Computational GOMS Modeling of a Complex Team Task: Lessons Learned

David E. Kieras
University of Michigan
Electrical Engineering & Computer Science Department
Ann Arbor, MI 48109-2110
kieras@eecs.umich.edu

Thomas P. Santoro
Naval Submarine Medical Research Laboratory
Human Performance Department
SUBASE BOX 900, Groton, CT 06349
santoro@ nsmrl.navy.mil

ABSTRACT
This paper presents the lessons learned when a computational GOMS modeling tool was used to evaluate user interface concepts and team structure designs for a new class of military shipboard workstations. The lessons are both encouraging and cautionary: For example, computational GOMS models scaled well to a large and complex task involving teams of users. Interruptability and working memory constructs had to be added to conventional GOMS model concepts. However, two surprises emerged: First, the non-psychological aspects of the model construction were the practical bottleneck. Second, user testing data in this domain were difficult to collect and lacked definition, meaning that the model provided a better characterization of the design details than the user testing data. Included in these lessons are recommendations for future model applications and modeling methodology development.

Categories & Subject Descriptors: H.5.2 [Information Interfaces and presentation]: User Interfaces — evaluation/methodology, theory and methods, GOMS
General Terms: Design, Human Factors
Keywords: human performance modeling

INTRODUCTION

GOMS models, introduced by Card, Moran, and Newell [5] are a way to characterize the procedural knowledge required to use a system. Explaining the acronym, to construct a GOMS model, one determines the user's *G*oals, lists what *O*perators can be executed in the interface, discovers the *M*ethods, which are sequences of operators that will accomplish the goals, and the *S*election rules that pick out which method to use to accomplish a goal when more than one applies. When the model is elaborated down to the keystroke level of detail where the operator execution times can be estimated with standard approximations, GOMS models yield quantitative predictions of human performance times. When the methods are written in a standardized form such as NGOMSL (Natural GOMS Language) [7, 8], their length and similarity can be counted and used to predict relative learning times. Thus GOMS model predictions can be used to evaluate these *procedural* aspects of usability of a computer interface design early in the development process. While GOMS models are useful only for tasks that involve substantial amounts of routine procedure execution, they can often enable interface designers to start evaluating usability and making design iterations before the investment in prototype development. Furthermore, once the initial model is constructed, it is usually very easy to determine the effects of variations on the design. Prototype construction and empirical user testing can then begin with more confidence that the initial implemented design is likely to be basically good. As argued in a survey of model-based evaluation techniques [10], being able to do some of the design iterations quickly and cheaply makes it possible to do more iterations, which enables a better design outcome.

As summarized by John and Kieras [5, 6], since the original Card et al. proposal, considerable progress has been made in developing this concept into several useful techniques, and connecting GOMS models with *cognitive architectures*, especially those in computational form such as ACT-R [1] and EPIC [11] (see [2] for an overview); such work is connecting the practically-oriented GOMS methodology with fundamental mechanisms of human cognition and performance. A step in this direction is GLEAN (GOMS Language Evaluation and ANalysis), a tool for constructing and running computational GOMS models; it has been under development for several years [14, 18], and is currently available for research purposes [9]. GLEAN provides an executable programming language for GOMS models, GOMSL (GOMS Language), that resembles a familiar programming language, making the models relatively easy for system developers or other non-specialists to construct. A sample of this notation appears later in this paper. The GOMSL program is interpreted and executed by a simplified computational cognitive

CHI 2004, April 24–29, 2004, Vienna, Austria.

architecture (Figure 1) that incorporates some basic facts and parameters about human performance in addition to the conventional GOMS keystroke-level model. The simulated human on the right-hand side of Figure 1 interacts with a simulated device on the left-hand side. Simplified perceptual processors translate simulated sensory input from the simulated device display to the cognitive processor, and the cognitive processor can command vocal and manual motor processors to produce simulated movements on the device's inputs, such as simulated keystrokes or mouse movements. The device can then change the contents of the simulated display accordingly.

