Automating Interface Evaluation

Michael D. Byrne[1], *Scott D. Wood*[2], *Piyawadee "Noi" Sukaviriya*[1], *James D. Foley*[1], *David E. Kieras*[2]

[1]Graphics, Visualization, and Usability Center
Georgia Institute of Technology
Atlanta, GA 30332
E-mail: {byrne, noi, foley}@cc.gatech.edu

[2]Electrical Engineering and Computer Science Department
University of Michigan
Ann Arbor, MI 48109
E-mail: {swood, kieras}@engin.umich.edu

ABSTRACT

One method for user interface analysis that has proven successful is formal interface analysis, such as GOMS-based analysis. Such methods are often criticized for being difficult to learn, or at the very least an additional burden for the system designer. However, if the process of constructing and using formal models could be automated as part of the interface design environment, such models could be of even greater value. This paper describes an early version of such a system, called USAGE (the UIDE System for semi-Automated GOMS Evaluation). Given the application model necessary to drive the UIDE system, USAGE generates an NGOMSL model of the interface which can be "run" on a typical set of user tasks and provide execution and learning time estimates.

KEYWORDS: GOMS, Usability, User Interface Design Environment, Interface Evaluation, Formal Models of the User, UIMS

INTRODUCTION

Often, it appears as if the field of HCI is divided into two camps, the technologists and the psychologists. Despite efforts at interdisciplinary partnership, most real interfaces are fundamentally technology-driven. One reason for this, perhaps, is that simply developing working code is difficult enough without having to worry about the most recent findings in the psychology of human-computer interaction. One possible avenue to pursue in solution to this problem would be to bring psychological models into the developer's technical arsenal. That is, build formal cognitive analysis techniques into the application development environment.

This paper describes a first attempt to do just that with a system called USAGE, which is short for the **U**IDE **S**ystem for semi-**A**utomated **G**OMS **E**valuation. USAGE is built out of a number of components, each of which will be described. An example of how USAGE might be utilized will be presented, as well as a discussion of current problems and possible future directions.

UIDE

UIDE (the User Interface Design Environment) is a model-based interface design tool which was developed in C++ and currently runs under SunOS. It is described in more detail in [6]. The aspects of UIDE that are most directly relevant are the model-based nature of the system and the planning engine.

In UIDE, an application is constructed by specifying a model of the interface and then "plugging in" the application-specific code. The interface model consists of a number of pieces, most importantly application actions, interface actions, and interaction techniques. Application actions are user actions at the user task-relevant level, such as "create a gate" in a digital circuit design application. Application actions are performed by executing one or more interface actions, which consist of interface-level constructs, such as "select a command from a pull-down menu." Interface actions are, in turn, composed of interaction techniques, which specify the actual physical actions, such as "click-left-mouse-button."

While UIDE requires that each application action be linked

to one or more interface actions at the level below it, no particular order is specified by this linkage. However, UIDE also contains a simple planning algorithm that can determine the "correct" sequence of interface actions to be performed to execute an application action.

```
APPLICATION: CircuitDesign

Application Action: CreateANDGate {location}
{gate}

1. SelectCommandIcon {CreateAND}
     1) ClickObject_LeftButton
2. SelectPosition-DM {canvas} {pos:location}
     1) ClickPosition_LeftButton
3. InvokeAction


Application Action: MoveGate {object}
{new_position}

1. DragObject-DM {graphical_obj}
{app_obj:object} {new_position:new_position}
     1) DragToPos_LeftButton
2. InvokeAction


Application Action: DeleteGate {object}

1. SelectCommandFromPulldownMenu {Gate}
{Delete}
     1) DragToObject_LeftButton
2. SelectObject-DM {graphical_obj}
{app_obj:object}
     1) ClickObject_LeftButton
3. InvokeAction
```

Figure 1. Example of UIDE output

As part of USAGE, UIDE can generate an ASCII description of an application in terms of the hierarchical decomposition of each application action; that is, a listing of all the interface actions and interaction techniques in sequence as necessary to perform each application action. An excerpt from such a file appears in Figure 1.

