GLEAN: A Computer-Based Tool for Rapid GOMS Model Usability Evaluation of User Interface Designs

David E. Kieras kieras@eecs.umich.edu (313) 763-6739
Scott D. Wood swood@eecs.umich.edu
Kasem Abotel kna@engin.umich.edu
Anthony Hornof hornof@umich.edu

Artificial Intelligence Laboratory
Electrical Engineering & Computer Science Department
University of Michigan
1101 Beal Avenue, Ann Arbor, Michigan 48109-2110

ABSTRACT

Engineering models of human performance permit some aspects of usability of interface designs to be predicted from an analysis of the task, and thus can replace to some extent expensive user testing data. The best developed such tools are GOMS models, which have been shown to be accurate and effective in predicting usability of the procedural aspects of interface designs. This paper describes a computer-based tool, GLEAN, that generates quantitative predictions from a supplied GOMS model and a set of benchmark tasks. GLEAN is demonstrated to reproduce the results of a case study of GOMS model application with considerable time savings over both manual modeling as well as empirical testing.

KEYWORDS

User-Interface Software and Technology: *Usability, usability evaluation, user models, GOMS models*

INTRODUCTION

Engineering Models for Usable Interface Design

The standard accepted technique for developing a usable system, empirical user testing, is based on iterative testing and design revision using actual users to test the system and help identify usability problems. It is widely agreed that this approach, inherited from Human Factors, does indeed work when carefully applied [7]. However, Card, Moran, & Newell [4] have argued, and many HCI researchers have agreed (e.g. [2]), that empirical user testing is too slow and expensive for modern software development practice, especially when difficult-to-get domain experts are the target user group. One response has been the development of "discount" or "inspection" methods for assessing the usability of an interface design quickly and at low cost [17]. However, another response, which has been evolving since the seminal Card, Moran, and Newell work, is the concept of engineering models for usability. Engineering models for usability are analogous to the models used in other engineering disciplines in that they produce quantitative predictions of how well humans will be able to perform tasks with a proposed design. Such predictions can be used as a surrogate for actual empirical user data, making it possible to iterate through design revisions and evaluations much more rapidly. Furthermore, unlike purely empirical assessments, an engineering model for an interface design can capture the essence of the design in an inspectable representation, making it easier to reuse successful design insights in the future.

The overall scheme for using engineering models in the user interface design process is as follows: Following an initial task analysis and proposed first interface design, the interface designer would then use an engineering model to find the applicable usability problems in the interface. However, because there are other aspects of usability that are poorly understood, some form of user testing is still required to ensure a quality result. Only after dealing with design problems revealed by the engineering model would the designer then go on to user testing. If the user testing reveals a serious problem, the design might have to be fundamentally revised, but again the engineering models will help refine the redesign quickly. Thus the slow and expensive process of user testing is reserved for those aspects of usability that can only be addressed at this time by empirical trials. If engineering models can be fully developed into computer-based tools, then the designer's creativity and development resources can be more fully devoted to more challenging design problems, such as entirely new concepts or approaches to the problem at hand.

The GOMS Model

The major extant form of engineering model for interface design is the GOMS model, first proposed by Card, Moran, and Newell. John & Kieras [8] list many successful applications of GOMS to practical design problems. A GOMS model is a description of the knowledge that a user must have in order to carry out tasks on a device or system;

To appear in Proceedings of UIST'95

it is a representation of the "how to do it" knowledge that is required by a system in order to get the intended tasks accomplished. The acronym GOMS stands for Goals, Operators, Methods, and Selection Rules. Briefly, a GOMS model consists of descriptions of the Methods needed to accomplish specified Goals. The Methods are a series of steps consisting of Operators that the user performs. A Method may call for sub-Goals to be accomplished, so the Methods have a hierarchical structure. If there is more than one Method to accomplish a Goal, then Selection Rules choose the appropriate Method depending on the context. Describing the Goals, Operators, Methods, and Selection Rules for a set of tasks in a formal way constitutes doing a GOMS analysis, or constructing a GOMS model.

Research summarized by John & Kieras [8] has resulted in a family of GOMS models and techniques for predicting key aspects of usability of an interface. In particular, execution time can be predicted by simulating the execution of the methods required to perform the task. The time to learn how to operate the interface can be predicted from the length of the methods and transfer of training from the number of methods or method steps previously learned. One important feature of these GOMS models is that the "how to do it" knowledge is described in a form that can actually be executed – the analyst, or an appropriately programmed computer, can go through the GOMS description, executing the described actions, and actually carry out the task.

The type of GOMS model used in the work reported here is known as the NGOMSL methodology [8, 9, 10] and is based on the cognitive modeling of human-computer interaction by Kieras and Polson [1, 13]. NGOMSL is an acronym for **N**atural **GOMS L**anguage, which is a structured natural language used to represent the user's methods and selection rules. This paper introduces GOMSL, (**GOMS L**anguage), which is a formalized, machine-executable form of NGOMSL. The NGOMSL type of GOMS model has an explicit representation of the user's methods, which are assumed to be strictly sequential and hierarchical in form, and is useful for many desktop computing applications (see [8] for more discussion).

