The Potential for Modeling Human-Robot Interaction with GOMS

Jill L. Drury[1], Jean Scholtz[2] and David Kieras[3]
[1]The MITRE Corporation, [2]Pacific Northwest National Laboratory,
[3]The University of Michigan
USA

1. Introduction

Human-robot interaction (HRI) has been maturing in tandem with robots' commercial success. In the last few years HRI researchers have been adopting—and sometimes adapting—human-computer interaction (HCI) evaluation techniques to assess the efficiency and intuitiveness of HRI designs. For example, Adams (2005) used Goal Directed Task Analysis to determine the interaction needs of officers from the Nashville Metro Police Bomb Squad. Scholtz et al. (2004) used Endsley's (1988) Situation Awareness Global Assessment Technique to determine robotic vehicle supervisors' awareness of when vehicles were in trouble and thus required closer monitoring or intervention. Yanco and Drury (2004) employed usability testing to determine (among other things) how well a search-and-rescue interface supported use by first responders. One set of HCI tools that has so far seen little exploration in the HRI domain, however, is the class of modeling and evaluation techniques known as formal methods.

1.1 Difficulties of user testing in HRI

It would be valuable to develop formal methods for use in evaluating HRI because empirical testing in the robotics domain is extremely difficult and expensive for at least seven reasons. First, the state of the art in robotic technology is that these machines are often unique or customized, and difficult to maintain and use, making the logistics of conventional user testing difficult and expensive. Second, if the evaluation is being performed to compare two different user interfaces, both interfaces must be implemented and ported to the robots, which is a significant task. Testing may involve using two robots, and even if they have identical sensors and operating systems, small mechanical differences often result in different handling conditions, especially when using prototyped research platforms. Thus there are serious problems even with preparing the robot systems to be tested.

Third, compared to typical computer usability tests, the task environments used to perform robot usability testing are complex physical environments, difficult to both devise and actually set up. Together with the fairly long time required to perform the tasks in a single situation, the result is that the cost per test user and trial is very high. Fourth, robots are mobile and whether the tasks being evaluated involve moving the entire robot or just moving parts of the robot such as a gripper, the probability that any two users will follow exactly the same path of movement through the test environment is extremely small. This lack of uniform use of the robots makes comparison between users difficult, especially when the robots get stuck on obstacles or hit structures, seriously disrupting the evaluation. Fifth, if the testing is done outside (necessary for larger robotic vehicles) the same environmental conditions (lighting, rain, snow, etc.) cannot be guaranteed for all participants in the evaluations.

Sixth, training to operate robots is slow and costly; to compare two different interfaces, users need to have approximately the same level of skill in operating the robots. This may take considerable practice time as well as a means of assessing the acquired skills. Seventh, safety issues are always a concern and additional personnel are needed to ensure that no robots, people, or facilities are harmed during the tests, further increasing the cost.

Given the above challenges, it is clear that it is beneficial to obtain as much usability information as possible using methods that do not involve actual user testing. Obviously user testing will be necessary before the user interface can be declared successful, but it will be much less costly if formal methods can be employed prior to user testing to identify at least a subset of usability problems.

1.2 The potential of GOMS models

Perhaps the most widely-used of the formal methods is the Goals, Operations, Methods, and Selection rules (GOMS) technique first presented by Card, Moran, and Newell (1983), and which then developed into several different forms, summarized by John and Kieras (1996a, 1996b). Depending upon the type of GOMS technique employed, GOMS models can predict the time needed for a user to learn and use an interface as well as the level of internal consistency achieved by the interface. GOMS has proven its utility because models can be developed relatively early in the design process when it is cheaper to make changes to the interface. Analysts can use GOMS to evaluate paper prototypes' efficiency, learnability, and consistency early enough to affect the design prior to its implementation in software. GOMS is also used with mature software to determine the most likely candidates for improvement in the next version. Since GOMS does not require participation from end users, it can be accomplished on shorter time scales and with less expense than usability tests. Based on its use as a cost savings tool, GOMS is an important HCI technique: one that bears exploration for HRI.

Very little work has been done so far in the HRI domain using GOMS. A method we used for coding HRI interaction in an earlier study (Yanco et al., 2004) was inspired by GOMS but did not actually employ GOMS. Rosenblatt and Vera (1995) used GOMS for an intelligent agent. Wagner et al. (2006) used GOMS in an HRI study but did so in limited scenarios that did not explore many of the issues specific to HRI. Kaber et al. (2006) used GOMS to model the use of a tele-operated micro-rover (ground-based robot). In an earlier paper (Drury et al., 2007), we explored more types of GOMS than was presented in either Kaber et al. or Wagner et al. within the context of modeling a single interface.

1.3 Purpose and organization of this chapter

This chapter describes a comparison of two interfaces using a Natural GOMS Language (NGOMSL) model and provides a more detailed discussion of GOMS issues for HRI than has been previously published. At this point, we have not conducted a complete analysis of an HRI system with GOMS, which would include estimating some of the parameters involved. Rather, our results thus far are in the form of guidance for using GOMS in future HRI

modeling and evaluation efforts. The primary contribution of this chapter is an illustration of the potential of this approach, which we think justifies further research and application.
The next section discusses GOMS and what is different about using GOMS for HRI versus other computer-based applications. Section 3 contains guidance for adapting GOMS for HRI. We present background information on the two interfaces that we have modeled in Section 4, prior to presenting representative portions of the models in Section 5. Finally, we provide a summary in Section 6 and conclusions and thoughts for future work in Section 7.

2. Why is HRI different with respect to GOMS?

Before discussing the nuances of using GOMS for HRI, we briefly describe GOMS. There is a large literature on GOMS and we encourage the reader who is unfamiliar with GOMS to consult the overviews by John and Kieras (1996a, 1996b).