This paper presents the lessons learned when GLEAN was used to evaluate interface and team structure designs for a new class of user interface that was being developed in a large-scale project for the U.S. Navy; see Osga, et al.[15] for a detailed presentation. Our modeling project was based on the work of that group, but was a separate, and much smaller-scale, activity. The detailed results of our modeling work are presented elsewhere [16], and necessarily involve domain-specific detail and applicability. The goal of this paper is present some of the *general* lessons that were learned from this experience. These lessons apply not just to GOMS modeling, but to other modeling approaches as well. The task domain, system design, modeling work, and specific results will thus be described in the context of the lessons learned, and only as much as necessary.

It is important to be clear about two aspects of this work. First, we made use of the results of the design effort and its major usability study described in [15], but we were not responsible for either the design or the usability study. Second, the conclusions presented in this paper are the personal opinions of the authors, and should not be mistaken for the policy or views of any part of the U.S. Navy, U.S. Department of Defense, or U.S. government.

BACKGROUND

This work was sponsored by the ONR SC-21 Manning Affordability Initiative, a large project that sought to explore how modern computer technology could reduce the size of warship crews. The focus of this project was on the crews in the Combat Information Center (CIC), the place where the sensing and weapons systems of a warship are brought together and controlled. The hope was that more sophisticated computer systems could automate some of the functions being done by humans, and bring the information from multiple systems together to fewer operators, some of whom are the actual decision makers. The result should be not only fewer required crew members, but more effective and reliable information integration and decision-making.

A major thrust of the Manning Affordability Initiative was the development of a workstation computer platform for use by CIC system operators, the Multi-Modal Watch Station (MMWS), and a new concept of how the CIC jobs associated with Air Defense Warfare (ADW) would be organized. A full description is provided by Osga and co-workers[15]; this group will be called the *MMWS group* in what follows. The MMWS group designed and evaluated the new system using conventional human factors and user testing techniques; a primary goal was to compare the new system with the current ADW systems in place in the Navy. Our project was a small-scale appendage to the main project, and had a very limited goal: to determine whether and how GOMS models could contribute to the design of complex systems, and in particular, what would be needed to make GLEAN useful at this level. Because of the small

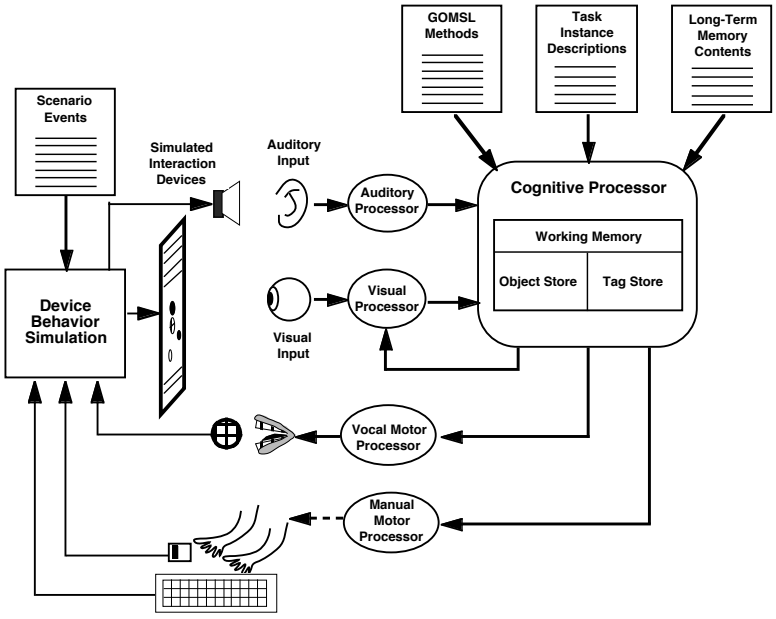

Figure 1. Architecture of the GLEAN system.

scale of our effort, we did not attempt to analyze both the current and new systems; rather we focused on trying to analyze a set of basic design issues in the new system being developed by the MMWS group.

THE OPERATOR'S TASK

The basic task of the CIC operator is to monitor a display, detect important events, and take certain actions. Each CIC team member normally has certain responsibilities; one is the team leader, and has an additional role of overall supervision. As discussed more below, defining the exact roles of the team members turned out to be a substantive problem both in the empirical data and the modeling work.