Figure 1 provides a small example in GOMSL of a set of methods for doing file moving and deleting on the Macintosh. Each method accomplishes its goal by either calling submethods to accomplish subgoals, or executing primitive ("keystroke-level") actions such as pressing the mouse button. Other low-level operators, such as Step 1 in the drag method represent how the user examines the screen to find an object and then remembers its location as the destination for a mouse point operation. Note how the underlying simplicity and consistency of the Macintosh methods is apparent from this small example – a single general method is used for two different user goals.

```
Method for goal: Move (file) to (destination).
Step 1. Accomplish goal: Drag (file) to
  (destination).
Step 2. Return with goal accomplished.

Method for goal: Delete (file).
Step 1. Accomplish goal: Drag (file) to ('trash').
Step 2. Return with goal accomplished.

Method for goal: Drag (source) to (destination).
Step 1. Find object whose label is source and
  store its location under source position.
Step 2. Decide: If hand is not at mouse then home
  to mouse.
Step 3. Point to source position.
Step 4. Hold down mouse button.
Step 5. Find object whose label is destination and
  store its location under destination position.
Step 6. Point to destination position.
Step 7. Release mouse button.
Step 8. Delete source position, delete destination
  position.
Step 9. Return with goal accomplished.
```

Fig. 1. An example GOMS model: Methods for moving and deleting files in the Macintosh Finder interface, written in the GOMSL notation used in GLEAN. The `move` and `delete` methods are called with arguments `file` and `destination` which are tags for file or folder names in working memory, and then they call the `drag` submethod. The `find` operator searches the screen for an object and stores its location in working memory. The `point` operator then uses this working memory information. See text for further explanation.

Strengths and Limitations of GOMS Models

It is important to be clear on what GOMS models can and cannot do (see [8] for more discussion). First, in order to apply the GOMS technique, the designer (or interface analyst, hereafter just referred to as the designer) must conduct a task analysis to identify what goals the user will be trying to accomplish. The designer can then express in a GOMS model how the user can accomplish these goals with the system being designed. Thus, GOMS modeling does not replace the most critical process in designing a usable system, that of understanding the user's situation, working context, and goals. Approaches to this stage of interface design have been presented in sources such [7, 15].

Second, GOMS models can predict the *procedural* aspects of usability; these concern the amount, consistency, and efficiency of the procedures that users must follow. Since the usability of many systems depends heavily on the simplicity and efficiency of the procedures, the narrowly focused GOMS model has considerable value in guiding interface design. The reason why GOMS models can predict these aspects of usability is that the methods for accomplishing user goals tend to be tightly constrained by the design of the interface, making it possible to construct a GOMS model given just the interface design, prior to any prototyping or user testing.

Third, there are other important aspects of usability that are not related to the procedures entailed by the interface design. These concern both lowest-level perceptual issues like the

legibility of typefaces on CRTs, and also very high-level issues such as the user's conceptual knowledge of the system, e.g., whether the user has an appropriate "mental model" [11], or the extent to which the system fits appropriately into an organization [see 8]. The lowest-level issues are dealt with well by standard human factors methodology, while understanding the higher-level concerns is currently a matter of practitioner wisdom and the higher-level task analysis techniques [15]. Considerably more research is needed on the higher-level aspects of usability, and tools for dealing with the corresponding design issues are far off. For these reasons, great attention must still be given to the task analysis, and some user testing will still be required to ensure a high-quality user interface.

Fourth, the development of the GOMS modeling techniques has involved validating the analysis against empirical data, as is also done in this paper. However, once the technique has been validated and the relevant parameters estimated, no empirical data collection or validation should be needed to apply a GOMS analysis during practical system design.

Fifth, there has been a widespread belief that constructing and using GOMS models is too time-consuming to be practical (e.g., [16]). However, the many cases surveyed by John & Kieras [8] make clear that members of the GOMS family have been applied in many practical situations and were often very time- and cost-effective. A substantial problem is that the calculations required to derive the predictions are tedious and mechanical. Eliminating this problem is the target of the work reported here.

Goals of this Work

Our goal is to develop a family of computer-based tools that will allow interface designers or analysts to easily develop and rapidly apply GOMS model techniques and extensions to them. The work reported here is part of the MUSETTE project (**M**odel-based **US**ability **E**valuation **T**ool and **T**echnique **E**nsemble). The first MUSETTE tool is called GLEAN, for **G**OMS **L**anguage **E**valuation and **AN**alysis. The GLEAN user (the interface designer), will develop a GOMS model for an existing or proposed interface. GLEAN will then calculate estimated procedure learning time and execution time for a set of benchmark tasks, and will supply additional information to help identify problems in the interface design.

As a test of the accuracy and functionality of GLEAN, we applied it to a user interface analysis and design case reported by Gong [5], who used GOMS analysis to identify usability problems in a software product, and to evaluate the improvement in the interface produced by correcting these problems. Gong then also conducted a formal empirical usability study to collect measures of actual usability. We attempted to reproduce Gong's models and predicted usability results with GLEAN.

In the remainder of this paper, we first describe the design goals for GLEAN and the current form of GLEAN. Then we describe the application of GLEAN to Gong's study and the accuracy of the usability predictions.

DESCRIPTION OF GLEAN

Design Goals

A first and fundamental design goal is that GLEAN must automate the tedious calculations required to generate usability predictions from a GOMS model. A prototype version of GLEAN, developed by Scott Wood [18], demonstrated that this was feasible. Part of this goal is that the designer should obtain the results of the GOMS analysis in a useful and intelligible form.

A second design goal is that it must be easy, fast, and simple for the designer to supply the inputs to GLEAN with relatively little training. Note that task analysis is required to determine what goals the user is trying to accomplish, and at least the top-level methods that map the user goals to the facilities available in the interface. So using GOMS methodology will always require a designer who is trained to perform appropriate task analysis. Also, it is reasonable that the designer have as much training as currently seems to be necessary to get started in GOMS analysis (e.g. a one-day short course; see [8]), but there should be very little learning required to use the GOMS tool itself.