2.1 The GOMS Family

GOMS is a "family" of four widely-accepted techniques: Card, Moran, and Newell-GOMS (CMN-GOMS), Keystroke Level Model (KLM), Natural GOMS Language (NGOMSL), and Cognitive, Perceptual, and Motor GOMS (CPM-GOMS). John and Kieras (1996a) summarized the four different types of GOMS:

- CMN-GOMS: The original formulation was a loosely defined demonstration of how to express a goal and subgoals in a hierarchy, methods and operators, and how to formulate selection rules.
- KLM: A simplified version of CMN was called the Keystroke-Level Model and uses only keystroke operators — no goals, methods, or selection rules. The modeler simply lists the keystrokes and mouse movements a user must perform to accomplish a task and then uses a few simple heuristics to place "mental operators."
- NGOMSL: A more rigorously defined version of GOMS called NGOMSL (Kieras, 1997) presents a procedure for identifying all the GOMS components, expressed in structured natural language with in a form similar to an ordinary computer programming language. A formalized machine-executable version, GOMSL, has been developed and used in modeling (see Kieras and Knudsen, 2006).
- CPM-GOMS: A parallel-activity version called CPM-GOMS (John, 1990) uses cognitive, perceptual, and motor operators in a critical-path method schedule chart (PERT chart) to show how activities can be performed in parallel.

We used the latter three types of GOMS in Drury et al. (2007). In this chapter we use NGOMSL only, because it emphasizes the interface procedures and their structure. We discuss our selection of NGOMSL in more detail in Section 3.2.

2.2 HRI challenges for GOMS

A first challenge is that traditional GOMS assumes error-free operation on the part of the user and predictable and consistent operation on the part of the computer. The human-error-free assumption for GOMS is often misunderstood, and so needs some discussion. In theoretical terms, one could write GOMS models that describe how users deal with their errors (Card et al., 1983; Wood, 1999, 2000; Kieras, 2005) and even use GOMS to help predict where errors could take place (Wood, 1999; Wood and Kieras, 2002). The reason why GOMS modeling of human error is not routinely done is that (1) the techniques need to be further developed, and this requires as a foundation a better theory of human error than is currently available; and (2) in many computer user interface design situations, making the interface easy to learn and easy to use (which can already be addressed with GOMS) "automatically" reduces the likelihood of human error, making it less critical to deal with. Safety-critical domains are the obvious exception; clearly, developing techniques for modeling human error should be an area of intensive future research.

However, the second assumption, that of predictable and consistent computer behavior, is much more important in the HRI domain. Even when operated without autonomy, in the search-and-rescue (SAR) domain robots can behave in unexpected ways. In addition, the HRI task environment is such that the user cannot easily predict what the situation will be, or what effects trying to interact with that environment will have. The fact that the user cannot predict the state of the robot or environment in the near future means that models must account for a great range and flexibility of users' responses to any given situation, and the application of the models must be done in a way that takes the great variability of the situation into account. Later in this chapter we illustrate one way of handling this situation: user activity can be segmented into phases and actions whose methods and their predicted execution time can be represented in GOMS, leaving the probability or frequencies of these activities, which GOMS cannot predict, to be dealt with separately.

A second challenge for HRI pertains to the seemingly simple task of maneuvering the robot, which normally occurs with a control device such as a joystick. While GOMS has long modeled the use of pointing devices to move cursors or select different items on the computer display, it has not developed mechanisms to model the types of movements users would employ to continuously or semi-continuously direct a robot's movement with a joystick. This missing element is important because there are fundamental differences in the amounts of time that are spent moving a cursor with a pointing device versus pushing a joystick to steer a robot's motion, and there are likely to be fundamental differences in the cognitive mechanisms involved.

Wagner et al. (2006) applied GOMS to HRI to model mission plan generation but does not include this basic task of driving a robot. GOMS has been used frequently in the aviation domain and so we scoured the literature to find an analogous case, for example when the pilot pulls back on the rudder to change an aircraft's altitude. To our surprise, we found only analyses such as Irving et al. (1994) and Campbell (2002), which concentrated on interactions with the Flight Management Computer and Primary Flight Display, respectively: interactions confined to pushing buttons and verifying the computers' responses.

Kaber et al. (2006) attempted to account for driving times using a simple GOMS model that assumed, in essence, that controlling the robot motion was directly analogous to the computer interface task of finding an object on the screen and pointing to it with a mouse. The resulting model seriously overpredicted driving task times, which is consistent with the possibility that driving tasks are processed quite differently from interface pointing tasks. How to deal with this problem is one of the issues addressed in this chapter.

A third challenge relates to modeling mental operations to incorporate the right amounts of time for the users' thought processes at each stage of using an interface. For example, previous empirical work has shown that it takes a user an average of 1.35 seconds to mentally prepare to perform the next action when executing a routine task in a predictable environment (John and Kieras, 1996b). But robot operations are notoriously non-routine and unpredictable, as discussed above. Luckily, GOMS has always assumed that application-specific mental operators could be defined as necessary: what is difficult is determining the mental operators that make sense for HRI.

A fourth challenge is that the mental and perceptual operators in GOMS do not account for the effects of having varying qualities of sensor data, either within the same system at different times or on multiple systems that are being compared. For example, if video quality is bad on one system but exceptionally clear on another, it will take more time to extract video-based information via the first system's interface than from the second's interface. Each GOMS operator is normally assigned a single value as its typical time duration, such as the 1.35 seconds cited above for mental preparation. Unless a perceptual operator is assigned one time value in the model for the first system and a shorter time value for the second model (to continue the example), the models will not take into account an important difference that affects performance.

A fifth challenge pertains to different levels of autonomy. We believe it would be very useful, for example, for GOMS models to tell us whether it is more efficient for the robot to prevent the user from getting too close to objects, as opposed to requiring the user to spend time and effort watching out for obstacles immediately around the robot.

As we present our adaptations to GOMS and our example models in the following sections, we provide guidance for overcoming these challenges.

3. Adapting GOMS to HRI

3.1 Procedures vs. perception

The design process for HRI breaks naturally into two major parts: the perceptual content of the displays and the overall procedural operation.

Designers need to define the perceptual content of the displays so that they can be easily comprehended and used for HRI tasks. This design challenge will normally need to be accomplished using traditional interface design wisdom combined with evaluation via user testing; GOMS does not address whether one visual presentation of an item of information is easier to interpret than another. Rather, GOMS assigns a "mental operator" to the part of the task that involves interpreting display information, but this does not shed any light on whether one presentation facilitates users extracting information more quickly than another presentation—the standard mental operator is a simple "one size fits all" estimate. If modelers can define different types of domain- or task-specific mental operators for displays, then competing designs can be examined to see which requires more instances of one type of mental operator versus another. If these operator durations can be measured empirically, then the GOMS model can make a more accurate quantitative contribution.