GOMS

GOMS (goals, operators, methods, and selection rules) was introduced in the early 1980s as a technique for interface analysis by Card et al. [2]. The validity of this technique as a method for quantifying the procedural knowledge necessary to operate and interface has been greatly reinforced by Kieras and colleagues [1,5].

Kieras [3] presents a formalism for performing GOMS analysis called NGOMSL (Natural GOMS Language). This language is simple to learn and use for the purposes of interface analysis, and in many ways resembles a programming language. Figure 2 presents a common method as it would be represented in NGOMSL. NGOMSL models may be used to estimate learning time for an interface, as well as execution time for tasks using the interface [3]. While it is not difficult to learn how to construct models in NGOMSL, constructing such models and then using them to predict execution times can be a tedious and time-consuming process.

One of those problems has recently been addressed, however. Recently, we have constructed an NGOMSL "interpreter" [7]. This interpreter takes as input an NGOMSL model of an interface and a description of the tasks to be performed in that interface and produces an estimate of the time it will take to perform the tasks. This makes evaluation of alternative interfaces simple and rapid if NGOMSL models are available for the alternative interfaces. Construction of such models, however, is not aided by the NGOMSL interpreter and is still left entirely to the analyst.

```
Method for goal: select <command> from <menu>
menu
Step 1 Recognize <menu>
Step 2 Move cursor to <menu>
Step 3 Press left mouse button
Step 4 Recognize <command>
Step 5 Move cursor to <command>
Step 6 Release left mouse button
Step 7 Return with goal accomplished
```

Figure 2. Example NGOMSL

THE USAGE TRANSLATOR

The last essential piece of USAGE is the translator. The basic function of the translator is simple: it translates the action sequence file generated by UIDE into an NGOMSL model. The NGOMSL model, along with a description of the tasks to be performed, is then fed into the NGOMSL interpreter. The role of the translator is clearer when viewed in the context of the flow of USAGE, which is depicted in Figure 3, with file icons representing ASCII files and ovals programs.

The translator works by reading from the UIDE-generated application description one application action at a time. For each application action, a new NGOMSL method is generated. The steps in the method are the interface actions necessary to complete the application action. Since the

planner in UIDE ensures the correct and complete action sequences, this guarantees an accurate NGOMSL model.

The interface actions are processed by lookup. Within the translator code, a library of "common" NGOMSL methods are maintained. These include methods for simple actions like clicking on objects, selecting from pull-down menus, and dragging objects with the mouse. The name of the application action and the interaction technique(s) linked to it are used as the index for this library. For instance, the "SelectCommandFromPullDownMenu" in Figure 1 would cause the translator to include a step in the DeleteGate method to accomplish the goal of selecting from a pull-down menu, and would include the method in Figure 2 as part of the NGOMSL model for the interface.

The USAGE translator makes it possible to quickly and effortlessly generate an NGOMSL model of an interface created in UIDE. The power of this tool should be apparent: it makes possible rigorous formal modeling of the user interface without creating additional work for the interface designer. The logic for estimating learning time for the interface has not yet been implemented in the interpreter, but is roughly proportional to a simple count of the number of NGOMSL statements in the model of the interface, and execution time for a set of tasks can be directly estimated by employing the NGOMSL interpreter.

Such a tool has a variety of applications, one of the most obvious being rapid comparisons between interfaces. This particular application of USAGE is important enough to merit more extended treatment.

USAGE AT WORK: CIRCUITDESIGN

In the description of UIDE presented in [6], the digital circuit application which was constructed using UIDE was presented as an example, and it will be used again here. CircuitDesign is a simple demonstration application in which users can create and manipulate simple digital circuits made up of logic gates. Figure 4 depicts the CircuitDesign interface. The current instantiation of CircuitDesign uses a combination of command icons (on the left of the Figure 4) and pull-down menus, a fairly typical GUI.

