeling language*. Developers do not need to know the MIMIC modeling language. The environment tools provide the functionality to achieve editing operations without directly manipulating language structures.
- *Open architecture*. Third-party developers can enhance the environment by incorporating their own design tools. Such tools need only to adhere to the MIMIC language. This feature is key in supporting machine-learning and other techniques for user-task automation.

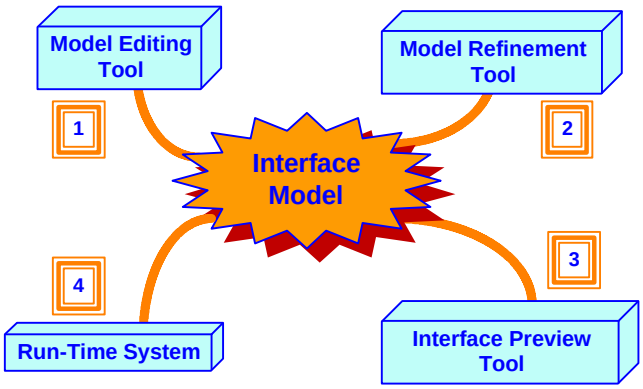

Figure 3. The Mobi-D development environment.

Discussion

The model-based approach and the Mecano environment are to be seen as *enabling technologies* for adaptation and automation of user tasks. The Mecano framework facilitates *design-time* and *runtime* capabilities for automation that can be utilized by traditional artificial-intelligence approaches to user-task automation.

Any tool or technique that targets user automation can take advantage of the open nature of the Mobi-D architecture. Thus, for example, a programming-by-

demonstration tool can be incorporated into Mobi-D as a runtime tool that operates on the generated interface. Other tools and techniques can be used at appropriate times in the development process. The only requirement is that these tools be able to read and write the MIMIC language

The main advantage of Mecano, however, is not in the open nature of its architecture. The key gain is that any inferences made by any tools associated with Mecano are done within the context of an interface model. Therefore, Mecano tools can directly relate their reasoning to the user tasks, goals, and preferences that drive the interaction process. In general, the inability to do so has been a shortcoming of many AI approaches to user automation.

User-task automation requires the coexistence of useful reasoning techniques and comprehensive knowledge bases of the interface and its users. Mecano provides the infrastructure to develop such knowledge bases and to incorporate tools based on appropriate reasoning techniques for user-task automation.

References

Cypher, A. Programming Repetitive Tasks by Example, , in Proceedings of CHI'91 Conference on Human Factors on Computing Systems, pp. 33-39.

Foley, J., Kim, W., Kovacevic, S. and Murray, K., UIDE - An Intelligent User Interface Design Environment, in J. Sullivan and S. Tyler (eds.), *Architectures for Intelligent User Interfaces: Elements and Prototypes*, Addison-Wesley, 1991, pp. 339-384.

Johnson P., Wilson, S. and Johnson, H. Scenarios, Task Analysis And The Adept Design Environment, in J. Carroll (ed.), *Scenario Based Design*, Addison Wesley, 1994.

Puerta, A., Eriksson, H., Gennari, J.H., and Musen, M.A. Model-Based Automated Generation of User Interfaces, in Proceedings of AAAI'94 (Seattle, July 31 to August 4 1994), AAAI Press, pp. 471-477.

Puerta, A. L-CID: A Blackboard Framework to Experiment with Self-Adaptation in Intelligent Interfaces, Ph.D. Dissertation, University of South Carolina, July 1990.

Szekely, P., Luo, P., and Neches, R. Beyond Interface Builders: Model-Based Interface Tools, in Proceedings of INTERCHI'93 (Amsterdam, Netherlands. April, 1993), ACM Press, pp. 383-390.

Vanderdonckt, J. M., Bodart, F. Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection, in Proceedings of INTERCHI'93 (Amsterdam, Netherlands. April, 1993), ACM Press, pp. 424-429.

Wiecha, W. Bennett, S. Boies, J. Gould and S. Greene. ITS: A Tool For Rapidly Developing Interactive Applications. *ACM Transactions on Information Systems* 8(3), July 1990. pp. 204-236.