GOMS can clearly help with the procedural implications of display design. For example, perhaps one design requires actions to toggle the display between two types of information, while another design makes them simultaneously visible; GOMS would highlight this difference. Designers also need to define the overall operation of the user interface in terms of the procedures that users would follow to carry out the task using the interface. Evaluating the procedural design challenge can be done easily and well with GOMS models if the perceptual design challenge can be handled so as not to confound any comparisons that modelers might make between competing designs.

This brings us to our first guideline for modeling HRI using GOMS:

1. Don't get bogged down in modeling the perceptual content of the displays; focus on the procedures instead.

The modeler should focus instead on the step-by-step procedures used when interacting with the interface, keeping in mind that issues in perception might determine and dominate issues regarding procedures. Getting bogged down in the perceptual issues is tempting because this part of the task is obviously important, but current modeling technology doesn't allow modelers to make much progress *a priori*. Many tasks demand that the operator view video or dynamic sensor data. They may need to do this multiple times in order to understand the situation. There is no way to predict the number of times a user may consult a display because doing so is situation-dependent and also dependent upon environmental conditions (such as complexity of the scene and video quality), skill levels, and physical capabilities of the users (eyesight acuity, dexterity, etc.). Even if we could know how many times users would need to refer to a particular part of the displays, it is difficult to assign accurate times to the actions.

GOMS modeling can be used to characterize a single interface, but it becomes much more useful when comparing two interface designs. The difficult-to-model aspects such as perceptual processes can often be held constant between the two designs, enabling the models to pinpoint the procedural differences and highlight their consequences. When improving designs incrementally, however, a modeler can model a single interface to the point where it exposes inconsistencies or inefficiencies that can become the focus of suggested design improvements. The improved design can then be compared to the first design using GOMS to identify the degree of improvement attained.

3.2 Choice of GOMS technique

We present our models using NGOMSL because this form of GOMS is easy to read and understand while still having a relatively high level of expressive power. A further advantage of NGOMSL is that it can be converted relatively easily into the fully executable version, GOMSL notation (see Kieras, 2005; Kieras and Knudsen, 2006). NGOMSL can be thought of as stylized, structured pseudocode. The modeler starts with the highest level goals, then breaks the task into subgoals. Each subgoal is addressed in its own method, which may involve breaking the task further into more detailed subgoals that also are described in their own methods. The lowest-level methods contain mostly primitive operations. Design consistency can be inferred by how often "basic" methods are re-used by other methods. Similarly, efficiency is gained when often-used methods consist of only a few steps. The number of methods and steps is proportional to the predicted learning time. Because NGOMSL lacks the ability to describe actions that the user takes simultaneously, we adopt a bit of syntax from GOMSL, the keyword phrase "Also accomplish goal…", when we need to show that two goals are being satisfied at the same time.

Since all detailed GOMS models include "primitive operators" that each describe a single, atomic action such as a keypress, we discuss primitives next.

3.3 Primitives

At the lowest level of detail, GOMS models decompose a task into sequences of steps consisting of *operators*, which are either motor actions (e.g., home hands on the keyboard) or cognitive activities (e.g., mentally prepare to do an action). As summarized by John and

Kieras (1996a, 1996b), the following primitive operators are each denoted by a one-letter code and their standard time duration:

K	to press a key or button (0.28 seconds for average user)
B	to press a button under the finger (e.g. a mouse button) (0.1 seconds)
M	perform a typical mental action, such as finding an object on the display, or mentally prepare to do an action (1.35 seconds)
P	to point to a target on a display (1.1 seconds)
H	to home hands on a keyboard or other device (0.4 seconds)
W	to represent the system response time during which the user has to wait for the system (variable)

As an example of the **W** operator, the system associated with Interface A (described below) has a delay of about 0.5 seconds before the user can see a response from steering the robot; the value for Interface B (also described below) was 0.25 seconds. This time difference was noticed and commented on by users and so needs to be reflected in the models' timing calculations.

As discussed above, none of these primitives are especially suited to describing manipulating the robot; thus we define a "steer" operator **S** and introduce our second guideline:

2. Consider defining and then assigning a time duration to a robot manipulation operator that is based on typical values for how long the combination of the input devices, robot mechanics, and communications medium (especially for remote operations) take to move the robot a "reference" distance.

This guideline is based on the fact that the time required to manipulate the robot is driven more by the robot mechanics and environment than by the time needed by the human to physically manipulate the steering input device. The ultimately skilled user would perform all perceptual, navigation, and obstacle avoidance subtasks while keeping the robot moving at a constant speed, thus making execution time equal to the time it takes to cover an area at a given speed.

We assigned a reference **S** time of 1 foot/second to the interactions with the two robots modeled in this chapter. In accordance with our guidance, we observed operations and determined that the mechanics of the robot were the dominant factor in determining how quickly, on average, a user would steer the robot to accomplish moving one foot. This is necessarily a somewhat crude approach to assigning a time because the robots could run at various speeds which changed based on lighting conditions, proximity to obstacles, and users deciding to speed up or slow down. When two interfaces are being examined, however, using a single representative speed for each robot should not harm the comparison of their respective GOMS models.