However, the designer of CircuitDesign could have chosen differently, even within the same interface paradigm; for example, all of the commands could have been placed in the pull-down menus. Conversely, all of the commands could have been icon-based—on the surface, there does not seem to be much of a difference between the two command styles. Such decisions are often made casually, based on

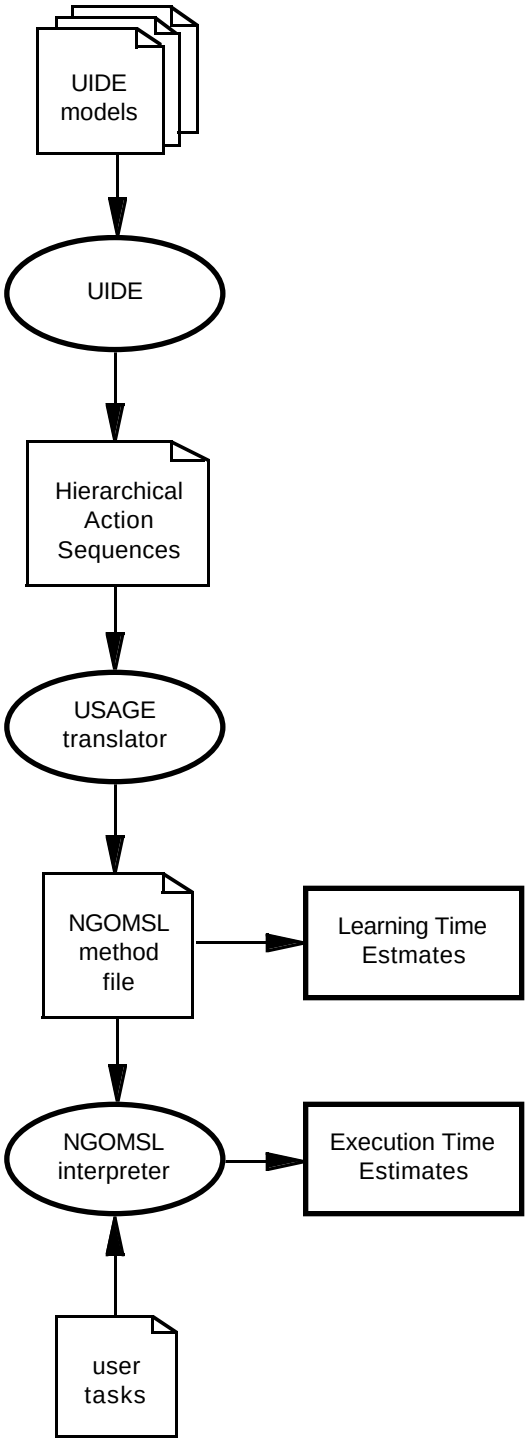

Figure 3. USAGE flow

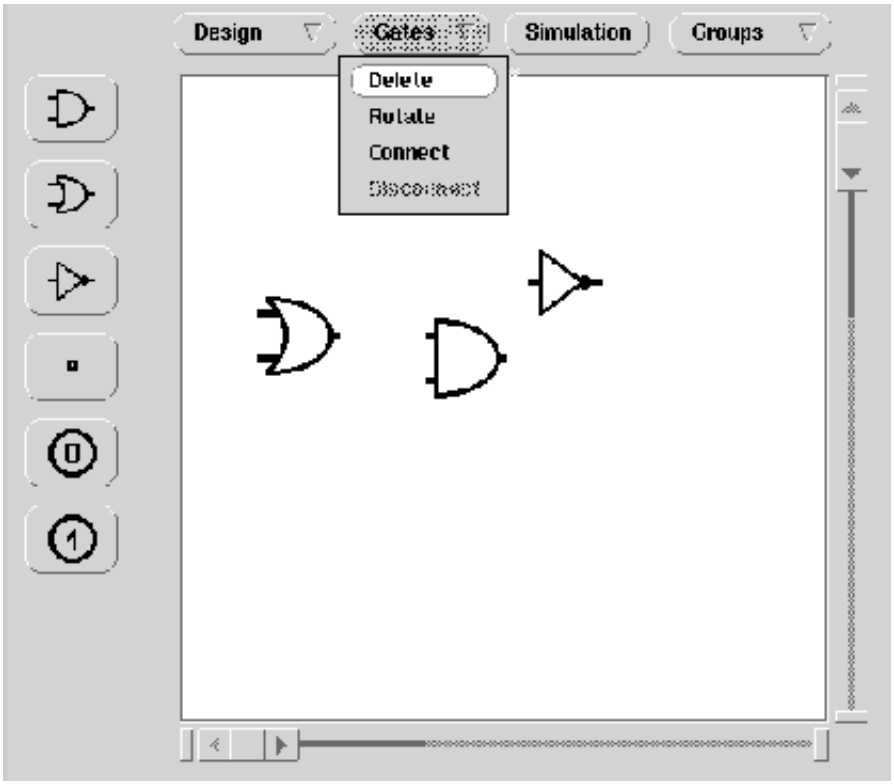

Figure 4. Interface to the CircuitDesign application with some gates in the design

little more than the whim of the designer. The ramifications of such decisions are not often addressed, since the time and effort of doing so is typically presumed to outweigh any minor gains.

With USAGE, however, the impact of such changes can be assessed rapidly, easily, and cheaply. The process of comparing the three interfaces (all icon, mixed, and all menu) is a simple one. First, three different UIDE action listing files need to be generated. This can be done by constructing three separate UIDE instantiations and having UIDE create a new file for each one or by creating one UIDE instantiation and simply altering the action listing file. Second, a set of representative tasks must be defined and constructed for use by the NGOMSL interpreter. Next, the action listing files must be translated to NGOMSL by the USAGE translator. Finally, each NGOMSL model is "run" by the interpreter and the estimate of execution time recorded.

A word needs to be said about the task set used. In order for *any* modeling, whether GOMS or not, to be truly informative, a representative set of tasks must be used. It is not reasonable to construct a model and then estimate execution times for a non-representative set of tasks. If the users of the interface will actually be spending 60% of their tasks on creating new gates, for instance, it makes little sense to use a set of tasks that contain an equal balance of all the tasks possible in the interface. The simple availability of features does not ensure that they will be employed by real users. Thus, while the USAGE process is intended to be as automated as possible, the only way to ensure correct predictions is through the use of task files that are actually representative. Real users, or at least real usage statistics, are the only reliable source of such task information.

