# A Review of Human Performance Models for the Prediction of Human Error

**Prepared for:**
National Aeronautics and Space Administration
System-Wide Accident Prevention Program
Ames Research Center
Moffett Field, CA 94035-1000

**Prepared by:**
Kenneth Leiden
K. Ronald Laughery, Ph.D.
John Keller
Jon French, Ph.D.
Walter Warwick, Ph.D.
Micro Analysis & Design, Inc.
4949 Pearl East Circle Suite 300
Boulder, CO 80301

And

Scott D. Wood, Ph.D.
Soar Technology, Inc.
3600 Green Court, Suite 600
Ann Arbor, MI 48105

September 30, 2001

# Executive Summary

The NASA System-Wide Accident Prevention Program is intended to promote the development of new technologies to reduce accidents in aviation. Reducing human errors is a significant part of this effort. The Human Error Modeling element within this program is supporting this effort by focusing on the development and application of computer simulation and modeling tools for the *prediction* of human error. The goal of this particular research, headed by Micro Analysis & Design, is to look at existing human performance modeling architectures to determine their suitability for human error prediction. Specifically, this report reviews a diverse range of human performance modeling architectures to highlight the aspects of each that are most useful to error prediction. In addition, hybrid approaches are explored, where features from different modeling architectures are combined to enhance error prediction capabilities.

A compilation of human error taxonomies was assembled in an attempt to define the scope of errors that human performance models would potentially need to predict. The taxonomies are predominantly conceptual in nature – they focus on understanding the cognitive process involved in the production of human error rather than describing the observable characteristic of the error. The four error taxonomies are:

- Situation Awareness (SA) – Endsley, 1998
- Model of Internal Human Malfunction – Rasmussen, 1982
- Model of Unsafe Acts – Reason, 1990
- Information Processing Model – Wickens and Flach, 1988

In addition, a method for modeling errors and error chains is described to illustrate the importance of determining the needed scale of the problem that human error models should address. By determining at what level in the error chain an error should be considered unsafe or otherwise significant, error modeling research can bound the problem to pursue efficient and realizable objectives. Experience has shown that many human performance modeling studies have failed because of the inclusion of too many factors that were not system performance drivers.

The human performance modeling architectures included in this report have been classified into three general types –task network, cognitive, and vision. Task network models use human/system task sequence as the primary organizing structure and are often considered top-down approaches. The task network models reviewed are Micro Saint, IPME, WinCrew and IMPRINT. On the other hand, cognitive models are more typical of a bottom-up approach as they simulate the mechanisms that underline and cause human behavior. Cross-pollination has blurred some of the original distinctions between these two types. The cognitive models reviewed are ACT-R, Air MIDAS, Core MIDAS, APEX, COGNET, D-OMAR, EPIC, GLEAN, SAMPLE and Soar. Lastly, vision models focus specifically on the use computational algorithms to simulate the human visual processing of an image. The vision models reviewed are ORACLE, OptiMetrics, and the Georgia Tech vision model.

In the last section of this report, guiding remarks about the application of selected architectures to future human error modeling work are given. Recommendations are given on two fronts. First, how well the individual architectures stand on their own merits for human error prediction is discussed. Second, how individual architectures can be combined to improve error prediction capability through a hybrid approach is presented.

# Table of Contents

# 1 Introduction

> *On the USS Vincennes this number [altitude] was embedded in a list of other numbers showing range, speed, bearing, and so on. This small screen was hard to read, especially if a crewmember had to abandon what he was watching on the large screen to search through the small one. The more critical problem noted by the Fogarty report was that the trend was not given. Crewmembers could not easily check whether an airplane was ascending or descending. [Also]…due to a system weakness, the Vincennes crew may have been seeing two different airplanes. The fact that the track number had been changed had not been clearly announced at the time it occurred. To find out the altitude of a track, you can use a trackball to hook it on the screen. If the ship is pitching violently, as the Vincennes was, it might be safer to punch in the track number into a keypad.*
>
> *Sources of Power*
> G. Klein, 1988

Tragic accidents frequently leave a convoluted trail of human and mechanical errors. As the excerpt above from Klein's 1988 book indicates, the USS Vincennes incident is a good example of how errors can interact to cause a chain of events between air and ground platforms that could lead to the shoot down of an Iranian commercial airliner on a routine commercial flight across the Persian Gulf. Interrupting the cycle of errors at any one of several key points above likely would have prevented this tragedy. Much can be learned in hindsight by having panels of experts review the accident chains; studying them and the events that precipitated them can lead to an explanation of the causes and identification of what events went wrong. This is the traditional accident investigation procedure.

In addition to expert panels for catastrophic accidents, other methods for determining the effect of human error on safety call for analysis of incident and accident report databases from the National Transportation Safety Board (NTSB) and the Federal Aviation Administration (FAA). In addition, the NASA Aviation Safety Reporting System (ASRS) was developed to give pilots and controllers the opportunity to voluntarily report safety related occurrences that fall short of *formal* incidents or accidents addressed by the other reports. These efforts have provided much insight, as will be shown in this paper, into understanding causal effects behind human error. However, with respect to ASRS, because it is voluntary, one can assume that there are safety related occurrences that go unreported. Thus, the rate at which similar types of instances have arisen is unknown. Additionally, the person writing the report determines what aspects of the safety related occurrence are important, which can miss important contributing factors. Together, these provide an incomplete database for statistical analysis, potentially resulting in deficient conclusions about the causal factors. Even if it was assumed that the conclusions were sufficiently

accurate, it would be difficult to know how to extrapolate from them to new systems or new operations concepts, for which no data are available.

Another alternative to understanding error might be controlled experimentation that could, theoretically, fill in some of the gaps mentioned above. Current and new systems could be investigated as they provide an excellent means for collecting quantitative data. However, there are two significant drawbacks with this approach. The first drawback is the high cost of human-in-the-loop simulation. Pragmatically, only a modest number of hours of data can be collected. The second, and more serious drawback, is that human performance errors are relatively rare – and come in many different sub-types with still lower frequency of occurrence. Human errors in reasonably well designed systems occur, at most, a few percent of the time and usually with a probability of more like $10^{-4}$ or $10^{-6}$. Even in the most ambitious experiments, it is virtually guaranteed that most types of errors will not occur even once, much less enough times to get a stable estimate of error probability. Hence, to understand human error and its impact on system performance there is a need for another approach to supplement what is learned from experimentation and natural observation.

One possible solution is computer modeling and simulation, where computational models of human behavior can be used to support error prediction and safety analysis. Assuming that sound models of the causal factors behind human error can be built to augment the error data we already have, we can then use simulation to study the aggregate effects of these errors on higher level system performance and safety. Furthermore, sound first-principled models will allow us to examine the underlying causes of error (e.g., system design and operating strategies), providing a way to study the sources of human error and ways to reduce their likelihood or compensate for their occurrence. Finally, error prone conditions and scenarios predicted by the models could be replicated in human-in-the-loop experiments for more focused study and analysis. The experimental data, in turn, can be used for validation of the human performance models.

An ongoing project focusing on ways to reduce human error in the cockpit is the NASA System-Wide Accident Prevention Program. The goal of this multi-year program is to foster the development of new technologies to reduce aviation accidents. Among the specific technologies currently being developed is the synthetic vision system. Synthetic Vision provides pilots in poor visibility conditions with an artificially rendered image of what the terrain would look like if the visibility were not impaired. To support this program, the Human Error Modeling (HEM) element is investigating the application of human performance models to study the types of errors operators could make when utilizing new technology such as Synthetic Vision. Since aviation errors can occur at all levels of human activity from perception to cognition to motor movement processing, human error models must have very broad coverage to be of use in this project. The models also need to represent the error chains that develop when human operators interact with changing and uncertain environments. Therefore, the objective of the HEM element is to develop models of both the human operators and the new technologies with which they interact that will enable us to predict the type of errors that may result and the associated safety and performance consequences. The ultimate goal is that these human performance

models will complement the findings from human-in-the-loop experiments and flight tests to support development and evaluation of strategies for reducing human error.

Although there have been many advances in human performance modeling over the last decade, the focus has been on the prediction of human-system dynamics in typical system operation rather than focusing on the underlying dynamics behind and the consequences of human error. Of course, there is good reason for this – if models cannot predict typical performance well, they will not be able to support the more challenging demands of predicting and dealing with error-laden performance. In short, we must "walk" with human performance models before we can expect to "run." However, the past decade has seen significant advances on many fronts of human performance modeling, from models of memory and perception to models of high level recognition-primed decision making. The goal of the portion of the HEM element that is described in this report is to look at the existing human performance modeling architectures to determine their suitability for human error analysis. Specifically, this report reviews a diverse range of human performance modeling architectures to highlight the aspects of each that are most useful for human error prediction. In addition, we explore hybrid approaches, where features from different modeling architectures are combined to assess potential enhancements in error prediction capabilities in the foreseeable future. In the next section, the approach for collecting and presenting this information is described.

# 2 Approach

As stated above, the goal of this research was to assess the state-of-the-art in human error modeling as it can be applied to the NASA System-Wide Accident Prevention Program. To do this, it was necessary to understand the type of human errors that the human performance models would need to predict. Therefore, the first step was a literature search to learn about human error and the way the different error types are classified. Our literature review had an aviation domain focus. Although a considerable number of documents from the Nuclear Regulatory Commission (NRC) appeared in our search that we are familiar with from other programs, the time constraints of the project demanded that we focused on the ample body of aviation literature.

The search for aviation-related documents pertaining to error taxonomies was fruitful, resulting in the development of several taxonomies to help guide this research by providing insight into causal factors of human error and bounding the error categories that this research would evaluate. It provides a framework for asking the right types of question about what we should expect from model prediction capabilities. The taxonomies are discussed in detail in Section 3.

Next we focused on computational approaches to human error prediction. We performed an intense search for any computational approach, regardless of the domain. Unfortunately, this search came up relatively empty with only few efforts aimed specifically at the prediction of human error. This confirmed our beliefs that human error modeling is still in its infancy. Interestingly, several "human error models" showed up in our search but these were not computational approaches, but rather just another term for taxonomy.

Then, we began to formulate ideas on how to present information regarding the specific modeling architectures that would be reviewed during this research. (The rationale for architecture selection is discussed in Section 5.) We settled on the idea that each architecture is given a brief overview, discussing high level features as well as pragmatic aspects such as what computer platforms support it. This is followed by presenting the architecture's underlying theory in a form that was descriptive and not overly technical. The process of turning theory into a mathematical formula or algorithm is generally omitted but can be obtained through the references. Comparisons between architectures is made in this section only where appropriate to clarify similarities or differences. Because of the number of architectures reviewed, the comparisons are minimized to keep the sections readable and focused. Lastly, the architecture's capability to predict human error is discussed, tying it back to the architecture's underlying theory and, in the rare case, actual application.

With the architecture review process established, we began collecting information on the individual architectures. Most of the architectures have websites where at least some top level information and point of contact information could be obtained. For every architecture except ACT-R (the ACT-R website provided everything we needed), we contacted someone knowledgeable with the architecture to assist us with information collection. Late in the project, we learned of an individual, Scott Wood, who had written

his Ph.D. dissertation on extending GOMS (the acronym for Goals, Operators, Methods, and Selection rules) to human error prediction. As his dissertation has not yet been published, our literature search did not uncover this important finding. We brought Dr. Wood onboard to assist with the GOMS-based architectures. In addition, since he is now an employee of Soar Technologies, he assisted with the Soar architecture as well. The chapter of his dissertation that specifically applies to human error prediction is listed in Appendix A.

The last step was to utilize our understanding of each architecture's strength and weakness to make recommendations to NASA about which architectures we believed could most efficiently and effectively predict human errors in future research. In addition, hybrid approaches, where features from different modeling architectures are combined, were assessed based on their potential to enhance error prediction capabilities in the foreseeable future.

# 3 Taxonomies of Human Error

The large corpus of aviation post-accident information has provided accident investigators and human factors researchers with an opportunity to investigate the causal factors behind human error. The goal of the investigations, of course, is to understand the causal factors, and ultimately implement satisfactory resolutions or mitigation strategies. A literature review revealed that no single taxonomy of human error is generally accepted for addressing all causal factors. In some instances, there is considerable overlap between taxonomies for classifying causal factors, but in other instances some factors are not represented at all (Wiegmann et al., 1995). It is beyond the scope of this research to resolve this discrepancy. Rather, we will present these schemes “as is” as the basis for qualifying the types of human errors that human performance modeling should address.

The taxonomies that are discussed in this paper are predominantly conceptual in nature. That is, they focus on understanding the cognitive process involved in the production of human error rather than describing the observable characteristic of the error (Wiegmann et al., 1997). The purpose of presenting these error taxonomies is twofold:

1. The error taxonomies provide a framework from which researchers can glean relevant error-causing information and understand underlying error mechanisms that should be accounted for in the error prediction models. This will enable researchers to more readily determine the preferred computational approach for a given error type. Furthermore, if the error-causing information is highly correlated to specific conditions or scenarios, the focus could be on replicating the error-prone conditions or scenarios in the simulations so as to cultivate error production. For example, if certain types of errors predominantly occur in high workload situations, a simulation scenario to study those errors should generate the proper dynamic events so as to emulate high operator workload.

2. The accident/incident statistics associated with each error taxonomy provide an estimate of the frequency of occurrence for the various error types, enabling researchers to focus on human error modeling for the error types that are more likely to occur and have consequences. This will help us to determine what types of errors our models must be able to predict.

## *3.1 Error Taxonomy for Situation Awareness*

Prior to discussing the Situation Awareness (SA) error taxonomy, a brief explanation of SA for our purposes is warranted. Though several definitions of SA exist, one of the most applicable definitions is that provided by Endsley (1988): "Situation awareness is the perception of the elements in the environment within a volume of time and space, the comprehension of their meaning, and the projection of their status in the near future." Expanding upon this to describe pilot situation awareness, the following levels of SA were developed by Endsley (1999) and are presented here verbatim:

- **Level 1 SA – Perception of the elements in the environment -** The first step in achieving SA involves perceiving the status, attributes, and dynamics of relevant elements in the environment. The pilot needs to accurately perceive information about his/her aircraft and its systems (airspeed, position, altitude, route, direction of flight, etc.), as well as weather, air traffic control (ATC) clearances, emergency information, and other pertinent elements.

- **Level 2 SA – Comprehension of the current situation -** Comprehension of the situation is based on a synthesis of disjointed Level 1 elements. Level 2 SA goes beyond simply being aware of the elements that are present to include an understanding of the significance of those elements in light of the pilot's goals. Based upon knowledge of Level 1 elements, particularly when put together to form patterns with the other elements, a holistic picture of the environment will be formed, including a comprehension of the significance of information and events. The pilot needs to put together disparate bits of data to determine the impact of a change in one system's status on another, or deviations in aircraft state from expected or allowable values. A novice pilot might be capable of achieving the same Level 1 SA as a more experienced one, but may fall short in the ability to integrate various data elements, along with pertinent goals to comprehend the situation as well.

- **Level 3 SA – Projection of future status -** It is the ability to project the future actions of the elements in the environment, at least in the near term, that forms the third and highest level of situation awareness. This is achieved through knowledge of the status and dynamics of the elements and a comprehension of the situation (both Level 1 and Level 2 SA). For example, the pilot must not only comprehend that a weather cell—given its position, movement and intensity—is likely to create a hazardous situation within a certain period of time, but s/he must also determine what airspace will be available for route diversions, and ascertain where other potential conflicts may develop. This ability gives the pilot the knowledge (and time) necessary to decide on the most favorable course of action.

Based on this SA model, Endsley (1999) developed an error taxonomy for SA that was essentially a more detailed version of the SA model (see Table 1).

| Table 1 - SA Error Taxonomy | |
|---|---|
| **Error Type** | **Error Description** |
| **Level 1: Failure to correctly perceive information** | |
| Data not available | Data is not available due to failure of the system design to present it or failure in the communication process. |
| Data hard to discriminate or detect | Examples are poor runway markings or inadequate lighting, noise in the cockpit, or obstructions blocking view. |
| Failure to monitor or observe data | Data is available, but is not scanned due to simple omission, attentional narrowing, distractions due to multi-tasking, or high workload. |
| Misperception of data | Data is misperceived due to influence of prior expectations or misunderstood due to task distraction. |
| Memory loss | Forgetting information is due to disruptions in normal routine or high workload. |
| **Level 2: Failure to correctly integrate or comprehend information** | |
| Poor mental model | Poor mental model does not enable the combining of information needed to meet goals. Primarily associated with automated systems. |
| Use of incorrect mental model | Interpretation of cues through an expected, but wrong, mental model of a system's behavior leads to the incorrect assessment of the situation. |
| Over-reliance on default values | Routine expectations of the system is assumed even though conflicting information is available, but not accessed. |
| Other | Information is not properly integrated or comprehended due to working memory lapses or other undetermined cognitive reasons. |
| **Level 3: Failure to project future actions or state of the system** | |
| Poor mental model | Information of current state is correctly understood, but projection of that state into the future fails because of poor understanding of how to do so. |
| Over-projection of current trends | The current state is projected into the future correctly. However, it is projected further into the future than for which the data is realistically valid. This, combined with not updating the projections at appropriate intervals, can lead to incorrect plans for the future. |
| Other | Projection of current state into the future fails because it is a demanding task that in a multi-tasking environment is not always performed. This is possibly due to the lower priority it is given or due to limits in cognitive resources. |
| **General** | |
| Failure to maintain multiple goals | Failure to maintain multiple goals in memory degrades SA across all three levels. |
| Executing habitual schema | Performing task automatically can result in important system cues being overlooked. |

Endsley (1995) applied the three-level SA error taxonomy to NTSB accident reports during a four year period. 71% of the accidents could be classified as having a substantial human error component. Of those accidents, 88% (32) involved degraded SA. Specifically, 72% were Level 1 errors, 22% were Level 2, and 6% were Level 3. In a much more detailed application of the SA error taxonomy, Jones and Endsley (1996) utilized voluntary reports from NASA's Aviation Safety Reporting System for 111 incidents involving pilots and 32 incidents involving controllers. The results are shown in Table 2.

**Table 2 - SA Error Taxonomy:**

**Percentage of Incidents by Error Type**

| | Error Type | Frequency (%) |
|---|---|---|
| General | Failure to maintain multiple goals | 0 |
| | Executing habitual schema (less receptive to cues) | 0 |
| Level 1 | Data not available | 13 |
| | Data hard to discriminate | 11.1 |
| | Failure to monitor data | 35.1 |
| | Misperception of data | 8.7 |
| | Memory loss | 8.4 |
| Level 2 | Lack of poor mental model | 6.9 |
| | Use of incorrect mental model | 6.5 |
| | Over-reliance on default values | 4.6 |
| | Other | 2.3 |
| Level 3 | Lack of poor mental model | 0.4 |
| | Over-projection of current trends | 1.1 |
| | Other | 1.9 |

Although the percentages of Level 3 errors are quite low for the two studies mentioned above, its importance should not be undervalued. Level 3 errors are the leading cause of air traffic control operational errors for TRACON and Center operations at 29% and 32.8% respectively. Given that a controller manages and provides separation assurance for air traffic almost exclusively by propagating current states into the future, these statistics are no surprise. What is important to glean from this data is as the United States transitions to a Free Flight paradigm, pilots will become more responsible for their self-separation from other aircraft. In order to do this, pilots must understand how current or pending actions can affect their projected aircraft state relative to other air traffic. Although flight deck decision aids will assist the pilot, Level 3 SA of the pilot in a Free Flight paradigm will become paramount to aircraft safety.

## *3.2 Error Taxonomy from the Model of Internal Human Malfunction*

Rasmussen (1982) outlined a decision-making process that led to the development of multi-step chain for diagnosing cognitive failure (O'Hare et al., 1994). The model assumes

that information is processed beginning with the detection of cues in the environment and ending with action execution.

The types of error described by O'Hare can be summarized in Table 3 as follows:

| Table 3 - Error Taxonomy from Model of Internal Human Malfunction | |
|---|---|
| **Error Type** | **Error Description** |
| Error other than human (structural, mechanical, electrical, etc.) | Pilot intervention could not prevent the accident/incident. |
| Information error | The pilot did not detect cues arising from the change in system states. |
| Diagnostic error | The pilot did not accurately diagnose the state of the system based on the information available. |
| Goal Setting error | The pilot did not choose a goal that was reasonable given the circumstances. |
| Strategy Selection error | The pilot did not choose a strategy that would achieve the intended goal. |
| Procedure error | The pilot did not execute procedures consistent with the strategy selected. |
| Action error | The pilot did not execute procedures as intended. |

Using this model, Wiegmann and Shappell (1997) categorized flight related mishaps (Class A, B & C) for the US Navy and Marine Corps between 1977 and 1992. 91.3% (264) of the mishaps attributed to pilot causal factors fit this model. The results are shown in Table 4.

| Table 4 - Error Taxonomy from the Model of Internal Human Malfunction: Percentage of Mishaps by Error Type | |
|---|---|
| **Error Type** | **Frequency (%)** |
| Information | 6.1 |
| Diagnostic | 21.72 |
| Goal Setting | 11.55 |
| Strategy Selection | 12.95 |
| Procedure | 39.48 |
| Action | 8.19 |

## 3.3 Error Taxonomy from the Model of Unsafe Acts

Reason (1990) classifies human actions into three levels:

- **Skill-based** actions are routinely practiced and highly automatic (e.g., keeping a car in its lane on a windy mountain road). Conscious thought is used sporadically to verify progress.

- **Rule-based** actions are a combination of conscious and unconscious processes to determine responses for situations that have been encountered before, either through experience or training (e.g., routine takeoffs and landings). Unconscious pattern recognition matches the indicators of a problem to a potential solution. The mind then rationalizes consciously if the solution is appropriate. If it is not, then other stored rules are considered as solutions.

- **Knowledge-based** actions require slow, demanding, and highly-error prone conscious thought to determine a response when other methods have proven unsuccessful (e.g., multiple system failures). The broader and deeper the knowledge base, the more likely a good solution will be determined. Trial and error approaches can be successful if there is sufficient time.

The three levels of actions described above, if performed incorrectly, are categorized as either **errors** or **violations**. Errors differ from violations in that errors are unintended whereas violations are deliberate. Violations are deviations from normal operating procedures (e.g., cutting corners while performing a sequence of tasks), though in most instances the human committing the violation does not consider it an unsafe act. Another important difference between errors and violations is the means in which they are reduced. Whereas errors can be reduced by improving the quality of the information processed by the human, solutions to reduce violations require motivational, (e.g., enforcement of compliance, improved morale, show of concern), cultural, and organizational changes that are quite broad and far-reaching. Below we will discuss each of the three error types in the Reason taxonomy.

### 3.3.1 Error Types

When a **skill-based** action is not performed as intended, the result is a skill-based error that can be categorized as a slip, lapse or perceptual error:

- **Attentional slips** occur when we fail to monitor the progress of routine actions at some critical point, usually when our situation or intention has just changed. Then, actions of habit for the old situation or intention override the new actions.

- **Memory lapses** occur when we forget what we earlier intended to do or when we omit steps from a plan of action.

- **Perceptual errors** were later added to the model (Maurino, Reason, et al, 1995). They occur when we don't recognize some object or situation and is often based on either habit or expectation (e.g., Train conductor expecting a green light, but light is actually red)

When a rule-based action does not produce results as intended, the result is a **rule-based mistake** that can be categorized into one of two levels:

- **Misapplication of good rules** occur most often when several elements in a situation are familiar, but other aspects of the situation are distinctly different. The person applies a good rule for the familiar aspects of the situation, but overlooks the other critical aspects that make the good rule invalid. (e.g., braking on an icy road for a pedestrian in the crosswalk)

- **Application of bad rules** occur when an incorrect rule has been used repeatedly in the past with no negative consequences. The repeated use of the bad rule reinforces the person's belief in the rule, but eventually a situation arises where something goes wrong (e.g., not wearing a seat belt and being involved in a car accident).

Lastly, a **knowledge-based mistake** occurs when a person is solving a unique problem that involves conscious reasoning. However, the solution is error-prone because of working memory limitations, inaccurate mental model of the situation, confirmation bias, frequency bias, similarity bias, and over-confidence.

### 3.3.2 Violation Types

- **Skill-based violations (i.e., routine violations)** are typically actions that cut corners due to an attitude of indifference about the environment (e.g., using the turn signal when changing lanes). The violator is rarely punished for incorrect actions or rewarded for correct actions.

- **Rule-based violations (i.e., situational violations)** are actions that assess the cost benefit trade-off of complying with proper rules and procedures vs. the time or effort saved by skipping steps. Although the person fully intends to violate the rules, they do so with the belief that nothing unsafe will happen in doing so.

- **Knowledge-based violations (i.e., exceptional violations)** in many cases occur when the situation preceding the violation was not covered by training or procedure. The violations can also occur in situations in which an unlikely combination of familiar aspects presents itself.

Wiegmann and Shappell (1997) again, like the Model of Internal Human Malfunction in Section 3.2, applied the Model of Unsafe Acts to flight related mishaps (Class A, B & C) for the US Navy and Marine Corps between 1977 and 1992. 91.3% (264) of the mishaps attributed to pilot causal factors fit this model. In this case, the taxonomy did not follow

the Model of Unsafe Acts as rigorously – rule-based and knowledge-based mistakes were lumped together as were the three types of violations. The results are shown in Table 5.

**Table 5 - Error Taxonomy from the Model of Unsafe Acts: Percentage of Mishaps by Error Type**

| Error Type | Frequency (%) |
|---|---|
| Slip | 14.28 |
| Lapse | 11.18 |
| Mistake | 57.13 |
| Violation | 17.42 |

## *3.4 Error Taxonomy based on the Information Processing Model*

Although not specifically developed to address human error, the classic Information Processing model (Wickens and Flach, 1988) provides another framework for classifying human error through a series of mental operations beginning with sensory stimuli and ending with response execution.

- **Sensory store** converts physical phenomena (e.g., light, sound) into neural manifestations. The information lasts briefly and does not require attention resources.

- **Pattern recognition** maps the physical codes of the sensory stores into meaningful elements (markings into letters, letters into words). This mapping is very complex and the least understood of the stages. Different physical memory codes may map to a single memory code, or conversely, a single physical code may map to several memory codes. Pattern recognition is strongly influenced by signal detection, data sampling, and linguistic factors.

- **Decision/response selection** is the next step and depends on the options available. The information can be stored in working memory to be used in the near future; the information can be combined with other information; or the information can initiate a decision process to immediately select a response. The information itself is often probabilistic so the consequences of the action for valid vs. invalid information needs to be considered. Cost benefit tradeoffs are often weighed in the decision making process.

- **Response execution** is initiated by the response selection. This stage takes the high level response and decomposes it into the required auditory, motor and cognitive steps. The execution of the response then becomes the feedback mechanism to the sensory stores.

- **Attention Resources** do not fit directly into the "sensory stimuli to response execution" sequence. Rather, attention resources can be viewed as a limiting factor for the last three stages – pattern recognition, response selection, and execution. For pattern recognition, attention resources may or may not limit perceptual processes (e.g., radio communication may be disregarded until the correct call sign is given or an inflection in voice captures the pilot's attention). Attention resources can delay or render the restart of both the response selection and response execution stages.

Wiegmann and Shappell (1997) again, like the Model of Unsafe Acts above, applied the Model of Information Processing to flight related mishaps (Class A, B & C) for the US Navy and Marine Corps between 1977 and 1992. 86.9% (251) of the mishaps attributed to pilot causal factors fit this model. The results are shown in Table 6.

**Table 6 - Error Taxonomy from the Model of Information Processing: Percentage of Mishaps by Error Type**

| Error Type | Frequency (%) |
|---|---|
| Sensory error | 2.84 |
| Pattern recognition error | 14.87 |
| Decision/response selection error | 29.54 |
| Response execution error | 45.48 |
| Attention resources error | 7.28 |

# 4 Human Performance Modeling to Predict Human Error

The error taxonomies described in the previous section provide a foundation for discussing how human performance models can be used to predict human error. These taxonomies provide a sense of what is important in error production and, therefore, what may need to be represented in a model. Furthermore, the taxonomies offer insight into what *really* is important in describing the system and human elements that lend themselves to error production. In applying human error models, perhaps the most significant task will be to determine what aspects of the human/machine system to include in the model and what to leave out. Our experience has shown that many human performance modeling studies have failed because of the inclusion of too many factors that, while a part of human/system performance, were not system performance drivers. Consequently, the models become overly complex and untenable. We expect for human error modeling that this will present an even larger risk to model development. Extrapolating from our experience, it is better to begin any human error modeling project with a small, but very specific set of human/system representations and then add to it as necessary, rather than to begin a modeling project by intending to represent everything. We believe the first approach may succeed while the second is virtually doomed.

In the next section, we will discuss error chains and how they necessitate a system level model as a means for providing a framework so errors can propagate into something eventful or relevant. Then, we will discuss how each type (i.e., cognitive, task network, information processing, perception) of human performance model can lend itself to the prediction of errors as defined in the taxonomies.

## *4.1 Error Chains*

In the error taxonomies described in Section 1, both accidents and incidents were used to generate statistics for the different error types. For self reported incidents, the value of this data should not be underestimated. In some situations, the only difference between an incident and an accident is that the chain of events was broken by lucky timing rather than any active correction by a human. For example, a runway incursion could become a deadly accident if the timing coincided with the taxiing aircraft crossing the path of a landing aircraft.

From a HPM modeling perspective, we need to ask how errors and error chains should be modeled and at what level in the error chain should the impact be considered unsafe or otherwise significant. For example, consider the simple case of controller-pilot communication in Class A airspace for a change of altitude (i.e., flight level). Because errors frequently occur in the communication process, procedures are in place to ensure that those errors are corrected prior to the subsequent aircraft clearance maneuver. The steps below describe the interaction and events that occur:

1. The controller tells the flight deck to change flight level.
2. The flight deck acknowledges the communication by repeating the controller's order (i.e., the "readback").
3. The controller listens to ensure that the readback is correct.

4. The flight deck changes flight level.
5. The controller monitors for compliance.

This simple procedure is designed to ensure that the flight deck maneuvers their aircraft as directed by the controller. However, a sequence of errors occurring in a given order can result in the flight deck maneuvering the aircraft to an incorrect flight level. This error could result in an operational error (e.g., for en route airspace, separation between two aircraft is less than 5 nm) for the controller and, in a worst-case scenario, a mid-air collision between the aircraft. Two potential error chains are described below.

**Error Chain #1:** This chain is initiated in one of two ways: The controller has an attentional slip and says the wrong flight level during his clearance to the flight deck, or the flight deck mishears the flight level assignment.

- Level 1 – The slip or miscommunication is caught by the controller when he/she listens to the readback of the flight level by the flight deck, breaking the chain.
    - Level 2 – The controller fails to listen attentively during the readback and thus doesn't discern that the flight deck is proceeding with the wrong flight level clearance. Instead, when the controller monitors for maneuver compliance, the controller realizes the aircraft is not heading to the cleared flight level and immediately corrects the mistake, breaking the chain.
        - Level 3 – The controller fails to monitor for compliance. If the flight level in question is clear of traffic, the error might go unnoticed with no ramifications at all.
            - Level 4 – If the flight level in question is not clear of traffic, Conflict Alert (i.e., software that projects current aircraft states into the into the future) will notify the controller of the pending conflict, breaking the chain. An operational error may result if the point of closest approach between the two aircraft come within the separation standard (e.g., 5 nm.).
                - Level 5 – Conflict Alert fails to notify the controller or the controller fails to react to it. If the aircraft are close enough, TCAS (Traffic Alert and Collision Avoidance System) will alert the flight decks of the two aircraft via a traffic advisory or, if a potential collision is imminent, a resolution advisory.
                    - Level 6 – One or both of the flight decks fails to receive or respond to the TCAS resolution advisory. At this point, a mid-air collision is potentially imminent.

**Error Chain #2:** This chain is initiated as follows: The flight deck hears the correct flight level and reads it back correctly, but makes a blunder by typing the wrong flight level into the flight management system (FMS) computer.

- Level 1 – The controller realizes the aircraft is not heading to the cleared flight level and immediately corrects the mistake, breaking the chain.
    - Level 2 – The controller fails to monitor for compliance. If the flight level in question is clear of traffic, the error might go unnoticed with no ramifications at all, thus breaking the chain.
        - The remainder of the chain follows the same the same steps as in Error Chain #1 (i.e., Level 4 - 6).

For either of the two chains described above, the occurrence of a mid-air collision or near-miss would almost always involve more than human error. Most likely it would be the result of failures in ground-based and/or air-based equipment combined with poorly timed stochastic air traffic patterns. Although from a real world perspective this is clearly significant, from a HEM perspective it is less interesting because of the infinitesimal probabilities of getting to that point in the chain. On the other hand, if one assumes that safety is first compromised when the aircraft leaves the current flight level for an *uncleared* flight level, then the HEM scope becomes much more focused. For example, error chain #2 can evoke this unsafe situation with just two human errors – the flight deck/FMS blunder and the controller failing to monitor for compliance (Level 2). Likewise, error chain #1 would require three consecutive errors to reach the same unsafe situation.