While all the other "standard" primitive operators apply to HRI, the **M** operator requires close scrutiny. As used in modeling typical computer interfaces, **M** represents several kinds of routine bits of cognitive activity, such a finding a certain icon on the screen, recalling a file name, making a routine decision, or verifying that a command has had the expected result. Clearly, using the same operator and estimated time duration for these different actions is a gross simplification, but it has proven to be useful in practice (see John and Kieras, 1996a, 1996b for more discussion). However, consider data being sent to the user from a remote mobile robot that must be continually perceived and comprehended (levels 1 and 2 of Endsley's (1988) definition of situation awareness). This is a type of mental process that is used to assess the need for further action triggered by external, dynamic changes reflected in the interface (e.g. as seen in a changing video display). Intuitively, this mental process seems qualitatively different, more complex, and more time-consuming from those traditionally represented with **M**. Since this type of mental operation is nontrivial, we propose representing it as a separate operator, and then the analysis can determine if one design versus another requires more instances of this type of mental assessment. For example, if one interface splits up key sensor information on separate screens but another provides them on a fused display, the latter design would require fewer mental operators in general and fewer operators of the type that assesses dynamic changes in the environment. This leads us to an additional guideline:

3. Without violating guideline number 1, consider defining HRI-specific mental operator(s) to aid in comparing the numbers of instances these operators would be invoked by competing designs.

We define a **C** (Comprehend) operator to refer to a process of understanding and synthesizing complex display information that feeds into the user's continually-changing formulation of courses of action. This operator is expected to take longer than the conventional **M** operator, and will be used to represent the interpretation of the complex displays in this domain.

Once the set of primitives has been finalized, the next step is to assign times to the various operators unique to the HRI domain. To a certain extent, the exact times are not as important as the relative differences in times that result from competing interface designs. The time required for our mental operator **C** for robotics tasks will depend on the quality of the video, the complexity of the situation being viewed, and the design of the map and proximity sensor displays. Thus, if two systems being compared have radically different qualities of sensor data, we suggest the following guideline:

4. Without violating guideline number 1, consider assigning time duration values to HRI-specific mental operators that reflect the consequences of large differences in sensor data presentation.

In the absence of detailed empirical data specific to the system and conditions being modeled, we estimate the time required by the **C** operator to be 2 seconds. This estimate was derived based on observing search-and-rescue operators working with each system in remote operations. Again, this is a crude estimate that varied substantially among the users. In the future, eye tracking may be a possible means of determining how long users gazed at particular part of the display.

4. Example interfaces

This next section presents some sample results from applying GOMS as adapted to HRI. But first, we need to describe the interfaces we modeled. User interface analysts use GOMS to model human activity assuming a specific application interface because the models will be different for each application. Our decision regarding which interfaces to analyze was not important as long as the chosen interfaces contained representative complexity and functionality. We chose two interfaces, illustrated in Figures 1 and 2, that use the same basic set of urban search-and-rescue (USAR) functionality and the same robotic platform.

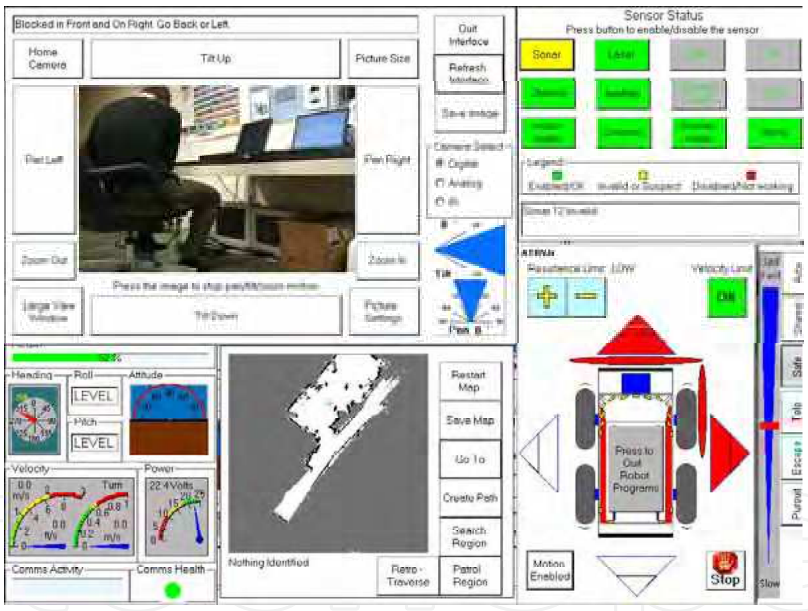
4.1 Interface "A"

The architecture for the system underlying Interface A was designed to be flexible so that the same interface could be used with multiple robot platforms. We observed the interface operating most frequently with an iRobot ATRV-Jr.: the same one used for Interface B.

Interface A (Figure 1) was displayed on a touch screen. The upper left corner of the interface contained the video feed from the robot. Tapping the sides of the window moved the video camera left, right, up or down. Tapping the center of the window re-centered the camera. Immediately to the right of the video display were pan and tilt indicators. The robot was equipped with two types of cameras that the user could switch between: a color video camera and a thermal camera; the camera selection radio buttons were also to the right of the video area.

The lower left corner contained a window displaying health status information such as battery level, heading, and attitude of the robot. A robot-generated map was placed in the lower central area. In the lower right corner, there was a sensor map that showed red arrows to indicate directions in which the robot's motion was blocked by obstacles.

The robot was controlled through a combination of a joystick and the touch screen. To the right of the sensor map in the bottom right hand corner of the touch screen, there were six mode buttons, ranging from autonomous to tele-operation. Typically, the user touched one of the mode buttons, then used the joystick to steer the robot if not in the fully autonomous mode.


Blocked in Front and On Right. Go Back or Left

Nothing Identified

Figure 1. Interface A, one of the two example interfaces that we analyzed

When the user wished to take a closer look at something, he or she touched the video window to pan the camera. For victim identification, the user often switched to the thermal or infrared (IR) sensor (displayed in the same space as the videostream and accessed via a toggle) to sense the presence of a warm body.

The proximity sensors were shown around a depiction of the robot in the bottom right hand side of the display. The triangles turned red to indicate obstacles close to the robot. The small,

outer triangle turned red when the robot first approached objects, then the larger, inner triangle also turned red when the robot moved even closer to an obstacle. The location of the red triangles indicated whether the blockage was to the front, rear, and/or sides.
Note that System A's interface did not incorporate menus. Visual reminders for all possible actions were present in the interface in the form of labels on the touch screen.
While the organization that developed System A has explored other interface approaches since this version, users access almost the same functionality with all interface designs to date. Also, many other USAR robots incorporate similar functionality.