A third design goal is that the GOMS model notation in GLEAN should be readable and comprehensible with little or no training. That is, a designer using GLEAN should be able to show an actual model to someone unfamiliar with GOMS, and this other person must be able to grasp the gist of the model with only a small amount of explanation. If so, a designer will be able to explain, display, and justify the results of the analysis to other parties (e.g. project management) without a serious language barrier. Hence we have tried to preserve the English-like flavor of NGOMSL in the formalized GOMSL notation. The gamble is that this English-like GOMSL might be harder to write than a more compact notation, but almost anybody should be able to read it with only a little explanation. The example shown in Figure 1 illustrates this goal.

A final design goal was to simplify the development and standardization of GOMS models by supporting the reuse of GOMS methods. In many cases, the modeled interface is being developed for a standard interface paradigm or platform, such as the Macintosh or Windows. Many of the interface methods are in fact supposed to be standard for the platform and these standard methods should be available in a library.

Structure of GLEAN

Figure 2 shows the structure of GLEAN. In essence, GLEAN is a facility for simulating the interaction between a *simulated user* who interacts with a *simulated device* (the

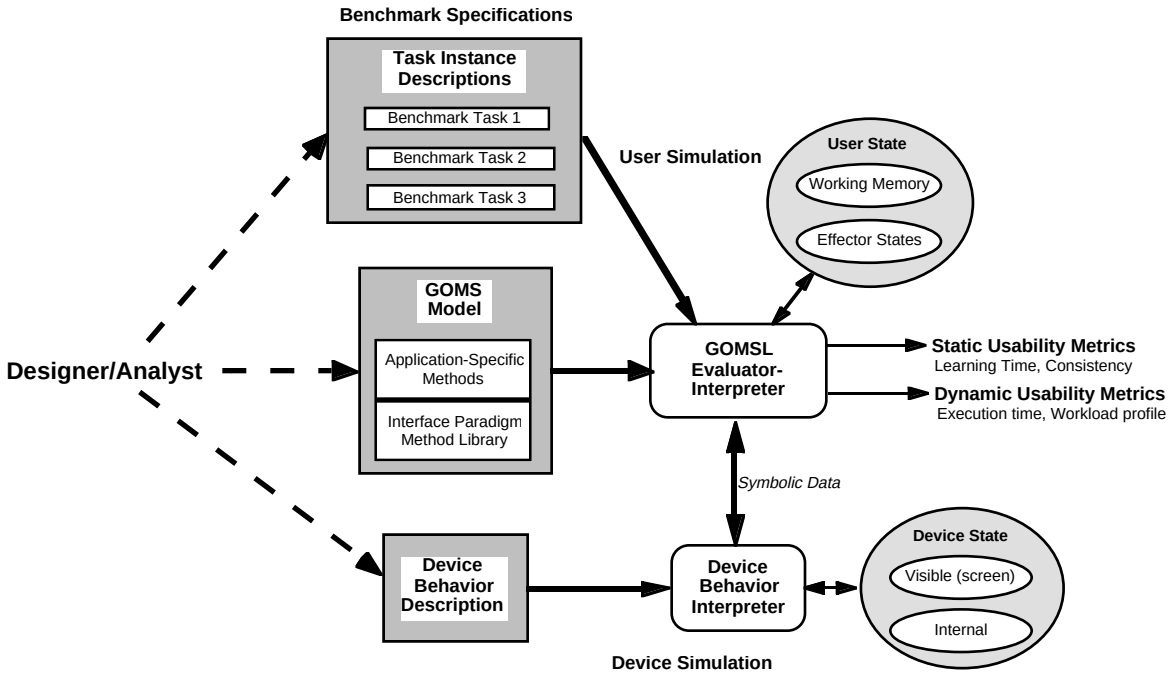

Fig. 2. Structure of GLEAN. The designer supplies benchmark tasks, a GOMS model, and a description of interface behavior. GLEAN simulates the user-device interaction and generates usability metrics.

interface) to execute a set of specified benchmark tasks. To set up the simulation, the designer supplies three representations contained in simple text files and expressed in a defined notation. The Task Instance Descriptions specify the benchmark tasks. The user's procedural knowledge is represented with the GOMS model expressed in the GOMSL notation. The behavior of the simulated interface is specified by the Device Behavior Description, which specifies the objects in the interface (e.g. icons on the screen) in terms of abstract properties, such as their location and appearance, and their behaviors in response to user input.

The GOMSL Evaluator/Interpreter generates static measures of usability from the GOMS model, such as the predicted procedure learning time. It also generates dynamic measures by executing the methods to accomplish each of the specified benchmark tasks. During this execution, the GOMS methods modify the user state, such as the contents of working memory, and execute operators that represent interactions with the device, such as locating an icon on the screen or pointing with the mouse. The Device Behavior Interpreter supplies information about the state of the device, such as the current location of a specified icon, or updates the state of the device, such as changing the cursor location in response to mouse movement operators.

The main purpose of the simulated device is to automatically generate the feedback required by the GOMS model; for example, if the GOMS model needs to drag an icon from one place to another, the device simulation supplies the current location of the object and the mouse cursor, and the new locations, along with the information that the destination icon has become reverse-video. Such a facility makes it much easier to debug and run the GOMS models. However, to permit writing and partially testing a GOMS model quickly, GLEAN permits the designer to run a GOMS model without a specified simulated device.