For the example here, the task set includes three gate creations, one gate move, and one gate deletion for a total of five tasks. Using the procedure outlined previously, three execution times were obtained, and these are presented in Table 1. If the current mixture of icons and menus is taken as the base for comparison, one can see that the choice of command icons vs. menus does, in fact, make a difference in the estimated execution time. While the absolute times are not of particular interest, the relative times are. While changing to an all-icon interface yields a speed increase of 7%, changing to a completely menu-based interface hurts

	Sec	Ratio
All Icons	36.7	0.93
Mixed	39.3	1.00
All menus	47.1	1.20

Table 1. USAGE execution time estimates

performance considerably more: the decrement is approximately 20%. Certainly, interface designers have other considerations to contend with; e.g. there is only so much screen space available for command icons. On the other hand, USAGE can be an extremely useful aid when making decisions about such tradeoffs.

A system such as USAGE, as a part of the larger UIDE system, is a powerful aid and represents a real advance in bringing formal modeling techniques to the interface designer. As pointed out by Nielsen and Phillips [4], GOMS models constructed by even moderately trained analysts is a reasonably cost-effective analytical technique. An automated GOMS system as part of UIDE provides all the advantages of formal analysis at a substantially reduced cost, as well as all the other advantages of UIDE (see [6]).

LIMITATIONS

While the advantages of USAGE are considerable, it is not without its shortcomings. Currently, one of the major limitations of USAGE is the translation process. As of this writing, the translator is not able to handle application or interface actions that have more than one way available for the user to perform them, such as in the case of command-key equivalents for menu-based commands. This is not a major hurdle, though, and will require only relatively simple additions to the code.