4.2 Interface "B"

The ATRV-Jr. robot used with Interface B was modified to include a rear-facing as well as forward-facing camera. Accordingly, the interface had two fixed video windows (see Figure 2). The larger one displayed the currently selected camera (either front- or rear-facing); the smaller one showed the other video feed and was mirrored to emulate a car's rear-view mirror.
Interface B placed a map at the edge of the screen. The map window could be toggled to show a view of the current laser readings, removing the map from the screen during that time.

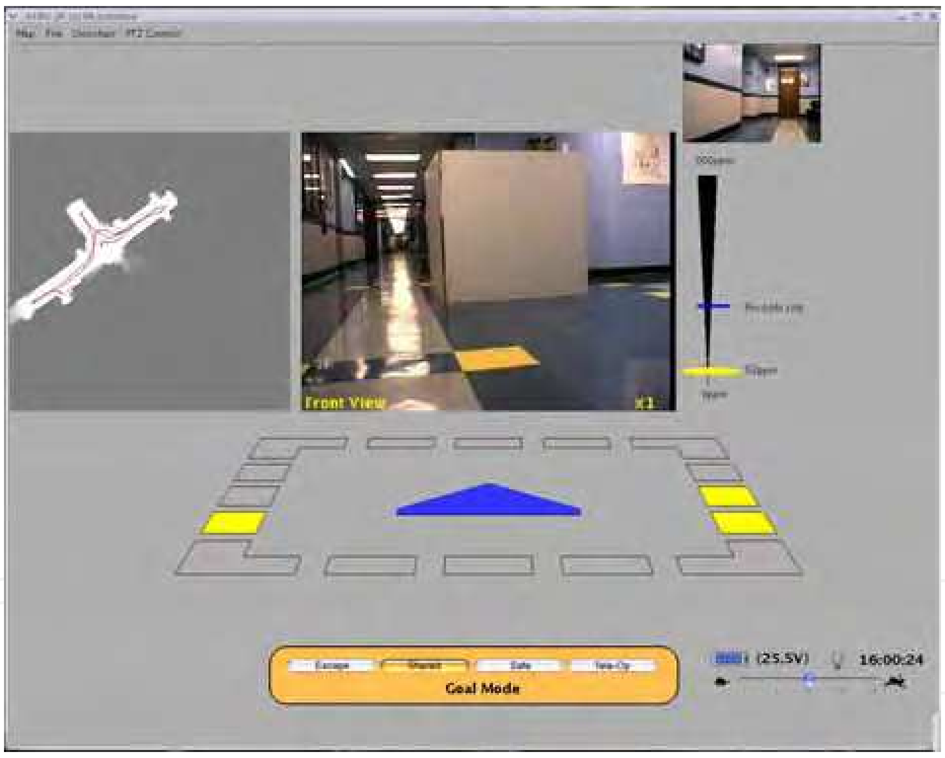


Figure 2. Interface B

Information from the sonar sensors and the laser rangefinder was displayed in the range data panel located directly under the main video panel. When nothing was near the robot, the color of the box was the same gray as the background of the interface, indicating that nothing was there. As the robot approached an obstacle at a one foot distance, the box turned to yellow, and then red when the robot was very close (less than half a foot away). The ring was drawn in a perspective view, which made it look like a trapezoid. This perspective view was designed to give the user the sensation that they were sitting directly behind the robot. If the user panned the camera left or right, this ring rotated opposite the direction of the pan. If, for example, the front left corner turned red, the user could pan the camera left to see the obstacle, the ring would then rotate right, so that the red box would line up with the video showing the obstacle sensed by the range sensors. The blue triangle, in the middle of the range data panel, indicated the true front of the robot.

The carbon dioxide meter to the right of the primary video screen showed a scale in parts-per-million (PPM) and also indicated the level at which "possible life" was detected as a blue line. (The platform used for Interface A had an IR sensor to serve this same purpose.) The bottom right hand corner showed battery life, whether the light on the front of the robot was illuminated, a clock, and the maximum speed. The level of autonomy was shown in the bottom right hand set of buttons (Shared/Goal Mode is illustrated in the figure).

4.3 Tasks Analyzed

We analyzed tasks that are typical of a search-and-rescue operation: maneuver the robot around an unfamiliar space that is remote from the user, find a potential victim, and confirm the presence of a victim.

5. Example modeling results

Given the novelty of the adaptations, at this point we can present only illustrations of how GOMS can be applied to answer questions about HRI designs. Clearly additional research will be needed to provide a complete assessment of the accuracy and value of these adaptations and any required corrections to them.

In Section 2.2 we pointed out the need to deal with the flexibility and unpredictability of the HRI domain. Now we illustrate a simple approach to this problem: the user's activity is decomposed into segments corresponding to GOMS methods. Often, the activity within a method, and the time to perform it, is fairly well-defined. In many cases, what is unpredictable is simply how likely or how frequently that activity will be required. As long as the likelihood or frequency of an activity is similar for the two interfaces, useful comparisons can then be made from the models.

5.1 Top level model

A major part of creating a model for a task is to characterize the top level of the task. Figure 3 contains a fragment from a preliminary model for the top level of the robot search-and-rescue task. Due to space reasons, we cannot show all of the methods, so we only show those methods that lead to the user determining whether she has spotted a victim. This "thread" of methods is shown in the figure by the bold-face goals.

This top-level model shows the overall assumed task structure. After getting some initial navigation information, the user repeatedly chooses an area to search until all areas have been covered. Each area involves driving around to different locations in that area and looking for victims there. Locating a victim involves repeatedly choosing an area to view, viewing it, and then checking the sensors to see if a victim is present. This last goal will be examined in more detail in the next subsection.

```
Method for goal:  **Perform search and rescue mission**
1.    Accomplish goal: obtain global navigation information.
2.    Choose next local area.
3.    If no more local areas, return with goal accomplished.
4.    Accomplish goal: search local area.
5.    Go to 2.
Method for goal:  search local area
1.    Accomplish goal:  drive to new location.
The following step applies 70% of the time.
2.    Also accomplish goal:  locate victim.
3.    Return with goal accomplished.
Method for goal:  locate victim
1.    Choose next area of location.
2.    If no more areas, return with goal accomplished.
3.    Accomplish goal: view area of location.
4.    Accomplish goal:  view sensors for indication of victim.
5.    If indication shown, return with goal accomplished.
6.    Go to 1.
```

Figure 3. Top-level methods for search-and-rescue

The top-level method focuses attention on a basic issue in the task, namely the extent to which the user can simultaneously drive the robot to cover the area, and locate a victim using the video and sensors. Both interfaces seem to be compatible with simultaneous operation, compared to some other interface that, for example, used the same joystick for both camera motion control and driving. The method shows this simultaneity assumption with the use of the "Also accomplish goal" operator. However, Yanco and Drury (2004) observed that users were able to drive and look for victims simultaneously only about 70% of the time, and often had to pause to reorient themselves. Currently, GOMS lacks a direct way to express this sort of variability, so we have commented this step in the method as a place-holder.