If limited resources were available for HEM development and only one of the chains as described above could be investigated, there needs to be a method for down-selecting between error chains to maximize the value of the research. One option is to perform a reliability assessment based on estimates of the error rates for each level in each chain to approximate the probability of getting to the level in the chain in which safety is first compromised, and yet still statistically significant. For example if we assumed that the error rates for each level of chains #1 and #2 were approximately equivalent, say 0.01, then the chance of reaching the level that safety is compromised would be 100 times higher for error chain #2 because one less error must occur to reach the level that safety is first compromised. Perhaps in this situation it would be more fruitful to pursue HEM of chain #2 over chain #1. For new systems or for some existing systems, the reliability assessment approach may not be feasible because error rates will be unavailable. However, researchers should consider extrapolating from existing task data if at all reasonable, as even a crude reliability assessment will provide value when determining the required depth and breadth of error chain modeling for a particular HEM project.

We should note that this approach to examining error chains has been studied extensively as part of the Probabilistic Risk Assessment (PRA) process and is used extensively in some domains, such as in the analysis of nuclear power plant risk. In this domain, it is referred to

as Human Reliability Assessment (HRA). Many tools and techniques are available for this and can be found on the Nuclear Regulatory Commission web-site at http://www.nrc.gov/NRC/NUREGS/indexnum.html. However, our frequent dealings with this literature and experts in the field tell us that they have the same problem that led to this research – there is neither sufficient data nor predictive models of human error to determine the individual event human error probabilities. While HRA analysis lets us assess how error chains will play out on to system performance, the technology has found itself in need of the very same underlying human error models that we are seeking through this project.

Another option is simply to focus the HEM on task failure just prior to the point in the chain where safety is first compromised; for error chain #1 that would correspond to the controller monitoring (or failing to monitor) for compliance. From a HEM perspective, the drivers that would most likely cause a failure in this task would be task shedding due to high workload or task disruption followed by a memory lapse to complete the task. The approach to modeling these types of errors would most likely require a representation of the complete system at the appropriate level of fidelity so that workload and potential disruptions could be sufficiently represented.

The two options presented here are straightforward examples to illustrate the importance of determining the scale of the problem that HEM should address. As mentioned earlier, our experience has shown that many human performance modeling studies have failed because of the inclusion of too many factors that were not system performance drivers. It is important that any HEM project bound the problem and pursue realizable objectives that are statistically relevant.

# 5 Technical Approaches to Predicting Human Error

One of the more difficult decisions during the course of this research was to decide which human performance modeling approaches would be examined and included in this report. We use the term 'modeling approach' broadly to mean a range of theories, methods, algorithms, and architectures that represents, predicts, or replicates at least a single facet of human performance. Using this definition, the number of human performance modeling approaches is certainly in the hundreds, and many of these could be applied in some manner to model human error. However, for pragmatic reasons, the approaches considered for this report were limited to existing architectures for modeling human performance in complex systems rather than theories or algorithms focused on specific aspects of human performance. We do not feel that this is a significant limitation to the findings presented here as we expect that over time the theories and algorithms that prove to be meaningful will be incorporated into the larger modeling architectures, such as those described here, as needed. Proof of this can be seen over the few years where cross-pollination of good ideas between existing architectures has become commonplace, particularly with respect to hybrid approaches being developed to address progressively more complex problems.

Still, even by limiting our findings to existing architectures, we needed to restrict our focus to those architectures that would lend themselves most effectively to either the prediction of human error or the prediction of conditions where human error is likely. Since we were aware that HEM was a cutting-edge application of human performance modeling, we did not anticipate and, in fact, we did not find much literature about the use of existing architectures for human error prediction (with the exception of HRA, as discussed above, which fills a particular niche). Hence, much of the decision about which architectures to consider was based on our historical expertise and current awareness of developments in the human performance modeling field. One final consideration was that all the architectures in the current NASA taxiway HEM study would be included in this report (with the exception of Wickens' error algorithm, which did not meet our definition of an existing architecture.)

The architectures included in this report have been classified into three general types –task network, cognitive, and vision. In the next sub-sections, these three types will be discussed briefly. Although cross-pollination has blurred some of the original distinctions between these different types, they will be described based on their original, more generic attributes. Specific capabilities and features based on recent developments will be mentioned in Sections 6.1 – 6.11, where individual architectures are discussed.

## *5.1 Task Network Models*

Task network models use human/system task sequence as the primary organizing structure. The individual models of human behavior for each task or task element are connected to this task sequencing structure. Task network modeling is often referred to as a reductionist approach because the process of modeling human behavior involves taking the larger aspects of human system behavior and then successively reducing them to smaller elements of behavior. One can also think of this as a top-down approach to modeling human/system performance. The level of system decomposition (i.e., how finely we

decompose the tasks) and the amount of the system which is simulated depends on the particular problem. The task network models discussed in this paper are Micro Saint, IPME, WinCrew, and IMPRINT.

## 5.2 Cognitive Models

The second fundamental approach to modeling human performance is based upon the mechanisms that underlie and cause human behavior. Since this approach is based on fundamental principles of human cognition, it is referred to as cognitive modeling. A diverse group of models falls into this category. This diversity is due to the level of fidelity that perception, attention, working memory, and decision making are addressed by the individual modeling architectures. Some models attempt to represent the steps of cognition at a very low level to simulate human memory while others focus on higher level decision making. In addition, there are differing opinions on underlying theory as well as the differing solution needs of the respective research teams. The cognitive models reviewed in this paper are ACT-R, MIDAS, D-OMAR, SAMPLE, Soar, GLEAN, EPIC and APEX.

## 5.3 Vision Models

In aviation, the analysis of human visual behavior is sufficiently important to merit special consideration. The vision models reviewed in this paper use computational algorithms to simulate the human visual processing of an image. They simulate human visual performance by predicting the probability that a target within the image will be detected. They also predict something called a false alarm that is the incorrect indication that a target is present when one is not. The predictions are based on how variations within the image such as luminance, edges and movement are received by either the receptors in the retina or the visual cortex. The models have most often been used to either assess vehicle detectability or evaluate the functioning of various vision aiding systems such as infrared imaging. The three vision models reviewed during the course of this research are ORACLE, the OptiMetrics Visual Performance Model, and the Georgia Tech Vision Model.

# 6 Human Performance Model Architectures

In this section, we describe the characteristics of individual human performance modeling architectures. Each architecture is given a brief overview, discussing high level features as well as pragmatic aspects that were determined by a NATO panel (AGARD Report, 1998) to be important issues in selecting a human performance model. This is followed by presenting the architecture's underlying theory for normative human behavior. Lastly, the architecture's capability to predict human error is discussed as it pertains to the architecture's underlying theory and, in the rare case, actual application. Task network modeling is discussed first, followed by the cognitive models, and then the vision models.

## *6.1 MA&D Task Network Suite*

### 6.1.1 Overview

Over the past 20 years, MA&D has developed a suite of task network-based simulation tools for that have been used for evaluating human performance in complex systems. Each simulation tool uses the same underlying discrete event simulation engine, but the individual tool characteristics depend heavily on the type of human performance application for which the tool was designed and is currently employed. Below is a brief description for each of the available simulation tools.

**Micro Saint** is a Windows-based general purpose, discrete-event simulation. Micro Saint provides a friendly GUI for constructing and simulating a sequence of tasks. The most general of the tools in the MA&D suite, there are few built-in features for representing human performance. On the other hand, there are few restrictions on the type of algorithms, whether relating to the human or the system, that can be developed and simulated. In terms of model reuse, existing models of human performance can be carried forward into new models with little effort. Simulations can be observed with animation to verify correct model design and/or to highlight unique human/system interactions.

**IPME** – Integrated Performance Modeling Environment – is a UNIX-based (specifically, Linux, Solaris, and IRIX) integrated environment that combines both the top-down approach of task network modeling with bottom-up approaches for simulating human behavior. IPME allows the user to select from two different workload models. The first is known as the Information Processing / Perceptual Control Theory Dynamic Scheduler (IP/PCT) model. The IP/PCT scheduler, which will be discussed in greater detail later in this section, simulates operator loading rather than task loading. The second workload model is the prediction of operator performance (POP). POP estimates when operator task demands exceed capacity. POP was developed by the British Center for Human Sciences. In addition to the workload models, IPME supports performance shaping function (PSF) approaches and built-in micro models. A PSF is a user-defined function which dynamically modifies individual operator performance. Specifically, a PSF defines how environment variables, operator trait, and operator state (e.g., temperature, skill-level and fatigue, respectively) affect operator performance for 'time to perform' and 'probability of failure' values. Built-in micro models are functions that represent basic human actions and behaviors such as the rate at which text is read or the time to reach or move a motor control.

**WinCrew** is a Windows-based workload tool to predict crew operator workload. WinCrew has a direct link between task-induced workload and the effect on system performance. With WinCrew, you can predict how the individuals in a crew will dynamically alter their behavior when they encounter high workload situations. Tasks are dynamically assigned depending on workload levels and task priority. Workload levels also impact task time and accuracy.

**IMPRINT** is a Windows-based tool widely perceived to represent a breakthrough in the application of modeling and simulation-based tools to applied manpower and personnel analysis. In addition to a heavy simulation component, these tools include a variety of mathematical programming and optimization methods. IMPRINT supports simulation-based acquisition by 1) supporting the development of clear system performance requirements and conditions so that hardware/software designers will know what the system will have to do to achieve mission success, 2) providing crew size limits to hardware/software designers so that they do not design for unavailable numbers of operators and maintainers, and 3) providing designers with a description of the significant soldier characteristics that explain and limit the probable operator and maintainer populations.

### 6.1.2 Theory of Operation

Task network models use human/system task sequence as the primary organizing structure (Laughery and Corker, 1997). The individual models of human behavior for each task or task element are connected to this task sequencing structure. Task network modeling is often referred to as a reductionist approach because the process of modeling human behavior involves taking the larger aspects of human system behavior (e.g., “perform the mission”) and then successively reducing them to smaller elements of behavior (e.g., “perform the function, perform the tasks”) until a level of decomposition is reached at which reasonable estimates of human performance for the task elements can be made. One can also think of this as a top-down approach to modeling human/system performance. The level of system decomposition (i.e., how finely we decompose the tasks) and the amount of the system which is simulated depends on the particular problem.

Task network modeling is an approach to modeling human performance in complex systems that has evolved for several reasons. First, it is a reasonable means for extending the human factors staple – the task analysis. Task analyses organized by task sequence are the basis for the task network model. Second, task network modeling is relatively easy to use and understand. Our experience has been that individuals with no training in computers or modeling can still look at a task network model and understand the basic flow of the human process being modeled. Finally, task network modeling has been demonstrated to provide efficient, valid and useful input to many types of issues. With a task network model, a system design (e.g., control panel redesign) can be examined to address questions such as "How much longer will it take to perform this procedure?" and "Will there be an increase in the error rate?"

Task network modeling greatly increases the power of task analysis since the ability to simulate a task network with a computer permits *prediction* of human performance rather than simply the *description* of human performance that a task analysis provides. What may not be as apparent, however, is the power of task network modeling as a means for modeling human performance in *complex systems*. Simply by describing the systems activities in this step-by-step manner, complex models of the system can be developed where the human's interaction with the system can be represented in a closed-loop manner. This, more than any other reason, gives task network modeling a distinct advantage over the other architectures discussed in this paper.

Task network models of human performance have been subjected to validation studies with favorable results (e.g., Lawless, Laughery, and Persensky, 1995; Allender et al, 1995). While validation issues must be considered with respect to a particular model and not a modeling approach, these studies have demonstrated that a task network strategy for modeling human performance, if properly applied, can be a productive one that provides meaningful and useful predictions of human performance in complex systems.

This basic task network is built via a point and click drawing palette. Through this environment, the user creates a network as shown in Figure 1. Networks can be embedded within networks allowing for hierarchical construction. To reflect complex task behavior and interrelationships, more detailed characteristics of the tasks need to be defined. For example, definition of task time, whether for a human task or machine task, is provided by choosing a mean time and standard deviation and the type of distribution to use (e.g., Gaussian, beta, exponential). Task time is quite important because it represents the temporal flow through the network. If available, this data can be derived from micro-models of human behavior. Otherwise, it is based on empirical data or estimates by the modeler. Another important parameter to define is the task release condition, which determines under what condition the task will begin to execute (e.g., a task cannot start until the operator to perform the task is available). These are just two examples of the definitions that must be provided by the modeler.

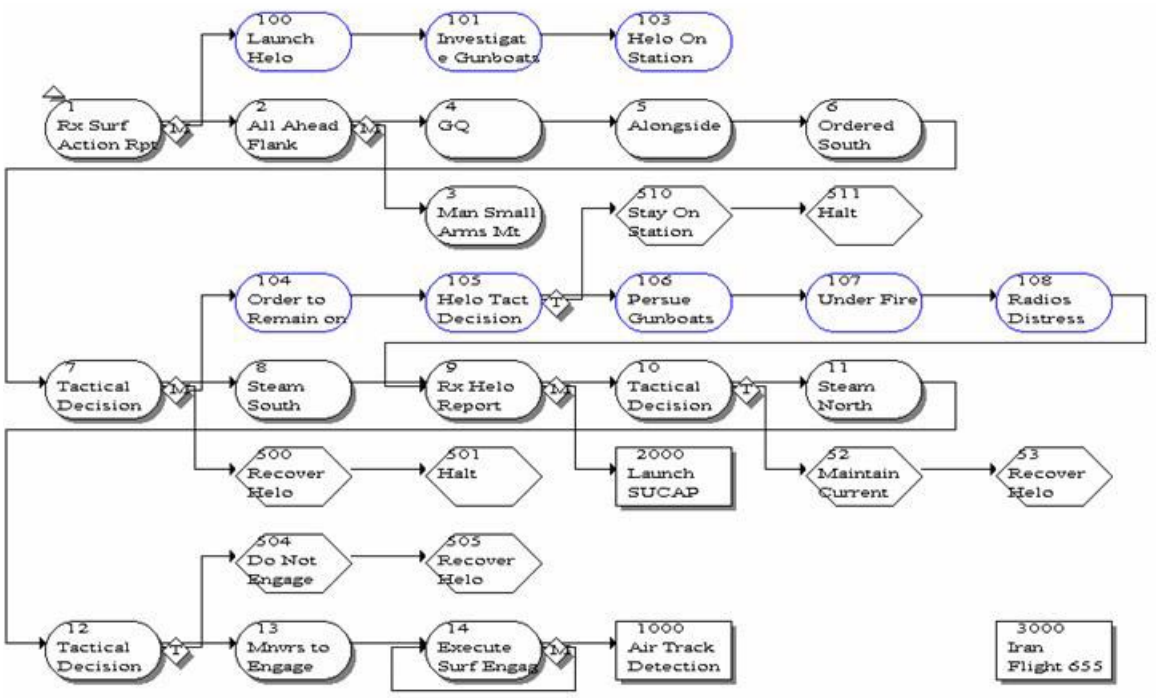



**Figure 1 Task Network for USS Vincennes Accident Recreation**

#### 6.1.2.1 Multi-tasking

Multi-tasking is one of the key features of task network models. Multi-tasking is performed by permitting the operator to move through more than one task path at a time. The task network GUI for constructing parallel networks provides a multi-tasking capability with minimum difficulty, especially when compared to other architectures that must construct multi-task activities through procedural languages. Although multi-tasking in Micro Saint must be done with great care to inhibit an operator from working on more tasks than is realistically possible, IPME and WinCrew have built-in workload models that constrain multi-tasking to levels represented by their underlying theory. Although detailed descriptions of all the workload models embedded in these architectures is outside the scope of this paper, a short description of the IP/PCT model within IMPE will be presented to demonstrate that the traditional top-down approach to task network modeling has evolved quite substantially in the last three years to take on a more bottom-up approach.

The **IP/PCT model** embedded in IPME was developed by the Defence Civil Institute of Environmental Medicine (DCIEM) in Canada and described in detail in Hendy et al (1997). The IP/PCT model assumes that human operators adapt to excessive processing load by changing their processing strategy to reduce the amount of information to be processed, or by increasing the time available before an action has to be performed. This process is under the operators control, is initiated in response to the perceived time

pressure, and is driven by the motivation to achieve a certain level of performance. If no effort is made to cope with the excessive load, the performance degrades. The time pressure is only reduced by the efforts of the operator.

The task network for a simulation can contain tasks that are initiated external to an operator and tasks that are initiated by an operator. Together these tasks create a *demand load* on an operator. The *demand load* is the task time-line that the operator must follow to meet the mission requirements. The *operator load* consists of the tasks that the operator can execute simultaneously. Tasks are considered to be in conflict when an operator tries to do more than one task at a time. Numerical coefficients describing the task interference are empirically derived.

For concurrent task performance, the IP/PCT model recognizes two types of interference: *structural* interference and *resource-limited* interference. *Structural* interference is used to describe effects due to limitations such as the inability to focus at two or more different images, problems associated with operating different controls with the same hand or limb, or the inability to speak two messages simultaneously. These structural limitations are driven by physical rather than information processing limitations. Physical domains in the IP/PCT model include vision, auditory, and manual/kinesthetic. The vision and auditory domains can have interference coefficients between zero and one inclusive, but interference in the manual/kinesthetic domain is assumed to be all or nothing, that is, interference coefficients are either zero or one.

Competition for common processing structures controls *resource-limited* interference and comprises the cognitive domain. Within this domain, the degree of interference is graded, so task interference coefficients range from zero to one. All operator tasks are *by default* assigned a cognitive category. When the *demand load* exceeds the *operator load*, the IP/PCT model uses a Scheduler to determine the order in which the tasks should be executed. The Scheduler uses queues to store different tasks. The *active queue* contains the tasks the operator is currently performing. The *working queue* contains the tasks available for scheduling. The *system queue* contains tasks or events that are externally initiated. When conflicts occur between task performance, tasks can be interrupted or deferred. Interrupted tasks can be resumed later or started from the beginning. Deferred tasks start at a later time. The Prospective memory stores tasks that are interrupted or deferred. A task can be shed or removed from the Prospective memory if there is a memory overflow. These types of tasks are abandoned and are not rescheduled.

#### 6.1.2.2 Recognition-primed Decision Making

Decision making in task network models has traditionally been implemented through rule-based approaches. Although useful in some situations, rule-based approaches have significant shortcomings when decisions must be made under uncertain conditions, with incomplete information, severe time pressure and/or dramatic consequences (e.g., the decisions a pilot would face in some type of emergency situation). Fortunately, the development of recognition-primed decision making (RPD) was introduced to address exactly these types of issues. Furthermore, RPD theory is aimed at a “decision-specific” approach that precludes the need for an integrative cognitive architecture, and is, in fact,

well-suited to a task network framework. Hence, RPD will be discussed below, primarily from work outlined in Warwick et al (2001), as a further example that the top-down approaches of task network modeling are more and more evolving to include features previously considered viable only in bottom-up methodologies.

The RPD model explains how people can use their experience to arrive at good decisions without having to compare the strengths and weaknesses of alternative courses of action. The claim is that people use their experience to size up a situation, and thus form a sense of *typicality*. Typicality amounts to the recognition of goals, cues, expectancies, and a course of action. Where classical decision theories postulate an analytical agent who carefully considers a host of alternatives, often against a background of perfect information, the RPD model postulates an agent poised to act who depends on his expertise to assess the available information and identify the first workable alternative. Figure 2 shows the flow of activities in the RPD model.

There are several noteworthy contrasts between RPD theory and traditional theories of decision-making. First, RPD theory emphasizes situation assessment over the comparison of options. The idea is that a suitable solution will emerge once the situation is understood. Hence, the focus of decision-making in the RPD model shifts away from the kinds of activities traditionally assumed to drive decision-making behavior (e.g., multi-attribute analysis) and onto diagnosis and the evaluation of expectancies. Second is the emphasis on *decision cycle* as opposed to a decision event. For example, a decision-maker might recognize a situation and the concomitant expectancies only to discover that the expectancies are violated, at which point the decision-maker will re-assess the situation and evaluate a new set of expectancies. Third, the RPD model postulates a decision-maker who implements the first *workable* solution he considers rather than a decision-maker who strives for an optimal solution. Finally, RPD is a *descriptive* model of decision-making; it does not dictate how decisions ought to be made and, by the same token, it does not specify the lower-level cognitive processes that are necessary to realize the flow of activities depicted in Figure 2.

Two aspects of RPD influenced the resulting computational approach in the task network implementation. First, RPD requires a long-term memory structure to represent the attribute of *experienced* decision-making. Second, a structure for *recognizing* a current situation based on experiences from past situations led to the implementation of Hintzman's *multiple-trace memory model* (1986). The basic idea behind a multiple-trace model is that each experience an agent undergoes leaves behind its own *trace*, even if that experience happens to be exactly like another experience the agent has undergone. A multiple trace memory is a collection of episode tokens rather than a store of episodic types. Recognition in a multiple trace model is a process of comparing a given situation against every trace in memory, computing a *similarity* value for each trace and then using these values to form an *echo*. Thus, typicality is assessed against the sum of experience rather than against a single experience. Moreover, because the process is driven by an assessment of similarity among multiple traces rather than a search for the perfect match between individual traces, recognition can be "fuzzy."

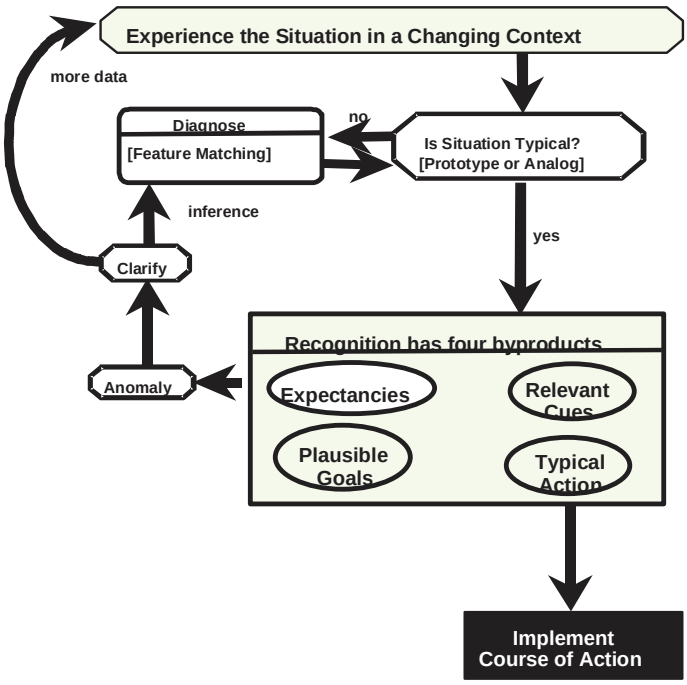



**Figure 2 RPD Flow of activities**

In terms of model output, the *echo* parameter is analyzed to determine what, if anything, has been recognized. The echo is compared to a threshold to determine if the echo demonstrates systematic associations in long term memory or whether it can be ignored as noise. The values of the echoes corresponding to the possible courses of action are compared to see whether any meet the criteria. If a single echo meets the criteria then only the course of action that corresponds to that echo is recognized. If more than one echo meets the criteria then each of the corresponding courses of action is recognized and the situation is declared *ambiguous*. If no echo meets the criteria then no course of action is recognized and the situation is declared *uncertain*.

### 6.1.3 Error Prediction Capabilities

The most useful stand-alone application of task network modeling for error prediction is in the context of multi-tasking activities and high operator workload. In this context, the modeling technique focuses on predicting error-prone operator conditions, situations or environments as well as the specific human errors that result. Embedded workload models (e.g., IP/PCT) predict when task shedding occurs due to high workload demands. Depending on the types of tasks being shed, this may produce an error-prone condition. For example, in a high workload situation, a radar controller, upon issuing a clearance, may shed the task related to the confirmation of the pilot readback of the clearance to focus on other required actions [In fact, the radar associate controller's role is to assist the radar controller with pilot readback of clearances (Leiden & Green, 2001), but there are many times when a sector is staffed with only one controller]. If the pilot mishears the clearance, the pilot of course would readback the incorrect clearance and maneuver the aircraft incorrectly. However, the radar controller would not perceive the error until the task of monitoring for maneuver compliance is performed. Thus, one may argue that the high workload that led to the task shedding has contributed to an error-prone situation and should be investigated further, despite the fact that the controller clearly skipped a procedural step.

To carry the high workload implications one step further, clearly the potential exists for human error anytime a task is shed *and* that task involves some type of information collection that is needed by a downstream task. Although these types of error are directly related to the *failure to monitor or observe data* in the Model of SA and *information errors* in the Model of Internal Human Malfunction, what is more important is investigating why the high workload condition exists because this ultimately is the source of the error. Besides the error types mentioned above, high workload is also an underlying cause for Level 1 *memory loss* and *misperception of data* in the Model of SA. Also, the *other* categories in Level 2 and Level 3 SA (i.e., information not properly integrated or comprehended, and projection of current state into the future fails in a multi-tasking environment, respectively) can be caused by high workload. Furthermore, *memory lapses, mistakes, and rule-based violations* in the Model of Unsafe Acts can result from high workload. Although the path from high workload to the production of error for some of the error types (e.g., memory loss) is not an error mechanism that currently exists in the suite of MA&D task network architectures, the workload prediction itself provides important parameters that may be useful with hybrid approaches (e.g., ACT-R maintains the working memory model while IPME passes workload parameters to ACT-R). On the other hand, other error types can more easily be represented in task network models. For example, rule-based violations can be represented by building shortcuts in task sequence, corresponding to an operator's intention of saving time, and determining the effect of the shortcut on downstream decisions.

Task network models are particularly well-matched for human error production due to the mismatch between task demands and resources. The detailed modeling of the human and the system provides a straightforward means for establishing whether an operator would perceive cues from the environment. This type of error directly supports the *failure to*

*monitor or observe data* in the Model of SA and *information errors* in the Model of Internal Human Malfunction.

Task network models have been employed in the past for human reliability modeling. In this application, task network models do not predict human error per se, rather they measure the effect of human error on system performance and reliability. The USS Vincennes incident was recreated in Micro Saint using this approach. One method of reliability modeling involves identifying potential errors in the task network representation and then assigning error rates to the corresponding tasks or subtasks. The tasks are presumably decomposed to the level where empirical data on error rates is available (e.g., keyboard entry errors). The task network model is then executed in a Monte Carlo manner to provide a realistic framework for representing human/system interaction, which in turn provides a more realistic representation or how errors propagate through the system. The results can be used to develop design enhancements to reduce the effect of common human errors on system performance (e.g., the need for a confirmation dialogue box when entering critical information into the system).

There are two drawbacks to this approach. First, it assumes that error rates are available at the task or subtask level, which is often not the case. Without this data, the modeler is forced to estimate the potential range of an error rate for a given subtask and then run the model parametrically through that range to look for system performance sensitivities. If the system is sensitive to the error rates, choosing potential solutions can become difficult if the solutions themselves are the result of a system tradeoff. Furthermore, depending on the complexity of the model, the parametric combinations necessary to represent the error rates of many different tasks can become computationally cumbersome when combined with Monte Carlo techniques. Second, this approach focuses on the external appearance of errors and does not address their underlying causes. Hence, error rates for tasks that rely on cognitive processes are usually omitted from the analysis.

## 6.2 ACT-R and ACT-R/PM

### 6.2.1 Overview

The acronym ACT-R was originally formed from the phrase *Adaptive Control of Thought – Rational*, but it appears of late that the phrase *Atomic Components of Thought* has become the more accepted expression. Indeed, the latter phrase appears to more accurately reflect the method for which human cognition is represented in both detail and fidelity by the ACT-R architecture. The 'PM' extension refers to 'perception' and 'motor', where 'motor' generally implies manual motor movement.

ACT-R was developed at Carnegie Mellon University under sponsorship from the Office of Naval Research. Anderson and Lebiere's book, "The Atomic Components of Thought" (1998) methodically elaborates on the underlying theory and experimentally grounded performance of ACT-R. Lebiere and Anderson (2001) provide a concise, but encompassing reference for ACT-R. ACT-R is open-source and the associated software, models, published papers, and tutorials are available from the ACT-R web site (http://act.psy.cmu.edu). ACT-R is implemented in the common LISP programming language as a collection of LISP functions and subroutines. ACT-R runs on MacOS, Windows and Unix platforms. A number of tools are available for model development and execution, including a graphical user interface to author and run ACT-R models, a parameter optimizer that automates the task of model fitting, and a multi-model extension that enables multiple ACT-R models to run concurrently and communicate with each other and with an interactive simulation. Most recently, ACT-R was demonstrated to run in a High Level Architecture (HLA) simulation environment (Lebiere and Anderson, 2001).

One of the strongest factors contributing to ACT-R's utility is that it is experimentally grounded, both from an architecture and a model perspective. Architecturally, the 10years of ACT-R development follow an even longer history of experimental work and theory development. From a modeling perspective, good comparisons of ACT-R model output to experimental data are substantial. Just a few examples are classic list memory comparisons from data collected in 1962 (Anderson, Bothell, Lebiere & Matessa's chapter in Anderson & Lebiere, 1998, p. 237); complex problem solving task comparisons (Salvucci & Anderson's chapter in Anderson & Lebiere, 1998); and comparisons with simplified air traffic control tasks (Lebiere and Anderson, 2001).

### 6.2.2 Theory of Operation

The mechanisms applied in ACT-R are based on the assumption that human cognition should be implemented in terms of *neural-like computations* so as to more realistically simulate brain processes. In ACT-R, performance is predicted on a timescale from approximately 50 ms to a few hundred ms. This level of granularity is considerably above basic brain processes but considerably below higher level tasks such as operating an aircraft or managing air traffic. ACT-R attempts to represent the steps of cognition by using production system theory. A sequence of production rules triggers the coordinated retrieval of information from memory and the environment. Memory is divided into three types – a goal stack, procedural memory, and declarative memory. The goal stack contains the ranking of priorities for guiding behavioral actions. Procedural memory contains

production rules (i.e., productions), in terms of a condition-action pairs, that determines what action is initiated when a particular condition arises. Production rules can trigger actions with the outside world, change the goal stack, or retrieve information from declarative memory. Lastly, declarative memory contains knowledge structure elements known as *chunks*, which are stored in labeled slots. Chunks can be initialized prior to a run to emulate the knowledge acquired from previously solved problems, or added dynamically during a run based on solutions from current goals or perceptions from the environment.

The three memory types are accessed according to the current goal that represents the focus of attention. When the current goal is achieved or otherwise no longer needed, it is removed from the stack (i.e., popped) and a new goal is retrieved from the stack. Alternatively, the current goal can be suspended when a new or existing goal of higher priority is moved to the top of the stack (i.e., pushed). A conflict resolution process determines which production rule to apply when the condition corresponding to the current goal matches more than one production rule. New production rules can be created by combining chunks with existing production rules and then transforming them into a single production cycle. Figure 3 depicts the interaction between the current goal, the three memory types, and the outside world.

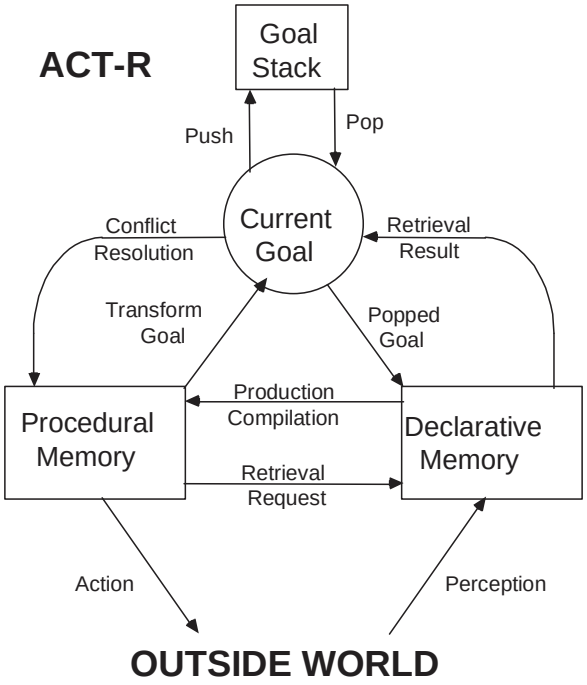



**Figure 3 - The overall flow of control in ACT-R. (From Lebiere and Anderson, 2001)**

The representation of qualitative human cognition is facilitated by ACT-R's neural-like activation methods for both knowledge accessibility and goal-oriented conflict resolution. Specifically, the neural-like activation determines the speed and success at which chunks are retrieved and production conflicts are resolved. The chunk activation is a function of the attentional weight given to the current goal so as to represent individual performance of

working memory. Root level activation is shaped from a Bayesian statistics mechanism to reflect the historical utilization of the information contained in the chunk.

Despite ACT-R's strong goal-directed behavior, multi-tasking has been implemented in ACT-R (Lebiere, Anderson and Bothell, 2001) by adding the capability to identify the inception of new events and then suspend the current task in favor of a more urgent one.