In more detail, the workstation display consisted of a large radar-like display (the *tactical situation* display) on which aircraft and surface ships (conventionally referred to as *tracks*) are represented as icons whose position, color, shape, and additional details represent the location, course, and speed of a track, and its *ID* - whether it has been identified by the system as friendly, neutral, potentially hostile, or a commercial aircraft, or is unknown. One of the workstations also presents information about the various radar and radio emissions (termed *ESM*) that can be used to identify a track or characterize its activity.

Depending on the team member's role, the specific tasks are to verify the ID of new tracks or changed tracks, make reports over radio communication channels about tracks of interest, and issue queries and warnings to tracks that for example, are approaching too close for comfort to the warships in the group. As a general background task, the operator needs to monitor suspicious tracks; this repetitive activity (called *hooking and looking*) involves choosing a track for examination and then selecting (*hooking*) its icon on the display. This brings up a sub-display of detailed data on the track's course, speed, altitude, and other characteristics, which the operator examines. The MMWS group proposed workstation designs with various levels of support for these activities, ranging from improved display representations to automated facilities. More details can be found in [15].

The MMWS group developed a single large task scenario that guided the design effort and was used in the user testing. We used a somewhat simplified form of this scenario in the modeling work, which spanned about 1.5 hours of real time, and involved a total of 70 tracks, most of which were simultaneously present on the simulated display. The scenario was a list of about 650 track events, corresponding to appearance, disappearance, and ESM events, and events for course, speed, and altitude changes. In the models, the state of the simulated display was updated every 1 sec of simulated time, and the GLEAN architecture itself executes on a grain size of 1 ms of simulated time. The GLEAN models themselves required only a few minutes to run the scenario.

THE LESSONS

The nine lessons presented below are gathered into four groups: design coverage, validation, practical concerns, and psychological theory.

Design Coverage Lessons

Lesson 1. GOMS models are useful in complex domains.

Our first modeling efforts focused on issues at the level of the single operator. The MMWS group had developed a body of subject-matter expert (SME) opinion on how the task should be conducted in order to follow the official rules of engagement. Basically, the GOMS models were simply programmed to carry out the stated tasks on the specified interface according to the SME prescriptions. To illustrate the GLEAN models, Figure 2 shows a sample of the GOMS methods from these models; due to the limited space, a full explanation is not possible, but it is hoped that the reader can get a impression of how the models were written. The first method describes how to select a track: first a mouse point, followed by a mouse button click, followed by waiting for the table of track data to appear. The terms <table> and <current_track> are working memory *tags* - named "slots" that hold the identity of the visual objects currently being examined.

The second method in Figure 2 illustrates some of the decision-making methods. This method examines various visual features of the current track, such as whether it is

```
Method_for_goal: Hook Track
Step 1. Point_to <current_track>.
Step 2. Click B1.
Step 3. Wait_for_visual_object_whose Label is
   "Track Data" and_store_under <table>.
Step 4. Return_with_goal_accomplished.

Method_for_goal: Review Track_profile
Step look_back. Store NONE under <action>;
   Look_at <current_track>.
Step check_com. Decide:
   If Color of <current_track> is Purple,
      Then RGA.
Step check_tripwires. Decide:
   If O40 of <current_track> is YES,
      and IOB of <current_track> is YES,
      Then Store WARN under <action>; RGA;
   If O60 of <current_track> is YES,
      Then Store QUERY under <action>; RGA;
   If C75 of <current_track> is YES,
      and O60 of <current_track> is NO,
      Then Store VID under <action>; RGA.
Step check_asp. Decide:
   If ASP of <current_track> is INT,
      and IOB of <current_track> is YES,
      and <RNG> is_less_than "180",
      Then Store QUERY under <action>; RGA.
Step check_IOB. Decide:
      If IOB of <current_track> is YES,
      and <RNG> is_less_than "110",
      Then Store QUERY under <action>; RGA.
Step. Return_with_goal_accomplished.
```

Figure 2. Sample GOMSL Methods. RGA is an abbreviation for Return_with_goal_accomplished.

inbound (IOB = YES) and decides what action to perform. This action is stored in a tag, <action>, for use by the calling method. The resulting models could predict observed single-operator task execution times reasonably well, as would be expected from the considerable work in the literature on validating GOMS models [5, 6]. The key measure is how soon certain critical actions would get done, given the large number of tracks on the display that had to be examined. In general, display designs that make the hook-and-look process more efficient will result in the important events being detected and acted upon sooner.