Thus, to predict the usability of a proposed design with GLEAN, the designer performs the following steps: (1) chooses and represents the benchmark tasks; (2) writes the GOMS model entailed by the user interface design; (3) describes the behavior of the interface at the abstract level required to provide any necessary feedback for the methods; (4) debugs the model by running it with GLEAN and correcting any errors; (5) obtains the predictions of usability by running GLEAN on the final model; (6) examines the usability predictions, the execution time profile, and the structure of the GOMS methods to identify problems in the interface design and suggest solutions; and (7) modifies the GOMS model and device description to reflect the changes in the interface design, conducts a new evaluation, and uses the results to drive further design.

GOMS Model Representation

The GOMS model is expressed in the GOMSL notation and consists of a set of application-specific methods, a library of general methods for the interface paradigm (platform), and a list of of additional required user knowledge in the form of associations in long-term memory (LTM), such as under which menu a particular command may be found. Due to

limitations of space, a detailed presentation of the syntax and semantics of GOMSL is not possible, but examination of the examples should clarify most of the properties of GOMSL.

The representation of the user's working memory (WM) requires discussion. A key requirement for an executable GOMS model is the ability to explicitly represent how the user is assumed to store and retrieve items in WM. Our goal in designing this aspect of GOMSL was to enable methods to be written in a simple style in which these WM operations would not be obtrusive or clumsy. Thus, the user's working memory (WM) is assumed to be a simple list of properties (tags) and associated values. A `store` operator can store a value in working memory under a specified tag, and any operator can retrieve a WM item by referring to the tag as an argument. When the information is no longer needed, a `delete` operator will delete the tag and value. For example, in the `drag` method shown in Figure 1, the location of an object on the screen is stored under a tag which is then used as an argument for a `point` operator, and then finally discarded.

Information is passed from a method to a submethod through WM; as shown in the Figure 1 example, a parenthesized tag, or "pseudo-argument," in a goal description specifies which value in the current WM contents is to be copied under a new tag for a submethod. This approach allows submethods to be encapsulated to some extent, analogously to conventional programming languages, and thus enabling the use of method libraries. However, there are several difficult issues about the psychological and programming implications of this representation that we have not fully resolved; WM tags are used similarly to variables in ordinary programming languages, but are fundamentally different. Since GOMSL is intended to represent human procedural knowledge and human limitations and abilities; it can not be a full-fledged programming language that supports facilities such as locally scoped variables and recursion. On the other hand, the way GLEAN represents WM may be puzzling to the intended audience; we will have to gain further experience and feedback.

Benchmark Task Description

The Task Instance Description describes a set of specific task instances to be used to predict execution times. These descriptions are supposed to be as independent of the interface design as possible, containing only declarative information about the parameters and requirements of the task. An example task representation from the Gong study described below is shown in Figure 3 and illustrates some important features. A task description is a set of objects, each of which can have property-value pairs that specify task parameters or subtasks. Our goal is to allow arbitrary mixtures of task information whose organization is

```
Top level goal is Use the 3DSSPP.
Task: Use the 3DSSPP.
Task sequence is Specify anthropometry, Specify
 the action at the hands, Specify the force at the
 hands, Predict by hand locations.

Task: Specify anthropometry.
  Gender is Male.
  Height&Weight is 50th %ile.
Task: Specify the action at the hands.
  Action is Lift Up.
Task: Specify the force at the hands.
Force sequence is Right Magnitude, Left Magnitude.
Number values are 13, 16.
```

Fig. 3. An example task instance description.

hierarchical, sequential, or unordered. In many task domains, an instance of a task contains task parameters or subtasks that are unordered and always available; that is, they have meaning independent of other parts of the task, the order in which they appear in the description, or the order of use during task performance. However, task instances also can contain ordered subtasks or parameters, which may have to be performed or used either in some arbitrary order or an order determined by the structure of the task or the task methods. Our task instance description notation allows us to define such sequences and associate them with different portions of the task.

The Device Behavior Description and Interpreter

The Device Behavior Interpreter applies the device behavior description to simulate at an abstract level of detail the behavior of the device during task execution. This description is an abstract representation of the device states and transitions, and what feedback is supplied to the GOMS method from the device; no actual implementation of device functionality is required. For example, a GOMSL `find` operator being executed will result in a query about the current location of an icon on the device's screen; the device simulator will supply the location to the user simulation, where it will be placed in the working memory partition of the user state. Then a GOMSL `point` operator could be executed to move the cursor to that location. The device simulator would receive the point command, and update the current location of the cursor in the device state. If the icon is subsequently dragged, then its location would also be updated.

This approach of simulating the abstract behavior of the device was followed in the original Kieras and Polson modeling work with a specialized transition-network representation of the device behavior [1, 12, 13]. Our current description notation is intended to be much easier to use, being similar in syntax to GOMSL. The Device Behavior Description includes the objects in the interface their properties, and their behavior in response to user input. For example, for the Macintosh, the device description would contain an object for the trash can icon, its location, and appearance, and how the appearance changes when the user acts on the trash can; for example, it reverse-videos

when another object is dragged onto it. In this notation, the behavior is described by a procedure for each user operator that describes how the device state should change. Using this representation, we have developed a simulation of many Macintosh interface behaviors. The adequacy of this facility, and its possible future development, are discussed more in the conclusion.

GOMSL Evaluator/Interpreter

When GLEAN first loads a model, the GOMSL Evaluator/Interpreter calculates estimated learning times. When the model is run, the state of the simulated user is updated during execution; for example, the contents of working memory change and the hands move from place to place. The user state can be interrogated by the methods, for example, to determine whether the hand needs to be moved from the keyboard to the mouse. At the designer's option, a full trace of each method step can be displayed, along with the current contents of the simulated user's working memory and the current state of the device simulation. The final output is a detailed profile showing the frequency, average time, and total execution time of each operator and each method. This profiling facility helps the designer to determine the source of any differences in task execution time when comparing interface designs.