ACT-R/PM is an extension of ACT-R to improve the modeling capabilities for perception and motor movement. As with the other facets of ACT-R, the adherence to psychological theory is paramount. In ACT-R/PM, communication between the outside world (i.e., environment) and the cognition level is made possible by the perceptual/motor layer. In fact, ACT-R/PM precludes a direct communication link between the environment and the cognitive layer. The following summary of ACT-R/PM is based on information collected from the ACT-R/PM website.

The perceptual-motor layer is comprised of four modules. The Vision module manages all vision-related processing. The Motor module manages basic motor action preparation and execution. Motor movements to date have primarily emphasized keyboard actions and mouse movements. The Speech and Audition modules provide for speech and hearing, respectively. Figure 4 below depicts the system configuration.

Production rules specified in the cognition layer maintain responsibility for shifts of attention and motor action initiation. Conversely, the perceptual-motor modules pass information collected from the environment to the cognition layer by updating chunks in declarative memory. A noteworthy feature of ACT-R/PM is that the environment can be the same one as which human subjects interact (assuming the human experiments are implemented in Macintosh Common LISP). Alternatively, the environment can be simulated. Finally, multi-tasking between the cognition layer and the perceptual-motor layer work is possible, though coordinating parallel activities is difficult.

As evidence of further cross-pollination between architectures, the Motor module is borrowed largely from EPIC (which is discussed in detail in Section 6.10). The Speech and Audition modules are still quite rudimentary, though they do provide the capabilities necessary to model a great many results in those areas.

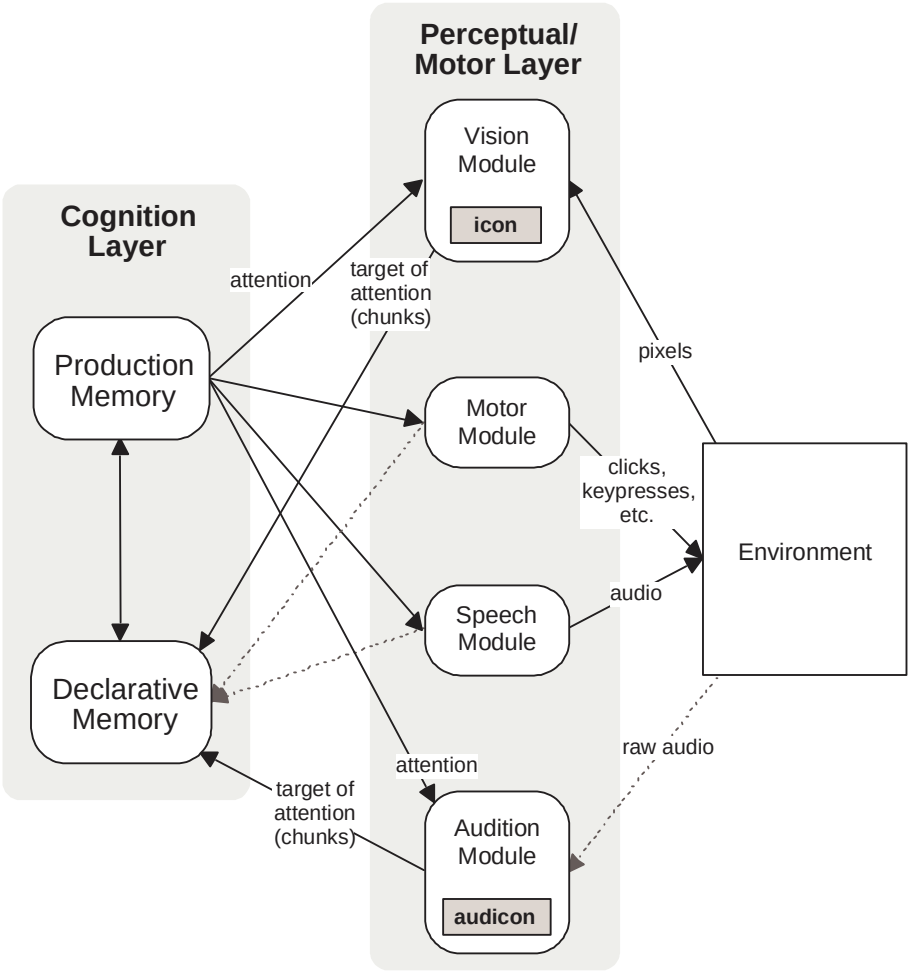



**Figure 4 - The information connections in ACT-R/PM.**

### 6.2.3 Error Prediction Capabilities

When considering the human error taxonomies of Section 1, memory retrieval in ACT-R provides a built-in mechanism for modeling two error types explicitly – *memory lapses* from the Model of Unsafe Acts and *memory loss* from the Model of SA. ACT-R refers collectively to these error types as errors of omission and errors of commission (Lebiere, Anderson, and Reder, 1994 and Lebiere, Anderson and Bothell, 2001). Errors of omission occur when memory retrieval fails whereas errors of commission occur when the retrieval is successful, but the wrong information is retrieved.

For errors of omission, the level of activation of a chunk establishes both its probability of retrieval and the retrieval time. Retrieval latency is inversely proportional to the exponential of the chunk activation (and the production strength). If a chunk cannot gather enough neural-like activation, its retrieval time will eventually exceed a predetermined latency threshold, resulting in memory retrieval failure. To produce human-like error rates, a stochastic component is included in the chunk activation. However, it is important to

state that errors of omission occur rarely as humans most often will respond with an erroneous answer rather than no answer at all. Consequently, the ACT-R approach to errors of commission is much more interesting and certainly more applicable in most situations.

Errors of commission are produced by enabling imperfect matching in the 'condition' portion of the production rule. If an activation were large enough to overcome a mismatch penalty parameter, then the wrong chunk would be retrieved. As an example, consider the modeling of arithmetic errors. When retrieving the sum of 3 and 8, both numbers are made source chunks and contribute activation to the correct chunk: 3+8=11. Many other chunks also receive activation from the source chunks. In fact, the other chunks might even gather more activation than the correct chunk because of the pattern of sources and the stochastic contribution to the activation values. To favor close matches, associations between numbers are set to reflect their similarity. Hence, the penalty for mismatching 7 with 8 (resulting in 3+7=11) will be less than the penalty for mismatching 1 with 8 (resulting in 3+1=11), because 8 is more similar to 7 than it is to 1. Other effects being equal, if the correct retrieval (3+8=11) fails, the chunk '7' will be more active than the chunk '1' and thus will be more likely to be retrieved for the *incorrect* formulation of the solution.

As this example should demonstrate, there is no magic in the ACT-R production of error. For error prediction to be reasonably accurate *and* extendable to new environments, it must follow experimentally grounded principles that sufficiently describe the mismatching of information in human cognition that leads to retrieval errors. If such principles are unknown (e.g., in arenas of pattern recognition), then curve fitting/calibration to experimental data is still possible, but the extensibility of the model will be greatly reduced. Fortunately, validated models of human performance in ACT-R become, in essence, part of the baseline architecture so that new research builds upon validated work. This was demonstrated in ACT-R during the Agent-based Modeling and Behavioral Representation (AMBR) program (Lebiere, Anderson, and Bothell, 2001). An accurate model of a simplified air traffic control task was constructed with only five declarative chunks of memory and 36 production rules.

Lastly, a final example of the application of ACT-R for error prediction is presented. A modeling project to predict *situation awareness errors* (corresponding to Level 1 and Level 2 SA) of soldiers using helmet-mounted displays for navigation (Kelley, Patton and Allender, 2001) was conducted. The ACT-R model for SA error prediction matched human subject errors with little tweaking by the modelers to achieve the superior results. Stated differently, the model was first developed independent of the experimental data and only then was it slightly modified to more accurately predict the SA errors collected experimentally from human subjects. This is yet another example that the ACT-R architecture is experimentally grounded. Because of the insight that this SA error research provides and the fact that it is one of the few examples of the application of cognitive modeling to predict higher level errors, the resulting conference paper has been included its entirety in Appendix B.

## 6.3 MIDAS

### 6.3.1 Overview

Development of the Man-machine Integration Design and Analysis System (MIDAS) began in 1985 during the joint Army-NASA Aircrew/Aircraft Integration program to explore the computational representations of human-machine performance to aid crew system designers. In the years since then, different MIDAS user communities developed different goals and priorities for MIDAS refinements. This led to what are today *two distinct versions of MIDAS*. *Air MIDAS* refers to the version that follows the original LISP implementation. Air MIDAS has been utilized continuously over the last four years to support research overseen by Dr. Kevin Corker at San Jose State University in conjunction with NASA Ames Research Center. On the other hand, *Core MIDAS* refers to the C++ version of MIDAS that has been in a state of redesign over approximately the same time period. The goal of the Core MIDAS redesign is to improve tool usability by providing a single GUI for data input, run time 3D visualization, and data analysis that executes on one single hardware platform. Although Air MIDAS and Core MIDAS share a common underlying theory, pragmatically they are two separate architectures. As such, in the Theory of Operation Section (Section 6.3.2), Air MIDAS and Core MIDAS will be discussed collectively and referred to simply as 'MIDAS'. Then, in the sections following that, Air MIDAS and Core MIDAS will be discussed separately.

### 6.3.2 Theory of Operation

MIDAS is a modeling tool specifically designed to assess human-system interaction in complex, dynamic environments. Utilizing an integrated approach to human performance modeling, MIDAS gives users the ability to model the functional and physical aspects of the operator, the system, and the environment, and to bring these models together in an interactive, event-filled simulation for quantitative and visual analysis. Through its facilities for constructing, running, and analyzing simulations, MIDAS can provide measures of operator workload and task performance across a range of conditions. Additional metrics of human-system effectiveness can be derived by users for a given scenario based on observed system or vehicle operating parameters during simulation. Such MIDAS output is intended to support the evaluation of current or future equipment, procedures, and operating paradigms. In particular, MIDAS is well suited for use early in the design cycle of new systems by exposing potential human performance difficulties related to a given design concept or by differentiating between the effectiveness of competing designs – all prior to the expenditure of resources needed to fabricate prototype hardware and test real humans.

Operator behavior within a MIDAS simulation is driven by a set of user inputs (see Figure 5) specifying operator goals, procedures for achieving those goals, and declarative knowledge appropriate to a given simulation. These asserted knowledge structures interact with and are moderated by embedded models of perception for extracting information from the modeled world and embedded models of cognition for managing resources, memory, and actions. In this way, MIDAS seeks to capture the perceptual-cognitive cycle of real world operators who work towards their most immediate goals given their perception of the situation and within the limitations of their physical and cognitive capacities. In

MIDAS, as in the real world, perceived changes in the world – new information or events – may cause changes in the adjudged context of the situation triggering new goals or altering methods to achieve current goals. Such perceived changes may be precipitated through the behavior of other modeled entities or by user specified time-based, condition-based, or probabilistic events set within the simulation (e.g., a system failure or the receipt of an incoming air traffic control clearance). It may, in fact, be the impact of the operator's own actions which lead to changes in context and goals.

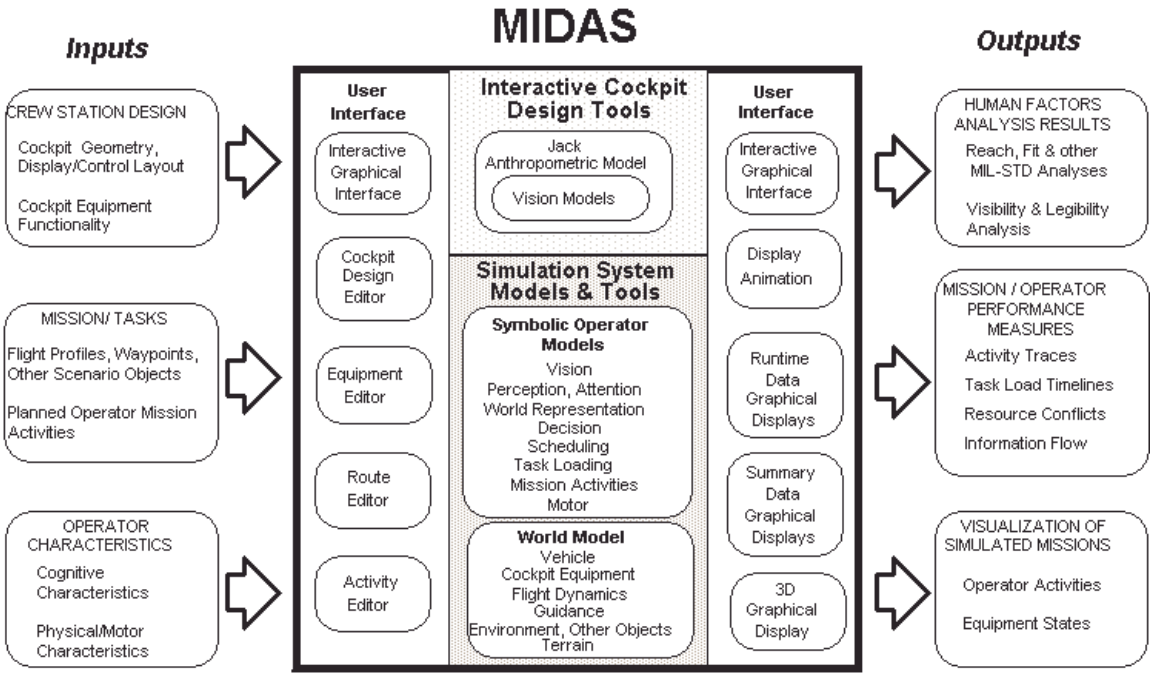



**Figure 5 – MIDAS Overview**

This complex interplay between top-down and bottom-up processes makes possible the emergence of unforeseen, unpredicted behaviors. Importantly, task time-lines are not an input requirement of a MIDAS simulation, but rather one of its outputs. Behavior flows from and is sensitive to the goals, knowledge, and domain expertise imparted to the virtual operator, the perceptual availability of information in the constructed environment, and the nature and timing of external events. This process is partially enabled by micro-models of human performance that feed forward and feed back to the other constituent models in the complex human/system representation. Together, this affords MIDAS users flexibility in selecting the focus of study as each of these parameters can be manipulated individually or in concert in order to assess the resulting effects on operator and system performance. Investigative approaches can range from single factor sensitivity analysis to "what if" studies that explore large test matrixes of interacting manipulations and conditions.

While the MIDAS framework simulates human behavior, the structure is not intended to be a "unified theory of cognition", perception, or action in the terms specified in either Soar or ACT-R. The MIDAS system does not make "structural" assumptions about human performance representation and, as such, it cannot be considered "unified". However, MIDAS does exhibit some of the characteristics of good unified theory such as:

- Providing systematic conceptual coherence for examining human performance
- Supporting (but not generating) models that can be systematically refined, and that produce behavior that is verifiable relative to human performance of similar tasks in similar environment
- Use of explicit functional requirements for the generation of human performance
- Use of formal models that meet those functional requirements

### 6.3.3 Air MIDAS

In the last four years, Air MIDAS refinements have been tailored to 1) expand the representation of cognitive tasks for both flight deck operators and controllers (Abkin et al., 2000); 2) represent the inter-agent coordination between controller/controller and controller/pilot interactions (Abkin et al., 2000); 3) represent an operator's affective (e.g., emotional) state (Hudlicka and Corker, 19????); and 4) represent the context (e.g., physical, procedural, organizational and cultural) in which operators perform their tasks (Verma and Corker, 2001).

One of the distinctions between Air MIDAS and Core MIDAS is that the original anthropometrics component known as Jack was replaced in Air MIDAS with a simpler, computationally less intensive algorithm for the time and accuracy of operator movements (the old Fitts Law). The equipment representation is in three dimensional space around the operator, with multiple operators and equipment representations possible.

#### 6.3.3.1 Error Prediction Capabilities

Clearly, the aviation-specific domain that has been the focus of Air MIDAS development will provide a strong foundation for human error modeling of flight deck operations. In this section, the features of Air MIDAS that lend themselves to error modeling will be discussed.

Long-term memory is represented in Air MIDAS through declarative and procedural information. It is assumed by the current architecture that long-term memory does not vary during the course of a simulation. However, it would be possible to use Air MIDAS to simultaneously model two operators performing similar functions but with different knowledge. In terms of error modeling, this might be useful for simulating coordination-type errors between controllers where declarative knowledge determines the type of procedure to perform. Controller A assumes one procedure (correct) while controller B assumes another (incorrect). The consequence of controller B's incorrect procedure may induce an error that *appears* to be caused by controller A.

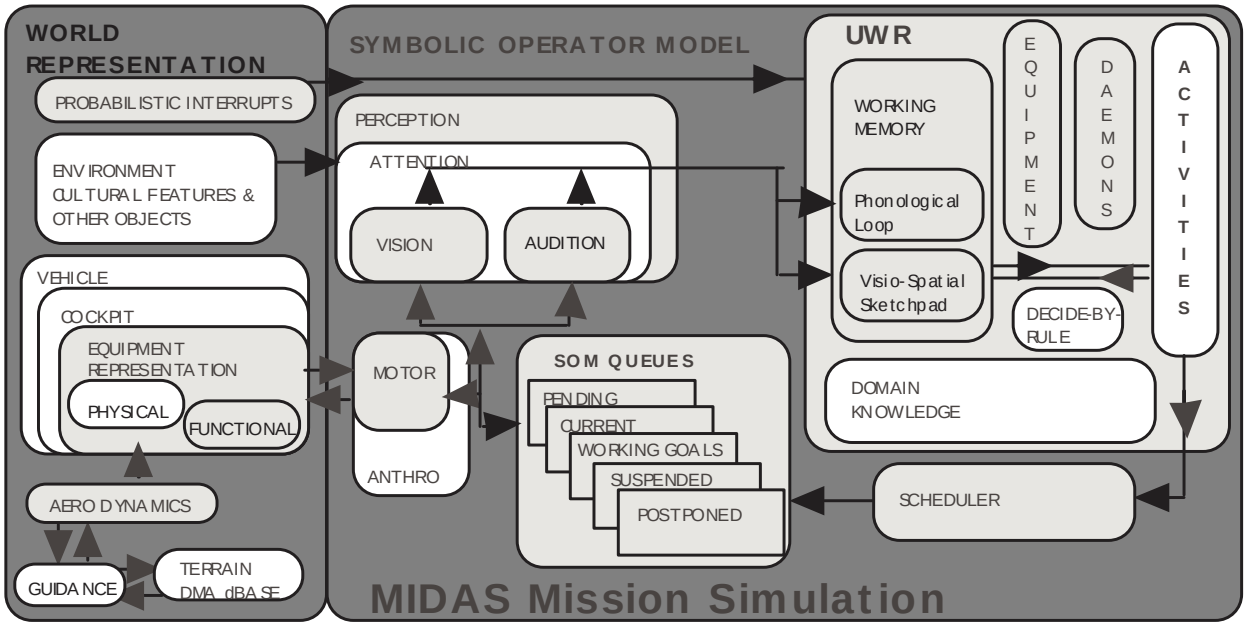
Working memory is modeled in Air MIDAS using a semantic net and is implemented as a central control processor of limited capacity interacting with temporary storage of both speech-based information (termed the articulatory loop) and spatial information (termed the visuo-spatial scratch pad). Unlike ACT-R, Air MIDAS will either return the correct information or no information at all – it can produce errors of omission, but not errors of commission. The resulting actions or decisions are then based on how this piece of

information (or lack of) influences the outcome. Hence, error modeling due to memory loss is dependent upon three different methods of decision making available in Air MIDAS. First, skill-based responses (e.g., scanning a display in response to an auditory alert) to perceptions from the environment are represented by *daemons*. Daemons get activated when values corresponding to perception elements reach a required threshold. The perception values themselves are determined based on extensive perceptual literature (Gore and Corker, 2000). Skill-based errors from the Model of Unsafe Acts could be emulated by stochastically changing the threshold (resulting in a *lapse*) or triggering the wrong daemon (resulting in a *slip*).

Second, rule-based responses can be represented by production rules that are activated when conditions in the simulated world match those defined in the rules. Like skill-based errors, error production for rule-based responses from the Model of Unsafe Acts (i.e., mistakes) is enabled by either stochastically changing the condition that triggers a production rule or matching the condition to the wrong production rule.

Lastly, decision making that is more complex and optimizing than rule-based approaches is represented by a set of six prescriptive algorithms (Payne et al. 1988) that utilize a combination of attribute values, weights, and cut-off values to select the best option from a list of potential solutions. Errors associated with this more complex decision making algorithm would still be producible, but the mapping of empirical data needed for any semblance of validation would be very difficult. Indeed, the validity of all three methods for error production is dependent on unique empirical data. Whether the error modeling is extensible to other work station designs or operational paradigms would have to be examined individually for each error type within a model. Extensibility of the prediction/production of one error type certainly does not guarantee extensibility of other error predictions within the same model.

Air MIDAS is particularly well-matched for human error production due to the mismatch between task demands and resources. A complex representation of both the human and the system is depicted in Figure 6. The system description is known as the World Representation (WR) and includes vehicle state, the physical environment within which the vehicles operate, and the information displayed by the crew station and other equipment. The operator's knowledge and perception of the system is known as the Updateable World Representation (UWR). Together, WR and UWR provide a straightforward means for establishing whether an operator would perceive cues from the environment. A simple example is the case where a controller misses a handoff indicator on his display because his visual attention has been focused on the flight strip bay for a period of time. This type of error directly supports the *failure to monitor or observe data* in the Model of SA and *information errors* in the Model of Internal Human Malfunction.




Source: Corker, 2000.

**Figure 6 - Air MIDAS Architecture**

More complex examples of error production (e.g., *goal setting errors* in the Model of Internal Human Malfunction) can occur when an operator is multi-tasking (e.g., a controller observing the radar display, listening to a request from an aircraft, and determining what to do next.) Directing attention to one task can cause another to be suspended or forgotten. In Air MIDAS, task priority is governed through a task scheduler that represents the current queue of tasks to be performed by the operator, given the current goals and context. Multi-task sequencing is performed by executing tasks based on the available perceptual and cognitive resources. The task resources are defined using a seven-point scale, where a value of seven implies that the task requires the highest amount of the perceptual or cognitive resource. Multiple tasks can utilize a given resource simultaneously, as long as the sum of their resource demands does not exceed a value of seven.

*Action errors* from the Model of Internal Malfunction can be produced by the anthropometrics model. Procedural sequence in the model determines what movements to what equipment are required, and movement time is calculated. If the time required is less than the time available then the probability of error increases.

Finally, it is important to note that although no Air MIDAS research specifically aimed at error modeling has yet been completed, work is underway on two projects – the NASA runway incursion study and a large multi-agent simulation for NASA safety analysis using Georgia Tech's Re-configurable Flight Simulator.

### 6.3.4 Core MIDAS

Although Air MIDAS is certainly the most powerful human performance tool available for research in the aviation domain, it is quite difficult to use. Hence, the primary reason behind the Core MIDAS redesign was to improve tool usability. Although the redesign effort conducted by Ames Research Center and US Army invested a large sum of resources to develop version 2.0 beta software, the project is not quite complete and requires further resources. MA&D was contracted by NASA to assess version 2.0 beta by developing a model to compare with the Vertical Motion Simulator (VMS) rotary aircraft experiment conducted by Ames Research Center in March-April 2000. The objective was to attempt to simulate the man-in-the-loop simulator experiment within the simulation environment and determine whether simulated target detections and pilot situational awareness corresponded with the results of the VMS experiment. MA&D was also asked to make recommendations on the feasibility of commercializing Core MIDAS. The following information is based on MA&D's assessment of Core MIDAS

Unique features contained only in the Core MIDAS environment are a novel animated manikin capability (see Figure 7) for visualizing a human's interaction with a complex user interface. The integration of a human performance model with a 3-D geometric view that includes an animated manikin is a difficult problem that the MIDAS team worked diligently to solve. Several human-performance simulators have been considering a link between the human-performance model and animated manikin representations. None to date have been successful other than Core MIDAS.

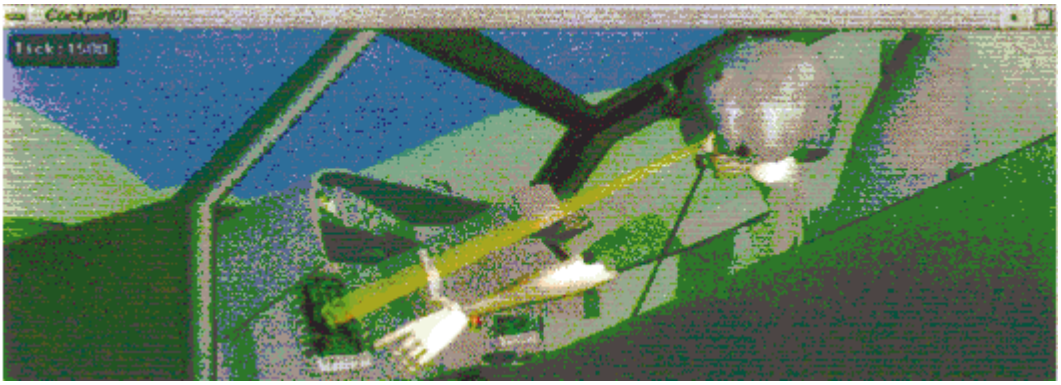


**Figure 7 - Cockpit with "Jack" ™ anthropometrics**

Research from several Ames projects have been used to formulate the Human Performance model engine. This human performance model includes memory and belief, situation awareness, attention, vision, auditory, and motor movement component models. In addition, a procedure scripting language called the Operator Procedure Language (OPL) provides a Scheme-like scripting capability for managing the memory and belief model and integrating the human actions with scenario events.

Although the GUI is almost complete, there is no integrated GUI for writing the OPL scripts. The help system does not exist and the user documentation is incomplete and sketchy. None of these problems are insurmountable, but MA&D believes there is at least

another year of development for a team of three people to complete a releasable version of Core MIDAS 2.0.

#### 6.3.4.1 Error Prediction Capabilities

In the 2.0 beta version of Core MIDAS, the working memory model had not yet been completed. Thus, the ability to model memory loss is impossible. At this time, the most promising aspects of error prediction are associated with the model of situation awareness (Shively, Brinkner, and Silbiger) incorporated into 2.0 beta. Although this SA measure has not been validated, it does provide a reasonable approach to gauge the data available from the system versus the data known by the operator and provides a graphical output display depicting the ratio during run-time. The current implementation of the model in Core MIDAS follows the general description of the above reference, but it lacks the important feature that computes SA error. Nonetheless, the SA model is described here because of its potential future relevance to SA error prediction. (The following text was excerpted from an informal NASA document that was emailed to MA&D in support of the 2.0 beta assessment.)

The SA model is comprised of three key features: 1) situational elements, 2) categories (context sensitive nodes), and 3) a regulatory mechanism, the SA manager. For each context, a set of related situational elements specify the ideal SA. When the operator's context changes due to new environmental conditions, the weightings on the categories change their values. Each of the model's key features is discussed below.

The *situational elements* can either be the operator's perception of components in the environment that define a situation or they may represent information in the operator's memory. Examples are features from the environment such as trees, hills, buildings, vehicles, equipment components, etc. or internal objects such as facts about a certain scenario (e.g. weather conditions, time). Each situational element is associated with a category that is relevant to the situation. The level of SA is determined from the number of the situational elements that populate each category together with their respective weights. In the current implementation, all situational elements are defined as having the same level of importance. The amount of SA each situational element contributes to the operator's overall SA is determined by the perception level of the situational element. As the perception level increases for objects exterior to the crewstation from detection to recognition to identification, the situational element is assigned a higher percentage of SA (see Table 7). Similarly, there are different levels of perception for objects within the crewstation (see Table 8).

| Level | Characteristic | % SA |
|---|---|---|
| Undetected | No info | 0 |
| Detected | Location | 33 |
| Recognized | Category | 67 |
| Identified / comprehended | specific type, contextual meaning | 100 |

**Table 7 - Perception levels of objects external to the crewstation.**

| Level | Characteristic | % SA |
|---|---|---|
| Unread | no info | 0 |
| check-read | Approximate value | 50 |
| Read | exact value | 100 |

**Table 8 - Perception level of objects internal to the crewstation.**

*Categories* are semantically related collection of situational elements. Each node is assigned a weight for each context. The weight determines the importance of the category in a specific context. When the context changes, the weight on each category changes to accurately reflect the ideal SA. The current implementation of the SA model sets no limitation on the number of categories that can reside in working memory and there is no temporal decay of information.

In the current implementation, the *SA manager* uses the information stored in the categories and situational elements to compute the operator's SA continuously. A dynamic runtime display for each operator outputs the level of SA throughout a simulation.

## 6.4 D-OMAR

### 6.4.1 Overview

The Distributed Operator Model Architecture (D-OMAR) was developed by BBN Technologies under sponsorship from the Air Force Research Laboratory. D-OMAR is an event-based architecture that provides a broad range of software tools for developing human performance models or agents for agent-based systems. The agents or human performance models operate in a simulation environment running in real-time or fast-time. D-OMAR supports the notion of an agent whose actions are driven not only by actively seeking to achieve one or more defined goals, but also by reacting to the input and events of the world. The D-OMAR knowledge representation languages were designed to *facilitate the modeling of the human multi-tasking behaviors* of team members interacting with complex equipment in pursuing collaborative enterprises.

The knowledge representation languages that form the basic building blocks for the modeling effort are the:

- Simple Frame Language (SFL) for defining the agents and objects in the simulation world.
- Simulation Core (SCORE) procedure language for defining the behaviors of agents and objects.
- Flavors Expert (FLEX), a rule language for defining rule-based, contingency behaviors.

The D-OMAR languages are extensions of the Common LISP Object System (CLOS), the object-oriented substrate on which all system components are built. All user interface components of D-OMAR, the graphical editors and browsers, the simulation control panel, and the timeline displays, have been built in Java. D-OMAR is open source software available at http://omar.bbn.com. In addition, the website has D-OMAR references and an online user manual. D-OMAR runs in Windows or Solaris and requires Allegro Common LISP and the Java Development Kit. External simulations can communicate with D-OMAR via HLA, CORBA, or a higher performance native-mode D-OMAR communication protocol.

### 6.4.2 Theory of Operation

The D-OMAR architecture is not limited to any particular theoretical approach in the design of human performance models. Rather, the flexible architecture enables users to develop models according to their own psychology, philosophy, or inclinations. The underlying theory driving the behavior of an agent is whatever the user chooses to implement as long as it is of the perspective that the agent demonstrates both proactive and reactive behaviors.

Although D-OMAR does not preclude any theoretical approach, it certainly was developed as a framework to facilitate multi-tasking activities in complex environments. In fact, this framework was pursued in D-OMAR (Deutsch, 1998) in part because of research by Dennett (1991), who stated “all varieties of perception—indeed all varieties of thought or mental activity—are accomplished in the brain by parallel, multi-track processes of

interpretation and elaboration of sensory inputs." Following Dennett's suggestion, the architecture (see Figure 8) was developed so as to *not* employ an executive or controlling process typical of the cognitive architectures described in this paper.

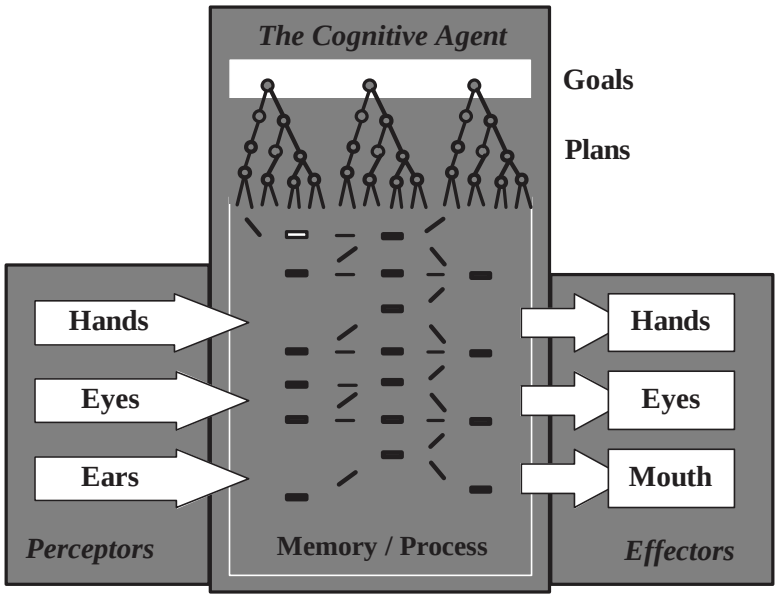



**Figure 8 The D-OMAR Cognitive Agent**

D-OMAR models contention between competing tasks in two ways. The first method simply inhibits or interrupts, depending on priority, parallel tasks that require the same perceptual or motor movement resources (e.g., two tasks that each require the dominant hand of an operator). The second method models contention and resolution based on established policy (Deutsch, 1998). Rather than being the subject of a rule-based decision, established policy-driven behaviors are viewed as a cognitive form of automaticity (Logan, 1988). Policy-based decisions are viewed, not as the product of a centralized executive process (for which there is little evidence), but rather as the outcome of contention among the particular subset of tasks competing to execute in response to events initiated externally or internally. Events, be they externally or internally initiated, impinge, not on short-term memory, but on activated long term memories in the form of schemata with well established policy-based priorities. When contention between tasks is present, policy-based priorities determine the next action. In addition, policy-based decision determines if interrupted tasks are reinitiated from the beginning, resumed from the point of interruption, or suspended altogether. A simple, but clarifying example of policy-driven behavior is communication between controllers and the flight deck. Conversations between individual aircrew members of the flight deck are immediately halted when a radio message from a controller is heard. The flight deck listens to the message, even if it is directed towards another aircraft. Likewise, flight deck-initiated communication must wait until the party-line is clear before the aircrew member can speak.