Of more interest, these models made clear that some of the workstation designs did not support a key part of the operator's task - remembering past actions taken on individual tracks. We were able to demonstrate how very simple facilities could alleviate a serious memory load problem. In another design exploration, we determined that a relatively simple piece of additional automated functionality to guide the operator to examine the highest-priority track would substantially improve performance beyond relying only on the manual selection process. That these analyses could be easily done suggests that other modeling approaches that work at least as well as GOMS will similarly scale to such complex procedural tasks.

Lesson 2. Modeling a team can be done with a team of models.

In this project we demonstrated a rather straightforward approach to analyzing team performance: if one can simulate an individual user acceptably well, then one can model a team of such users by setting up a model of each user and having the models interact with each other according to specified team procedures or team strategies. These are simply part of each individual's methods. For example, the GOMS model for the ESM operator specifies that when the operator notices a new ESM event on the workstation display, the operator will announce it by speech over the intercom. The methods for another team member would specify that upon hearing this announcement, the track should be re-evaluated. Like GOMS in general, this team modeling approach would be expected to work well only in highly proceduralized tasks.

Figure 3 shows the overall structure of a typical team model. There are four simulated humans, each performing a specific role; three of them are using the basic simulated workstations, while one is using a workstation that includes specialized displays for ESM information. The scenario events are generated by a master device which takes a scenario file as input, and broadcasts the corresponding event information to each simulated device, insuring that the track information is in synchrony, even though the devices will all be in different states as their simulated humans interact with them.

The four simulated humans communicate with each other via speech over an intercom channel; a vocal output from one of the operators is broadcast to the other operators as auditory input. Each simulated human also communicates by speech with outsiders over radio channels through their simulated workstations. The key feature is that all of the team interaction takes place via speech interactions over the intercom, while the individual activities of team members take place in interaction with their individual workstations.

With this basic framework, we explored different team designs in terms of whether and how the team members cooperated on the task. Different team organizations were simply represented in the individual GOMS methods that specified when announcements would be made over the intercom, and what actions would be taken in response. Different team procedures produced clear differences in predicted overall team performance. The relative ease of this team modeling approach suggests that it will be very useful; it should be applicable to other types of cognitive-architectural models of human cognition and performance.

Validation Lessons

Lesson 3. Validating a model against data may be impractical.

Research on GOMS and other modeling methodologies has usually tested the validity of the model by comparing its predictions to empirical data collected with actual human users in the same tasks and interface. In the case of GOMS and related methods, there is enough of a record of validation success to accept that the GOMS model methodology is basically valid [5, 6]. But a GOMS model is based on a task analysis that identifies the user's goals and procedures, and can be wildly inaccurate if the task analysis is wrong. The usual way to identify such an error would be to compare the model's predictions to empirical data.

However, the whole rationale for modeling human performance is to reduce the amount of empirical data collection needed to arrive at a usable design; clearly if validating the model requires as much data collection as user testing, there is little point to doing modeling. We

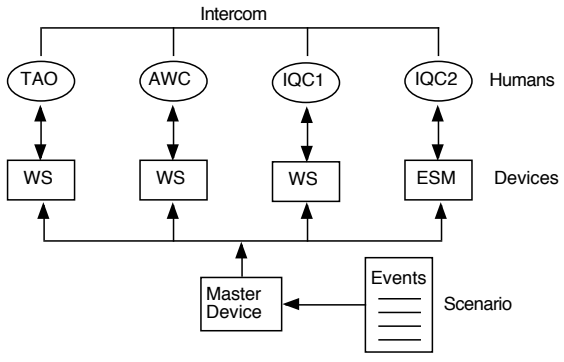

Figure 3. Structure of the team model.

discovered that the situation was actually more serious: in this very complex task, it was actually impractical to collect enough data to credibly validate a model.