5.2 Comparing Interfaces A and B

In addition to showing the overall flow of control, the top-level model acts to scope and provide a context for the more detailed modeling, such as the consideration of how different displays might support the goal of viewing sensors for indication of a victim. This is illustrated by critiquing the methods for this goal supplied by Interface A, and then comparing them to Interface B. Note that because the two sensors are different for the two platforms we are comparing the procedure necessary for viewing a thermal (infrared) sensor as illustrated in Interface A with the procedure for viewing a carbon dioxide sensor as illustrated in Interface B.

The method for Interface A is shown in Figure 4A. The method shows that the display must be toggled between the normal video display and the infrared display. Since it may be done frequently, the time cost of this operation could be significant. While the GOMS model cannot predict how often it would be done, the preliminary estimate is that it would take about 2.2 seconds per toggling (two **P**, or Point, operators). Clearly this aspect of the design could use improvement. Hestand and Yanco (2004) are experimenting with a USAR interface that places infrared data on top of video data. While research is needed to determine if it takes longer for a user to comprehend combined video/infrared data, this model provides a design target: since it takes about 2.2 seconds to toggle the displays in the uncombined form, in order to do better, the combined display should not take any longer to comprehend.

In addition, the infrared display is color-coded in terms of temperature, and there is no on-screen cue about the color that indicates possible life, suggesting that the user will need to perform extra mental work. We have represented this as a step to recall the relevant color code.

In contrast, Interface B shows a different approach for another sensor datum, carbon dioxide level. The method is shown in Figure 4B. There is an on-screen indication of the relevant level, requiring a simple visual position judgment rather than a comparison to a memorized color.

Method for goal: **view sensors for indication of victim**

1. Look at and interpret camera display (**C**).

Using a touchscreen is similar to pointing with a mouse.

2. Point to touchscreen IR button to toggle display (**P**).
3. Recall IR display color-code that indicates possible life (**M**).
4. Look at and interpret IR display (**C**).
5. Decide if victim is present (**M**).

Need to restore display to normal video to support next activity

6. Point to touchscreen Digital button to toggle display (**P**).
7. Return with goal accomplished.

Figure 4A. Fragment of a GOMS model for Interface A

Method for goal: **view sensors for indication of victim**

1. Look at and interpret camera display (**C**).
2. Look at and determine whether carbon dioxide level is above "Possible life" line (**M**).
3. Decide if victim is present (**M**).
4. Return with goal accomplished.

Figure 4B. Fragment of a GOMS model for Inteface B

Interface B's method is shorter than Interface A's for several reasons. Interface A cannot show video and infrared sensor information (to show the presence of body heat) at the same time, incurring costs to switch between them, whereas Interface B can show video and carbon dioxide (present in humans' exhalations) sensor readings simultaneously. Also, Interface B explicitly shows the level above which the presence of nearby human life is likely, whereas users looking at Interface A will need to remember which color-coding in the infrared display indicates heat equivalent to human body temperature. This difference in approaches requires one less operator (to recall the appropriate color) as well as changes the nature of the mental operator (from a **C** to an **M** indicating a simple comparison). For one pass through the method, Interface A requires 2 more steps, two **P** operators, and an additional **C** operator. If we use the estimates for **C**, **P**, and **M** previously discussed, Interface A would require 8.9 seconds for this fragment versus 4.7 seconds for Interface B.

Even though GOMS cannot predict the interpretability of display elements, this example shows that the costs and benefits of different design decisions can be modeled, and even quantified, to some extent. Design decisions must balance the effectiveness of providing different sensors with the work they require on the part of users in order to benefit from having those sensors. If a number of sensors are present and are helpful, then procedures for viewing all of the sensors and comprehending the information can be modeled and thus account for the time tradeoffs involved.

5.3 Modeling different levels of autonomy

Previously we have stated our contention that GOMS would be useful for showing the impact of differing autonomy levels on the user's efficiency. We illustrate this point by showing the difference in workload between extricating a robot when in tele-operation mode versus in escape mode. In tele-operation mode, the user directs all of the robots' actions in a complete absence of autonomy. In contrast, once the user puts the robot into escape mode, the robot itself figures out how to move away from all obstacles in the immediate environment and then, once clear of all obstacles, stops to await further commands from the user. Our experience is that robots become wedged into tight quarters surprisingly often, which motivated the development of the escape mode.

Figure 5 illustrates the portion of the GOMS model that pertains to getting a robot "unstuck": the unenviable condition where it has few options in how it can move. Figure 5 pertains to Interface A, but the model for Interface B is similar (only a few less steps).

Note that Figure 5 employs an informal means of passing a variable to a method. We denote the passing of a variable by a phrase in square brackets.

Method for goal: get unstuck when tele-operating

1. Accomplish goal: determine direction to move
2. Accomplish goal: drive to new location
3. Accomplish goal: check-movement–related sensors
4. Return with goal accomplished.

Method for goal: determine direction to move

1. Look at and interpret proximity display (**C**)
2. Accomplish goal: move camera in direction of obstacle
3. Return with goal accomplished

Method for goal: move camera [movement direction]

1. Point to touchscreen button for [movement direction] (**P**)
2. Look at and interpret video window (**C**)
3. Decide: if new movement direction is needed (**C**), go to 1.
4. Return with goal accomplished.