Because D-OMAR is an event-based simulation, it is more computationally efficient than time-incremented, cognitive simulations. D-OMAR accommodates the particular and

varied time steps at which each of several concurrent processes can be expected to operate. On the other hand, in time-incremented simulation environments such as Soar or EPIC, each decision might be revisited numerous times before it is resolved and the concurrent nature of the ongoing tasks might dictate that several separate rule sets be evaluated at each tick.

### 6.4.3 Error Prediction Capabilities

D-OMAR is not limited to any particular theoretical approach in the design of human error models so the modeler's theory, skill, and resources becomes the limiting factor in the development of error prediction capabilities. From a more pragmatic perspective though, errors associated with working memory (e.g., forgetting) that are readily modeled in ACT-R, are an area that will need to be addressed in the D-OMAR model.

The policy-based approach for modeling contention between competing tasks provides a framework for exploring human error due task distraction, task disruption, multi-tasking, or attentional narrowing. In addition, D-OMAR is well-matched for human error production due to the mismatch between task demands and resources. The detailed modeling of the human and the system provides a straightforward means for establishing whether an operator would perceive cues from the environment. This type of error directly supports the *failure to monitor or observe data* in the Model of SA and *information errors* in the Model of Internal Human Malfunction.

Finally, it is important to note that although no D-OMAR research specifically aimed at error modeling is yet complete, work is underway on the NASA runway incursion study.

## 6.5 SAMPLE

### 6.5.1 Overview

The Situation Awareness Model For Pilot-In-The-Loop Evaluation (SAMPLE) is an information processing model of an operator of a dynamic system grounded in modern control and estimation theory. SAMPLE uses a probabilistic representation of human information processing and situation assessment as the foundation for modeling complex decision-making behavior in multi-task environments. SAMPLE was developed by Charles River Analytics with support from Wright-Patterson AFB. The agent-based architecture is used to represent individual human entities. Entity representations to date have included air combat pilots, commercial pilots, air traffic controllers and airline dispatchers. Because it is agent-based, the architecture is intended to be embedded within a new or existing simulation. SAMPLE agents can run on Windows NT, Windows 2000, SGI Irix 6.5, and SunOS. The modular architecture is written in C++. A SAMPLE agent can be constructed without writing any new code, provided the model is limited to using the currently implemented technologies of fuzzy logic, Bayesian reasoning, and rule-based expert systems. Integration of the developed agent model within a target simulation environment requires additional C++ programming. Currently, building knowledge representation into an agent is a rather labor-intensive process. However, work is underway to address this issue through an US Air Force sponsored effort to develop a Graphical Agent Development Environment (GRADE) to provide easy-to-use graphical interfaces to support not only agent construction, but also validation and visualization of cognitive functions.

### 6.5.2 Theory of Operation

As the name implies, the Situation Awareness Model For Pilot-In-The-Loop Evaluation was developed to emphasize situation-centered (or SA-centered) decision-making (Mulgund et al, 2000). There is a substantial difference between the SA-centered approach and the conventional decision-centered approach. The conventional approach views the decision maker as “faced with alternatives, and considering the consequences of each alternative in terms of analysis of future states (odds/probabilities) weighed against alternative goals” (Klein, 1989). In the SA-centered model, no alternative is considered; instead, SA becomes the focus of all human actions. It defines a decision maker’s view of the environment and characterizes the information needs that drive his/her experiential (if-then) decision-making. This view of human decision-making provides the basis for the intelligent agent representation in SAMPLE.

The current version of SAMPLE encapsulates many theories that have been incrementally integrated into prior versions of SAMPLE and its predecessors. A more complete description of SAMPLE is presented by Mulgund et al (2000). The SAMPLE block diagram is depicted in Figure 9. The block diagram is separated into two major components: one representing the aircraft and the other representing the pilot. The display/control module acts as the interface between these two components. Although not obvious from the figure, the path between *Filter Continuous Inputs* and the *Control Channel* represents skill-based behavior while the path between *Extract Features* and *Select Procedures* represents rule-based behavior.

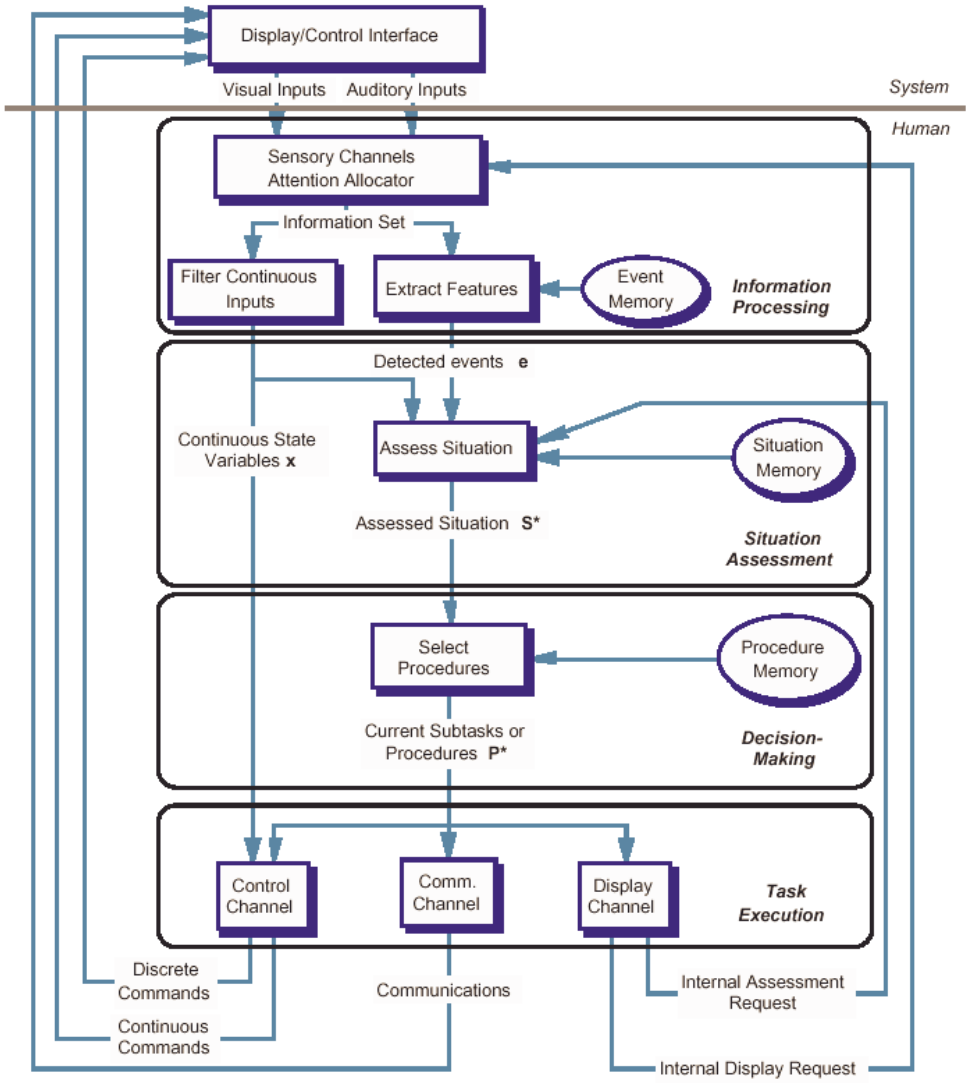



**Figure 9 SAMPLE Block Diagram**

*Sensory channels* of the pilot are driven by the *display/control interface.* For the world external to the cockpit, the visual channel simply receives map-based navigation information based on vehicle state and terrain. Within the cockpit, the visual channel receives information displayed on the instruments. The auditory channel is used for communication and auditory alerts. Lastly, there is a tactile/vestibular channel for cues not related to vision or audition such as rotational accelerations.

The *attention allocator* limits what flows into the pilot's sensory channels as well as deciding how to allocate attention among competing information sources. The *information processor* consists of two sub models, a continuous state estimator and a discrete event detector. The estimator uses a Kalman filter for estimates of the current vehicle/system state. The discrete event detector determines whether external events are perceived by the

pilot based on the dynamic information available at that instant combined with fuzzy logic. (Zadeh (1973) proposed fuzzy logic to deal with uncertainty in human decision-making.)

The *situation assessor* employs Belief networks (Pearl, 1988) to represent probabilistic reasoning in the presence of uncertainty. Belief networks provide the capability and flexibility of modeling SA without arbitrary restrictions. They provide a comprehensible picture of the SA problem by indicating the dependent relationships among the variables, both high-level (symbolic) and low-level (numeric), that are relevant to the SA problem. Compared to neural networks, this approach provides a clearer view of how each individual piece of evidence influences the high-level situation assessment. Each piece of evidence can be updated incrementally as perceptions from the environment arrive. To sum it up, Belief networks can be said to emulate a skilled human's information fusion and reasoning process in a multi-task environment. In its application, Belief networks enable the situation assessor to combine the estimated states and detected events to generate a situation state. This assessed situation state in actuality is a multi-dimensional vector defining the occurrence probabilities of the possible situations facing the pilot. For model tractability, a pre-defined set of static candidate situations are assumed based on their task relevance.

The *decision maker/procedure selector* generates a selected task or procedure based on the assessed situation state. Skill-based and decision-making behavior is implemented as a production rule system, with a general structure given by:

*If (Situation = Si) then (Procedure Set = Pi)*

This supports selection of a procedure set that is pre-assigned to the assessed situation specified in the human operator's mental model. The procedure set, and its linkages to the associated situation, are maintained in a procedural knowledge base. A procedure set may contain one or more sub-procedures. After a procedure set is selected, each firing of a sub-procedure is determined using a forward chaining production system in response to event and state evolution.

The three types of actions modeled within SAMPLE are *control actions, display requests, and communications.* Control actions fly the aircraft. Display requests result from the need for specific information and focus attention to a particular information source.

### 6.5.3 Error Prediction Capabilities

SAMPLE maintains a SA-focused approach for the agents that are modeled, whether they are controllers, air combat pilots or commercial pilots. Given that the error taxonomy developed by Endsley focuses only on errors due to degraded SA, the SA-centered decisions in SAMPLE provide a solid foundation for investigating these types of errors. Assuming a validated SA model (which is arguably a very difficult undertaking for Level 2 and Level 3 SA), the SAMPLE situation assessor could be employed as a powerful tool for investigating SA errors. One method is to look at the range of actions a pilot would choose to perform based on sufficient awareness and then systematically remove pieces of information from that awareness while keeping everything else the same. As each piece of

information is removed from his/her awareness, the pilot would begin to move from good decision making towards poor decision making until at some point an error or unsafe condition results. If the analysis was run in a parametric manner, a sensitivity analysis could determine which pieces of information are most relevant to error prone or unsafe conditions. Then mitigation strategies could be focused directly at keeping that information in the pilot's awareness.

With respect to specific types of errors in the taxonomies, skill-based and rule-based errors from the Model of Unsafe Acts could be emulated by stochastically changing thresholds in the production system.

Although no error modeling studies have been run to date, previous and current work in the domain of aviation and air traffic control suggests that SAMPLE should be a strong candidate for future NASA studies.

## 6.6 GLEAN

### 6.6.1 Overview

GLEAN (GOMS Language Evaluation and ANalysis) is a software tool for constructing simulations of a human user interacting with a simulated device or system. The simulated user is "programmed" with a GOMS model for how to use the system to perform tasks as they arise in one or more scenarios supplied to the software package. The GOMS model is specified in a formalized version of the structured-natural-language notation developed by Kieras(1988, 1997), which has enjoyed some popularity with user interface designers and researchers.

The first version of GLEAN was written in LISP/CLOS (Wood, 1993), and was used in an interesting demonstration of interoperability with an experimental user-interface software-development environment (Byrne, Wood, Sukaviriya, Foley, & Kieras, 1994). The second version (Kieras, Wood, Abotel, & Hornof, 1995), also in LISP/CLOS, was developed further under ARPA support and was validated against some empirical data collected by Gong (Gong & Kieras, 1994). A third version, GLEAN3, was then developed in C++, and was based on a simplified version of the EPIC architecture for modeling human cognition and performance (Kieras & Meyer, 1995), and has been used in some advanced modeling projects (Santoro, Kieras, & Campbell, 2000). Wood (2000) developed a variant of GLEAN3, called EGLEAN (Error-extended GLEAN), which represented important hypotheses about the sources of user errors and how users would recover from their errors.

The current version of GLEAN3 is being used for modeling human performance of a team of operators of proposed advanced tactical workstations in a project conducted by the U.S. Navy (Santoro, et al., 2000). This work has led to a further elaboration of the tool to allow it to easily represent complex and sophisticated tasks. The groundwork is underway for an even more advanced version that will have the EPIC architecture as a substrate. Since EPIC is currently the most accurate and comprehensive modeling system for representing human abilities and limitations in complex multi-task situations that stress perceptual and motor systems, this future version of GLEAN will have much greater accuracy, fidelity, and flexibility than any other GOMS tool, but at the same time, because it is programmed in a GOMS language, it will be far easier to use than comparably accurate research-grade modeling systems.

#### 6.6.1.1 GOMS

The best developed engineering technique for gaining deep insights into user needs is GOMS analysis (Card, Moran, and Newell, 1983). GOMS is an acronym that stands for Goals, Operators, Methods, and Selection Rules, the components of which are used as the building blocks for a GOMS model. *Goals* represent the goals that a user is trying to accomplish, usually specified in a hierarchical manner. *Operators* are the set of atomic-level operations with which a user composes a solution to a goal (Note: throughout this discussion, *operators* refers to GOMS operators. Human system-operators are referred to as users). *Methods* represent sequences of operators, grouped together to accomplish a single goal. *Selection Rules* are used to decide which method to use to achieve a goal when several are applicable.

Selection rules for a goal are grouped into a single collection called a Selection Rule Set. The set of Methods and Selection Rule Sets necessary to complete a task are called task procedures, which represent all of the procedural knowledge necessary to complete a given set of tasks. A task instance description represents the declarative knowledge necessary to perform a specific instance of the task. For example, a telephone dialing procedure might be very general, allowing it to be used to dial any number. A task instance description for that procedure would include a specific number or person to dial. Long-term memory (LTM) is another form of knowledge that can be necessary to complete a task. For instance, a user's LTM may include an item representing that the telephone number for an emergency is "911." Task procedures, task instance descriptions, and long-term memory are referred to collectively as a GOMS task model. This is differentiated from the conceptual GOMS model, which represents the ideas that constitute the family of GOMS techniques.

Most GOMS techniques are at least partially based on a simple cognitive architecture known as the Model Human Processor (MHP). This representation of human cognition consists of separate components for cognitive, motor, and perceptual processors (and associated buffers), as well as for long and short-term memory. The components of GOMS map onto this model in one form or another. For instance, control in the MHP is provided by the cognitive processor, where execution of methods and selection rules is assumed to take place. Likewise, the execution of operators can be seen as the issuance of commands by the cognitive processor to the other components.

#### 6.6.1.2 GOMS Analysis

GOMS models are used to represent the procedural knowledge required to perform tasks from the user's perspective. That is, they represent the "how to do it" information about a task, rather than "how it works" (Kieras and Polson, 1985). This makes GOMS useful for answering questions external to the user, such as "How long does this task take?," but not for questions related to aspects such as user acceptance (Olson and Olson, 1990).

GOMS models can form the basis for both quantitative and qualitative predictions of how well people can perform tasks using a particular interface, but GOMS analysis cannot replace empirical user testing entirely. It can, however, catch many flagrant usability problems (c.f. Nielsen and Phillips, 1993), and can replace expensive user-testing during initial design iterations. This is especially important when the intended user audience consists of highly skilled people whose time is expensive and difficult to obtain. In addition, GOMS can be used very early in the design process, before an interface is built, to rapidly explore the design space. John and Kieras (1996a) review several design cases highlighting the successful use of GOMS in a wide variety of domains.

A GOMS analysis consists of two major parts: defining a general model of the users of the system to be examined (and possibly portions of the system itself), and running that model on a set of task instances. The model consists of the procedures for a task and a set of task instance descriptions. Creating the model requires a prior task analysis and a determination of representative scenarios to include in the task instance description. To run the model, the

analyst must enumerate the steps taken from the methods in the task procedures, as specified by the information in the task description, and use this enumeration as the basis for calculating performance statistics. Kieras (1997) provides a detailed treatment of GOMS analysis.

GOMS analysis starts with a task analysis, and consists of working out the task procedures that users will perform to accomplish their goals using the new system. These procedures are then written in a GOMS notation consisting of a series of steps that will accomplish each possible user Goal. Steps consist of a series of operators (actions) and are grouped into methods (procedures) which can also include sub-methods. The result is a hierarchical decomposition of the task procedures. Given a set of top-level user Goals, the complete GOMS model specifies exactly which sequence of Operators needs to be executed to accomplish each Goal using the interface under design.

Thus the GOMS model is a complete description of the human user's procedural knowledge that a specific user interface design will require. Empirical research has shown that the amount and complexity of the Methods predicts the time to learn the procedures. Furthermore, the total execution time for tasks can be predicted by tracing the execution pathway through the Methods and summing the time for the Operator execution, using standard parametric time estimates for the different primitive Operators. In effect, the GOMS model acts as a simulated user, and so it can be used to predict the usability properties of an interface design, instead of empirically measuring these properties using human users operating expensive prototypes. Thus the design can be rapidly iterated to maximize usability early in the design process; the expensive human testing can be reserved for final testing and issues that the GOMS model cannot yet address.

While GOMS analysis is straightforward, and can be done by hand, it is complex and tedious enough to benefit from automated calculation This is because each step that is executed must be enumerated and its corresponding time estimate added to the overall execution time. Each scenario, as specified in the task instance description, can have a different execution path, so execution times must be obtained for each one. Any changes to the task procedures require that a new set of execution times be obtained. Furthermore, once the simulated user specified by a GOMS model is implemented in the form of a running computer simulation, there are additional possibilities for having the simulated user interact with simulated systems and other simulated users on a large scale. Such simulations of human performance, like EPIC models (Kieras & Meyer, 1995), are routine in cognitive science research, but they suffer from being extremely specialized and esoteric; in contrast, GOMS is simple and straightforward enough to be usable by people who are not cognitive scientists, such as user interface developers and human factors professionals. Over the years, several computational GOMS tools have been created (see Baumeister, John, & Byrne, 2000), but none have been developed commercially. Among these, GLEAN has been the best-developed scientifically, and benefits the most from practical experience with GOMS analysis.

### 6.6.2 Theory of Operation

GLEAN simulates the interaction between three main GOMS components: the simulated user, a task-model, and a simulated device interface. While similar in concept to the Card, Moran, and Newell (1983) Model Human Processor, the GLEAN architecture is actually a simplified version of the EPIC cognitive architecture (Kieras and Meyer, 1997). Figure 10 (modified from Kieras, 1998) shows the GLEAN architecture and the relationship between the simulated user and the simulated device. The area within the dashed box represents the portion of the architecture that remains fixed regardless of the task. The area outside the dashed box represents task-specific aspects of the model and is customized for each type of task. The analyst defines the device behavior, specifies the human procedures, and provides a set of task descriptions to run. Different scenarios can be tested by changing the task descriptions while leaving the device and procedures fixed. Device designs can be compared by changing the device and procedures, but leaving the task descriptions fixed.

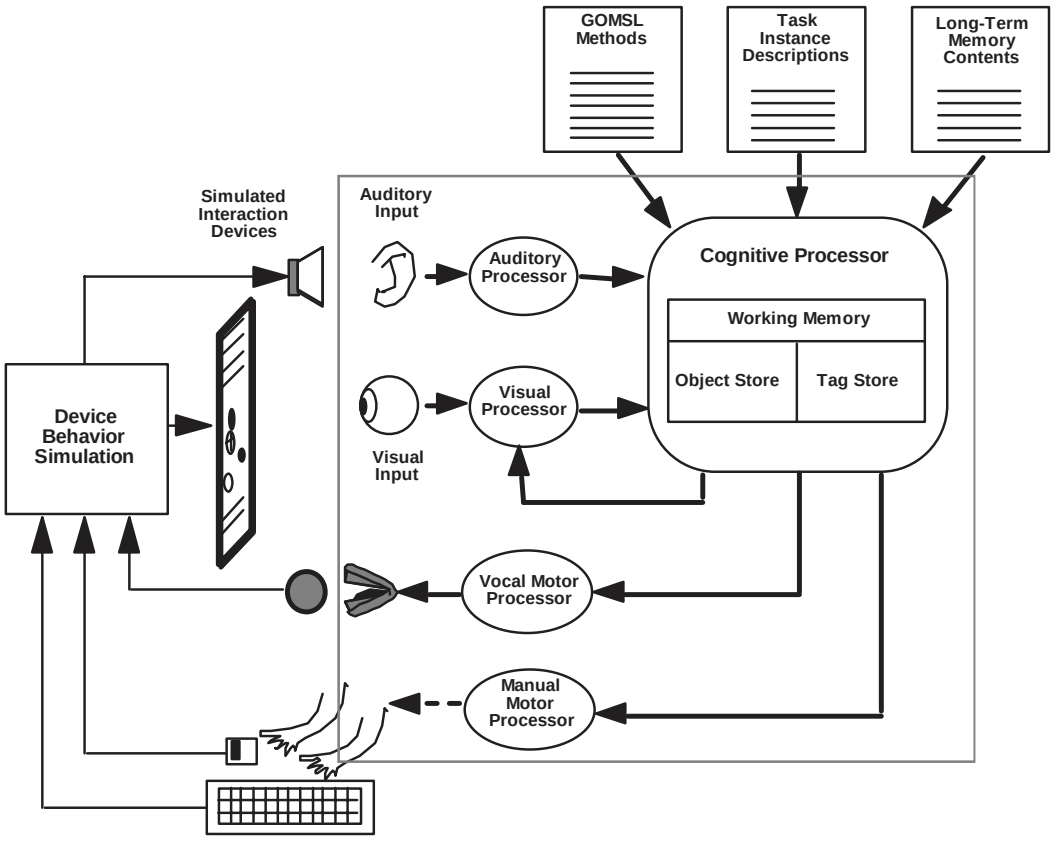



**Figure 10 GLEAN architecture and the relationship between the simulated user and the simulated device.**

At the core of the GLEAN architecture is the simulated user, formed around a central cognitive processor. It also includes processors for perceptual, motor, and working memory components. Each processor has a set of associated GOMS operators that they are

responsible for executing. The cognitive processor has the additional responsibility of controlling model execution.

The GLEAN architecture also includes a device processor for simulating the device interface. The device processor is a replaceable component that can be customized for different task domains. A customized device allows for realistic behavior modeling that can accurately determine many human performance limitations.

#### 6.6.2.1 GLEAN Control Structures and Operation

GLEAN takes as input a set of task procedures, a set of task instance descriptions, and a simulated device. The task procedures and task instance descriptions are written in a formalized GOMS dialect called GOMSL (GOMS Language) (Kieras, 1998). The simulated device is written in C++ and is compiled into GLEAN. As output, GLEAN produces a trace of the operators that the simulated user executes, as well as various run-time statistics such as execution time and working memory load.

The GLEAN cognitive processor maintains a goal stack which allows a top-down, hierarchical execution of a model's methods. It also maintains variables that represent the current method (the method currently being executed) and the current step (the step currently being executed).

GLEAN begins each simulation by getting the initial goal from the model, and reading the corresponding method from the method list. That method then becomes the current method, and first step in the method becomes the current step. Each step within the current method is executed in sequence. After a step is executed, the next step in sequence becomes the current step (the only exception to this is the Goto operator, which has the ability to change the current step to anywhere in current method). When sub methods are called, the cognitive processor creates a stack frame, saving a pointer to the current method and a pointer to the next step to execute in that method. The cognitive processor then pushes the new stack frame onto the goal stack and the called sub method becomes the current method. When a called sub method's execution is complete, the stack frame on top of the goal stack is popped and execution resumes with that frame's method and current step. The simulation is complete when there are no more steps to execute.

### 6.6.3 Error Prediction Capabilities

The use of GOMS analysis in error research has generally been as a method for framing and understanding a problem. Lerch (1988) built a series of GOMS models for the task of financial planning. He used the models as a method for rigorously defining the problem and as a basis for predicting relative error frequency between two interfaces. An important aspect of his approach was that he used GOMS to help explain how skilled financial planners translate complex problem solving goals into system goals. For instance, to accomplish the user goal of producing a profit-and-loss statement using a computer spreadsheet, the user must enter a series of calculations into appropriate cells in the spreadsheet corresponding to revenues and costs for the business in question. The cell calculations are system goals whose procedures may change depending on the software used. This demonstrated that GOMS was indeed suitable for addressing complex problem solving when computers are involved because there is always some amount of translation

between user goals and system goals. Lerch found that even small differences in the way the software required the user to enter cell formulas had a dramatic impact on how long it took the user to complete the task, the amount of additional mental calculation required, and the number of errors that were produced.

Smelcer (1989) created GOMS models of users forming SQL queries to determine possible error locations within a query formation. He used GOMS to formulate four possible explanations for a common SQL programming error. These explanations included necessary conditions for accepting each hypothetical cause for the error type. The GOMS models were used both as a systematic means for evaluating prior error data from SQL programming experiments and as a way to design new experiments that would capture the data necessary for determining causality. Using GOMS, Smelcer found that increased memory load resulted in increased SQL errors. He also used the GOMS models as a basis for a cost-benefit analysis of possible error reduction remedies (e.g. training and interface changes).

In a similar manner, Byrne (1993) developed GOMS models for an analysis of post-completion errors. He used GOMS to capture and formulate the procedural knowledge necessary to do a task thought likely to produce post-completion errors. From this he built a production rule model and determined working memory load for the task. The model predicted that increased working memory load would make post-completion errors more likely in specific locations in the task procedures. Experiments were designed using the model, the results of which confirmed the model's predictions.

Most of these projects focused on the effects that task structure had on error production, rather than focusing on error-producing mechanisms or error recovery. The one architectural assumption that each of these relied upon was that of a limited working memory. While Lerch and Smelcer focused on the number of items in working memory, Byrne also considered the decay of goal items in working memory. In addition to using structured methods to frame their problems, Smelcer and Byrne created computational models to make precise quantitative predictions. The computational models forced a higher degree of precision in the problem formulation (fewer unspecified assumptions) than non-computational models, and allowed for rapid evaluations under different assumed conditions.

To explore and enhance the error prediction capabilities of GOMS, Wood (1999) proposed a technique using GOMS for locating error prone locations within a task or interface. These techniques were applied to the redesign of a web-based financial application that was instrumented to collect human subject data (Wood 2000; Wood and Kieras, 2002). Modifications included both procedural changes, changes to the action sequences directly addressed by a GOMS analysis, and non-procedural changes, changes to the perceptual and conceptual qualities of the environment that are indicated by a GOMS analysis, but where a designer's expertise is still very important. To test the relative effectiveness of the procedural versus the non-procedural changes, only half the tasks were improved procedurally while the entire interface included the non-procedural improvements. The original and improved interfaces were then tested empirically to compare error types, error

rates, task completion time, and other usability factors. Over 200 errors were collected, and the results indicated a 34% decrease in errors to the tasks improved only in a non-procedural way, and an 83% decrease in errors for the tasks improved using both procedural and non-procedural means. These results demonstrate that using GOMS techniques can be very effective in predicting and reducing certain, prevalent types of human error. Wood (2000) also contains a detailed discussion of necessary components for extending any GOMS-like architecture to model human error. That chapter is included as part of this report in Appendix A.

#### 6.6.3.1 Using GOMS to Identify Error Sources

As proposed by Wood (1999), the basic approach for using GOMS to design for error is to first construct a GOMS model for the task and then to examine both static and dynamic aspects of the model to identify sources of error. These sources can be classified as either procedural or non-procedural aspects of the design. Procedural aspects concern the effects of the action sequences in the methods, while non-procedural aspects concern the perceptual and conceptual qualities of the objects in the environment. For example, consider the GOMS methods sketched in Listing x for selling a stock in a notional stock management application. The procedural aspects involve the steps that the user must perform to sell a stock, namely, looking for and remembering share and stock symbol (e.g. MSFT) information, then performing a selling procedure. The methods require finding and remembering two values, the stock's symbol and the number of shares to be sold, and then going to a second page where this information must be typed in; this pattern indicates a possible memory load problem. In addition, the procedures are relatively slow to execute because the application has not prepositioned the cursor in the first field to be typed into.

Method for goal: Sell stock
Step 1. Look_for symbol and store under <symbol_to_sell>.
Step 2. Look_for shares and store under <shares_to_sell>.
Step 3. Accomplish goal: Sell stock using <shares_to_sell> and <symbol_to_sell>.

Method for goal: Sell stock using <shares> and <symbol>.
Step 1. Press "Sell" button.
Step 2. Wait for page to load.
Step 3. Look_for "Shares" field.
Step 4. Point_to "Shares" field.
Step 5. Type_in <shares>.
Step 6. Look_for "Symbol" field.
Step 7. Point_to "Symbol" field.
Step 8. Type_in <symbol>.

Listing x. GOMSL excerpt showing potential memory overload error.

To illustrate the identification of non-procedural aspects, notice that by inspection of the methods, one can tell that on one page the user must visually search for the stock symbol, and on another page must find the proper field to type the information into. Thus the form,

typography, and layout of the symbol text and field label on the screen will affect the time and reliability of the "Look_for" operators in the methods. In contrast to procedural changes, correcting any problems with these aspects of the interface does not change the structure of the user's procedures, but will affect how quickly and accurately they can be executed. Also note that the model identifies the essential information for this task. Thus the GOMS model is a straightforward way to identify critical visual aspects for relevant interface elements.

A static analysis of a GOMS model considers just the content of methods in the model, such as that shown in Listing 1. The above examples of the appearance of the symbol information and field, and the lack of cursor-prepositioning can be deduced by such a static inspection, in which one simply examines the GOMS methods for certain patterns of steps and operators (see Wood, 1999). A dynamic analysis involves running the model as a simulation to execute a set of specific task instances in which all necessary information such as stock symbol and shares are specified. For example, the memory load during the execution of a method may be very high in one task instance (e.g. if multiple stocks must be compared and traded), but low in another. Likewise, some tasks may result in different patterns of visual search, some of which might be more error prone than others, depending on the non-procedural aspects of the information involved.

Using GOMS to design for error prevention yields three main benefits. First, GOMS models can address procedural sources of error by identifying locations, times, and conditions within a task where errors are likely. Second, GOMS models can be used as a road map to guide static and dynamic analyses of non-procedural sources of error. Third, since GOMS models are psychologically motivated and can provide specific information for design improvement, they address many of the criticisms discussed by Kirwan (1992). To help demonstrate these claims further, these analysis techniques were applied to the redesign of a web application and tested for their effectiveness at reducing human error.

## 6.7 APEX

### 6.7.1 Overview

APEX is a software architecture for modeling human performance in complex, dynamic environments. It consists of a suite of software tools for creating, simulating and analyzing human performance models, plus a methodology for using those tools effectively for a variety of modeling goals. In particular, APEX reduces time and expertise requirements for modeling behavior at levels of detail ranging from discrete perceptual/motor actions to intelligent management of multiple, long-duration tasks. Applications include time-analysis of skilled behavior, partially-automated human-factors design analysis, and creation of artificial human participants in large-scale simulations.

APEX was created by Michael Freed as part of his doctoral dissertation and continues to be developed by researchers at NASA Ames Research Center and elsewhere. It is an evolving system with applications and tool improvements continuously underway. APEX is envisioned as a distributed development effort. APEX uses Common LISP. Tutorials, downloads, and other documentation is available at the APEX website: www.APEXsystem.org. A depiction of the simulation environment is depicted in Figure 11.