We attempted to validate the team models against the user testing data collected by the MMWS group [15]. They compared the performance of real U.S. Navy CIC teams with the current system to the performance of similar teams who received some training on a prototype of the new system and performed the same, single scenario. The study was extremely difficult logistically, required a huge software development effort for the prototype system, and was quite expensive. The sample sizes were quite small, and there were few data points, little replication of conditions, and tremendous variability in the task strategies followed by the participants, both individually and at the team level.

This is not meant as a criticism of the MMWS group; such limitations are to be expected in this sort of domain. In the best traditions of user testing, the MMWS group chose in favor of a limited amount of data that was very face-valid, rather than the larger amounts of data that could have been collected with inexperienced subjects in an unrealistically simplified task. Furthermore, despite the limitations, the study answered the basic question posed by the MMWS group: Their results supported the conclusion that the new system would allow the reduced CIC crew size; the teams using the new system with half as many people did as well or better on several measures than did the larger teams using the current system. See [15] for details.

However, a scientifically-sound attempt to validate a model for the task would require larger sample sizes, multiple scenarios, and more experimental control over team procedures along with the same equipment and real test users. The huge expense of such an effort means that it could never be done. Rather than forgo either modeling or any attempt at model validation, it would be a better idea to focus on the most likely source of serious error in the model, which is the underlying task analysis and its representation in the model.

That is, instead of comparing the model behavior to limited and costly empirical data, it might be more profitable to have SMEs criticize the behavior of the model in how it acts in specific situations; if the model performs the task incorrectly, it could then be determined whether the task analysis underlying the model is incorrect, or whether the model is an incorrect representation of the task analysis. The task analysis and model could then be corrected. Note that it is difficult to critique an abstractly-stated task analysis, but since the model is executable and will produce specific and concrete behaviors, this evaluation can be definite and precise. If the task analysis and its representation in the model appears to be correct, then the model predictions have some degree of limited validity, and can serve as useful guidance for the design.

Lesson 4. The model might look wrong because the validation data are not right.

In our attempt to validate our team model against the user testing data, two problems emerged that involved user or team strategies. First, while the model predicted several types of action timings quite well (within about 10%), the times for new track reports and queries were seriously mispredicted (33 - 46% error); the model performed these actions much earlier than the human teams. This was puzzling because the model simply followed the SME rules for when a new track should be reported or a query issued (see Figure 2). Apparently, the human teams were not doing the same thing as the SMEs said they should be. A deeper look at the data would be required to determine why. See [16] for more discussion. At first glance, the model appears to be seriously inaccurate at predicting this aspect of performance, but the real question is whether the SMEs were wrong, the teams were wrong, or the experiment was wrong in some way. Perhaps other test users would follow the SME rules, or other SMEs would have different rules.

Second, the team interaction and communication did not follow any easily discernable pattern, nor were the teams required to follow any specified procedures. It is a U.S. Navy tradition that each team is allowed to organize itself and determine its own procedures. The task analysis underlying the design and the models did not specify the details of the team organization and procedure. Furthermore, trying to determine what strategy the human teams actually followed would have required a much larger sample of activity than was possible.

The lesson is two-fold: (1) empirical data in which the humans behave inconsistently or differently from the expert task analysis cannot support a model based on the task analysis; but (2) unless one knows that such data truly represent how the system should and will be used, they also do not discredit the task analysis and the model based on it. In short, the difficulty of task analysis and data collection in these complex domains means that any simple concept of validation against data is unworkable; ambiguity could be the norm.

Lesson 5. Model evaluation can be more definitive than user testing.

In contrast to the uncertainties in the user testing data, our exploration in modeling different team organizations produced very clear conclusions. As described fully in [16], following the general logic suggested in [12], we first constructed models that bracketed the required performance. The first model made unrealistic assumptions that every team member noticed and acted upon all critical events and worked completely independently. This superhuman non-team performed the task very well. A second model made more realistic assumptions about attention to events, but still lacked any cooperation. This model performed quite poorly - too many events were missed while the team members worked on their individual tasks. With the next models, we systematically attempted to

find the amount and kind of cooperation that would result in fewer missed events. In the better models, if a simulated operator was not busy with a specific task, it would announce critical events to the rest of the team; other team members could hear this announcement and remember it long enough to act on it after completing a task in progress. The model teams that performed well thus define a good team strategy, and the evaluation of the interface then becomes very clear: For each specified set of team organization and procedures, a particular interface design will result in a predicted level of performance.