Method for goal: drive to new location

1. If hands are not already on joystick, home hands on joystick (**H**)
2. If movement direction not yet known, Accomplish goal: determine direction to move
3. Initiate joystick movement (**W**)
4. Move joystick until new location is reached or until stuck (**S**)
5. If stuck, then accomplish goal: get unstuck when tele-operating
6. Return with goal accomplished.

Method for goal: check movement-related sensors

1. Look at and interpret video window (**C**)
2. Look at and interpret map data window (**C**)
3. Look at and interpret sonar data window (**C**)
4. Return with goal accomplished.

Figure 5. Model Fragment for Tele-Operating Stuck Robots

As might be expected, getting a robot unstuck can be a tedious process. Not counting the shared method for checking movement-related sensors, there are 17 statements in the methods in Figure 5, and it often takes multiple iterations through to completely extricate a robot. Each iteration requires at least 6 **C** operators, a **P**, a **W**, and possible **H**, in addition to the robot movement time included in the **S** operator. These actions will require 15 seconds assuming the robot moves only a foot and only 6 **C** operators are needed: and all of this activity is attention-demanding.

Figure 6 shows the simple method for using an automony feature to extricate the robot. The user changes the autonomy mode from tele-operation to escape and then simply watches the robot use its sensors and artificial intelligence to move itself from a position almost completely blocked by obstacles to one that is largely free of obstacles so that the user can resume tele-operation. In contrast to the Figure 5 methods, the single method shown in Figure 6 lists only 5 statements. This method assumes that the user will do nothing but wait for the robot to finish, but clearly, the user could engage in other activity during this time, opening up other possibilities for conducting the task more effectively.

While this comparison might seem obvious, these models were simple to sketch out, and doing so is a very effective way to assess the possible value of new functionality. Using GOMS could save considerable effort over even simple prototype and test iterations (see Kieras, 2004 for more discussion).

Method for goal: get unstuck when using escape
1. Point to touchscreen button for escape (**P**).
2. Wait for robot to finish (**W**).

Same method called as in manual get unstuck method

3. Accomplish goal: check movement-related sensors
4. Decide: if robot still stuck, go to 1.
5. Return with goal accomplished.

Figure 6. Model fragment for extricating robots using escape mode

6. Summary

In this section we summarize the five primary GOMS-related HRI challenges and how we recommend addressing them. Specifically, using GOMS it is challenging to model:

A. Working with a difficult-to-predict environment or robot state.

The dynamic situations common with many robot applications require flexible modeling techniques. One way to deal with this challenge is to segment user activity into phases and methods whose times can be estimated, leaving the probability or frequencies of these activities to be addressed separately.

B. Maneuvering the robot.

We recommend using a new "steering" operator, **S**, and assigning a standard time to that operator that is characteristic for the robot platform (Guideline 2).

C. Describing users' mental operations as they synthesize complex display information.

While we do not feel it is useful to focus on modeling the perceptual content of displays (Guideline 1), it can be useful to define HRI-specific mental operators (Guideline 3). We recommend defining a mental operator **C** (Comprehend) to refer to understanding and synthesizing complex display information.

D. Accommodating the effects of having sensor data of various qualities.

We recommend assigning time duration values to the HRI-specific mental operators that reflect the different qualities of sensor data presentation (Guideline 4). Determining these time duration values is a topic for future work (see below).

E. Handling different levels of autonomy.

A complete model would need to include different methods that identify users' actions under the various levels of autonomy possible for a particular robotic system. More interestingly, GOMS provides an avenue for investigating the utility of potential new autonomy levels (see below).

7. Conclusions and future work

In this chapter we have shown how GOMS can be used to compare different interfaces for human-robot interaction. GOMS is useful in determining the user's workload, for example when introducing different displays for sensors.

GOMS models are also useful in determining the upper and lower time limits for different procedures. For example, checking movement-related sensors will depend on how many sensor windows the user has to look at and how many of these windows have to be manipulated to be viewed. This can be extremely helpful in deciding what should be visible at what time to maximize the user's efficiency.

GOMS can also be used to evaluate potential new autonomy modes. In our example we used the escape mode on Interface A to show how autonomy modes can be modeled. In actuality, the escape mode was originally incorporated into robots because it was clear to everyone that enabling this robotic behavior could save the user a lot of work and time. In other words, designers did not need a GOMS model to tell them that such functionality would be worthwhile. However, this example shows how GOMS can be used to model other autonomy modes to determine possible human-robot ratios. By examining the maximum expected times for different procedures, it is possible to use Olsen and Goodrich's equations on "fan out" (Olsen and Goodrich, 2003) to determine the upper bound on the number of robots that a single person can control simultaneously.

While GOMS was designed to model user behavior with a particular user interface, what it has done in the escape/tele-operation example is to render explicit the effect of both the robotic behavior and the user interface on the user. GOMS could be used in less obvious cases to "test drive" the effects that new robot behaviors, coupled with their interaction mechanisms, might have on users. To the extent that the effects of the interface design and robot behavior can be teased apart, GOMS can have utility in the process of designing new robot behaviors.

Future work might productively include experimentation with the effect of degraded video on the time necessary for users to perceive and comprehend information. Simulators could introduce a controlled amount of noise into the videostream in a repeatable fashion so that user tests could yield empirical data regarding average times for comprehending video data under various degraded conditions. This data could be codified into a set of "reference" video images. By comparing a system's video quality with the reference images, modelers could assign a reasonable estimate of video comprehension times *a priori*, without further empirical work.

Another future work area might be to examine whether it is useful to employ GOMS to model the robot's performance in interacting with the environment, other robots, and/or humans. Perhaps such an analysis would be desirable to help determine how many autonomous robots would be needed to complete a task under critical time limitations such as a large-scale rescue operation, for example. Such a modeling effort would likely uncover additional issues and would depend on the nature of the robot's environment, the behaviors that were programmed for the robot, and the mechanical limitations inherent in the robot platform.