GLEAN calculates execution time predictions by following the recommendations in Kieras [9, 10] and Gong [5, 6]. For example, each GOMSL step requires 0.1 s, each keystroke, 0.28 s. Gong [5] found that the 1.1 s mouse pointing time recommended by Card, Moran, & Newell [4], which was based on text editing activity, is actually quite inaccurate for GUI interfaces, because mouse movements are often made to large or close targets, such as activating windows, clicking on buttons, and so forth. GLEAN maintains the current position of the mouse cursor, and so can use Fitts' Law [4] to calculate the time required to move the cursor to a target object whose size is specified.

GLEAN calculates predicted procedure learning time following the formulas recommended by Kieras [9, 10] and Gong [5, 6]. GLEAN tallies the total number of method steps to be learned, taking account of which methods the designer has designated as already known to the user. For example, a typical Macintosh user should already know the basic Macintosh methods such as dragging and selecting, so the learning time for a new Macintosh application should not include the time for these methods. Thus the designer can indicate that single methods, or all of the methods in a file (e.g. the Macintosh basic methods library), are already learned. In addition, GLEAN calculates a refined transfer of training predictor [1,9] involving identifying identical steps in methods with similar goals. Finally, the time to memorize LTM associations is included in the learning time as well, based on the number of chunks specified by the designer.

However, at the time of this writing, the learning time predictions are problematic and require recalibration. Since the time to learn the procedures depends on the number of steps in the methods, the learning time predictions are sensitive to the "programming style" used in writing the methods. The style rules and model structure suggestions in the NGOMSL methodology [9, 10] are rather loose and informal compared to GOMSL. NGOMSL was intended to be suitable for easy construction and manual application, while GOMSL is machine-executable and much more tightly specified, especially with respect to WM usage and task instance definitions. Consequently, we will have to develop some new and more rigorous style rules for certain aspects of GOMS model structure and representation, and recalibrate the learning time prediction formulas accordingly. This work is in progress.

In the meantime, note that the number of method steps determines the amount of "cognitive overhead," but does not affect which or how many keystroke-level operators are executed. Since the cognitive overhead contributes only a little to the predicted execution time, GLEAN's execution time predictions are relatively unaffected by programming style differences.

A final feature of interest is that the designer can define special operators to represent special-case types of user activity, and can override the default time values associated with operators. Thus, following Gong's recommendations, in the results reported below, the experienced users are assumed to overlap many mental operations (e.g. `locate`, `verify`) with the accompanying physical actions, and so these operators were set to zero time.

Technical Notes

GLEAN is currently implemented in Common Lisp/CLOS and normally runs under Macintosh Common Lisp with a rudimentary menu interface. Our current effort is to keep the tool as platform-independent as possible, so the Macintosh interface is purely a convenience. Future versions are expected in be in C++ and use the Amulet interface toolkit.

An important design feature is that all of the notations used in GLEAN for the GOMS model, device description, and task instance description, are parsed by a translator that converts the representations into a set of internal objects used by evaluation and simulation functions in GLEAN. Thus details of the notation syntax can be modified independently of the evaluation and simulator facilities and it is possible to support multiple notations if desired. A second noteworthy design feature is that the task instances, contents of LTM, and the device state, are represented in a uniform object-property-value structure, which simplifies the definition of GOMSL operators and the notations.

DEMONSTRATION OF GLEAN: REPRODUCING THE GONG RESULTS

As a test of the accuracy and usability of GLEAN, we applied it to a user interface analysis and design case study conducted by Richard Gong [5, 6], who used GOMS analysis to identify usability problems in a software product, and to evaluate the improvement in the interface produced by correcting these problems. Gong then also conducted a formal empirical usability study to collect measures of actual usability which he used to verify and correct the GOMS prediction methodology. We attempted to reproduce Gong's models and predicted usability results with GLEAN.

Summary of Gong's Study

Gong applied the GOMS model methodology described in [9] to the design of a full-sized Macintosh computer software application. This application is a CAD system for the ergonomic design of factory workstations; it allows an ergonomics analyst to determine whether a particular job (e.g. installing automobile batteries on an assembly line) will pose a risk of injury to the worker. The *original* interface was constructed according to standard rules for the platform. The main display is illustrated in Figure 4. Conventional usability evaluation of this interface was difficult because the user base consists of a small expert population not readily available for full-scale usability tests, and an informal user survey produced only information about missing functionality or violations of expectations, rather than difficulties in learning or use.

Working independently, Gong constructed a GOMS model in NGOMSL of this original interface, and following the procedures in [9], evaluated it both qualitatively and quantitatively. The results of the GOMS evaluation was far more detailed and specific than the user survey results. Gong discovered a variety of problems in the interface procedures that were predicted to produce longer learning time and slower execution time than necessary. He redesigned the interface in response to these problems; the main display of the revised version is shown in Figure 5. Note that the revised display appears to be considerably more cluttered than the original, which violates one of the most widely-followed traditional usability guidelines.

Despite the more cluttered display, Gong found that the quantitative predictions from the GOMS analysis indicated that the revised interface would be 46% faster to learn and 40% faster to use. In a formal empirical comparison of the two versions, these predicted improvements were accurate within 10%, which is clearly accurate enough for engineering design purposes.