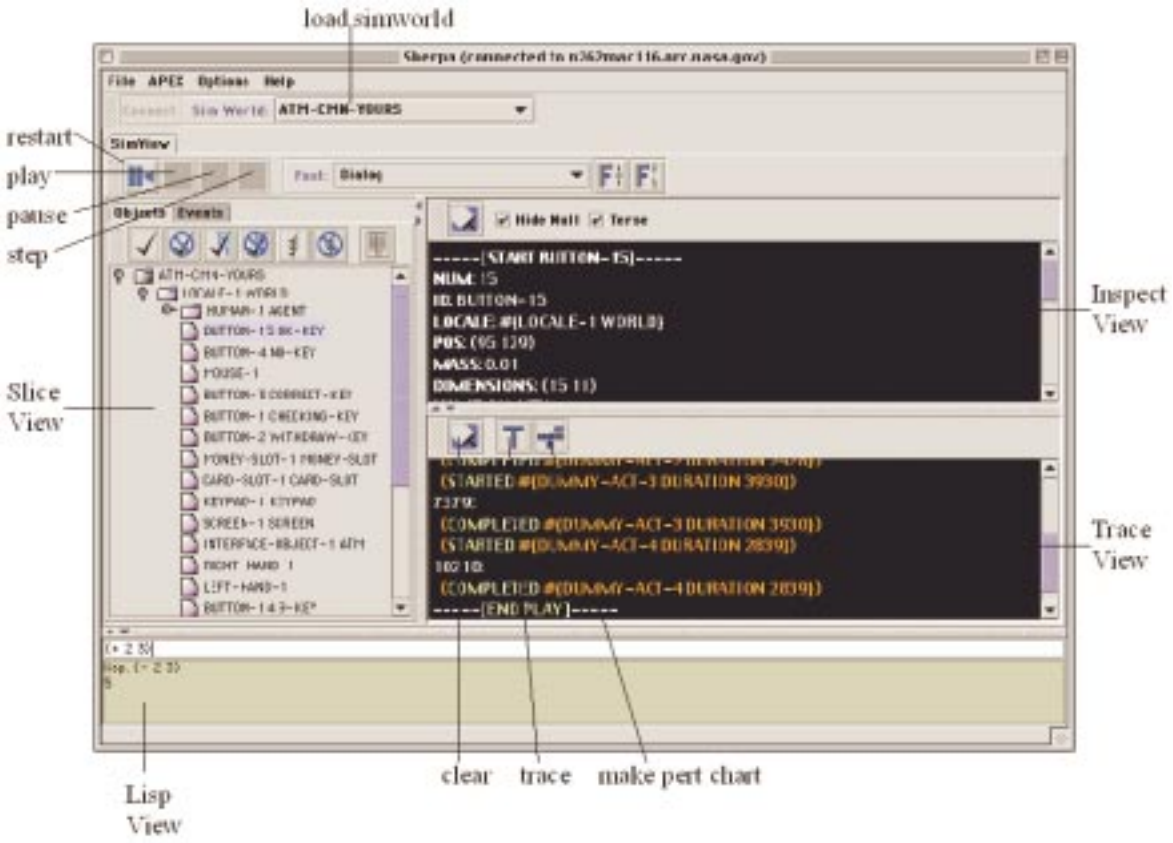



**Figure 11 APEX simulation environment**

### 6.7.2 Theory of Operation

Predicting performance in more challenging task domains requires an operator model that can function effectively in demanding task environments. Existing human models have typically lacked several very important capabilities including those needed to cope with varied forms of uncertainty inherent in many task environments; manage limited cognitive, perceptual, and motor resources; and, manage multiple, periodic tasks. These capabilities have been incorporated into a human operator model called APEX (Freed, 1997a; Freed, 1997b) by adapting techniques from the field of artificial intelligence.

APEX is based on GOMS (which is described in detail in Section 6.6.1.1), but places a particular emphasis on cognition, perception, and motor movement (Vera, John, Matessa, Freed, & Remington, submitted). Hence, the APEX version of GOMS is often referred to as CPM-GOMS. CPM-GOMS is the most powerful extension of the GOMS methodology, tying low level operators to timing predictions derived from the Model Human Processor (Card, Moran, and Newell, 1983). The advantage of CPM-GOMS is that models of tasks from one domain may carry over to other domains, allowing for model reuse through the creation of low-level behavior template libraries. A significant advantage of APEX is that it automates what had previously been a difficult, time consuming CPM-GOMS task – interleaving the low-level templates manually to carefully coordinate the cognitive, motor, and perceptual resources. The APEX user can perform these functions without specific expertise in cognition or CPM-GOMS.

#### 6.7.2.1 APEX hierarchical goal decomposition

The following information is presented verbatim from Vera, John, Matessa, Freed, & Remington (submitted). The hierarchical goal structure of a GOMS model can be expressed in APEX with the Procedure Description Language (PDL). In PDL, procedures (GOMS goals) consist of a number of steps (GOMS methods). These steps may be calls to other procedures or they may call tasks (GOMS operators) which make time demands on resources. The decision to perform a particular step can be mediated by preconditions (GOMS selection rules).

- **Serial step ordering** – With PDL, steps can be assigned serial order. This is done by having the precondition of one step be the completion of another step. Without such preconditions, steps will run in parallel, subject to resource constraints.

- **Cognitive, perceptual, and motor resources** – Tasks make time demands on one or more resources. Current resources include memory, vision, gaze, left hand, and right hand.

- **Virtual resources** – Virtual resources represent the intention to use perceptual and motor resources. Current virtual resources include gaze intent, left hand intent, and right hand intent.

- **Resource constraints** – Tasks are subject to certain resource constraints. These constraints are used by APEX to determine which tasks are able to run at any particular time. Tasks using different resources are able to run in parallel, but tasks

using the same resource are run serially. The ordering of tasks using the same resource can be manipulated by assigning priorities to tasks.

- **Task time assignment** – Tasks are assigned durations which can be constants or a function of the environment. For example, a button press task is assigned a MHP value of 100 ms, while a mouse movement task is assigned a time calculated by Fitts' Law. The overall time to run several tasks is calculated by APEX, which takes into account when the tasks start and what tasks may be running in parallel at any particular time.

### 6.7.3 Error Prediction Capabilities

In the domain of air traffic control, APEX has been employed to predict what Freed & Remington (1998) refer to as *habit capture* errors. This error type appears to be very similar to the *executing habitual schema* in the Model of SA error taxonomy. Since operators often recognize their error shortly afterward, a habit capture error appears to occur not because of a failure to successfully retrieve information, but rather a failure to make a retrieval attempt. Habit capture errors occur when the human fails to allocate the minimal cognitive resources to retrieve task-relevant information from memory. Without this information, decision making mechanisms rely on default assumptions, resulting in error if these assumptions are not suitable to the situation.

In APEX, the method to support this type of error requires the modeling of continuous coordination of concurrent activities such as task interruption, task switching, task and resumption. In addition, the model must account for uncertainty inherent in complex, dynamic environments. Lastly, the model must monitor for and recover from task failure.

As APEX is a GOMS-based approach, other errors that APEX should be suitable for predicting are discussed in Section 6.6.3.

## 6.8 COGNET

### 6.8.1 Overview

COGNET, for COGnitive NEtwork of Tasks, is a modeling framework developed by CHI Systems to simulate human expertise (Zachary, et al., 1992). It attempts to codify ways that subject matter experts in a given area might receive, process and act on information represented in a computer simulation. iGEN™ is the software tool that refines and then executes COGNET, interactively with other programs or independently. iGEN creates pseudo-operators or agents, based in COGNET, to perform an unscripted interactive decision-making processes, in real time or scenario time. This makes it comparable to ACT-R, EPIC and OMAR. However, rather than follow expert rule-based steps to a solution, like many other tools, iGEN has been refined over the last 12 years to simulate how humans solve problems. The agents created by iGEN do not currently mimic flawed human decision making, only expert actions. It runs executable cognitive models using an interface shell generated by C++ to obtain information about the real or simulated world to make decisions appropriate to the simulation. See http://www.chiinc.com/index.shtml for citations and other examples.

The COGNET/iGEN system will run on a Windows based PC (Windows 95, 98, NT version 3.51 and above) with a Pentium II chip, 64 MB of RAM and 10 MB of hard disk space at a minimum. For one machine, a developer's and a modeler's license, which includes all the necessary software and software keys, costs about $19,995 with a yearly maintenance fee of $2,995. Training is supplied with a license and usually takes about a week before simple models can be constructed.

An important practical consideration of simulation architecture is the cost and the training time to produce useful models. CHI systems believe the learning time for COGNET and iGEN is generally thought to compare with other modeling tools like Soar.

### 6.8.2 Theory of Operations

COGNET and iGEN™ are PC-based tools that allow a simulated system operator or agent to recognize a problem and act on creating a solution germane to the scenario. They can interact with other databases or tools to accomplish the desired goals of the simulation. COGNET is a system of task networks based on subject matter expert inputs that simulates their decision making processes. iGEN is a software tool that aids in refinement of the COGNET model for execution and can provide for sensory perceptual inputs to the model as well as outputs that can feed other computer tools (Zachary (2000). COGNET models have been designed for a broad range of applications. The combination of COGNET and iGEN has primarily been applied to the creation of expert models for instructional agents. These include ship based (AEGIS) air defense systems, piloting, including piloted and remotely piloted aircraft and ships, telecommunication system operators, expert medical tools such as oncological diagnosis and command and control tasks for Army, Navy and Air Force assets.

COGNET is not restricted to any unified theory of human memory or memory processes. Instead, it allows for different theories to be embedded and actualized. Expertise

representation is based on naturalistic theories such as recognition primed decision theory. Subject matter expert knowledge and decision steps are structured in large chunks as opposed to vast minutiae of knowledge details common in other models. The system matches the best chunk with the current scenario information to affect a solution. COGNET uses additional, smaller algorithms to estimate sensory or motor constraints, such as task time or accuracy requirements, in completing a task. For example, how long and the retention needed for reading a portion of text can be estimated to varying degrees of time/accuracy granularity. It can model expert competence for training or guidance or model simulated behavioral performance for evaluating system designs. This capability allows human frailties like fatigue to effect the model's execution. COGNET has the capability to be aware of characteristics of its internal information processing ability. This metacognitive feature is called cognitive proprioception. It provides the model with metacognitive controls over decision conflicts that might develop during execution. An example for Combat Generated Forces model is shown in Figure 12, which depicts the incorporation of metacognitive self awareness in resolving conflicts between input demands and outputs required of cognitive models.

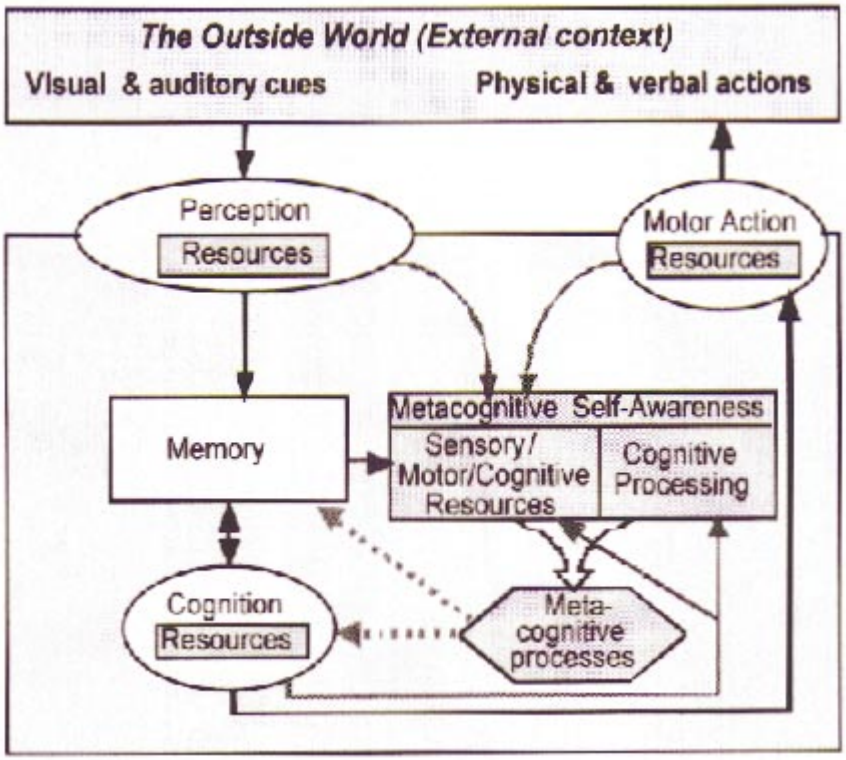



**Figure 12 - A schematic of a COGNET computer generated forces model.**

COGNET allows the creation of human performance models that represent expert human knowledge and information manipulation in a given computer scenario. COGNET modeling components consist of a 'blackboard' user interface where the problem to be solved or the expertise to be codified are represented and stored in three knowledge bases; perceptual, declarative/problem, procedural/control/action knowledge bases. The perceptual knowledge base, called the 'perceptual daemons', is a mechanism for sensing and checking on the external environment. Declarative knowledge refers to

conceptualizations about the situation. The procedural/action knowledge describes what can be done about or to the situation. Behavior is generated or tasks are completed after decision processing mechanisms are applied to these internal expertise and external contexts (Ryder et al, 2000). iGEN (Zachary, et al., 1996) is a software tool that simulates the ways that expertise models created by COGNET are processed and actualized. It can be focused to model team interactions.

The COGNET cognitive models can be generated in text form and compiled or in the text editor supplied (WIMP). Subject matter experts have built up their actions and decision structures from years of training. Accordingly, the COGNET and other cognitive models must be painstaking in detailing the procedures the expert follows into the model. There is onscreen help and extensive debugging tools to aid in model development.

### 6.8.3 Error Prediction Capabilities

iGEN most often is used to build (1) synthetic agents that are based on models of expert cognition and behavior, (2) training-application agents that detect deviations between trainee behavior (e.g., key-presses) and modeled expert/ideal behavior. In some cases, it attempts to link behavioral deviations to candidate rule and/or knowledge deficiencies. Finally (3) iGEN can generate decision support/intelligent agents that detect changes in the external environment and/or track users' activities in order to provide decision/performance guidance at appropriate times. In the latter application, there is little need to replicate human errors (just to detect and/or prevent them from occurring). However, the architecture generally supports the modeling of error commission.

The agents created by iGEN do not currently mimic flawed human decision making, only expert actions. Although iGEN generally supports the concept of producing errors of commission, it does not provide an elegant means of producing all errors that could occur. In these cases, the modeler may need to define specific points in task performance where they can occur, and probabilistic equations that drive their occurrence may need to be defined in the *shell* of the model (the shell is a C++ program that allows the iGEN model to coordinate/communicate with the external environment. It is also used to represent complex and mathematical functions that iGEN cannot easily accommodate).

However, the iGEN architecture can easily represent other types of errors, especially those that have to do with goal selection and attention, using its sophisticated controls for managing multiple goals or tasks. These controls include:

- a goal/task scheduling mechanism,
- priority formulas,
- goal triggers,
- demons that constantly monitor the task environment for new information, and
- the recognition of new information as a trigger to switch to a new goal.

The iGEN architecture continues to be improved. For example, improvements made to the in-house version (not yet available to the public) include more sophisticated management of multiple goals and tasks (e.g., dynamic priority formulas) and the capability to equate

the current blackboard configuration to an event, thereby making it possible to represent event recognition in a more realistic manner.

## 6.9 Soar

### 6.9.1 Overview

Soar is a general and robust architecture that can be used to develop complex human-like simulations as well as smaller-scale psychological models. There exist a variety of publications describing the Soar theory, the Soar cognitive architecture, the Soar implementation, and individual Soar models. In addition, the computer implementation of the Soar architecture is freely available from http://ai.eecs.umich.edu/soar and runs on all common operating system platforms, including MacOS, Windows, and Unix. Because Soar emphasizes a small set of uniform low-level principles, it is somewhat difficult to learn, and sometimes difficult to create particular types of models. The Soar theory sacrifices flexibility for the power of uniform models that include identical theoretical assumptions, and this can have an impact on ease of use. However, various members of the Soar community have recently created a number of development tools (as well as adjustments to the architecture's implementation) to make it easier to learn Soar and construct models within Soar.

Soar is a theory of human cognition that attempts to unify all intelligent behavior within a relatively small set of functional principles (Laird, et. al. 1987). The theory is also realized as a software architecture that implements the principals, allowing the construction of fine-grained, executable models of various types of human reasoning and learning. The Soar architecture grew out of 1970s and 1980s research on production systems, and began serious development in 1982 by John Laird, Paul Rosenbloom, and Allen Newell at Carnegie Mellon University. Begun as an architecture for creating generally intelligent systems, the principles behind Soar were increasingly mapped to human cognition, culminating in Newell's (1990) proposal of Soar's principals representing a "unified theory of cognition". As the architecture has matured, it has also been used to develop a wide array of psychological models and applied AI systems (Rosenbloom, et al. 1992). Perhaps the most ambitious application of the architecture was in the development of TacAir-Soar (TAS) (Tambe, et. al. 1995; Jones, et. al. 1999). TAS is an autonomous synthetic entity that carries out airborne missions in simulated military aircraft. TAS "pilots" have the knowledge to fly simulated fixed-wing aircraft for a variety of combat missions, including tactical missions, reconnaissance, and command and control. TAS is widely used in defense simulation technology training exercises and experiments, as well as serving as the basis for a number of research studies into complex human cognition (e.g. Nielsen, 2000).

### 6.9.2 Theory of Operations

Soar grew out of Newell's prior 30 years of research developing symbolic approaches to cognitive modeling and artificial intelligence (AI) (Newell, 1990). This tradition is adopted by many contemporary cognitive architectures, essentially viewing cognition as the application of *operators* in the course of a deliberate search through various *problem spaces*. Soar's development has consistently focused on finding a small set of core functional principles that, taken together, can generate general intelligent behavior. In this vein, the architecture posits (among other things) a single low-level representation for long-term knowledge (production rules), a single low-level representation for working memory (data objects with associated attributes and values), a single low-level mechanism

for learning new long-term knowledge (chunking), and a relatively simple cycle of operations for producing behavior (the decision cycle).

Once these core principles had solidified, Newell (1990) proposed their union as a candidate unified theory of cognition. He based this proposal on the principles themselves, the tradition of symbolic cognitive modeling, and experience with a large number of cognitive and AI models developed within Soar. Figure shows a diagram of Soar's basic architecture. It incorporates three memory stores; production memory, working memory, and preference memory. Long term memory storage is represented by production rules, which can be thought of as a set of logical 'if-then' conditional statements, or alternatively as a set of relational pattern matchers. Working memory represents a Soar agent's current perceptions, beliefs, goals, intentions, and actions. These are all represented in a uniform object-attribute-value symbolic representation, and they can represent the range of internal representations required for general intelligence, such as hypothetical future states for planning, correct (or incorrect) situation assessments, partially instantiated schemas, operators, etc. In addition to these memory areas, Soar includes a learning mechanism called 'chunking', which can store new rules into production memory.

As suggested above, Soar's basic unit of operation is the *decision cycle*. In a single decision cycle, Soar repeatedly fires and retracts the actions of productions, until it reaches *quiescence*, a state where no more changes to working memory are proposed. Within the decision cycle, all productions that match at a particular moment in time fire in parallel. It is up to the automatic mechanisms in preference memory to resolve any potential resource conflicts. This allows Soar systems to pursue multiple courses of action in parallel as long as they do not conflict with each other. When there are potential conflicts preferences and the decision cycle combine to impose serial action instead of parallel. As a very simplistic example, if a system may have simultaneous goals to not be hungry and not be thirsty, each achievable by eating and drinking, respectively. But an agent cannot eat and drink at the same time. This might be represented in the following if…then statements:

If we desire to be not hungry AND
our current state is hungry
Then we propose to eat.

If we desire to be not thirsty AND
our current state is thirsty
Then we propose to drink

These two statements impose a potential resource conflict, so there must be additional knowledge (which may ultimately come from a variety of sources, including chunked past experiences) that expresses context-dependent preferences between these two actions. Both of these productions may fire in parallel, put the preferences only allow one operator (eat or drink) to be selected as the next deliberate action. If the system does not have appropriate preference knowledge, the conflict remains, and Soar automatically generates an *impasse*, with an associated explicit goal to resolve the impasse (by determining a missing preference that would resolve the conflict). Each decision cycle ends with either

the selection of an operator to apply or the creation of an impasse. Either of these events leads to changes in working memory, which serve as initial production triggers at the beginning of the next decision. In addition, prior to each decision, simulated sensory systems are allowed to change restricted portions of working memory, which will also lead to new sets of production matches.

For simple problems, a Soar agent's top-level goal would be well defined, and productions that detect achievement of the goal would issue a command for the agent to halt. For more complex agents, the top-level goal is often much more vague (such as "be a pilot" or "be a robot"), and the agent's essentially run until a human intervenes. As with any AI system or cognitive model, the initial long-term knowledge must be engineered by a human modeler. Depending on the application, the modeler may be a psychologist implementing a cognitive task analysis, or it may be a knowledge engineer working in concert with subject-matter experts to encode expertise for a variety of domain tasks.

One of the most complex intelligent systems developed within the Soar architecture is TacAir-Soar (TAS). TAS is well suited for high fidelity training simulations. This type of simulation requires automated agents or *synthetic forces* that generate realistic, intelligent behavior at the level of individual human performance. TAS models this level of behavior for pilots in the tactical air domain. Among the advantages that Soar provides to TAS are appropriate symbolic knowledge representations, an explicit structure for goals, preference-based decisions, and the ability to generate complex goal-directed and reactive behavior in real-time (Soar includes a very efficient pattern-matching algorithm for its productions). Because TAS is based on a model of human cognition, it chooses the actions it takes and makes decisions for reasons similar to those which a human would employ. It enjoys an enormous knowledge base consisting of over 7000 rules, making it among the largest of the fielded expert systems. These rules contain all the knowledge for perception, decision, and action that a synthetic pilot needs for all the missions that TAS carries out. For any particular mission, only a subset of that knowledge base actually matches and fires, but all the rules are in there "just in case".

### 6.9.3 Error Prediction Capabilities

The Soar architecture focuses on supporting models of the cognitive band of human behavior (deliberate actions that occur at about the 50-100 msec time scale). This implies that Soar is strongest at modeling symbolic knowledge-level human behavior, and knowledge-level errors. Soar itself does not make particularly strong claims about perceptual and motor systems, so it would not impose any strong explicit constraints on errors in those systems. However, Soar does focus strongly on models that interact with external environments, so its cognitive components include explicit types of interfaces to simulated perceptual and motor systems. In addition, the Soar architecture does impose constraints on how knowledge must respond to and compensate for errors in perceptual and motor systems.

Timing of behavior and errors is also strongly supported by the Soar architecture. Since its inception as a cognitive architecture, the Soar theory has made strong claims about the mapping of the decision cycle to the timing of neural and cognitive processes in the brain.

Older Soar models explicitly hypothesize times for execution of cognitive and interactive tasks, while new models are able to predict such times by running in real-time simulations.

Probably Soar's primary strength in modeling human error would lie in the knowledge-based ways that people commit reasoning errors or respond to perceptual and motor errors. Because Soar does not commit to particular perceptual and motor models, any errors from those sources would have to be hand-simulated. However, this is not unprecedented in Soar models. TacAir-Soar integrates with realistic sensor simulations in JSAF, which sometimes fail, and so the knowledge base includes knowledge of how to react when such failures occur. Sometimes, such perceptual failures lead to the same types of errors that human pilots exhibit. Additionally, some experiments have been carried out with models built within a hybrid architecture combining Soar and Epic, which *does* include explicit models of perceptual and motor processes.

Some research has also focused on memory retrieval errors within Soar models. One criticism often leveled at production systems is that their long-term memories are "too perfect". However, this is an over-simplification. It is true that the matching of individual productions is deterministic, but individual production firings are at the lowest level of the memory representation. For Soar models that implement concept learning, for example, it is still the case that the model must piece together various retrieved symbols, and it does not always know how to combine them correctly (Miller and Laird, 1996; Pearson, 1996. Work with Epic-Soar has also investigated using types of working-memory decay (similar in spirit to ACT-R), which requires a task model to focus explicitly on attention, visual search strategies, and compensating for forgetting (Chong, 2001). As each of these cognitive skills is explicitly represented to perform a task, it becomes a potential source of error for human error modeling.

There have been a variety of Soar models and research projects relevant to modeling human errors:

TAS itself must robustly respond to low-fidelity perceptual information. This means that it sometimes makes mistakes in identifying radar contacts and piecing together, for example, enemy formations and intent. These mistakes become exacerbated in the face of information and attention overload (presumably similar to humans) (Jones, et al. 1999).

A mechanism has been built into the Soar architecture to simulate cognitive slowing due to fatigue. The fatigue system artificially slows down the decision cycle. Because TAS already has to have lots of knowledge to respond to information failures (in order to remain robust and autonomous), this leads to behavior errors without adding any extra knowledge. For example, the slowdown can cause failures in identifying contacts, which leads to particular types of errors in situational awareness. The slowing also leads to missed opportunities, such as not firing a weapon in time, in which case TAS immediately has to take compensatory actions (such as evading instead of firing) (Jones, et. al. 1998).

Nielsen's group is working on the "robustness" project to make TAS fall back on generalized "mental models" when it detects itself performing any anomalous behaviors.

The mental models serve as a more abstract type of default knowledge that will be helpful in most cases, but will lead to specific types of errors in other cases.

Early work by Laird focused on "Recovering from incorrect knowledge" (Laird, 1988). This highlights the fact that Soar sometimes exhibits knowledge-based errors by learning incorrect production rules via chunking over particular experiences. Laird demonstrates how Soar can learn new (presumably correct) rules that "mask" the older incorrect rules (there is no mechanism for "forgetting" productions in Soar).

Polk and Newell accomplished some of the most serious psychological modeling ever accomplished with Soar, with their work on "Modeling human syllogistic reasoning in Soar" (Polk and Newell, 1988; 1995). A large amount of the data on how humans process syllogisms focuses on the particular types of errors they make. Polk used the principles within Soar plus a Soar-based implementation of "mental models" together with some simple assumptions about working memory limitations to reproduce many of the regularities (including errors) observed in humans.

Other work that focuses on learning also takes into account errors that arise from "bad" learning, and how to continue to learn to improve behavior. Some of this work has been performed by Laird, Hucka, Yager, & Tuck (1991); as well as the dissertation work of Miller (1993; 1996) and Pearson (1996). Much of this work did not focus specifically on modeling *human* error, but it does lead to predictions about potential human errors.

Lewis (1992) has done extensive work using Soar to model how humans process various perverse natural language constructs, such as garden-path sentences and deeply nested sentences. This work focuses specifically on modeling limitations and errors in the human ability to process such sentences.

Howes and Young (1991) used Soar to evaluate the potential "learnability" of different graphical user interfaces for a text editor. This focused explicitly on predicting the types of errors that different types of users might make.

Simon, Newell, and Klahr (1991) built a Soar model for "A computational account of children's learning about number conservation". This is again a model that primarily looks at errors as part of a learning process. But the key to the model is to match children's learning behavior, both in terms of when they act correctly and when they commit errors.

## 6.10 EPIC

### 6.10.1 Overview

The EPIC (Executive Process-Interactive Control) architecture (Kieras and Meyer, 1994; Meyer and Kieras, 1994) is a computational architecture of the conventional information-processing style (see the section on GLEAN). As such, it has a production-rule cognitive processor surrounded by sensory and effector organs, and perceptual and motor processors. More precisely, EPIC contains separate auditory, visual, and tactile processors for perceptual inputs, and separate vocal, ocular, and manual motor processors for producing actions. All of these are controlled by the cognitive processor.

The key relevance of EPIC here is that the sharpest and most profound limitations on human ability are in the use of the peripheral perceptual-motor systems, and how they are used to support thinking and decision making. Humans are limited in terms of visual and auditory input capacity, and also are limited to a single vocal output channel and also have a single manual output channel except in special circumstances involving extreme amounts of practice. But in addition, many thinking and decision-making processes require use of verbal working memory, or verbally assisted reasoning, which ties up the vocal and auditory channels internally. Using visual working memory or internal visualization strategies during thinking and decision-making will likewise conflict with the use of the eyes and the visual system for handling immediate tasks.

EPIC represents these peripherally based limitations in computational form more directly than any other cognitive modeling system, making it a good choice for modeling complex interactive tasks. In addition, EPIC has focused directly on the representation of multitask performance. The goal of the EPIC project has been to resolve the long-standing scientific puzzle that people can in fact multitask to some extent, but *only* to some extent (Meyer and Kieras 1997a, b). EPIC explains how the details of the task structure and requirements determine the boundary between successful performance of multiple complex tasks and serious failure due to information overload. EPIC models can thus identify the critical bottleneck points in task structures. Because EPIC already addresses the complex issues involved in multiple task performance, it is an important foundation for any study aimed at improving performance. Despite the apparent potential of EPIC for system design and error analysis (Kieras, in press; Kieras and Meyer, 2000, 1997; Kieras, Wood, and Meyer, 1997), it should be noted that the primary focus of EPIC research has been basic science (e.g. Kieras, et al. 1999; Meyer and Kieras, 1994; Mueller et al., submitted; Schumacher et al., 1999); EPIC has not been applied to system design.

### 6.10.2 Theory of Operation

***Perceptual Processors***

The perceptual processors in EPIC accept input from task environment and place symbolic descriptions of individual events into working memory for use by the cognitive processor. The auditory processor places time-stamped data items in working memory representing individual word- or phrase-meanings. The visual processor handles both detection events

(onsets and offsets) as well as perceptual events (such as color, size, etc.). The tactile processor places events into working memory indicating as feedback from motor actions.

***Cognitive Processor***

The cognitive processor functions as a central executive in EPIC. It controls the peripheral processors and is programmed with production rules. In addition, the cognitive processor operates on a 50 millisecond cycle-time where working memory items are matched against production rules at the beginning of the cycle, and actions resulting from executed production rules are performed at the end of the cycle. The major components of the cognitive processor are working memory and the production rule interpreter.

***Working Memory***

In EPIC, memory is divided into three sections: long-term, working, and production. Long-term refers to long-term declarative memory. Production memory refers to long-term procedural memory in the form of production rules. Working memory is a short-term declarative store which contains control information such as task goals and steps, as well as sensory, motor, and cognitive items. Collectively, these items are called working memory items (WMIs).

***Production Rule Interpreter***

The production rule interpreter used is the Parsimonious Production System (PPS; Covrigaru and Kieras, 1987). Like most production rule systems, PPS rules consist of a condition part and an action part. The condition part consists of a list of patterns, and the action part consists of a list of actions. When all of the patterns in the condition part are matched with WMIs, all of the actions in the action part are executed. Unlike most other production systems, PPS executes all matching rules on every cycle; there is no conflict resolution.

***Motor Processors***

The motor processors include an ocular motor processor, a vocal motor processor, and a manual motor processor. The ocular motor processor controls eye movements specified by the cognitive processor as well as reflex eye movements specified by the visual perceptual processor. The vocal motor processor is responsible for controlling speech as determined by the cognitive processor. The manual motor processor is the most fully elaborated processor in EPIC. It is responsible for controlling hand movements.

***Operation of EPIC***

EPIC receives input from the sensory organs through the perceptual processors, which then deposit detection and recognition representations regarding the input into working memory in the form of WMIs. The cognitive processor matches WMIs to production rules at the beginning of a cognitive cycle and at the end of the cycle produces output in the form of either WMIs or instructions to the motor processors. When a motor processor receives a command, it programs the action, after which the action is sent the output effector, where the action is executed.

### 6.10.3 Error Prediction Capabilities

Although no error modeling work with EPIC has been published, some research on modifying EPIC to produce errors has been conducted. It was found that by making minor changes to EPIC's working memory component, allowing goal and task elements to decay and to be corrupted (confused with similar items), a wide variety of error types were seen. These included errors of omission and commission, reversals, transpositions, capture errors and others. Since these basic (observable) error types are the building blocks for a large number of contextual errors, EPIC has very good potential as an error modeling system. Furthermore, many other error types, such as those that are caused by missing data and data corruptions, can already be seen in EPIC models of high performance tasks (Kieras, Wood, and Meyer, 1997; Meyer and Kieras, 1999). In addition, the types of errors that people make while managing multiple task strategies are very well represented (Kieras, et al. 2000; Schumacher, et al. 1999). It is common when building most EPIC models to start with a simple model, one that typically makes novice-like errors, and have to build expert strategies to achieve the type of error-free performance that is seen in most experiment data. What is important here is that these types of errors are produced without modification to the EPIC architecture; system parameters are not tweaked.

Another important error modeling component is error recovery. To make EPIC models robust against many forms of information failure, error recovery strategies must be built (Kieras, Wood, and Meyer, 1997; Meyer and Kieras, 1999). Typically, these strategies closely resemble actual observed recovery strategies.

The EPIC architecture is a generation beyond extant information processing architectures in terms of accurately representing a wide variety of cognitive and human performance behavior. The resulting models are generative, giving them the ability to model any combination of task instance (input) data without changing the model. In addition, there are very few free parameters in the EPIC architecture; only task procedures typically need to be changed to match human data. Because EPIC is a valid psychological model, errors that are predicted can more easily indicate interface and task improvements than other modeling systems. Thus, although little error research has been conducted using EPIC, it is very good at modeling certain types of task strategic errors and seems to have good future potential for many other error types.

## 6.11 Vision Models

The three vision models described here are ORACLE, the OptiMetrics Visual Performance Model, and the Georgia Tech Vision Model. This section of the report briefly summarizes the similarities and differences of each model. It goes on to describe how vision models can be used with other modeling environments and their uses for human error prediction. Finally, a brief description of each model is provided that includes history, theoretical basis, basic functioning, specific project uses, and current architecture.

Each of the three models simulates some portion of the human vision system based on vision research and use computational algorithms to simulate the visual processing of an image. The images usually consist of a cluttered background and a target image (usually a military vehicle) within it. The purpose of the vision models has been to predict human performance for target detection tasks. The models have most often been used to either assess vehicle detectability or evaluate the functioning of various vision aiding systems.