As argued in the Introduction, the use of GOMS models for comparing alternative designs can be much less costly than conducting user testing, once time values for domain-specific operators have been estimated. Once our example model has been elaborated to cover all of the critical parts of the task, we feel it would be fairly simple to modify the model to examine a variety of different interface designs and robot functions. The effort needed for modeling can be contrasted with that required for user testing, which necessitates building robust versions of user interfaces, obtaining sufficiently trained users, and having robots that are in operating condition for the number of days needed to conduct the tests.

While open issues exist with applying GOMS models to HRI, we are confident that it will help us develop superior HRI systems more quickly and effectively.

8. Acknowledgments

This work was supported in part by NSF (IIS-0415224) and NIST. We would like to thank Douglas Few and David Bruemmer of Idaho National Laboratories; Elena Messina, Adam Jacoff, Brian Weiss, and Brian Antonishek of NIST; and Holly Yanco and Brenden Keyes of UMass Lowell.

9. References

Adams, J. A. (2005). Human-Robot Interaction Design: Understanding User Needs and Requirements. In *Proceedings of the 2005 Human Factors and Ergonomics Society 49th Annual Meeting*, 2005, Orlando, FL.

Campbell, C. B. (2002). Advanced integrated general aviation primary flight display user interface design, development, and assessment. In *Proceedings of the 21st Digital Avionics Systems Conference*, Vol. 2.

Card, S., Moran, T., and Newell, A. (1983). *The Psychology of Human-Computer Interaction.* Hillsdale, New Jersey: Erlbaum.

Drury, J. L. , Scholtz, J., and Kieras, D. (2007). Adapting GOMS to model human-robot interaction. In *Proceedings of the 2nd ACM Conference on Human-Robot Interaction (HRI 2007)*.

Endsley, M. R. (1988). Design and evaluation for situation awareness enhancement. In *Proceedings of the Human Factors Society 32nd Annual Meeting*, Santa Monica, CA, 1988.

Hestand, D. and Yanco, H. A. (2004). Layered sensor modalities for improved human-robot interaction. In *Proceedings of the IEEE Conference on Systems, Man and Cybernetics*, October 2004.

Irving, S., Polson, P., and Irving, J. E. (1994). A GOMS analysis of the advanced automated cockpit. In *Proceedings of the 1994 CHI conference on Human Factors in Computing Systems*, Boston, April 1994.

John, B. E. (1990). Extensions of GOMS analyses to expert performance requiring perception of dynamic visual and auditory information. In *Proceedings of the 1990 Conference on Human Factors in Computing Systems*. New York: ACM, pp. 107 – 115.

John, B. E. and Kieras, D. E. (1996a). Using GOMS for User Interface Design and Evaluation. *ACM Transactions on Human-Computer Interaction*, 3(4), December 1996.

John, B. E. and Kieras, D. E. (1996b). The GOMS Family of User Interface Analysis Techniques: Comparison and Contrast. *ACM Transactions on Human-Computer Interaction*, 3(4), December 1996.

Kaber, D.B., Wang, X., and Kim, S. (2006). Computational Cognitive Modeling of Operator Behavior in Telerover Navigation. In *Proceedings of the 2006 IEEE Conference on Systems, Man, and Cybernetics*, Oct. 8-11, Taipei, Taiwan.

Kieras, D. E. (1997). A Guide to GOMS model usability evaluation using NGOMSL. In M. Helander, T. Landauer, and P. Prabhu (Eds.), *Handbook of Human-Computer Interaction*. (Second Edition). Amsterdam: North-Holland. 733-766.

Kieras, D. E. (2004). Task analysis and the design of functionality. In A. Tucker (Ed.) *The Computer Science and Engineering Handbook* (2nd Ed). Boca Raton, CRC Inc. pp. 46-1 through 46-25.

Kieras, D. E. (2005). Fidelity Issues in Cognitive Architectures for HCI Modeling: Be Careful What You Wish For. In *Proceedings of the 11th International Conference on Human Computer Interaction (HCII 2005),* Las Vegas, July 22-27.

Kieras, D., and Knudsen, K. (2006). Comprehensive Computational GOMS Modeling with GLEAN. In *Proceedings of BRIMS 2006*, Baltimore, May 16-18.

Leveson, N. G. (1986). "Software safety: why, what and how." *ACM Computing Surveys* 18(2): 125 – 162, June 1986.

Olsen, D. R., Jr., and Goodrich, M. A. (2003). Metrics For Evaluating Human-Robot Interactions. In *Proceedings of PERMIS 2003*, September 2003.

Rosenblatt, J. and Vera, A. (1995). A GOMS representation of tasks for intelligent agents. In *Proceedings of the 1995 AAAI Spring Symposium on Representing Mental States and Mechanisms*. M. T. Cox and M. Freed (Eds.), Menlo Park, CA: AAAI Press.

Scholtz, J., Antonishek, B., and Young, J. (2004). Evaluation of a Human-Robot Interface: Development of a Situational Awareness Methodology. In *Proceedings of the 2004 Hawaii International Conference on System Sciences*.

Wagner, A. R., Endo, Y., Ulam, P., and Arkin, R. C. (2006). Multi-robot user interface modeling. In *Proceedings of the 8th International Symposium on Distributed Autonomous Robotic Systems*, Minneapolis, MN, July 2006.

Wood, S. D. (1999). The Application of GOMS to Error-Tolerant Design. Proceedings of the 17th International System Safety Conference, Orlando, FL.

Wood, S.D. (2000). Extending GOMS to human error and applying it to error-tolerant design. *Doctoral dissertation*, University of Michigan, Department of Electrical Engineering and Computer Science.

Wood, S. D. and Kieras, D. E. (2002). Modeling Human Error For Experimentation, Training, And Error-Tolerant Design. In *Proceedings of the Interservice/Industry Training, Simulation and Education Conference*. Orlando, Fl. November 28 – December 1.

Yanco, H. A. and Drury, J. L. (2004). Where Am I? Acquiring Situation Awareness Using a Remote Robot Platform. In *Proceedings of the 2004 IEEE Conference on Systems, Man, and Cybernetics*.

Yanco, H. A., Drury, J. L., and Scholtz, J. (2004). Beyond usability evaluation: analysis of human-robot interaction at a major robotics competition. *Human-Computer Interaction*, Vol. 19, No. 1 & 2, pp. 117 – 149.