For validation purposes, Gong predicted and collected the task times for the two interfaces for a series of seven tasks, each described on a sheet of paper showing the biomechanical data input. The user had to obtain the desired result from the system. Gong actually worked in terms of execution times for a very stable subset of the complete task, which he termed "form fill-in," in which the user sets buttons and dialog fields to the proper values for the task. While this is a relatively simple type of interface activity, nonetheless it is heavily involved in many current interfaces.

GLEAN Models for Gong's Interfaces

In our replication of Gong's analysis, we recoded his informal NGOMSL into GLEAN's GOMSL for the two interfaces. Our models included just the methods required for Gong's "form fill-in" part of the task. We constructed a set of methods for the application- and interface-specific portions of each of the two interfaces, and a second set of methods for the generic interface procedures for the platform. Following Gong, these generic methods were constructed in two forms: A lower-limit form assumed that the user used accelerator keys whenever possible, while an upper-limit form represented the user always pointing and clicking with the mouse. Actual users would be expected to use some mixture of the two forms. In addition to the methods, we represented each of Gong's seven benchmark tasks using our task instance description notation. The models were tested to ensure that they generated the correct sequence of user actions for each benchmark task.

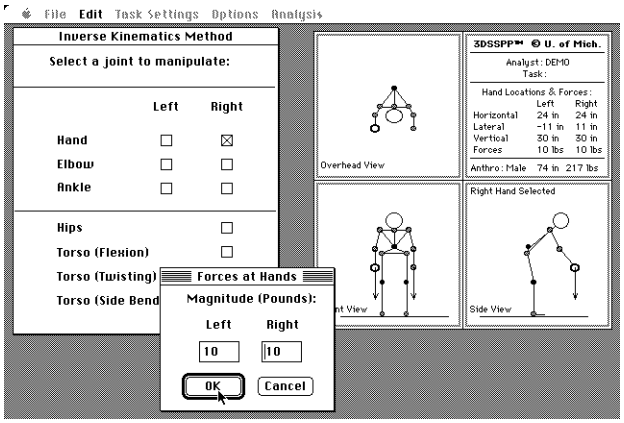

Fig. 4. The main display from Gong's original interface.

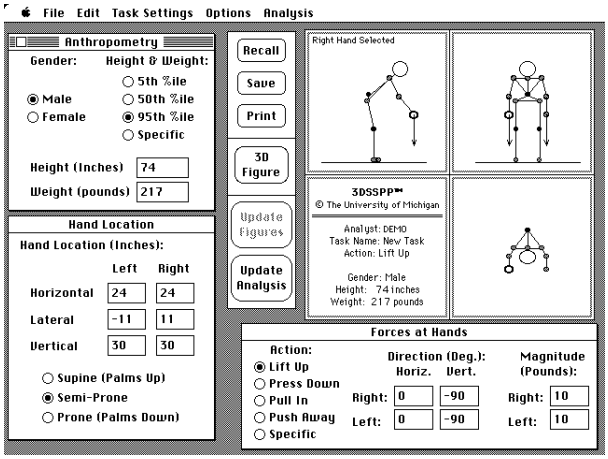

Fig. 5. The main display from Gong's revised interface.

Space limitations prevent illustrating the models in any detail. In summary, both interface models had the same task-domain-specific top level methods for cycling through the subtasks, and shared the same upper- and lower-limit libraries of generic methods. To illustrate the difference between the two interfaces, Figure 6 shows one of the interface-specific methods for the original interface, which requires opening a dialog box with a menu access, and Figure 7 shows the corresponding method in the revised interface, which requires only activating an on-screen window. The revised interface involves methods that are shorter and faster to execute.

```
Method for goal: Specify the force at the hands.
Step 1. Accomplish goal: Make sure the dialog
  ('Forces at Hands') is visible.
Step 2. Accomplish goal: Specify the field
  sequence ('Force') to contain the value sequence
  ('Number').
Step 3. Accomplish goal: Dismiss the dialog
  ('Forces at Hands').
Step 4. Return with goal accomplished.

LTM item: Forces at Hands.
   Access Menu is Task Settings.
   Command key is F.
   Chunks is 3.
```

Fig. 6. Example interface-specific method for the original interface, and the accompanying definition of command key and menu LTM associations. The fields to be filled in are in a dialog box that must be called up and dismissed. The LTM item is used by the method in Figure 8 to call up the dialog.

```
Method for goal: Specify the force at the hands.
Step 1. Accomplish goal: Make sure the window
  ('Forces at Hands') is the active window.
Step 2. Accomplish goal: Specify the field sequence
  ('Force') to contain the value sequence
  ('Number').
Step 3. Return with goal accomplished.
```

Fig. 7. Example interface-specific method for the revised interface. The fields to be filled in by the method are in an on-screen window.

Figure 8 gives some examples of the lower-limit generic library methods for performing lower-level interactions on the Macintosh platform. Both models used these libraries for the lowest level methods.

In addition to the method differences, the two models also differed in the number of Long-Term Memory associations required to specify which menu to open or command key to strike to invoke specific commands. These associations specifies with the methods, as shown in Figure 6, and are retrieved by the `Retrieve from LTM` operators in the library methods shown in Figure 8.