During a visual target search, the likelihood that attention is directed at an object is called the probability of fixation ($P_{fix}$). Objects that draw attention are usually distinct in some way from the background but may be non-target objects referred to as clutter. Fixation is directly related to the extent to which a given object 'pops out' at the viewer. This conspicuity is based on the distinction of the object from the background (Pew & Mavor, 1998). Discrimination is the process of separating a target from other clutter objects in the areas of the image that have been fixated. This is quantified as the probability of detection ($P_d$) for accurately identifying a target and the probability of false alarm ($P_{fa}$) for incorrectly indicating that a fixated object is a target.

All of the vision models predict performance for target detection ($P_d$). However, only ORACLE and the Georgia Tech models start with an estimation of fixation. The OptiMetrics model predicts detection given the object or area on which to fixate. This apparent limitation works when the model is used for assessments related to a specific target. However, its use becomes limited for environments in which a large viewing area requires a visual search.

The models can also be distinguished by their theoretical basis. The earliest of the three models, ORACLE, is a threshold model based on retinal and photoreceptor responses. The OptiMetrics and Georgia Tech models are based on more recent research on the functioning of the visual cortex. All three models report some level of prediction validation. In each case, the detection predictions of the model given a particular type of background and target image have been positively compared with the target detection performance of human test subjects using the same images. Therefore the difference between retinal versus cortical approaches does not seem to matter from an accuracy standpoint. Likewise, the differences in the detection prediction functions of each model seem more driven by the desired model uses and customer needs than by any potential limitations of the underlying visual theories and research. All three models also have the capability to analyze images based on either naked eye viewing or enhanced visual systems that translate wavelengths of the electromagnetic spectrum into a visual format such as

infrared imaging. An important distinction for this effort between the three models seems to be their age and the corresponding quantity of successful analysis projects and combinations with other modeling environments for which the model has been used. In this regard both ORACLE and the Georgia Tech models have some advantages over the OptiMetrics model.

### 6.11.1 Error Prediction Capabilities

The prediction capabilities of the vision models relate to human error in the sense that failures are based on limitations of the human visual system. The probability of detection assumes that a target is present and predicts the human visual capability to detect it. Likewise, the probability of false alarm assumes that either a target is not present or that a non-target object that has been fixated is identified as a target based on target-like cues. Within the error taxonomies described in this document there are several in the early stages of perception that relate to the prediction capabilities of these vision models.

For the error taxonomy for Situational Awareness, the errors related to the vision models are from Level 1, 'perception of the elements in the environment'. Specifically, target detection failures fall within the category of data that is hard to discriminate or detect. From the taxonomy of the Model of Internal Human Malfunction, the models represent information error. Target detection failures can represent the failure to detect a cue arising from the change in a system state. From the taxonomy of the Model of Unsafe Acts, target detection errors would seem to fit within the category of perceptual errors. The perceptual errors from this taxonomy are based on not recognizing some object or situation and might be predicted by the probability of false alarm. However, this recognition error is based on either habit or expectation. None of the target detection models take into account the information processing required to represent failures due to this level of cognitive processing. From the taxonomy based on the Information Processing Model, target detection errors fit within both the sensory store and pattern recognition categories. The vision models combine the functioning of the sensory input into the visual cortex with very early pattern recognition to predict fixation and detection.

### 6.11.2 Extension to Other Modeling Environments

Vision models can be used to drive Situational Awareness (SA) models by providing the input cues to the SA. The vision simulation is used to predict the probability that a given cue is detected. If the cue is detected, it is sent to the SA model as an input to the available information used for awareness generation and subsequent decision making. If it is not detected, then no cue is sent and the SA model continues with an incomplete cue set. Likewise, if the perception model predicts a false alarm, a cue that isn't really there will be sent to the SA model. The prediction capability of the vision models combined with an SA model allows for the analysis of situational awareness errors and subsequent decisions based on the use of a correct awareness model or method but using faulty or incomplete cue sets (Pew & Mavor, 1998).

The capability of predicting target detection can also be used with task network models. Given a task network model in which the sequence of actions includes a target or cue detection step, $P_d$ and $P_{fa}$ could be provided by the vision model based on scenario

information represented by an image. These values could then be used to affect state variables within the model, alter the subsequent sequence of steps that are followed, or affect other parts of the simulation based on their dependence on detection, failure or false alarm (Archer et al., 1999). Vision models may also be able to predict how long a fixation or detection would take. This information could be used as execution time for a detection task within a task network model.

### 6.11.3 Application to Aviation

Target detection issues in aviation are complex. They may include singularly or in combination such issues as in-flight versus ground movement situations, good visibility versus reduced visibility, aided target detection versus unaided target detection, moving objects versus stationary objects, and objects to avoid versus objects that help direct. While the vision models can predict target detection performance, it is rare that this information is useful in and of itself. For instance, during flight the detection of targets such as other aircraft by the pilots is often aided by radar and air traffic control. Although the vision models could be used to analyze in-flight target detection scenarios, the limitations of the pilots human visual system is not the only driver of detection performance. The external aids provided to the pilot should increase the target detection performance predicted by a vision model by providing not only the fixation location but also target identification information. Combining vision models with other modeling environments to simulate not only the cue detection but also the situational awareness and decision making required for the potential scenarios will prove more useful than the independent use of vision models.

### 6.11.4 Architectures

#### 6.11.4.1 ORACLE Vision Model

The ORACLE vision model is a product of the British Aerospace (BAE) Sowerby Research Center and has been under development since the late 1980s. A useful description of parts of the model is contained in, “The ORACLE Approach to Target Acquisition and Search Modeling” in the book, Vision Models for Target Detection and Recognition (Cooke et al., 1995). ORACLE is a statistical model of human vision based on research of the retinal and photoreceptor responses to targets. Performance is based on the contrast threshold response of the visual system as a function of target size and contrast. The model quantifies performance as probabilities for search, detection, recognition and identification. The model also includes representations of light level adaption, noise, and peripheral vision.

The theory for the model is based on how light entering the eye is optically spread before reaching the cones and rods in the retina. The photons of the image are mapped onto these receptors and are transformed into neural signals. ORACLE uses the image size and viewing angle across the distribution of receptions to statistically predict the resulting neural signals. Separate processing channels for different signals are modeled to represent static luminance response, moving target luminance response, color detection and low light vision. The individual signals are then combined to provide a total signal for the target. Target acquisition occurs at the point where, with foveal viewing, the total signal exceeds

the noise level, or threshold, for the observer's criteria. The model includes threshold variance to account for the visual diversity of the observer population.

Search involves discriminating between multiple objects that are all above the target threshold during peripheral viewing. Research showed that fixations occur more frequently on objects with target-like characteristics. Rather than model a secondary source of information for these characteristics the simulation uses an identification calibration that results in increased search time with increasing scene clutter.

Performance is based on a combination of independent channel calculations of threshold sensitivity. An estimate is made of the amount above the threshold level for the actual signal contrast combined with observer variations. These estimates equate to standard deviations above the threshold level for each channel that are then combined to gain the performance value for a specific retinal location.

The current software functions on Windows and Unix based systems. Numerous laboratory and field studies have been done to validate model components and across a range of sensor systems. The model has mostly been used within BAESYSTEMS to provide assessments of the capability of sensor/display design options and system evaluation in battlefield simulations for land, sea and air platforms. Most recently BAESYSTEMS and QinetiQ have used ORACLE in conjunction with the IPME task network modeling environment. IPME provides static target conditions that ORACLE uses to return detection probabilities. The analyst can choose different sensor types including binoculars, thermal sight, and naked eye viewing. The detection probabilities are then used to vary the flow of activities and change state variables within the network model for the specific scenario.

#### 6.11.4.2 OptiMetrics Visual Performance Model (VPM)

The development of the VPM was started as a joint effort of the Army Tank-Automotive and Armaments Command Research, Development and Engineering Center (TARDEC) and OptiMetrics, Inc and was called the TARDEC vision model (Gerhart et al., 1995). The original purpose was to address the detectability of military vehicles. Work continued as the National Automotive Center Visual Perception Model (NAC-VPM) (Pew & Mavor, 1998) examined the conspicuity of automobiles during driving tasks. Most recently, OptiMetrics has continued the development of VPM for the Air Force Research Laboratory (AFRL).

The stated purpose of VPM is to emulate human visual performance in order to predict performance for any real world situation of either unaided vision or through visual enhancement systems. The model predicts visual performance on detection tasks given target fixation based on color, brightness, motion, spatial patterns and ambient illumination (Gerhart et al., 1995). The inputs can be either a single digitized image or a sequence of digitized video images modified by the user to show only the target (fixation) area. The early visual processing uses temporal filtering to model the effects of motion, color opponent separation for changing color images to gray scale, and image decomposition for channels related to multi-resolution filtering and orientation filtering (Smith & Dunstan, 1998). Figure 13 shows a flow diagram of the early vision processing.

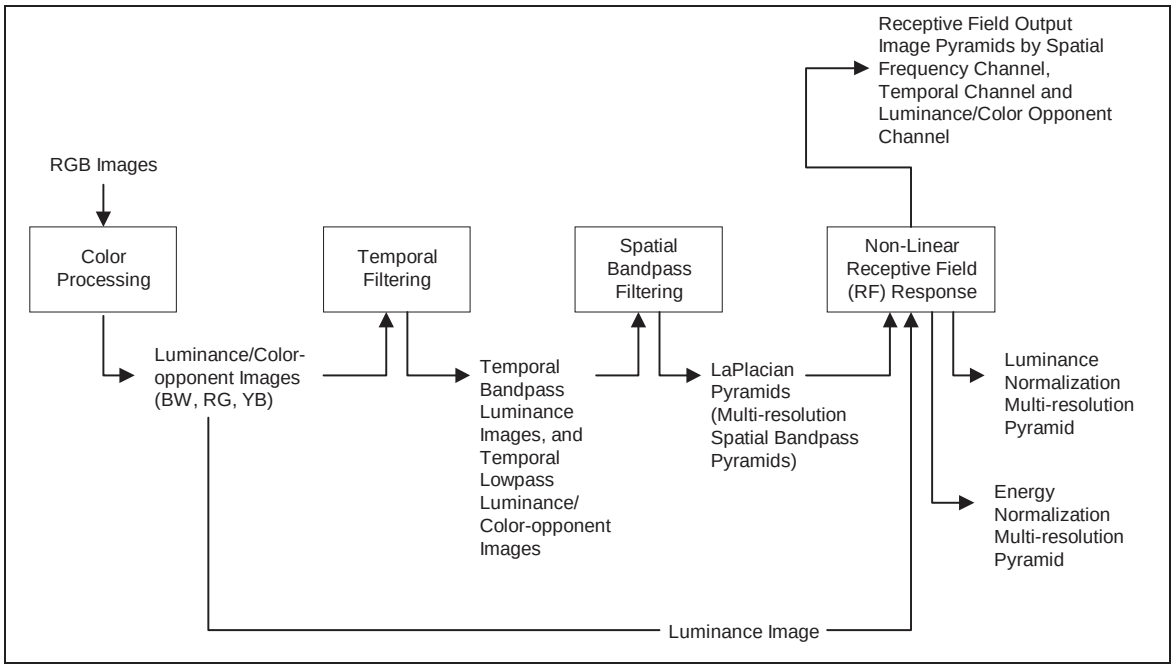



**Figure 13 VPM early vision processing model**

The performance prediction is based on a single measure of detectability formed by combining the calibrated signal to noise ratio from each channel. Signal detection theory is then used to compute the probabilities of detection, recognition and identification that are represented as tradeoffs between the probability of correct target discrimination and probability of false alarm (Gerhart et al., 1995). Additional output from the model can be used to analyze the relative contributions of each spatial channel to the overall predicted detectivity (Dunkel et al., 1999).

OptiMetrics has continued to work with the model on a number of different projects that include validation studies at different levels. The most recent is the work with the AFRL in which the detection performance for a series of FLIR images of large ground targets was correlated to empirical data from operator experiments (Smith & Dunstan, 1998).

#### 6.11.4.3 Georgia Tech Vision Model

The work on the Georgia Tech Vision (GTV) Model is being done at the Georgia Tech Research Institute and has been ongoing since the early 1990s (Pew & Mavor, 1998). The GTV is a general, high fidelity model of visual performance based on the principles of human visual processing. A good description and reference list of the model background and components can be found in the book, “Vision Models for Target Detection and Recognition,” (Doll et al., 1995).

The model incorporates findings from research on low-level visual processes, including texture segregation and pop-out, visual search, selective attention, and signal detection theory (Doll et al, 1995). Inputs to the GTV model are either a static image or a sequence of images representing a target moving through a cluttered background. The model

predicts search ($P_{fix}$) and detection ($P_d$) performance, including eye fixation locations during visual search and the discriminability of the target from clutter objects (Doll et al, 1997). The model also accounts for the effect of practice and experience on selective attention during visual search.

The algorithms used in the simulation are designed to be consistent with the neurophysiological structure of the human visual system. These include luminance adaption, pupil dilation, reception thresholds, and the processing of motion, flicker, color and edge information. Figure 14 shows the processing flow through the four modules of the GTV model.

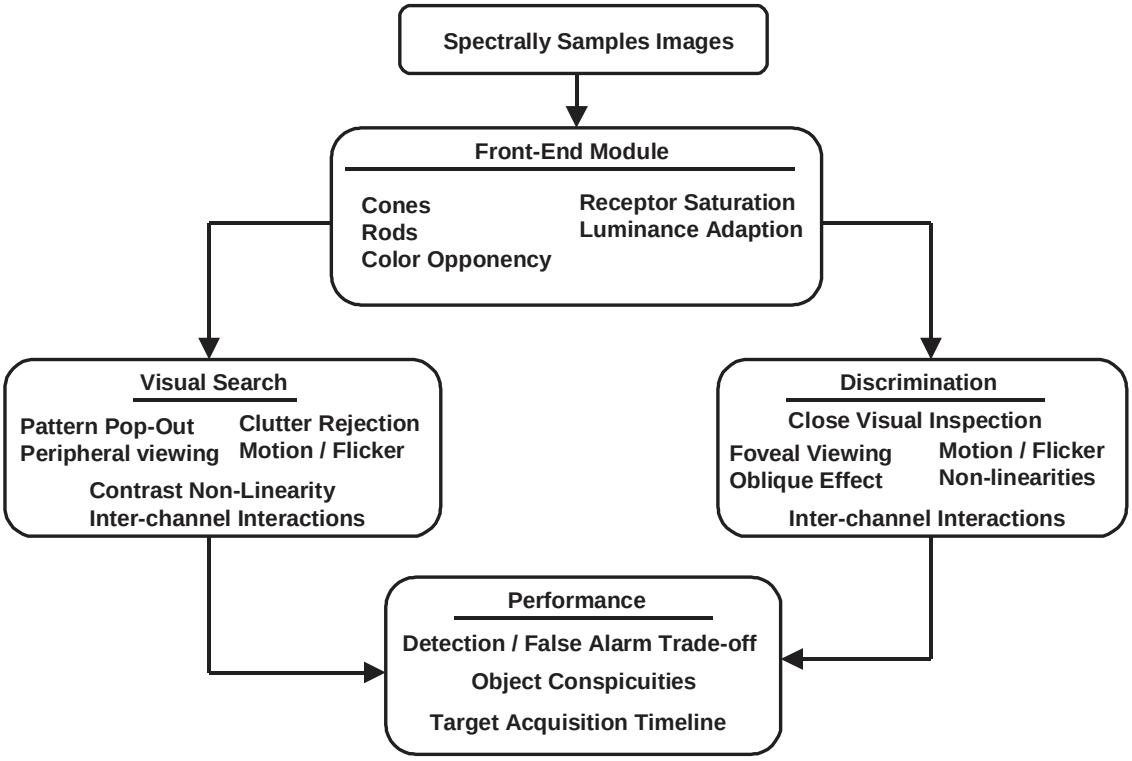



**Figure 14 Major GTV modules (reproduced from, "Vision Models for Target Detection and Recognition")**

The first module simulates the initial processing of the visual system. It transforms the visual image into 144 different channels designed to simulate the information available to the visual cortex. These channels represent differing spatial frequency/orientation characteristics, different color-opponent signals, and various types of retinal receptor outputs (Doll et al, 1997).

The multi-channel converted input is first analyzed by the search module to determine what areas within the image have the greatest conspicuity and would draw the attention of the viewer based on pattern perception in the peripheral visual field. The pre-attentive

processing using peripheral vision is simulated through a sequence of operations performed on different image channels. The operations include; 1) temporal filtering and integration to simulate the effects of motion and flicker, 2) pattern perception based on texture segregation, 3) simulation of peripheral sensitivity, and 4) weighting the pooling of individual channel outputs (Doll et al, 1995).

The discrimination module is essentially the same as the search module except that it simulates attention processing through foveal viewing rather than the pre-attention processing of peripheral viewing. It is based on the results of target detections experiments showing that observers perform detection tasks based on the discriminability of the target from the background clutter, rather than the detectability of the target pattern (Doll et al, 1995). The pre-attentive output image is segmented into objects that represent potential target locations. A quantification is then made of the extent to which each object attracts the observer's attention.

Viewer experience was determined to be an important factor in searching a cluttered image. Research showed that different input channels aid in fixation during visual search based on the type of background clutter and target type. Given several hours of practice, viewers are able to determine which types of cues help them distinguish probable targets from background clutter. The model uses a neural network to alter the weights of different input channels to simulate affects of experience. The neural network is trained on a specific type of background and target and the weights are altered in both the search and discrimination modules.

The performance module uses the outcomes of the search module to compute a probability of fixation of possible targets during visual search for each object in the pre-attentive output image. Discrimination performance is quantified as the probability of detection or false alarm for each object given that it is fixated ($P_{yes/fix}$). This allows the model to simulate a detection failure due to lack of fixation.

The GTV model can be used as a tool for diagnosing which features of a target contribute most to its conspicuity, and for designing treatments that increase or reduce conspicuity (Doll et al, 1995). The model is currently incorporated into a signature analysis tool called Visual and Electro-Optical (VISEO) detection for the Army Aviation and Troop Command, Applied Technology Directorate (ATCOM/AATD). VISEO is a suite of software tools designed to help a vehicle designer reduce the detectability of the vehicle. It places detailed graphical models of desired vehicles into cluttered background images and uses the GTV model to determine what parts of the vehicle (with or without camouflage) contribute most to detectability (Doll et al, 1997).

# 7 Recommendations

## 7.1 Single Architecture Remarks

The purpose of this section is to make some guiding remarks about the application of architectures described in Section 6 to future human error modeling research, in particular research concerning aviation error prediction. In order for these remarks to be held in the proper light, we felt it was necessary to emphasize some important points. First, we must reiterate that *aircrew-caused* aviation accidents, as well as many aviation incidents, result from a chain of errors (as discussed in Section 4.1) rather than a single error. Therefore, it is critical that the modeling approaches chosen for future human error prediction research be able to represent both plausible error chains and the *context* in which they occur. Such an approach will require that modelers and aviation safety experts work closely together to establish error chains of interest and the scope of the errors to be investigated. The human error taxonomies discussed in Section 3 should be of assistance for the latter. When developing models to predict *aviation errors*, we believe that validation will require validation not just at the architecture level, but at the model level. Based on our understanding of the degree that each architecture has been applied to aviation, we determined that seven architectures stood out among the rest: Air MIDAS, Core MIDAS, task network models, SAMPLE, Soar, D-OMAR and COGNET. We do not suggest that their use in modeling aviation activities necessarily proves that the architectures were both valid and practical. Indeed, few validation studies have been conducted that we could find, with some notable exceptions (e.g., Allender et al., 1995; Hoagland et al., 2001). However, the developers of these architectures can be contacted to provide insight on how they would extend existing models of aviation behavior to the prediction of human error.

It is clear that regardless of the underlying theory, all the architectures that have been discussed in Section 6 are comprised of computational algorithms that, in the end, represent the processing of zeros and ones. Thus, algorithms from one architecture can in many, if not all, cases be implemented into another architecture to represent a particular aspect of human behavior. For example, segments of the ORACLE vision model have been incorporated into IPME to represent visual scanning behavior that were part of a larger model of crew behavior. In another example, the goal-oriented, rule-based behaviors that are the focal point of Soar and COGNET models can be represented in task network-based tools, although the models will look very different. Therefore, we should be careful about stating what architectures *cannot* represent behavioral phenomena but, rather, understand that some architectures are better suited than others for modeling certain types of behavioral phenomena. While we do not suggest that these models are all functionally equivalent, we do suggest that they are more so than might appear at first glimpse. While we could use one architecture to model behaviors that are outside of the architecture developers original intention, in many cases this would be difficult and inefficient to do. Could the task network tools described in this paper model memory (and errors due to memory decay) in the same manner as ACT-R? Yes. Is it efficient to do so? Probably not from a model development time or computational

efficiency perspective. So, in evaluating models and conducting relative comparisons, we must keep in mind that there is far more functional equivalency than one might expect.

In a well worn saying about the very nature of models, "All models are wrong, but some are useful." The definition of "useful" is in the eye of the user. And the question the user must ask him or herself is, "What do I want to get out of the model?" In applying human error models, perhaps the most significant task will be to determine what aspects of the human/machine system to include in the model and what to leave out. Our experience has shown that many human performance modeling studies have failed because of the inclusion of too many factors that, while a part of human/system performance, were not system performance drivers. Consequently, the models become overly complex and untenable. We expect for human error modeling that this will present an even larger risk to model development. Extrapolating from our experience, it is better to begin any human error modeling project with a small, but very specific set of human/system representations and then add to it as necessary, rather than to begin a modeling project by intending to represent everything. We believe the first approach may succeed while the second is virtually doomed.

From the behaviors that need to be included in the representation, the next issue is, "What is likely to be driving human error rates in the system?" While this is a question that you will often want the model to answer, to the extent that you can find the likely sources of error, you will be more likely to hone in on the architecture that will best support your analysis. Again, the error taxonomies presented in this report are presented for your consideration in trying to find likely error sources.

Next, from our experience in the use of modeling and simulation, we know that practical matters such as time and funding available will *always* be central to determining what approach to use. Will you have that time to build the models with that architecture? Does using the architecture require individuals with skills that are not available to you? Will you need data to populate the models that will not be available? If you need to hook up with other models of system components (e.g., aircraft, ATC systems), will the architecture support the necessary strategy of model integration? An excellent source of guidance as to what factors might be considered is contained in a NATO-sponsored AGARD report (1998).

Although not nearly as important as other considerations, we know that for the foreseeable future, computational efficiency is a factor. Computers are not yet fast and powerful enough to simulate the simultaneous human behavior of every operator in the National Airspace System for any significant period of time. Therefore, we will often need to find ways of finding the appropriate balance between the best architecture to model particular human behaviors vs. models that will work within the constrained resources available. And, when we build models of larger systems (e.g., moving from a model of a TRACON to a model of the National Airspace), we may need to scale back human performance model complexity to achieve the required levels of computational efficiency. While this may seem disheartening to a purist, modeling and simulation is always a matter of compromising between the best possible model vs. one that can be

developed within the real-world constraints but that still addresses the issues for which it was designed.

To summarize, the selection of an architecture for human error modeling should consider many factors – ranging from the theoretically best-suited approach to practical considerations. In section 6 of this report, we have provided some insights into these issues, but, because of the rapidly changing nature of this technology, this information must be collected on any candidate architecture at the time the model development process is to begin.

## 7.2 Hybrid Modeling Approaches

As can be seen from the previous discussion in Section 7.1, we do not believe that there is "one right architecture" for modeling human error in aviation. If we had to recommend one, we would still have to begin any statement with, "Well, it depends…" and then list all of the factors above that the selection of a single architecture depends upon. However, this leads us to a higher level recommendation – that NASA consider an integral part of the model development strategy the use of a hybrid modeling approach. Hybrid modeling implies combining different modeling architectures to the same modeling problem, even to the point of using several architectures to represent different aspects of a single human's behavior.

For example, a model of overall task sequencing and flow built with a task network model can be combined with an ACT-R model of critical aspects of human cognition that lead to error, providing a more valid and usable environment than either architecture might individually. Another example is EPIC and Soar hybrid approach. Soar has no mechanisms for human perceptual or motor movement, but it does model learning. EPIC, on the other hand, has very detailed perceptual and motor processes and a non-learning cognition.

Hybrid modeling is made possible by the use of *distributed simulation technology*. Distributed simulation involves the use of two or more simulations representing different aspects of the system being simulated running on possibly different computers sharing enough data that they are able to represent the dynamic interactions amongst the system elements represented by each simulation. These simulations can also include man-in-the-loop simulators who can interact with other man-in-the-loop simulators along with models of human performance – all representing different system components and all interacting dynamically in real time.

There are two types of inter-model communication that might be needed to achieve distributed simulation for aviation error modeling:

> *Dynamic data exchange during simulation runs* whereby, one simulation relies upon another simulation federate to provide data during the simulation. This concept is illustrated in Figure 14 where different modeling architectures reflecting different aspects of behavior (e.g., decision making, procedure execution, goal balancing) interact to be able to form a better simulation of the human team.

*Sharing data between simulation runs* through a central data repository. For example, Figure 15 illustrates how, during different phases of system development, some models' outputs may serve as other models' inputs as shared through a central data repository.

There have been and continue to be attempts at developing and improving inter-model communication standards and technology such as the Department of Defense's High Level Architecture (HLA).

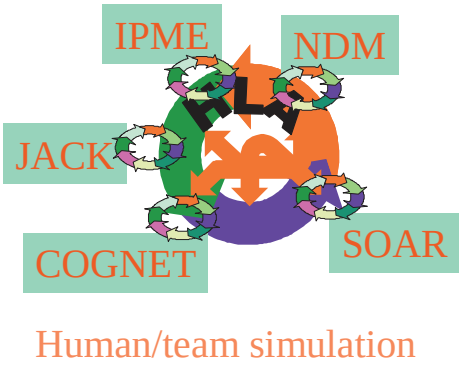



**Figure 14 HLA Architecture**

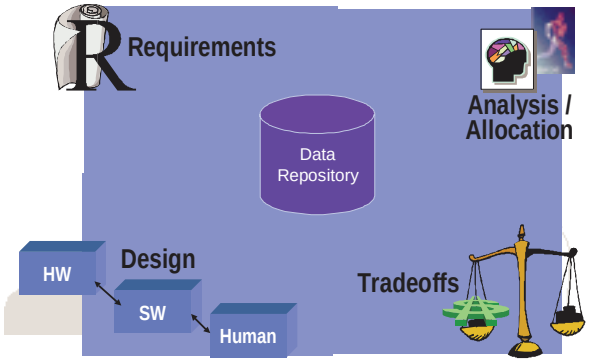



**Figure 15 Shared Data**

We believe that this hybrid approach should be seriously considered in this project. It is being tested in at least one of the experiments for this year's HEM element research program where ACT-R and Task Network models are being combined to determine how they can collectively provide a more powerful modeling architecture than they could individually.

# 8 References


Abkin, M. H., Bobick, J. C., Hansman, R. J., Reynolds, T. G., Hansen, M. M., Gosling, G. D., and Baumgardner, W. F. (2001). Development of fast-time simulation techniques to model safety issues in the National Airspace System. CY00 Final Report for NASA Ames Research Center.

AGARD Advisory Report 356 (1998). A Designer's Guide to Human Performance Modeling, AGARD-AR-356. Available through NASA CASI, #19990032479.

Allender, L., Kelley, T., Salvi, L., Headley, D.B., Promisel, D., Mitchell, D., Richer, C., and Feng, T., (1995). Verification, Validation, and Accreditation of a Soldier-System Modeling Tool. In *Proceedings of the 39th Human Factors and Ergonomics Society Meeting*, October 9-13, San Diego, CA. Human Factors and Ergonomics Society, Santa Monica, CA.

Anderson, J. R. & Lebière, C. (1998). *The atomic components of thought.* Mahwah, NJ: Erlbaum.

Archer, R., Keller, J., Archer, S., Scott-Nash, S. (1999). Discrete Event Simulation as a Risk Analysis Tool for Remote Afterloading Brachytherapy, NUREG/CR-5362, Vol 1 & 2, Micro Analysis & Design, Inc.

Baron, S., Zacharias, G., Muralidharan, R., & Lancraft, R. (1980). PROCRU: A model for analyzing flight crew procedures in approach to landing. In *Proceedings of the Eighth IFAC World Congress*, Tokyo, Japan. Eight

Baumeister, L.K., John, B.E., and Byrne, M.D. (2000). A comparison of tools for building GOMS models. In *Proceedings of CHI 2000*, The Hague, Amsterdam, April 1-6, 2000. New York: ACM, pp. 502-509.

Byrne, M. D. (1993). Systematic Procedural Error as a Result of Interaction Between Working Memory Demand and Task Structure., Georgia Institute of Technology.

Byrne, M. D., Wood, S. D., Sukaviriya, P., Foley, J. D., & Kieras, D. E. (1994). Automating Interface Evaluation. Paper presented at the Human Factors in Computing Systems, Boston, MA.

Card, S. K., Moran, T. P., & Newell, A. (1983). The Psychology of Human-Computer Interaction. Hillsdale, NJ: Lawrence Erlbaum.

Chong, R.S. (2001). Low-Level Behavioral Modeling and the HLA: An EPIC-Soar Model of an Enroute Air-Traffic Control Task. In *Proceedings of 10th Computer Generated Forces and Behavioral Representation Conference*. Norfolk, VA.

Cooke, K.J., Stanley, P.A., and Hinton, J.L. (1995). The ORACLE Approach to Target Acquisition and Search Modelling, in Vision Models for Target Detection and Recognition, Peli, E. editor. World Scientific Publishing Co. River Edge, NJ.

Corker, Kevin M. (2000). Cognitive Models & Control: Human & System Dynamics in Advanced Airspace Operations, *in* N. Sarter and R. Amalberti *(eds.)*, *Cognitive Engineering in the Aviation Domain*, Lawrence Earlbaum Associates, New Jersey.

Covrigaru, A., & Kieras, D.E. (1987). PPS: A parsimonious production system (Tech. Rep. No. 26, TR-87/ONR-26). Ann Arbor: University of Michigan, Technical Communication Program. (DTIC AD A182366)

Dennett, D. C. (1991). *Consciousness explained*. Boston, MA: Little, Brown and Company.

Deutsch, S. (1998). Multi-disciplinary foundations of multiple-task human performance modeling in OMAR. *In the Proceedings of the Twentieth Annual Meeting of the Cognitive Science Society*. Madison, WI. 1998.

Doll, T.H., McWhorter, S.W., Schmieder, D.E., and Wasilewski, A.A. (1995). Simulation of Selective Attention and Training Effects in Visual Search and Detection, in Vision Models for Target Detection and Recognition, Peli, E. editor. World Scientific Publishing Co. River Edge, NJ.

Doll, T.H., McWhorter, S.W., Schmieder, D.E., Hertzler, M.C., Stewart, J.M., Wasilewski, A.A., Owens, W.R., Sheffer, A.D.Galloway, G.L. and Herbert, S.D. (1997). Biologically-based vision simulation for target-background discrimination and camouflage/lo design. Paper No. 3062-29 in Targets and Backgrounds: *Proceedings of the International Society of Photo-optical Instrumentation Engineers*, Watkins, W.R. and Clemens, D., eds. Orlando, FL: Society of Photo-optical Instrumentation Engineers.

Dunkel, M.D., Smith, F.G., and Dunstan, A.W. (1999). Visual Performance Model Analysis of Human Performance in IR Target Recognition for Third Generation FLIR, (AF Report Number Unknown), OMI-663, OptiMetrics Inc.

Endsley, M. R. (1988). Design and evaluation for situation awareness enhancement. *In Proceedings of the Human Factors Society 32nd Annual Meeting* (pp. 97-101). Santa Monica, CA: Human Factors Society.

Endsley, M. R., & Jones, D. G. (1995). Situation awareness requirements analysis for TRACON air traffic control (TTU-IE-95- 01). Lubbock, TX: Texas Tech University.

Endsley, M. R., Farley, T. C., Jones, W. M., Midkiff, A. H., and Hansman, R. J. (1998). Situation awareness requirements for commercial airline pilots. International Center for Air Transportation, Massachusetts Institute of Technology, Cambridge, MA, 02139.

Endsley, M. R. (1999). Situation Awareness and Human Error: Designing to Support Human Performance. In *Proceedings of the High Consequence Systems Surety Conference*. Albuquerque, NM:

Freed, M. A. & Shafto M. G. (1997). Human-System Modeling: Some Principles and a Pragmatic Approach. *Proceedings of the 4th International Workshop on the Design, Specification, and Verification of Interactive Systems*.

Freed, M. A. & Remington, R. W. (1997). Managing decision resources in plan execution. *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*.

Freed, M. A. & Remington, R. W. (1998). A conceptual framework for predicting error in complex human-machine environments. *In Proceedings of the 1998 Meeting of The Cognitive Science Society*.

Gerhart, G., Meitzler, T., Sohn, E., Witus, G., Lindquist, G., and Freeling, J.R. (1995). Early vision model for target detection, In *Proceedings of the 9th SPIE AeroSense*, Orlando, FL.

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.