This experience suggests that in a complex task where user data are hard to collect and user strategies are hard to determine, it will be easier to evaluate the interface with modeling than with user testing. Of course, only a fool would skip user testing in the design of such a critical system, but our experience suggests that a model analysis may be the most practical way to arrive at detailed design decisions for complex team tasks; rather than guiding detailed design, user testing could be reserved for issues beyond the scope of modeling and ensuring that the final design is acceptable.

Practicality Lessons

Lesson 6. GOMS modeling is learnable and cheap.
Despite previous studies [e.g. 4], there is a common belief that modeling is too labor-intensive for practical application, both in terms of the difficulty of learning the methodology and of applying it. However, the second author gets the credit for constructing all of the GOMS models in this project, starting from a background that did not include any prior work in cognitive modeling. While the present case involves only a single modeler, it adds to the accumulation of experience that modeling is indeed cost-effective (cf. [5, 6]). The costs of setting up the model initially can be high, but in surprising ways (see the next lesson). However, both the learning time and setup time are one-time expenses that can be amortized over the many subsequent evaluations that are easy and cheap.

Lesson 7. Programming the simulated device is the actual practical bottleneck.
In a full user-device simulation with GLEAN, the simulated user needs to interact with a simulated device that produces the inputs to the simulated user, and reacts to the outputs from the simulated user. Logically, it suffices for the simulated device to produce and react to abstract inputs and outputs. An actual GUI that a real user might interact with does not have to be implemented. Also, only those parts of the device that the simulated user will interact with need to be simulated.

Despite these simplifications, programming the simulated device turned out to be a major bottleneck in building the models for two reasons. First, good programming design and technique is required to enable fast and reliable modifications to support design iteration; this requires a programmer with substantial experience, who can be hard to find. Second, the specialized domain of the project required "computational navigation" to generate track motions and compute values such as the range at the point of closest approach, and distances along great-circle routes. Odd as it may seem, the computer form of the required algorithms are not common knowledge even in the SME circles, and do not appear to be adequately documented in any readily available source. Thus considerable time was spent reinventing some of the functionality of the actual device. It is likely that similar programming problems will be present in any complex and specialized task domain.

On the whole, it appears that constructing the simulated device is the actual bottleneck that limits how quickly a model for a complex task can be developed. An alternative is to couple the simulated user directly to the interface of an intact piece of software, as in St. Amant's ingenious work [17]. This approach appears promising, but it is not a panacea: in the design situations in which models will be most useful, the system under design does not yet exist, and so there is no existing piece of software to couple to! Clearly, more work is required to alleviate this bottleneck to make modeling easier.

Psychological Lessons

Lesson 8. The relevant psychology is incomplete.
Several problems emerged concerning the psychological adequacy of the GLEAN cognitive architecture. While some of these can be addressed by upgrading the fidelity of the architecture to include mechanisms such as those in ACT-R [1] or EPIC [11], others reveal gaps in the science base that need to be filled. Two representative gaps will be briefly described.

GLEAN assumes a rather simple form of working memory based on the EPIC tag store and simple versions of visual and auditory memory [13]. These seem to be adequate for interacting with most computer interfaces, but in the CIC tasks, operators appear to remember several tracks that they deem "suspicious" and will repeatedly check on them. Some operators will take notes to maintain this list, but often they rely on some form of working memory; this is probably not the usual verbal working memory because these tasks involve considerable speech activity, which is known to powerfully interfere with verbal working memory. Another use of working memory was remembering critical events, such as new track appearances, that occur while other tasks are being executed; it seems even less likely that these are stored in conventional verbal working memory. The problem is that these kinds of task-support working memory have not really been studied in cognitive psychology, making any attempt to define them in the architecture speculative at best. To complete the models, we devised ad-hoc methods for keeping suspicious tracks and changed tracks in a working memory. Clearly research is needed to arrive at psychologically valid models of task working memory.