The two interfaces did not differ much in how many methods were required, but did differ in which generic methods were invoked, how long the interface-specific methods were, and how many LTM items were required to

```
Method for goal: Make sure the dialog (new dialog)
  is visible.
Step 1. Decide: If appearance of new dialog is
  visible then Return with goal accomplished.
Step 2. Retrieve from LTM command key of new dialog
  and store under key.
Step 3. Accomplish goal: Press the command key
  (key).
Step 4. Return with goal accomplished.

Method for goal: Press the command key (target
  key).
Step 1. Decide: If hand is not at keyboard then
  Home to keyboard.
Step 2. Keystroke 'Command'.
Step 3. Keystroke target key.
Step 4. Return with goal accomplished.
```

Fig. 8. Example lower limit methods from the generic method library. A command key associated in the user's LTM with the desired dialog is used to open the dialog.

specify the parameters for generic command-issuing methods. The revised interface had more of the controls visible on the screen at all times, so fewer LTM associations were required, and the methods would be shorter to learn and faster to execute, since it would not be necessary to open and dismiss as many dialog boxes. This contrast can be seen by examining Figures 6 and 7.

Accuracy of Predicted Times

As discussed above, the learning time predictions are in need of recalibration due to the differences in notation and programming style between Gong's informal NGOMSL models and our formalized GOMSL ones. GLEAN's predicted learning time for each interface is significantly larger than what Gong observed; but the predicted improvement between the original and revised interfaces is roughly correct, but is too small: Gong observed a 46% improvement in learning time, while GLEAN predicts a 31% improvement. Work to better characterize the relevant modeling rules and to recalibrate the learning time predictions is underway.

Figure 9 shows the observed task execution times measured by Gong for the original and revised interfaces for each of the seven tasks and the predicted times produced by GLEAN. Following Gong's suggestion, the predicted times were produced by averaging the predicted execution times produced by the upper- and lower-limit models. The difference between tasks and the two interfaces is predicted very accurately by GLEAN; the absolute error of prediction ranges from 1% to 16%, and averages 8%. Examining the execution time profile produced by GLEAN for the two interfaces shows that, as Gong described, the revised interface user is faster because the original interface user spends more time calling up and dismissing dialogs using menu accesses and retrieving items from LTM. Clearly, these results indicate that GOMS models using GLEAN are capable of making useful and accurate predictions of task execution time.

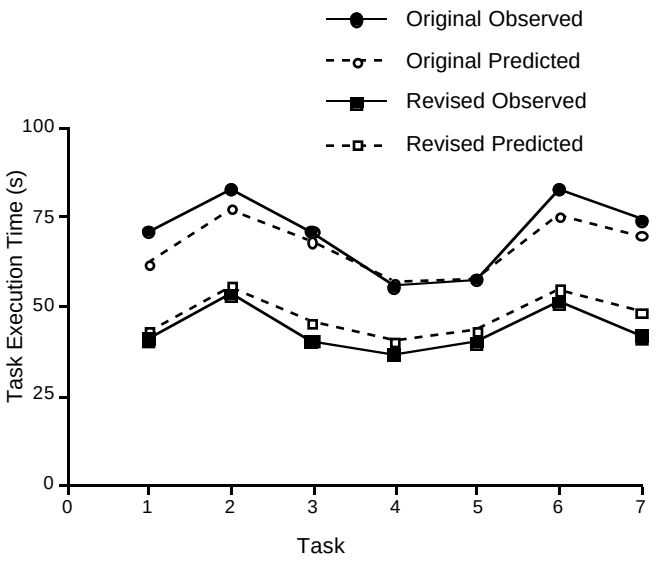

Fig. 9. Observed and predicted task execution times. The revised interface is faster than the original interface; the predicted values (small open shapes with dashed lines) average within 8% of the observed times (large solid shapes with solid lines).

Projection of Effort Savings due to GLEAN

Gong [5, 6] kept records of the amount of effort required to construct and apply the GOMS model; here we consider how the effort might have be decreased if GLEAN had been available for Gong's use.

Figure 10 shows a summary of the distribution of effort reported by Gong in developing the interfaces. Interface development includes all design drafting and interface coding; user assessment includes both a user survey and formal usability test; and the manual GOMS activity is the 11 days that Gong reported being spent on (1) creating the GOMS model, (2) generating the predictions, and (3) interpreting the modeling results. Of these 11 days, Gong (personal communication) spent about 4-5 days generating the predictions by first creating the action trace for each task by hand and listing the operator sequences, and then setting up and checking spreadsheets, and finally obtaining the predictions. This work GLEAN does within seconds; the resulting projected time savings are included in the GLEAN GOMS activity in Figure 10. This projection does not include any additional time that might be required to express the models in the formal GOMSL as opposed to the relatively free-form NGOMSL used by Gong; note that the work to express a GOMS model more formally might well be repaid by greater ease in ensuring that the model is correct and complete.

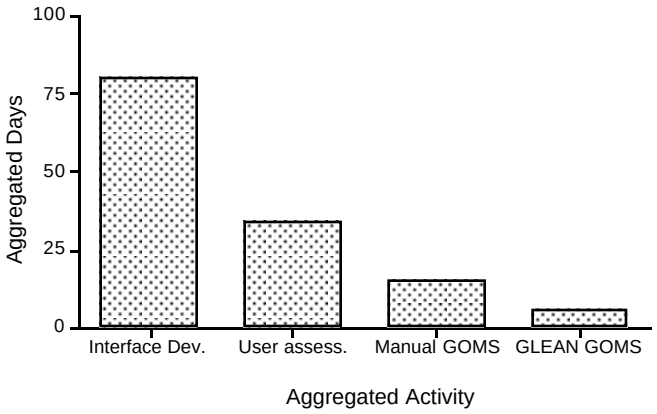

Fig. 10. Effort requirements reported by Gong for performing each type of activity, along with projected requirement if GLEAN had been used for the GOMS calculations.