Hendy, K. C., Liao, J., & Milgram, P. (1997). Combining time and intensity effects in assessing operation information-processing load. Journal of Human Factors and Ergonomics Society, 39(1). Santa Monica, CA.

Hintzman, D.L., (1986). *Judgments of Frequency and Recognition Memory in a Multiple-Trace Memory Model*, Institute of Cognitive and Decision Sciences: Eugene, OR. p. 1-94.

Hoagland, D. G., Martin, E. A., Anesgart, M., Brett, B. E., LaVine, N., Archer, S. (2001). Combat Automation Requirements Testbed (CART) Case Study One: Results from Modeling Human Behavior Within High Fidelity Constructive Simulations. In *Proceedings of the Tenth Conference on Computer Generated Forces*. Norfolk, VA.

Howes, A., and Young, R.M. (1991). *Predicting the Learnability of Task-Action Mappings*. In Proceedings of CHI, 1991, New Orleans, LA, ACM, NY.

Hudlicka, E. & Corker, K. (2001). Affective context and control in human-system modeling. In *Proceedings of the 3rd International Workshop: 2001 Human Error and System Safety Development*. Linkoping, Sweden, June.

John, B. E., & Kieras, D. E. (1996a). Using GOMS for User Interface Design and Evaluation: Which Technique? ACM Transactions on Computer-Human Interactions, 3(4), 287 - 319.

John, B. E., & Kieras, D. E. (1996b). The GOMS Family of User Interface Analysis Techniques: Comparison and Contrast. ACM Transactions on Computer-Human Interactions, 3(4), 320 - 352.

Johnson, T. R. (1997). Control in Act-R and Soar. In M. Shafto & P. Langley (Eds.), *Proceedings of the Nineteenth Annual Conference of the Cognitive Science Society* (pp. 343-348): Lawrence-Erlbaum.

Jones, D. G. & Endsley, M. R. (1996). Sources of situation awareness errors in aviation. Aviation, Space, and Environmental Medicine, 67(6), 507-512.

Jones, R. M., Neville, K., & Laird, J. E. (1998). Modeling pilot fatigue with a synthetic behavior model. In *Proceedings of the Seventh Conference on Computer Generated Forces and Behavioral Representation*. Orlando, FL.

Jones, R. M., Neville, K., & Laird, J. E. (1998). Modeling pilot fatigue with a synthetic behavior model. *Proceedings of the Seventh Conference on Computer Generated Forces and Behavioral Representation*. Orlando, FL.

Jones, R.M. (1999). Graphical visualization of Situational Awareness and mental state for intelligent computer generated forces. *Proceedings of the 8th CGF-BR Conference*, 11 May.

Jones, R.M. (1999). Graphical Visualization of Situational Awareness and Mental State for Intelligent Computer-Generated Forces. *In the Proceedings of the Eighth Annual Conference on Computer Generated Forces and Behavioral Representation*. May 11, 1999. Orlando, FL.

Jones, R.M., Kenny, P.K. (2000). Lessons learned from integrating STOW Technology into an Operational Naval Environment. Presented at Spring SIW Conference.

Jones, R.M., Laird, J.E., Nielsen, P.E., Coulter, K.J., Kenny, P. and Koss, F.V. (1999). Automated intelligent pilots for combat flight simulation. Artificial Intelligence, Spring.

Jones, Randolph M., Laird, John E., Nielsen, Paul E., Coulter, Karen J., Kenny, Patrick. and Koss, Frank V. 1999. *Automated Intelligent Pilots for Combat Flight Simulation*. AI Magazine, 20(1):27-41.

Kelley, T. D., Patton, D. & Allender, L. (in press). Error rates in mental manipulation of spatial images. Perceptual and Motor Skills.

Kelley, T. D., Patton, D. J., & Allender, L. (in press). Predicting situation awareness errors using cognitive modeling. In *Proceedings of the 2001 Human Computer Interaction International Conference*.

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

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: Elsevier.

Kieras, D. & 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.

Kieras, D.E., & Meyer, D.E. (1994). The EPIC architecture for modeling human information-processing: A brief introduction. (EPIC Tech. Rep. No. 1, TR-94/ONR-EPIC-1). Ann Arbor, University of Michigan, Department of Electrical Engineering and Computer Science.

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. Paper presented at the ACM User Interface Software and Technology, Pittsburgh, PA.

Kieras, D. & 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.

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., Wood, S.D., & Meyer, D.E. (1997). Predictive engineering models based on the EPIC architecture for a multimodal high-performance human-computer interaction task. *ACM Transactions on Computer-Human Interaction*. **4**, 230-275.

Kieras, D.E. (1998). A guide to GOMS model usability evaluation using GOMSL and GLEAN3. (Technical Report No. 38, TR-98/ARPA-2). Ann Arbor, University of Michigan, Electrical Engineering and Computer Science Department. ftp://ftp.eecs.umich.edu/people/kieras/GOMS/GOMSL_Guide.pdf.

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.

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.

Kieras, D. E., Meyer, D. E., Ballas, J. A., & Lauber, E. J. (2000). Modern computational perspectives on executive mental control: Where to from here? In S. Monsell & J.

Driver (Eds.), *Control of cognitive processes: Attention and performance XVIII* (pp. 681-712). Cambridge, MA: M.I.T. Press.

Kieras, D.E. Model-based evaluation (in press). In J. Jacko & A. Sears(Eds.), Handbook of Human-Computer Interaction in Interactive Systems. Mahwah, New Jersey: Lawrence Erlbaum Associates.

Kirwan, B. (1992). Human error identification in human reliability assessment. Part 1: Overview of approaches. Applied Ergonomics, 23(5), 299-318.

Klein, G. (1989). Recognition-Primed Decisions. Advances in Man-Machine Systems Research, Vol. 5: 47-92.

Laird, J., Yager, E., Hucka, M., Tuck, C. (1991). Robo-Soar: An Integration of External Interaction, Planning, and Learning using Soar, *Robotics and Autonomous Systems*, Vol. 8.

Laird, J.E. (1988). Recovery from incorrect knowledge in Soar. In *Proceedings of the AAAI-88*.

Laird, J.E., Newell, A. & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence, 33*, 1-64.

Laughery, K.R. Jr., & Corker K. (1997). Computer modeling and simulation of human/system performance. In Gavriel Salvendy (ed.), Handbook of human factors and ergonomics (2nd ed.), pp. (1375-1408), N.Y.: Wiley and Sons, Inc.

Lawless, M.L., Laughery, K.R., and Persensky, J.J. (1995). *Micro Saint to Predict Performance in a Nuclear Power Plant Control Room: A Test of Validity and Feasibility.* NUREG/CR-6159, Nuclear Regulatory Commission, Washington, D.C.

Lebière, C, Anderson, J. R., & Bothell, D. (2001). Multi-tasking and cognitive workload in an ACT-R model of a simplified air traffic control task. *Proceedings of the Tenth Conference on Computer Generated Forces and Behavior Representation*. Norfolk, VA.

Lebière, C., Anderson, J. R., & Reder, L.M. (1994). Error modeling in the ACT-R production system. *In Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*, 555-559. Hillsdale, NJ: Erlbaum.

Lehman, J.F., Laird, J. and Rosenbloom, P. A gentle introduction to Soar; an architecture for human cognition. http://www.soartech.com

Leiden, K. J., Green, S. M. (2001). Trajectory orientation: a technology-enabled concept requiring a shift in controller roles and responsibilities. In Donohue & Zellweger (Eds.) *Air Transportation Systems Engineering.* (pp. 627-646) American Institute of Aeronautics and Astronautics, Inc.

Lerch, F. J. (1988). Computerized Financial Planning: Discovering Cognitive Difficulties in Model Building. Doctoral dissertation, University of Michigan.

Lewis, R. L. (1992). *Recent developments in the NL-Soar garden path theory*. Technical Report CMU-CS-92-141, School of Computer Science, Carnegie Mellon University.

Logan, G. D. (1988). Automaticity, resources, and memory: Theoretical controversies and practical implications. *Human Factors, 30,* 583-598.

Maurino, D. E., Reason, J., Johnston, N., & Lee, R. B. (1995). *Beyond Aviation Human Factors*. (p. 18). Burlington, VT: Ashgate Publishing Company

Meyer, D. E., & Kieras, D. E. (1997). A computational theory of executive control processes and human multipletask performance: Part 2. Accounts of Psychological Refractory-Period Phenomena. *Psychological Review.* **104**, 749-791.

Meyer, D. E., & Kieras, D. E. (1997). A computational theory of executive control processes and human multipletask performance: Part 1. Basic Mechanisms. *Psychological Review,* **104**, 3-65.

Meyer, D. E., & Kieras, D. E. (1999). Precis to a practical unified theory of cognition and action: Some lessons from computational modeling of human multiple-task performance. In D. Gopher & A. Koriat (Eds.), *Attention and Performance XVII. Cognitive regulation of performance: Interation of theory and application* (pp. 17 - 88). Cambridge, MA: M.I.T. Press.

Meyer, D.E., & Kieras, D.E. (1994). EPIC computational models of psychological refractory-period effects in human multiple-task performance. (EPIC Tech. Rep. No. 2, TR-94/ONR-EPIC-2). Ann Arbor, University of Michigan, Department of Psychology.

Miller, C. S. & Laird, J. E. (1996). Accounting for graded performance within a discrete search framework. *Cognitive Science*, 20, 499-537.

Miller, C. S., Laird., J. E. (1996). Accounting for graded performance within a discrete search framework. *Cognitive Science*., pp. 499-537, Vol. 20, 1996.

Miller, C.S. (1993). *Modeling concept acquisition in the context of a unified theory of cognition*. Doctoral Dissertation, University of Michigan.

Mueller, S. T., Seymour, T. L., Kieras, D. E., & Meyer, D. E. (submitted). Theoretical implications of articulatory duration, phonological similarity, and phonological complexity effects in verbal working memory. Submitted to *Journal of Experimental Psychology: Human Perception and Performance*.

Mulgund, S.S., Harper, K.A., Zacharias, G.L., and Menke, T.E. (2000). SAMPLE: Situation Awareness Model for Pilot-in-the-Loop Evaluation, *Proceedings of the*

*9th Conference on Computer Generated Forces and Behavior Representation, Orlando, FL* (May).

Newell, A. 1990, Unified theories of cognition. Cambridge MA Harvard University Press.

Nielsen, J., & Phillips, V. L. (1993). Estimating the Relative Usability of Two Interfaces: Heuristic, Formal, and Empirical Methods Compared. Paper presented at the INTERCHI, Amsterdam.

Nielsen, P.E. (2000). Participation of TacAir-Soar in RoadRunner and Coyote Exercises at AFRL, Mesa. In *Proceedings of 10th Computer Generated Forces and Behavioral Representation Conference*. Norfolk, VA.

O'Hare, D., Wiggins, M., Batt, R. & Morrison, D. (1994). Cognitive failure analysis for aircraft accident investigation. *Ergonomics.* 37, 1855-1869.

Olson, J. R., & Olson, G. M. (1990). The Growth of Cognitive Modeling in Human-Computer Interaction Since GOMS. Human-Computer Interaction, 5, 221-265.

Payne, W. and J.R. Bettman, and E.J. Johnson (1988). Adaptive Strategy in Decision Making. Journal of Experimental Psychology: Learning, Memory and Cognition, 14, No. 3, pp. 534-552.

Pearson, D.J. (1996). Learning Procedural Planning Knowledge in Complex Environments. Doctoral Dissertation, University of Michigan.

Peli, E. (editor) (1995). Vision Models for Target Detection and Recognition. World Scientific Publishing Co. River Edge, NJ.

Pew, R.W. & Mavor, A. S. (eds) (1998). Modeling Human and Organizational Behavior: Application to Military Simulations. National Academy Press, Washington, D.C, pg 195.

Polk, T. A., & Newell, A. (1995). *Deduction as verbal reasoning*. Psychological Review, 102, 533-566.

Polk, T.A., Newell, A. (1988). Modeling human syllogistic reasoning in Soar. In *Proceedings of the Annual Conference of the Cognitive Science Society*, pages 181-187.

Rasmussen, J. (1982). Human Errors: A Taxonomy for Describing Human Malfunctions in Industrial Installations. Journal of Occupational Accidents, Vol. 4: 311-335.

Reason, J. (1990). *Human Error.* New York: Cambridge Press.

Rosenbloom, P. S., Laird, J. E. & Newell, A. (1992). *The Soar Papers: Research on Integrated Intelligence*. Volumes 1 and 2. Cambridge, MA: MIT Press.

Ryder, J., Santarelli, T., Scolaro, J. Hicinbothom, J. and Zachary, W. (2000). Comparison of cognitive model uses in intelligent training systems. Proceedings of the IEA 2000/HFES 2000 Congress, pp. 2-374-377.

Santoro, T.P., Kieras, D.E., and Campbell, G.E. (2000). GOMS modeling application to watchstation design using the GLEAN tool. Proceedings of the Interservice/Industry Training, Simulation, and Education Conference, pp. 964-973, Orlando, FL. November, 2000.

Schumacher, E. H., Lauber, E. J., Glass, J. M. B., Zurbriggen, E. L., Gmeindl, L., Kieras, D. E., & Meyer, D. E. (1999). Concurrent response-selection processes in dual-task performance: Evidence for adaptive executive control of task scheduling. *Journal of Experimental Psychology: Human Perception and Performance*, 1999, **25**, 791-814.

Simon, T., Newell, A., and Klahr, D. (1991). Q-*Soar: A computational account of children's learning about number conservation*. In Working Models of Human Perception. Morgan Kaufman, Los Altos, California, 1991.

Smelcer, J. B. (1989). Understanding User Errors in Database Query. Doctoral dissertation, The University of Michigan.

Smith, F.G. and Dunstan, A.W. (1998). Visual Performance Model Analysis of Human Performance in IR Target Recognition, AFRL-HE-WP-TR-1998-0065, (OMI-608), OptiMetrics Inc.

Smith, F.G., Riegler, J.T., and Kuperman, G.G. (1998). The Visual Performance Model: Predicting Target Recognition Performance with First Generation FLIR Imagery, IRIS Specialty Meeting on Passive Sensors.

Tambe, M., Johnson, W. L., Jones, R. M., Koss, F., Laird, J. E., Rosenbloom, P. S. & Schwamb, K. (1995). Intelligent agents for interactive simulation environments. *AI Magazine, 16*, 15-40.

Verma, S. A. & Corker, K. M., (2001). Introduction of Context in Human Performance Model as applied to Dynamic Resectorization. In *Proceedings of the 11th International Symposium on Aviation Psychology*, Columbus, OH.

Vera, A., John, B., Matessa, M., Freed, M., & Remington, R. (submitted). Automating CPM-GOMS. *Proceedings of ???.*

Warwick, W., McIllwaine, S., Hutton, R., McDermott, P. (2001). Developing computational models of recognition-primed decision making. *Proceedings of the Tenth Conference on Computer Generated Forces and Behavior Representation*. Norfolk, VA.

Wickens, C. D. and Flach, J. M. (1988). Information processing. In E. L. Wiener & D. C. Nagel (Eds.), *Human factors in aviation.* (pp.111-155). San Diego, CA: Academic Press.

Wiegmann, D. A., and Shappell, S. A. (1995). Human factors in U.S. aviation mishaps. *In Proceedings of the Eighth International Symposium on Aviation Psychology*, Columbus, OH..

Wiegmann, D. A., and Shappell, S. A. (1997). Human factors analysis of post-accident data: Applying theoretical taxonomies of human error. The International Journal of Aviation Psychology, vol. 7, no. 1, pp. 67-81, Lawrence Erlbaum Associates, Inc.

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

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., Kieras, D.E. (2002) Using GOMS in Design to Prevent Human Error. Submitted for publication.

Zachary, W., Le Mentec, J-C and Ryder, J. (1996). Interface agents in complex systems. In Ntuen, C. and Park, E.H. (Eds), Human interaction with complex systems: Conceptual Principles and Design Practice. Norwell, MA: Kluwer Academics Publishers.

Zachary, W., Ryder, J., Ross, L. and Weiland, M. (1992). Intelligent Human Computer Interaction in Real Time; Multi-tasking Process Control and Monitoring Systems. In M. Helander and M. Nagamachi (Eds) Human Factors in Design for Manufacturability. New York, Taylor and Francis, pp 377-402

Zachary, W.W. and Le Mentec, J-C. (2000). Modeling and simulating cooperation and teamwork. *Proceedings of the 2000 Advanced Simulation Technology Conference*.

Zadeh, L. (1973). Outline of a New Approach to the Analysis of Complex Systems and Decision Processes. IEEE Transactions on Systems, Man and Cybernetics, 3(1): 28-44.

# Appendix A
# Extending GOMS to Human Error

(From **Chapter 5** of Scott D. Wood's Doctoral Dissertation, *Extending GOMS to Human Error and Applying it to Error-Tolerant Design*, University of Michigan, 2000.)

Although GOMS techniques have been used for various types of error analysis, a theory for integrating human error into GOMS models has not been fully developed. This chapter reviews the original proposal to extend GOMS to human error, and develops a general framework for error recovery in a GOMS architecture. It frames error recovery as a multi- stage process and forms the basis for an Error-extended GOMS (EGOMS). The stages are related to the error recovery model, and fundamental issues for how each stage can be addressed within GOMS are discussed.

**5.1 CMN ERROR EXTENSIONS**

Card, Moran, and Newell (CMN)realized from the beginning of GOMS that error extensions were necessary to fully model even moderately complex tasks. In their observations of experiment subjects performing a typing task, they noted several interesting results regarding human error. The first result was that errors occurred in 36% of the experimental tasks. This indicates the pervasiveness of human error and the importance of designing to accommodate it. The second result was that the occurrence of an error in a task doubled the average task time. The errors and their correction accounted for an average of 26%of total task time. For one subject, error time accounted for 50%of the time to complete the tasks. Moreover, if an error required real problem solving to correct (e. g. finding one s place in a large text file), task time was increased by an order of magnitude. These results tell us that recovery methods need to be efficient and that they need to be designed such that their use does not require problem solving. A third result was that subjects tended to follow a common path during error recovery. CMN noted that when subjects committed errors, they seemed to progress through five distinct error stages. They described these stages in the following excerpt:

1. *Error.* The user makes a mistake.
2. *Detection.* He becomes aware of the error.
3. *Reset.* He resets the editor to allow correction.
4. *Correction.* He undoes the effects of the error.
5. *Resumption.* He resumes error-free activity. (p. 177)

CMN divided the observed errors into four categories:

1. *Typing* errors, where subjects pressed the wrong key.
2. *Method-abortion* errors, where subjects started a method, but halted it because they realized it wouldn't t work.
3. *Method-failure* errors, where a method was executed correctly, but didn't t have the desired outcome. In these cases, the errors were immediately detected and identified, so the users could recover in a routine manner.

4. *Big* errors, which were actually method-failure errors that required the subject to engage in real problem-solving. In contrast to *Method-failure* errors, these typically occurred when the user got lost and had to navigate within the device to find where to correct the error and where to resume normal performance after the errors were corrected.

To model the subjects progression through the five error stages, CMN proposed some modest extensions to the basic GOMS model. They assumed that correction methods existed for each of the error types, and could be represented in the same way as methods for normal tasks. So the extensions were only to incorporate the error correction methods. These extensions included simple goal manipulation operators, but did not include new operators for error detection, error identification, or for resuming the normal task methods. Furthermore, the extensions were specific to the error types and were applied only to the first three error categories. Big errors were not modeled because they involved problem- solving skills and were therefore considered beyond the scope of GOMS.

For *Typing* errors, the recovery procedure after error detection was: look at the display, then compare the typed character with the desired character, press the delete key, type the desired key, and resume normal task execution.

For *Method-abortion* errors, the method was halted prior to completion of the Edit method (possibly before the occurrence of any observable action), so the recovery procedure was to pop the goal stack and re-execute the aborted method. This required the addition of an *Abort-command* operator, which popped the stack back to the last Selection Rule set. In addition to its stack manipulation results, the *Abort-command* operator symbolized a mental reset by the user.

*Method-failure* errors were indicated by a failure from the *Verify* operator after the *Edit* method. That is, the *Verify* operator failed to verify that the method had succeeded. Either the method didn't t perform the desired task, or the method performed the task incorrectly. In the former case, the original, normal method was re-executed. In the latter case, a correction method was required so that the subject could undo an incorrect action. This required adding another operator, *Generate-unit-task* , which created a task to do this.

For each error type, resumption of normal tasks was modeled without modification to the underlying goal stack of the GOMS model. That is, error correction methods were pushed onto the goal stack and executed as normal methods. When they were completed, the correction methods were popped from the goal stack and normal task execution resumed as it would with the completion of any other method.

**5. 1. 1 Assessment of the CMN Extension Proposal**

Although the ideas of the CMN proposal for error extensions provide a clear direction, many critical details were not completely specified. For instance, how does detection occur? They assumed that it just happens. Moreover, they assumed that detection occurred immediately after an error was made, that the type of error was identified when

it was detected, and that the failed operator also automatically generated a correction goal. These assumptions are critical because when detection happens can determine how an error is classified. For example, pressing the wrong key was classified as a Typing error if it was fixed immediately, but the same erroneous keystroke might have been classified as a Method-abortion error and, if it were not detected soon enough, it would be classified as a Method-failure error. If detection were further delayed, it could, in fact, have required real problem-solving, and been considered a Big error.

Another set of problems revolves around the additional GOMS operators used in error methods and how they are performed within the scope of the Model Human Processor. Is it true that users can actually pop their goal stack? If so, what does this mean? What happens to items in working memory? Although it seems plausible that people can back up several steps (or methods)at a time, there is an implied assumption associated with stack-popping that working memory items are also removed. Yet, there is ample evidence that forgotten items (those that might be removed from working memory)don t just go away. An underlying error-model that represents stack-popping as an error recovery mechanism should also reflect the messiness of memory.

A third weakness of the CMN error extensions is that they do not necessarily generalize to other domains. For example, CMN assumed the user would pop the stack back to the last selection-rule-set. Is this the case in other domains? Is this even desirable? Because of the nature of the CMN text-editing task and the errors that they observed, the last selection rule set made sense as a place to do a mental reset. Generally though, this assumption seems too constraining. Also, what does it mean, in the general sense, to generate a unit task that undoes the last sequence of actions? Can users always do this? The problem is that the unit task that CMN assumed would be generated was dependent on the type of error encountered. A different error type might require a different unit task for recovery, and the knowledge that the user would require to generate the proper unit task has not been made explicit.

Despite the weaknesses listed here, the CMN work provides a solid foundation for modeling human error. My focus in the remainder of this chapter is to formalize and extend CMN s human-error work, and to create a more general framework for human error. One of the main strengths of the CMN error extensions is that they were structured around specific mental stages that the user goes through while correcting an error. These stages provide the basis for the current work.

**5. 2 DESIGN GOALS AND SCOPE OF AN ERROR-EXTENDED GOMS**

My overall goal is to extend GOMS sufficiently to allow erroneous behavior to be comprehensively modeled. This would allow engineering models to be built for a much wider set of human-computer interfaces than is currently possible. Since the purpose of these engineering models is to improve human-computer interaction, the proposed theory 4 will take an approach that best illuminates how user-interface changes can improve a system s error tolerance. An alternative approach might be to extend the model in a way that best matches internal error-handling mechanisms. For example, there are several different internal mechanisms that humans use to detect errors. However, user-interface

changes may not improve detection by some of these mechanisms, so focusing on such mechanisms is tangential. Generally, my focus in this work will be to model elements that can have a direct effect on usability.

I will frame my error extensions similar to the CMN extensions, around user stages of recovery. In place of their task-specific Reset stage, I propose *Error Identification.* Identification was implicit in CMN s *Detection* stage, but knowing that a problem has occurred does not necessarily tell a user what the problem is. By making *Error Identification* explicit, we can better focus on how the interface can better support it. Removal of the *Reset* stage makes the framework more general. Not all recovery paths require that the user first reset the device. So, the stages I will use to structure my extensions are:

1. *Error.* The user makes a mistake.
2. *Detection.* The user becomes aware that an error has occurred.
3. *Identification.* The user identifies the error s type.
4. *Correction.* The user corrects the effects of the error.
5. *Resumption.* The user resumes normal task execution.

I will refer collectively to the Identification and Correction stages as *Error Recovery.* In the next section I will present a General Framework for Error Recovery and show how the recovery stages fit into it. Then I will discuss the recovery stages and how they might appear in a GOMS model.

**5. 3 A GENERAL FRAMEWORK FOR ERROR RECOVERY**

An important characteristic of the recovery stages is that the user s progression through them is not necessarily a linear process. How and why users move from one stage to another is critical to understanding error recovery. To clarify this process, we can view the recovery stages as a state diagram. Progression through the stages is characterized by movement between a set of mental states.

Figure 5-1 illustrates how the recovery stages fit into a simple state diagram of the user s mental-states during error recovery. From the user s perspective, the error recovery model is straightforward. During normal, routine performance, the human does everything right and continuously executes correct actions. I refer to this as the *Normal* state. But, when an error occurs, the human enters a Quasi-Normal state where everything seems normal, but where some failure is imminent. The transition between the *Normal* and *Quasi-Normal* states reflects the Error stage. The user can continue performing correct actions within the *Quasi-Normal* state until the user detects that something is wrong, prompting him or her to recover. This transition from the *Quasi-Normal* state to the *Recovery* state reflects the Detection stage. Once in *Recovery* , the user identifies the error (the Identification stage)and takes the necessary corrective actions (the Correction stage). When error correction is complete, the user returns to normal operations (the Resumption stage).

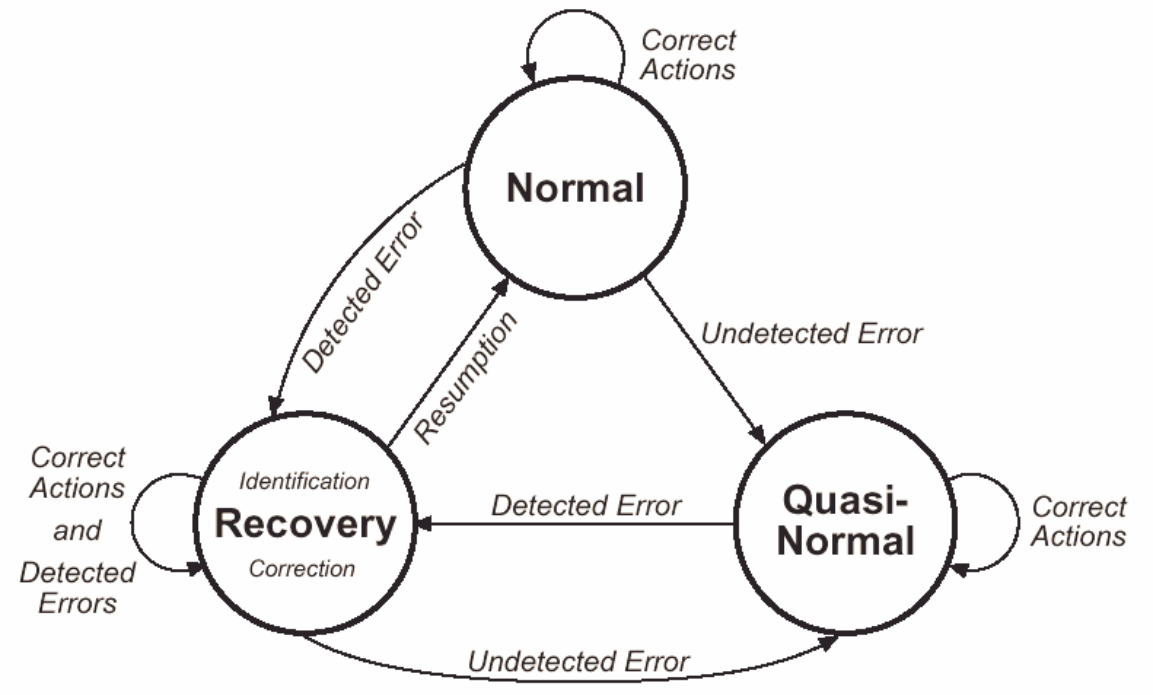



**Figure 5-1.** A general framework for error recovery. The state diagram illustrates user mental states while moving through error recovery stages.

Two additional transitions reflect the nonlinear nature of error recovery. The first of these occurs when the user detects an error as the action is performed (as with the CMN errors). In these cases, the user jumps immediately to the Recovery state. This transition can occur from any state, including Recovery. The second transition type can occur when an undetected error occurs during the Recovery state. In such a case, the user reenters a quasi-normal state, where error correction seems to be proceeding, but where another failure is imminent.

Movement through the mental states can be illustrated with a simple example: The task of entering your office in the morning. A possible procedure for this is represented in the following method:

Method_for_goal: Enter office
On_error: Remove key and restart current goal.
Step 1. Look for and choose correct key.
Step 2. Insert and turn key.
Step 3. Verify door is unlocked else go back to step 1.
Step 4. Open door and step through.

The On_error syntax indicates the name of a correction procedure, Remove key and restart current goal, with the normal procedure, Enter office. If any of the steps in the normal procedure fail, control would be transferred to the correction procedure. The task steps are: find the correct key, insert it into the lock and turn it, verify that the door is unlocked, and open the door and step through. If, during Step 1, you fail to find the right key and instead locate the wrong key, you would move from a Normal State to a Quasi-

Normal State. An error had occurred in Step 1, but it was not yet detected. In Step 2 the key may not fit or may not turn, so Step 2 would fail, indicating that something was wrong. You would then move to a Recovery State and start the procedure of removing the key and trying again. The correction method includes the transition from the Recovery State back to the Normal State.

Alternatively, if Step 1 had been successful, but you turned the correct key incorrectly (like in the wrong direction), the error would cause a transition from the Normal State to the Quasi-Normal State. Here, the error might only be detected when you attempted to verify that the doorknob turns. The explicit check in Step 3 would indicate a problem, which would cause the transition to the Recovery State. Here, the recovery plan specifies that you go back to Step 1 and resume normal task execution. So to recover, you simply to make the transition from the Recovery State back to the Normal State and try again.

The diagram is important for interface designers because it makes clear several important issues when designing for error tolerance:

1. There may be a delay between when an error occurs and when it is detected. Any actions while made during the Quasi-Normal stage may need to be undone anyway, so efforts should be made to help users detect errors as soon as possible after the errors are made. Furthermore, because there is a potential delay, it may be difficult for the user to identify the source of the error and choose an appropriate corrective procedure. Interface aspects that help identify errors and their corrective actions will reduce the time users need to spend in real problem-solving to correct problems.
2. Errors can occur while in a Recovery State, so error correction methods need to have their own recovery methods.
3. Correcting an error is not the end of recovery. Designers also need to consider how users will resume normal task performance, and system design should reflect this.

## 5. 4 STAGES IN HUMAN ERROR-RECOVERY

This section discusses in more detail the error recovery stages outlined earlier: Error, Detection, Identification, Correction, Resumption. Fundamental issues for how these stages can be addressed within GOMS are presented.

### 5. 4. 1 Error Stage

The error stage is the point at which the user commits an error. To model the occurrence of human error in GOMS models, there are four fundamental questions that I will address:

1. **How can the GOMS architecture fail?** Assuming an information processing architecture similar to that embodied in GLEAN (as discussed in Section 4. 2), how can the components of such an architecture fail? In Section , I will discuss how failures can occur within the scope of GOMS.

2. **In what form will errors be expressed?** That is, how can errorful behavior be differentiated from normal behavior based on a trace of actions? I will show in Section that by comparing the erroneous actions with those from a normal output we can see several different error types.
3. **What types of errors can be created in a GOMS architecture?** How can we create the types of errors that users make so we can use GOMS to help design systems that are error-tolerant? In Section I will show how a large variety of errors can be generated with a simple technique.
4. **How can errors be generated in a GOMS model?** What classes of algorithms are available for generating errors, and in which situations are they best employed? Section discusses various techniques for error generation and provides examples of when each technique may or may not be most appropriate.

*How can the GOMS architecture fail?* The GOMS architecture is a standard information processing system where information is passed from processor to processor as the simulated user performs actions. Failures can occur within the sending processor or the receiving processor. Failures can also occur within a processor as it is executing an operator.

**Cognitive Processor Failures.** The Cognitive processor acts as a central executive, stepping through task methods, and sending operators to peripheral processors for execution. In its executive role, the Cognitive processor executes steps, by examining each operator in the current step and sending it to the proper processor to be executed. It then determines the next step to execute, and makes it the current step. So, it iterates through the steps of a method, and it iterates through the operators of each step it executes. In this role, the Cognitive processor can fail in how it iterates through either the steps or the operators, or it can fail to send an operator to the proper peripheral processor. Iteration failures can take the form of omissions, repetitions, transpositions, or some random assignment. It is assumed that iteration failures at the step level will occur within a single method, such that if a step is incorrectly chosen, the chosen step will be within the same method as the correct step. Likewise, it is assumed that iteration failures at the operator level will occur within a single step.