Another gap appeared with the speech communication. In addition to the intercom, each operator uses an external radio channel. The intercom is fed to one earphone, the radio to the other, and the microphone is switched to either intercom or radio as needed. While an operator can tell if somebody is speaking over the intercom, apparently there is no way to tell if another operator is speaking over their radio channel. This means that an operator might be speaking over the radio when a message intended for them arrives over the intercom. Can one understand anything about speech input while one is talking? Again, this question does not appear to have been studied in the mainstream research literature, but related phenomena in the literature on attention and memory suggest that the answer might be neither obvious nor simple.

These two examples illustrate how there are many phenomena, both subtle and simple, that have not been adequately researched in experimental psychology, but which play key roles in important tasks and thus need to be included in human performance models. Perhaps more focused psychology research will fill these gaps in the future.

Lesson 9. GOMS needs interruptability.

The most interesting inadequacy of GOMS and GLEAN appeared very early in the project. Many of the tracks in the scenario disappear from the display as they go out of radar range, along with any data or data windows associated with the track. If the methods are working on the disappearing track, they will hang or fail when they attempt to look at or point to objects that are no longer present. It does not help to repeatedly check for the presence of the object, because it can potentially disappear at any time. The solution was to add an interrupt rule capability: The GLEAN model can include a set of if-then rules that are applied before every step execution cycle; these rules can detect the absence of the object currently being worked upon, and then execute methods to clean up and restart the model. The same interrupt mechanism turned out to be essential to detect critical events that happen during normal task execution, such as the changed tracks or speech input events; the triggered interrupt rules execute methods to save the information or alter task execution. Once the interrupt methods complete, execution resumes with the interrupted methods.

Figure 4 provides an example. The first if-then rule monitors for the appearance of a new track object (Blip), and executes a method to save the track in the above-mentioned working memory store for critical events. The second if-then rule watches for cases where the current track object disappears; the invoked interrupt method is also shown: it arranges for the disappearing track to be removed from the working memory list of suspicious tracks, and then aborts the current normal method execution and causes the model to restart with the top-level method. The cleanup will be completed and another track selected for examination.

The original GOMS concept [3] was based on a simple hierarchical-sequential flow of control regime similar to procedure-call hierarchies in a traditional programming language. In order to be useful in this domain, GOMS models required a fundamental extension to include interruptability and a more complex flow of control regime. Only then could the event-driven and asynchronous character of the multimodal team task domain be represented realistically. Perhaps a cognitive architecture such as EPIC that has a highly flexible control regime could be applied instead, and we suspect that other architectures could do as well if extended to have similar capabilities. But using a more complex architecture by itself does not solve the problem: Practical work requires a modeling methodology whose ease of learning and use is at least as good as GOMS has been, so further extending the GOMS notation might remain a better solution than trying to adopt and simplify a powerful but complex architecture.

CONCLUSION

The lessons learned in this project inform not only GOMS modeling, but also other modeling approaches that might apply to large and complex systems involving teams of humans. The overall positive lesson is that such modeling can be done, and done within practical constraints, and so future work to develop and apply modeling methodology in these domains should be successful.

The single most important negative lesson is the difficulty of validation of human performance models for complex team tasks. While we expected that models for such complex systems would be hard to validate for scientific and technical reasons, the surprising fact was that they are hard to validate for overwhelming practical logistical and economic reasons. Often overlooked is the fact that for the

```
Interrupt_rules

If Exists <new_track>
   Type of <new_track> is Blip,
   Event_type of <new_track> is New,
   and <new_track> is_not <last_new_track>,
   Then Accomplish_goal: Save New_track.

If <current_track> is_not nil,
   <current_track> is_not absent,
   and Status of <current_track> is
      Disappearing,
   and <current_track> is_not
      <last_disappearing_track>,
   Then Accomplish_goal:
      Prepare Disappearance_cleanup.

Method_for_goal: Prepare Disappearance_cleanup
Step. Decide:
If <current_track> is <last_suspicious_track>,
   Then Store <current_track> under
   <remove_from_suspicious_list>.
Step. Store <current_track> under
   <last_disappearing_track>.
Step. Abort_and_restart.
```

Figure 4. Sample interrupt rules.

same reasons, enough iterative user testing to fully refine a design is likely to be similarly impractical. Neither issue alters the fact that such systems are critical to develop correctly and at reasonable cost.