Furthermore, if more than one revision with additional interface changes had been performed, the savings would have been substantially more, both with using GLEAN, and with using GOMS. The time to modify the GOMS model would probably be very small for most interface revisions, and the time to recalculate the results with GLEAN would be almost zero, but manually would still probably take a few days. In contrast, the user assessment time for even a small interface variation would probably be doubled because an entirely new complete user test would likely be required, with little savings from the previous user test.

CONCLUSIONS AND FUTURE WORK

Our reproduction of Gong's study makes a strong case, along with the others summarized by John & Kieras [8], that GOMS modeling is an efficient usability evaluation technique, especially when augmented with automated tools such as GLEAN.

Further work on developing GLEAN needs to focus on several issues: In the immediate future, we plan to construct a variety of models using GLEAN, and then apply GLEAN to an actual interface being designed by a software development project. This activity will help check for adequate generality, scope, and ease of use of GOMSL and the task instance language, and to clarify the requirements for the device representation. In addition, we expect to gain a better understanding of the kinds of output that the tool should produce, such as more specific execution profiles. Finally, of course, we need to obtain feedback from representative software developers and interface designers on the ease of learning and using both GOMSL and the GLEAN tool itself.

In the longer term, we plan to incorporate results from research in progress such as [14] for estimating the time required for visual search and possibly identifying portions of the interface where user errors are especially likely.

The device simulator portion of GLEAN is problematic. Developing an accurate and complete GOMS model

requires testing the model against the proposed interface; having the proposed interface represented as a simulated device greatly assists this testing. However, the more elaborate and complete the device description and simulation becomes, the more it becomes a duplication of the actual interface implementation, and the less the justification for having a separate facility for device description and simulation in GLEAN. Coupling GLEAN to a suitable interface development environment would avoid this duplication.

An additional relationship between interface implementations and GLEAN is that there is some resemblance between the content of a GOMS model and some representation of an interface itself, as should be evident from the Figure 1 example above. GOMSL is defined so that when written in a certain style, important psychological aspects of the interface are captured, which is not a goal of current UIMS efforts for representing interfaces. But to some extent, each could be constructed from the other, as was demonstrated by Byrne, Wood, Sukaviriya, Foley, and Kieras [3] using UIDE and the prototype GLEAN. If so, constructing an elaborate device simulation just to assist the GOMS evaluation would be wasteful; it should be possible to take advantage of the redundancy between GOMS models and interface representations to even further reduce the time required to develop a usable interface.

ACKNOWLEDGEMENT

This work was supported by the Advanced Research Projects Agency under order number B328, monitored by NCCOSC under contract number N66001-94-C-6036.

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. Butler, K. A., Bennett, J., Polson, P., and Karat, J. (1989). Report on the workshop on analytical models: Predicting the complexity of human-computer interaction. *SIGCHI Bulletin*, 20(4), pp. 63-79.

3. Byrne, M.D., Wood, S.D, Sukaviriya, P., Foley, J.D, and Kieras, D.E. (1994). Automating Interface Evaluation. In *Proceedings of CHI,* 1994, Boston, MA, USA, April 24-28, 1994). New York: ACM, pp. 232-237.

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

5. Gong, R. (1993). *Validating and refining the GOMS model methodology for software user interface design and evaluation.* PhD dissertation, University of Michigan, 1993.

6. Gong, R., & Kieras, D. (1994). A Validation of the GOMS Model Methodology in the Development of a Specialized, Commercial Software Application. In *Proceedings of CHI,* 1994, Boston, MA, USA, April 24-28, 1994). New York: ACM, pp. 351-357.

7. Gould, J. D. (1988). How to design usable systems. In M. Helander (Ed.), *Handbook of human-computer interaction*. Amsterdam: North-Holland. 757-789.

8. John, B. E. & Kieras, D. E. (1994) The GOMS family of analysis techniques: Tools for design and evaluation. Carnegie Mellon University School of Computer Science Technical Report No. CMU-CS-94-181. Also appears as the Human-Computer Interaction Institute Technical Report No. CMU-HCII-94-106.

9. 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 Elsevier.

10. Kieras, D. (1994). GOMS Modeling of User Interfaces using NGOMSL. Tutorial Notes, CHI'94 Conference on Human Factors in Computer Systems, Boston, MA, April 24-28, 1994.

11. Kieras, D. E., & Bovair, S. (1984). The role of a mental model in learning to operate a device. *Cognitive Science,* 8, 255-273.

12. Kieras, D.E., & Polson, P. (1983). A generalized transition network representation for interactive systems. In *Proceedings of CHI '83 Human Factors in Computing Systems* (Boston, Dec. 13-15, 1983), ACM, New York, 103-106.

13. 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.

14. Kieras, D., Wood, S., & Meyer, D. (1995). Predictive Engineering Models Using the EPIC Architecture for a High-Performance Task. In *Proceedings of CHI'95 Human Factors in Computing Systems* (Denver, May 7-11, 1995), New York: ACM.

15. Kirwan, B., & Ainsworth, L. K. (1992). *A guide to task analysis.* London: Taylor and Francis.

16. Lewis, C. & Rieman, J. (1994) *Task-centered user interface design: A practical introduction.* Shareware book available at ftp.cs.colorado.edu/pub/cs/distribs/clewis/HCI-Design-Book

17. Nielsen, J. & Mack, R.L. (Eds). *Usability inspection methods.* New York: Wiley, 1994.

18. Wood, S. (1993). Issues in the implementation of a GOMS-model design tool. Unpublished report, University of Michigan.