Another weakness of the system comes from the library of NGOMSL methods that the translator must have available. In the current implementation of the translator, the methods are part of the source code and do not exist as a separate database; again, this is a relatively straightforward problem to resolve. More critically, the translator is only able to translate those interface actions/interaction techniques for which NGOMSL already exists. If a system designer is forced to come up with a new kind of interaction technique, they will also have to generate the NGOMSL to go along with it. While this is not an incredibly arduous task, it does require the interface designer be familiar with NGOMSL syntax and discourages the generation of new styles of interaction. But note that a particular interface paradigm, e.g. Windows, corresponds to a library of basic interaction methods; thus, the basic methods to support a paradigm would only be implemented once.

Finally and most critically, USAGE does not address the issue of higher-level goals or tasks. Currently, actions at the level of "delete a gate" are the highest-level tasks that USAGE can handle, as this is the highest-level description present in UIDE. Users, however, are likely to formulate goals at a much higher level, such as "create a flip-flop." This would have gate creation and connection as subgoals. Such hierarchical goal decomposition, the hallmark of much cognitive task analysis, is not available in USAGE, because it is driven by the UIDE description.

FUTURE DIRECTIONS

The obvious next step for USAGE is to solve the simpler technical problems just outlined, as well as to develop a larger NGOMSL library to handle a wider variety of interface actions/interaction techniques. Ideally, the translator would be built directly into the UIDE system and not a separate component.

Beyond that, though, there are other opportunities. One possibility that is intriguing is that of going in the other direction; that is, synthesizing an interface from an NGOMSL model. This would be a much more difficult process, since UIDE application models contain considerably more information than NGOMSL models, and are structured quite differently. However, reverse-translation in one form or another is not an impossibility. Because the NGOMSL form leads naturally to sound top-down design, particularly in the higher-level goal structure of the interface, such a tool has great appeal. The machinery would also be in place to automatically alter the NGOMSL models when technical concerns take the forefront and force changes in the design. Such a fully-integrated two-way system would be an invaluable asset in rapidly designing interfaces that are easy to learn and efficient to use.

CONCLUSIONS

USAGE brings together a state-of-the-art model-based interface design environment (UIDE) and a well-established formal interface analysis technique (GOMS modeling). Such a union allows for the automated generation of an NGOMSL model for an interface specified in UIDE. This system gives interface designers the ability to rapidly and effortlessly evaluate design alternatives. While much more work remains to be done, USAGE offers a tool previously unavailable in user interface development.

ACKNOWLEDGEMENTS

The UIDE project is supported by the Siemens Corporate R&D Systems Ergonomics and Interaction group of Siemens Central Research Laboratory, Munich, Germany and the Human Interface Technology Group of Sun Microsystems through their Collaborative Research Program. The first author is supported by a Graduate Fellowship from the National Science Foundation.

REFERENCES

1. Bovair, S., Kieras, D. E., & Polson, P. G. (1990). The acquisition and performance of text editing skill: A cognitive complexity analysis. *Human Computer Interaction, 5*, 1-48.

2. Card, S., Moran, T., & Newell, A. (1983). *The psychology of human-computer interaction.* Hillsdale, NJ: Erlbaum.

3. Kieras, D. E. (1988). Towards a practical GOMS model methodology for user interface design. In M. Helander (Ed.), *Handbook of human-computer interaction* (pp. 135-158). Amsterdam: North-Holland Elesevier.

4. Nielsen, J., & Phillips, V. L. (1993). Estimating the relative usability of two interfaces: Heuristic, formal, and empirical methods compared. In *Proceedings of INTERCHI'93*, 214-221.

5. Kieras, D. E., & Polson, P. G. (1985). An approach to the formal analysis of user complexity. *International Journal of Man-Machine Studies, 22*, 365-394.

6. Sukaviriya, N., Foley, J. D., & Griffith, T. (1993). A second generation user interface design environment: The model and the runtime architecture. In *Proceedings of INTERCHI'93*, 375-382.

7. Wood, S. (1993). Issues in the Implementation of a GOMS-model design tool. Unpublished report.