The challenge for future modeling work will be to understand how models can best be used in situations where we lack both the safety net of empirical model validation and also the old reliable standby of thorough user testing. Perhaps using the models as an aid to critically examining the task analysis will help solve the validation problem.

ACKNOWLEDGEMENT AND DISCLAIMER

This work was conducted in support of the SC21 Manning Affordability Initiative under the Naval Submarine Medical Research Laboratory Work Unit 62233N-R3322-50214, entitled "Human performance modeling of the MMWS Build 1 Watchstation." The opinions or assertions contained herein are the private ones of the authors and are not to be construed as official or reflecting the views of the Department of the Navy, the Department of Defense, or the United States Government.

REFERENCES

1. Anderson, J.R., & Lebiere, C. (1998). *The atomic components of thought*. Mahwah, New Jersey: Lawrence Erlbaum Associates.
2. Byrne, M.D. (2003). Cognitive architecture. In J. Jacko & A. Sears (Eds), *Human-Computer Interaction Handbook*. Mahwah, N.J.: Lawrence Erlbaum Associates.
3. Card, S., Moran, T. & Newell, A. (1983). *The Psychology of Human-Computer Interaction*. Hillsdale, New Jersey: Erlbaum.
4. 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.
5. John, B.E., & Kieras, D.E. (1996). Using GOMS for user interface design and evaluation: Which technique? *ACM Transactions on Computer-Human Interaction*, **3**, 287-319
6. John, B.E., & Kieras, D.E. (1996). The GOMS family of user interface analysis techniques: Comparison and contrast. *ACM Transactions on Computer-Human Interaction*, **3**, 320-351.
7. 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.
8. 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.
9. Kieras, D.E. (1999). *A Guide to GOMS Model Usability Evaluation using GOMSL and GLEAN3.* Document and software available via anonymous ftp at ftp://www.eecs.umich.edu/people/kieras
10. Kieras, D.E. Model-based evaluation (2003). In J. Jacko & A. Sears (Eds.), *The Human-Computer Interaction Handbook*. Mahwah, New Jersey: Lawrence Erlbaum Associates. 1139-1151.
11. Kieras, D.E. & Meyer, D.E. (1997). An overview of the EPIC architecture for cognition and performance with application to human-computer interaction. *Human-Computer Interaction*., **12**, 391-438.
12. Kieras, D.E., & Meyer, D.E. (2000). The role of cognitive task analysis in the application of predictive models of human performance. In J. M. C. Schraagen, S. E. Chipman, & V. L. Shalin (Eds.), *Cognitive task analysis*. Mahwah, NJ: Lawrence Erlbaum, 2000.
13. Kieras, D.E., Meyer, D.E., Mueller, S., & Seymour, T. (1999). Insights into working memory from the perspective of the EPIC architecture for modeling skilled perceptual-motor and cognitive human performance. In A. Miyake and P. Shah (Eds.), *Models of Working Memory: Mechanisms of Active Maintenance and Executive Control*. New York: Cambridge University Press. 183-223.
14. Kieras, D.E., Wood, S.D., Abotel, K., & Hornof, A. (1995). GLEAN: A Computer-Based Tool for Rapid GOMS Model Usability Evaluation of User Interface Designs. In *Proceeding of UIST*, 1995, Pittsburgh, PA, USA, November 14-17, 1995. New York: ACM. pp. 91-100.
15. Osga, G., Van Orden K., Campbell, N., Kellmeyer, D., and Lulue D. Design and Evaluation of Warfighter Task Support Methods in a Multi-Modal Watchstation. Space & Naval Warfare Center, San Diego, Tech Report 1874, 2002.
16. Santoro, T.P., Kieras, D.E., and Pharmer, J. (in press). Verification and validation of latency and workload predictions for a team of humans by a team of GOMS models. *US Navy Journal of Underwater Acoustics, Special Issue on Modeling and Simulation*.
17. St. Amant, R., and Riedl, M.O. (2001). A perception/action substrate for cognitive modeling in HCI. *International Journal of Human-Computer Studies*, 55(1), 15-39.
18. Wood, S. (1993). Issues in the Implementation of a GOMS-model design tool. Unpublished report, University of Michigan.