The Cognitive processor has the additional role of executing mental operators, such as control flow (Decide, Goto), stack manipulation (Accomplish goal, Return), task knowledge (Get), and analyst-defined operators. In this role, it can fail by not executing an operator or executing it incorrectly. Control flow can fail if the Cognitive processor makes a decision incorrectly (i.e., matching conditions incorrectly while executing a Decide operator)or by jumping to the wrong step during execution of a Goto operator. Stack manipulation can fail if the Cognitive processor calls the wrong method, or if it fails to pop a complete method from the goal stack. Task knowledge operators can fail if the Cognitive processor fails to get correct information from the task description. This can cause tasks to be performed incorrectly and it can cause iteration failures at the task level.

**Working Memory Failures.** Working memory (WM)serves as a central repository for all declarative knowledge that is passed between processors. Furthermore, items from Long-term memory must be placed in WM before they can be used. WM items are explicitly stored in the form of a tag-value pair. Stored items will overwrite other items with the same tag name, such that all tags within WM are unique. Once an item is in WM, it can 8 be used by any processor. WM items are explicitly deleted from WM when they are no longer needed. WM can fail by forgetting (deleting)items prematurely, or by confusing items when they are needed. It is assumed that failures will increase as the number of WM items increases. It is also assumed that individual items are more likely to be forgotten the longer they remain in WM without being used. The probability of a confusion between two WM items is assumed to increase with the number of features that the items share. That is, the more similar two items are, the more likely they are to be confused with one another.

**Perceptual Processor Failures.** The Perceptual processors are responsible for detecting (Wait-for operator), perceiving (Locate operator)and processing auditory and visual information from the simulated user s environment. Perceptual information is stored in WM. The Perceptual processors can fail by not detecting events that happen, falsely detecting events that don t happen, or misperceiving available information. Incorrect detections can cause Wait-for operators to execute too soon or too late. Misperceptions can cause perceptual confusions (e. g., locating the wrong object) or improper encodings (e. g., wrong object features are placed in WM).

**Motor Processor Failures.** The Motor processor translates motor operators (Move, Press, etc. )with corresponding arguments (e. g. location, button, key, etc. )into the proper motor movements. The Motor processor can fail by not executing an operator, executing an operator incorrectly, or spontaneously executing a motor movement (e. g. muscle twitch). It is assumed that motor movements consist of a set of features, such as digit or limb, direction, distance, force, etc. Failed movements where only a few features are incorrect are considered more likely than failures where numerous features are incorrect. So for instance, it is more likely that a motor error would cause an adjacent key to be mistakenly pressed than for some distant key to be pressed. This also assumes that motor failures will not be completely random.

*In what form will errors be expressed?* In what form do errors appear in a GOMS model? That is, if we compare a trace of correct actions with one that includes errors, how can we tell the difference? The problem with answering these questions is that each task domain will produce a different set of errors that are specific to that domain. For instance, an omitted keystroke may mean that a user navigates to the wrong location in one context, but could mean that a financial transaction failed in another context. In an action trace, both cases would appear as an omitted keystroke. When considering what how errors appear for simulated users, it is useful to strip away some of the task-specific context of these errors to arrive at a more general view of error expression. At an *observable* level, errors are classified in terms of the overt actions that they produce. In GOMS models we can see four fundamental types:

a. *Error of commission.* An action is performed incorrectly.
b. *Error of omission.* An action is not performed when required.
c. *Error of intrusion.* An action is performed when it is not required.
d. *Transposition error.* The performance of two actions is switched.

Given a task context, these simple error categories translate to a wide variety of more interesting error types. Note that in normal task execution, very little context is used by the GOMS architecture. The only context used is of the operator currently being executed, such as the contents of working memory and a limited view of the task environment. Other contextual information is not considered, such as how frequently the user executes each method, the order in which methods were learned, the user s history of performing the task (including past successes and failures), what the user is saying or thinking while performing the task, other visual and environmental factors outside visual focus, and other device and environmental elements not modeled, such as extra display items that are not considered part of the current task). For design and error-classification purposes, this contextual information is critical, but it is not necessary to generate errors. I will demonstrate in Section that we can generate a sufficient variety of error types to test error tolerance within the limited scope of GOMS.

To differentiate correct from errorful performance during a task model execution, the execution trace in question is compared with an execution trace that is known to be correct. Since we don t have any contextual information available in the trace, the only way to find errors that were produced is to look at the overt actions. At this level of analysis, errors appear as one of the four observable types. The architecture (and thus the analyst), of course, would know how an error was actually generated, but the form in which it appears could be much different. I will demonstrate this in Section 5. 4. 2.

*What types of errors can be created in a GOMS architecture?* How can we generate each of the four observable error types within a GOMS architecture? An error of commission can occur if a)a peripheral processor fails to execute an operator correctly or, b)an operator s arguments are corrupted. In both cases the execution trace would show an incorrect execution of the correct action. Argument corruption can produce a rich assortment of errors, especially with the Accomplish_goal operator. If the goal argument were to be corrupted, the wrong method would be executed.

An error of omission can occur if a)a peripheral processor fails to execute an operator, b)a goal is prematurely removed from the goal stack, or c)the Cognitive processor skips an operator while executing a method. In each case the error appears in the execution trace as omitted action.

An error of intrusion can occur if a)a peripheral processor spontaneously performs an action (e. g. a muscle twitch), b)the Cognitive processor experiences an iteration failure at the step or operator level that causes an extra operator to be executed, or c)the Cognitive processor spontaneously pushes a goal onto the stack. In the execution trace, these errors appear as an extra action between two correct actions.

A transposition error can occur if a)the Cognitive processor switches the order in which two operators are executed or, b)the Motor processor transposes the sequence of two motor movements. These errors appear in the execution trace as two correct actions that have been transposed.

Although the generation of these observable error types may seem very similar, the form of their expression points out their more apparent differences. To clarify how each of the error types could be created, consider four cases of a user performing a simple button-pressing task. The user s environment consists of three buttons on a workstation: Button-A, Button-B, and Button-C. The task requires that the user press Button-A first, then Button-C. The GOMS method for this might look like:

Method: Do task
Step 1. Press_button (A).
Step 2. Press_button (C).

**Case 1 -Error of commission.** If the Motor processor miss-aimed the finger for the first button press, B might be pressed instead of A. The same result would occur if the Motor processor were given the argument B instead of A (argument confusion). In either case, the user would have performed an error of commission. If Step 2 were performed correctly, this would appear in an execution trace as: B pressed, C pressed.

**Case 2 -Error of omission.** If the Motor processor failed to press anything for Step 1, or if the visual processor could not see Button-A, preventing the Motor processor from executing the step, the user would have performed an error of omission. This could also occur if the Cognitive processor skipped Step 1. If Step 2 were then performed correctly, this would appear in an execution trace as: C pressed.

**Case 3 -Error of intrusion.** If the Motor processor spontaneously pressed Button-B after Step 1 and before Step 2, the user would have performed an error of intrusion. This could also happen if a corruption in the Cognitive processor caused an extra Press operator to be executed. If Step 2 were then performed correctly, this would appear in an execution trace as: A pressed, B pressed, C pressed.

**Case 4 -Transposition error.** If the Cognitive processor switches the ordering of Steps 1 and 2, the user would have committed a transposition error. This would appear in an execution trace as C pressed, A pressed.

This example shows that by allowing for an architecture that can fail in some very simple ways, we can create each of the primitive error types. If we add more contextual information to these primitive error expressions, they can also be seen in their more familiar forms, such as errors from Reason s taxonomy. For instance, if the user in Case 2 indicated that the label for Button-A could not be seen, we could instead refer to that error as a perceptual confusion. Although the contextual information is unnecessary for generating errors, it is critical for discussing design issues, such as how to prevent perceptual confusions. I will discuss such design issues in Chapter 8.

It is useful to map the error types possible in a GOMS architecture to the error types classified by Reason (as discussed in Section 2. 6). Using Reason s taxonomy, we should be able to see all the errors at the skill-based level because they typically involve visible actions. Errors at his rule-based level could be generated in a GOMS architecture, but they could not be classified as such based solely on observable actions. Rule-based errors typically involve how rules (methods or operators)are selected and may not result in action patterns that would distinguish them from errors at the skill-based level. Errors at the knowledge-based level typically involve complex mental processes and are also beyond the scope of GOMS.

*How can errors be generated in a GOMS model?* Errors can be made to occur in a GOMS model by three approaches: a)Random, b)Systematic, or c)Model-based. Each approach has certain advantages, and might be used at different stages of the development process. A comprehensive modeling tool would make use of all the techniques.

**Randomly generated errors.** Perhaps the simplest form of error generation is to implement a stochastic algorithm in which each operator is assigned a failure probability. The GOMS Cognitive processor would then check each operator prior to executing it and generate an error with this probability. This probability could be adjusted so that the error rate of the simulated user would resemble error rates from real users performing a similar task. An alternative might be to assign failure probabilities to each of the GOMS processors (or classes of operators). This would allow each processor (operator class)to have a different failure rate, and might result in a more realistic mix of error types for a given overall error-rate.

Two benefits of the Random approach are that it is relatively simple to implement, and, if a large enough sample were taken, it would provide broad coverage when testing the robustness of error recovery routines. In other simulation domains, such as manufacturing, this is often the only reasonable method for testing a model because of the large number of simulation variables involved.

Random error generation is not always the best option, however. For example, model development is easiest if areas of the model can be isolated for testing. This allows the modeler to test one area completely before moving on to another. But random error generation, by nature, does not allow for such isolation, so modelers would want the option to use another error generation approach.

**Systematically generated errors.** Another simple way to generate errors is to allow the modeler to select specific operators to fail, and how they will fail. The GOMS simulator would then look for marked operators and cause them to fail. The advantage of this method is that it allows the modeler to target specific areas of a model to test. The main disadvantage is that testing recovery routines is a manual process and therefore tedious and error prone.

A variation on this method that would be useful for final testing would be to systematically test each operator in a task-model. The system would perform one complete model execution per operator in the model, ensuring that each operator failed exactly once. While comprehensive in theory, it would be very difficult to ensure full coverage of recovery routines using this variation. Because errors often occur in clusters and errors can occur during recovery routines, the system would have to test combinations of operator failures to be truly comprehensive. But, the number of failure permutations to ensure sufficient coverage would quickly make the problem intractable.

**Model-based error generation.** The most sophisticated method of error generation would be based on a psychological model of error-producing mechanisms. For example, a model of working memory loss could cause working memory items to decay, disappear, or become corrupted, based on the number of items in working memory, and the length of time each item was stored. This is the type of model used by Smelcer and Byrne, as discussed in Chapter 4. Similar failure models could be built for other components of the GOMS architecture. A disadvantage of this method is that, for many of the psychological 12 processes involved; the requisite failure theory does not exist. However, a relatively sophisticated failure model for working memory would not be difficult to implement. Such a model could allow a variety of error types to be generated with a single failure mechanism.

### 5. 4. 2 Detection Stage

The detection stage begins when the user first detects that an error has occurred. It is at the detection stage that user actions diverge from normal actions to error recovery actions. Sellen (1990)proposed several different classes of error detection mechanisms, only two of which are within the scope of GOMS.

*Action-based detection.* Action-based detection is based solely on the feedback from the erroneous action itself, what Sellen refers to as *self-detection.* Typically it occurs based on visual, auditory, proprioceptive, or mental (e. g. plan verification)feedback. It can also occur prior to any overt action (perhaps after method selection, but before first step). Examples include detecting a speech error by hearing yourself speak, and skilled data entry people who recognize a wrong keystroke because it didn't feel right.

This is the type of detection alluded to by CMN s Method-abort error. The user started a method, but realized before it was done that it was not the intended method. CMN could observe this type of error, even though there were no overt actions, because their subjects were thinking aloud as they were performing.

Action-based detection of errors from overt actions could only be implemented in a GOMS framework if appropriate sensory feedback were built in. The self-detection mechanism would then have to compare the intended action with the actual action. For example, a motor self-detection mechanism might match the intended target of a Press operator with what the feedback sensor perceived as the actual target. A mismatch between intended and actual would signal an error. Detection of errors before they occur (as with the Method-abort error)could be implemented by a meta-execution, in which

operators and their arguments (e. g. the target of a Press operator)were examined prior to execution to ensure that they were as intended. For instance, if the argument to a Press operator was a color instead of a physical object or location, the mismatch would be detected.

*Outcome-based detection.* Outcome-based detection occurs when something in the external environment alerts the person that the outcome of an action is not as desired. Where Action-based detection depends on feedback from the actual movement or action itself, Outcome-based detection depends on feedback from the environment as a result of the action. For example, if a computer user noticed the letter A on the screen after intending to press Z , the undesired outcome would alert the user of a problem. This form of detection relies on cues from the environment (device), and is represented by the GOMS Verify operator. The main assumption behind the Verify operator is that sufficient environmental cues exist for the user to make a deliberate check that the proper effects of an action are present. What is not currently specified in GOMS is what happens if the Verify operator indicates an error. An extension to the Verify operator is needed to re-route control flow if an error is detected.

*Detection through external limiting functions.* This form of detection occurs when something in the external environment prevents the person from performing the current action. When the device makes it impossible for normal task execution to continue, the user will usually be alerted to a problem. For example, if an automobile driver attempts to pull the keys out of the ignition while the car is still running, the action to remove the keys will fail because the lock design prevents it from happening. Likewise, computer operating systems often limit the movement of screen cursors to visible portions of the screen. This prevents dual-computer users from moving the cursor from one computer onto the screen of another computer. Another computer-based limiting function prevents disabled screen menus from being selected. Although the limiting functions used on the computer screen are not physical in same sense as the ignition lock, the technique does prevent certain actions from occurring, and they will often alert the user that something is wrong. This type of detection would only be possible in GOMS if there was tactile, perceptual or proprioceptive feedback that could indicate to the simulated user that an action could physically not be performed.

*Detection by uncertainty in action.* Here, the user realizes that they don t know what to do next. *Detection by uncertainty of action* occurs either when high-level goals are forgotten or when low-level goals or steps are forgotten. It is typical of error situations in which the user says, What was I doing? These types of errors could be modeled in GOMS by removing goals from top or bottom of the goal stack, however, modeling the detection of such goal corruptions would require more of an awareness of the situation than GOMS allows. Modeling uncertainty of what to do next in GOMS implies knowing that a task has not been completed even though the goal stack is empty. But, an empty goal stack is a normal situation for the simulated human and would not normally signal an error. Only by adding more awareness to the architecture, possibly with a meta-level process that examined stack integrity, could an empty or corrupted goal stack be considered detected.

*Detection by reminding.* Detection by reminding occurs in the context of an external event interrupting the primary task. Another goal has taken over, and later, some other event reminds the person that the primary task was not finished. These reminders can be external, task-dependent, or internal. For example, an internal reminder, hunger, tends to remind a person that they forgot to eat. This form of detection is beyond the scope of GOMS because it relies on some type of memory that is external to the task.

*Which forms of detection are most relevant for modeling interfaces?* From an interface-designer s perspective, two of the detection forms are most relevant: *Outcome-based detection* , and *Detection through external limiting functions.* These two forms of detection represent the soonest that a user can detect a problem through external means (outcome- based detection), and the latest a user can detect a problem through external means (limiting function). Internal forms of error detection, such as reminding or action-based, are fortuitous if they occur, but are not reliable enough to base a system s design on. For example, it a computer system responds to a user error by limiting further action, creating an auditory signal, or presenting a message in large red letters, we can be reasonably certain that the user will detect the problem. The conditions under which the other detection classes would be successful are not as well defined.

For design, two questions regarding error detection are most relevant:1)How can an interface be enhanced to improve the user s detection of errors, and 2)How long can an error persist before the user will detect it? The first question pertains to frequent error types, which are those we anticipate as being most likely, such as typographical errors for data entry tasks. To reduce the frequent errors, a designer could first model those interface aspects that lend themselves to deliberate checks. If there were not sufficient device cues such that the simulated user could verify that the required tasks were performed correctly, then the interface would need to be changed to allow such a model.

*Outcome-based detection* would allow the modeler to address deliberate environmental checks by users. This type of functionality was intended for the Verify operator. After an action or series of actions has been completed, the Verify is used to compare the state of the environment (device)with the intended outcome. If there is a mismatch, then the simulated user is alerted to a problem.

The second question asks how long a user can continue to perform actions after an error has occurred. It pertains to less predictable error types; those we do not necessarily anticipate, such as a system crash for computer-based tasks. It relates more to the overall robustness of the interface and raises two related questions:1)Are there any error types that can get through the normal defenses, and 2)Is there a path of graceful degradation for the user? What is needed is a way to catch errors other than what we anticipate. We also want to avoid overloading a GOMS model with more explicit checks than is realistic. In the extreme case, we might have a Verify operator after every operator, but it isn’t realistic to believe that humans perform that many explicit checks. There is also the possibility that the Verify operator will fail, or that correction methods will fail.

*Detection through external limiting functions* is the latest point in the execution of a task that the user is likely to detect an error. It is the point when some aspect of the interface forces the user to stop normal execution and realize that something is wrong. With the knowledge of where this detection will occur, the designer may be able to build better limiting functions into the interface, or move them closer to the point where the errors are actually occurring.

*The relationship between operator failure and error detection.* We can view human error in GOMS as a possible outcome of the execution of an operator. To simplify, we can view the outcome as binary: an operator either succeeds or it fails. If an operator fails, it is for either internal or external reasons. *Internal failure* refers to errors internal to a GOMS operator that occur because of a failure within the executing processor. *External failure* occurs when bad information being passed in as the argument for an operator causes that operator to fail. The external failure could be because of an internal failure in a previous operator, a failure in the device, or some other factor in the user s environment (such as smoke obscuring a critical display). For example, consider the following GOMSL steps:

Step 1. Look_for_object_whose label is isStartl_ and_store_under <Start_button>.
Step 2. Point_to <Start_button>.

If, during the execution of Step 2, the Motor processor caused the Point_to operator to miss its target, the Point_to operator would suffer an internal failure. If, however, the Look_for operator in Step 1 failed to acquire the correct target, the Look_for operator 15 would have suffered an internal failure. But the subsequent Point_to operator would be supplied the wrong object in working memory, and would point to the wrong object. This would be an external failure in the Point_to operator because the target argument was bad.

Thus, although the Look_for operator was the original problem, an error could not be detected until the Point_to was executed.

This example points out several interesting features regarding human error that are consistent with the literature. First, the operator whose failure is detected is not necessarily the primary source of the error. Second, errors may not be detected until they cause a subsequent operator to fail; this might be several steps later. Third, errors tend to cluster, such that the effects of one failure may directly or indirectly cause other failures within a relatively short time.

*The transition to the Recovery state.* The final action in the Detection stage is what happens after an error is detected. How should control be transferred from normal task execution (the Quasi-Normal state)to error identification and correction (the Recovery state)? There are two options for the type of recovery routine whereto control might be transferred:1)routines specific to an error type and, 2)general routines that can apply to multiple error types. A specific recovery routine would be appropriate if the simulated user knew the error type and the necessary recovery actions. Error identification is

implicit in this case, so the user can move directly to error correction. This is typically the case with Verify operators where a specific environmental cue is being checked.

A general recovery routine would be appropriate for situations in which the same recovery routine would apply to several different error types. For example, in the CMN text-editing task the first step in error recovery was always to reset the device, irrespective of the error type. If an interface is designed such that a few general recovery methods apply to any abnormal situation, each normal method can be associated with fewer recovery methods. This design strategy better supports recovery hierarchies (as opposed to each method having a unique recovery method). Any type of error that was not caught by a deliberate Verify would be sent to the general recovery routine for that method. In such cases, the exact error type may not be known, so the next step would be to identify the problem.

**5. 4. 3 Identification Stage**
The error identification stage is where the user determines what caused the problem. This stage is very important for interface design because it can determine whether the user goes directly to the most appropriate correction procedure, or instead drops into a problem- solving mode to further diagnose the situation. In Sellen s review of the error identification literature (Sellen, 1990), she notes that a common thread is that if people detect their own errors (through some form of self-detection, such as action-based detection), they are very likely to have also identified the cause. If, on the other hand, something in the person s environment points out the error (through outcome-based or limiting function-based detection), identification is very unreliable because it can involve complex problem solving and memory of past action events. Although modeling such behavior is beyond the scope of GOMS, we can use GOMS to determine how well a system supports error identification.

From a design perspective, what we are really interested in is how the interface can facilitate the identification process. How can we design systems such that the user does not have to engage in complex problem solving to recover from errors? If we ensure that the interface contains sufficient cues to allow errors to be identified uniquely and easily, we will have a robust system with at least one reliable recovery path. In addition, we can model such a device in GOMS using existing modeling constructs, such as perceptual operators and selection rule sets. For example, if a computer user were faced with a dialog box containing an error message, a general routine might be to read the error message (modeled by the Look operator), then use that message to decide how to correct the error (modeled by a Decide operator or a Selection Rule Set).

**5. 4. 4 Correction Stage** The correction stage is where the user corrects the error. Two issues need to be considered when modeling error correction: a)The separation of correction and normal tasks, and b) modeling the error correction goal stack.

*Correction versus normal tasks.* Where do we draw the line between error correction and normal task execution? Should the correction procedure duplicate steps in a normal procedure? Or should the correction procedure be limited to undoing damage? We have

two strategies that have a very strong effect on the way correction procedures might be modeled: a)We enforce a clear distinction between normal and correction procedures, or b)We use normal procedures whenever it is efficient to do so.

Enforcing a clear distinction between normal and correction procedures ensures that error correction is complete before resuming normal operations. For instance, if a web user had to navigate from Point A to Point B through a series of links, but accidentally arrived at Point C, the recovery method using this strategy might state that the user should attempt to locate a navigation path directly from Point C to Point B. Here, when the user gets back on the correct path (at Point B), error correction is complete. By maintaining a clear distinction, we can minimize the execution time at the cost of learning a specialized recovery method. The downside is that if too many specialized methods are required, the task will be too difficult to perform.

Making maximal use of normal procedures during correction requires that the user back up to a known point on the correct execution path to allow the initial, error-containing procedure to be re-executed. Using the same web navigation example, the correction procedure with this strategy would state that the user should back up to Point A and re-execute the task. Here, we keep recovery simple by making use of normal procedures, but at a cost of slower execution time. However, this strategy also requires that the modeler know how, and when, the user should return to normal tasks. So, the effort is shifted from writing correction methods to determining resumption logic.

An open question regarding error correction is which of the preceding strategies more closely matches actual behavior? The answer is probably that both strategies are used, depending on the structure of the task, how the task is learned, and the past experience of the user. The learning literature suggests that new procedures are learned in terms of existing procedures. So, if the user already knows the normal procedures, the correction procedures would only require learning what was different and when the new variation 17 should be used. This would support making maximal use of normal procedures. But, on the other hand, normal task resumption with this strategy often requires a more complex control structure. The trade-off in mental effort is learning specialized routines versus executing complex control logic. Observation of computer users suggests that people will often choose to perform sub-optimally if it means less thinking is required. How the task structure affects this trade-off will probably determine the strategy used.

**5. 4. 5 Resumption Stage**

The resumption stage is where the user resumes normal task execution. This is the most difficult part of extending GOMS to human error. How do we determine where and how a user can resume normal tasks? CMN assumed that the user would want to go back to the last Selection Rule set. This made sense in their simple text-editing task, but this isn't always the case. In a complex task, the user may need to go back several levels before normal tasks can resume. This is especially true in catastrophic failures where the user must essentially start over, or when errors occur while recovering from a prior error.

There are numerous ways that a user might wish to resume normal tasks. Each possibility would require a distinct resumption operator to return control to normal task execution. Six of these operators will be discussed in the following sections. Examples of how they are used in a GOMSL implementation are presented in the next chapter. For the first two operators, it is assumed that correction (and resumption)can be accomplished by re-executing the failed step or method, without the use of a recovery method (this assumes the error is detected within the method where it occurred). For each of the remaining operators, it is assumed that a correction goal is already on the goal stack and that error correction is complete. Thus, resumption for these operators includes some form of goal stack manipulation.

*Redo the last step.* For simple errors, such as motor slips, the most appropriate course of action is to re-execute the step that failed. For instance, if the task included pressing a button and the user omitted the Press step, the most appropriate course of action would be to attempt to press it again. This option would only work if the error were caught prior to executing another step. The simplicity of correction in this case obviates the need for a separate correction method.

*Redo the current method.* For errors that are not detected prior to executing another action, it may be best to re-execute the error-containing procedure again from the beginning. An example of this might be if a person were using a software wizard to create a document. Such wizards typically take the user through a series of questions to help solve a problem. If an error is made in the middle of this sequence, it may make the most sense to cancel the sequence and start over. This operator causes normal execution to resume at the first step of the current method (the method where the error was detected). This type of error resumption would typically only work if a deliberate Verify operator detected the error within the error-producing method. Otherwise, resumption would be taking place from another method (i.e., a recovery method). This distinction will be made clearer in Section.

*Redo the last method.* Consider a similar situation as described for the *Redo the current method* operator, but where the error was detected by something other than a deliberate 18 Verify operator. Then, identification would occur in the error recovery routine instead of the normal routine. Here, it wouldn't make sense to re-execute the current method, because the current method is a recovery method. Instead, we would need to *Redo the last method.* This operator would pop the correction goal off the stack and resume execution with the first step of the method where the error was detected.

*Restart a specific method.* In some situations, the user may need to go back further in the goal stack of normal procedures to resume execution (or to finish the correction process). This was the assumption by CMN, but they limited the restart to the last selection rule set. There is nothing special about selection rule sets that makes them a good place to resume normal task performance. There may be cases where more catastrophic errors (in the sense that their consequences are difficult to undo)require that users go farther back to restart. For example, the installation procedures for some computer operating systems are such that a single mistake by the user requires that the computer s disk drive be

reformatted. This might require going back several goals on the goal stack before an appropriate resumption point is found. It does make sense, however, to make sure that the goal that is resumed be on the goal stack. In other words, the user should not be able to restart with some arbitrary procedure. In this case, the goal stack would be popped back to the desired method and execution would resume with the first step of that method.

*Restart the current task.* The most extreme case of *Restart some other method* is where the error consequences are so far-reaching or the situation is so complex that the user must start the current task over. This would be the case if the user became so disoriented that restarting from the beginning was easier than trying to mentally reconstruct the current situation. This could be modeled the same as the *Restart a specific method* operator, but it seems that such an operator would be used often enough that having this special case would clarify the modeler s intentions.

*Return with goal accomplished.* This is the normal GOMSL method for returning control from one method to another. In some situations it can be used to return control to the normal task procedure after the correction goal has been accomplished. If correction is completed in the correction routine, and the state of the system is such that user can resume execution where he or she left off in the normal procedures (prior to error detection), then nothing special needs to be done to the goal stack or other control structures. The recovery method is popped from the goal stack and execution is resumed.

## 5. 5 A PROPOSED TECHNIQUE FOR APPLYING GOMS TO ERROR TOLERANT DESIGN

This section proposes a methodology for building error-tolerant systems using GOMS modeling techniques. The steps are listed as follows:

**1)Start with a GOMS model of correct performance.** This should be a refined model that includes all tasks related to either mission critical or safety critical goals. See Kieras (1999)for a tutorial on developing GOMSL models with GLEAN.

**2)Identify mission-critical and safety-critical portions of the interface.** Any method that can directly or indirectly affect the success of the primary tasks, or any method that can affect human safety should be considered mission-or safety-critical. It is for these methods that error tolerance is essential.

**3)Determine places where deliberate checks are applicable and verify that task methods provide adequate feedback for the user to determine success or failure of the method.** Check to see whether there is system feedback that will help the user determine whether individual methods have succeeded or failed. Where appropriate, this should be determined before leaving a method, but for generic or library methods, this may not be possible.

**4)Define recovery points within the interface for users to reset mentally.** Determine stable system modes or states where the user will have time and the necessary information to identify errors, take corrective actions and resume normal task execution.

**5)Design a hierarchy of recovery routines around recovery points to protect mission- critical areas.** Errors tend to cluster and recovery routines place an additional burden on working memory. Additional errors within recovery routines should therefore be considered likely. Error recovery routines should be designed to include additional mnemonic and visual aids, and they should be built hierarchically. That is, each recovery routine should have its own recovery routine should an error occur during error recovery execution. The hierarchy should be defined to the highest point where errors can be usefully handled within the context of the task. For example, the final recovery method for most computer tasks might involve re-booting the computer and starting the task over. But other recovery solutions could include reinstalling the system software, reformatting a disk drive, or replacing an entire hardware system. While these more complex remedies may be appropriate for technicians or systems administrators, they probably aren't useful for many non-technical users. The point is that the analyst will need to determine the proper stopping point as dictated by the task.

## 5. 6 SUMMARY:AN ERROR-EXTENDED GOMS (EGOMS)

CMN saw the necessity of modeling error recovery early in their development of GOMS. This chapter has discussed their early work, the stages of error recovery, and how those stages fit into the General Framework for Error Recovery, a state diagram of user mental-states. The topics covered in this chapter form the basis for an Error-Extended GOMS (EGOMS).

The discussion was framed around user mental-states of errorful task performance. The General Framework for Error Recovery illustrated user transitions between three key mental states: Normal, Quasi-Normal, and Recovery. These states were then related to a series of stages that users must traverse while recovering from errors: Error, Detection, Identification, Correction, and Resumption. The state diagram indicates why traversal through the recovery stages is not necessarily a linear process, and points out some key issues to consider when designing for error tolerance. Within each recovery stage, I addressed the fundamental issues required for extending GOMS theory to human error. The next chapter discusses an implementation of EGOMS and illustrates its use with example interfaces.

## 5. 7 REFERENCES IN THIS CHAPTER

Sellen, A. J. (1990). *Mechanisms of Human Error and Human Error Detection.* Unpublished Ph. D. , University of California.

Kieras, D. E. (1999). *A Guide to GOMS Model Usability Evaluation using GOMSL and GLEAN3* (Technical Report). Ann Arbor: University of Michigan.

# Appendix B

## Predicting Situation Awareness Errors using Cognitive Modeling

Troy D. Kelley
Debra J. Patton
Laurel Allender

Army Research Laboratory
AMSRL-HR-SE
APG, MD, 21005-5425

ABSTRACT

The U.S. Army Research Laboratory's Human Research and Engineering Directorate has developed a series of computational cognitive models within the Atomic Components of Thought-Rational (ACT-R) (Anderson & Lebiere, 1998) cognitive architecture to attempt to predict errors made by soldiers on navigation-related tasks while they wore helmet-mounted displays (HMDs). The study used 12 infantry soldiers who were required to perform a series of navigational tasks while wearing HMDs. During the exercise, the soldiers were asked a series of probe questions pertaining to information that had been displayed on the HMD. An hypothesis was developed about the causes of errors that soldiers made in response to the probe questions. The hypothesis was used as the basis for ACT-R models which, in turn, were evaluated against the data. The modeling effort here was not a typical example of "curve fitting," in which modelers examine data that they hope to later match by developing a cognitive model. Instead, these models were purely predictive from the beginning. The error data were not examined until after the initial models were completed. Results indicated that, after some adjustment, the computational cognitive models were able to predict the likelihood of soldiers answering a probe question correctly or incorrectly, based on the activation levels associated with the multiple memory chunks pertaining to a given question. Future directions of predictive cognitive models for interface design are discussed.

### 1. INTRODUCTION

The use of computational cognitive architectures for the investigation and prediction of human error is a new and important development for human-system design. In the past, traditional error analysis has taken the form of error classification or the development of error taxonomies (Norman, 1981; Rassmussen 1982; Reason, 1990). While error taxonomies can be useful for the identification of general types or classes of error, taxonomies do not necessarily provide the explanatory and predictive power required to understand the underlying cognitive processes or to prevent error occurrence.

Error classification alone does not provide the ability to examine "what-if" scenarios in the way that a cognitive architecture would.

Examination of leading cognitive architectures such as ACT-R and SOAR (Newell, 1990) reveals that their particular strengths are in the description of skill acquisition rather than the prediction of error. In the case of ACT-R, this largely attributable to its long history in the field of intelligent tutoring systems (Anderson, 1993). Because of this, some researchers have called for entirely new cognitive architectures to be developed, which would allow more complete modeling of errors within complex dynamic environments (Grant, 1996). While new architectures for error prediction may be useful in the future, existing architectures have a wealth of support and research that easily justifies using them for current error prediction modeling. Furthermore, the perceived limitation in error prediction with ACT-R, for example, perhaps lies more of how it has been used rather than a limitation of the architecture itself.

ACT-R was used for this study. It is freely available for government and academic research from Carnegie Mellon University. It is a symbolic, production system architecture, capable of low-level representations of memory structures. ACT-R is implemented in the common LISP programming language as a collection of LISP functions and subroutines, which can be accessed by the cognitive modeler. For this project, we used Macintosh Common LISP and ACT-R 4.0 running on a G3 Apple Macintosh computer running system 8.5.1.

The data used for development of the cognitive models reported here were collected by members of the Human Research and Engineering Laboratory's Soldier Performance Division as part of a study entitled "A Comparison of Soldier Performance Using Current Land Navigation Equipment With Information Integrated on a Helmet-Mounted Display" (Glumm et al., 1998). As part of the study, 12 male infantry soldiers performed a series of navigation tasks while wearing helmet-mounted displays (HMDs). During the navigation exercise, soldiers were asked a series of probe questions at pre-determined coordinates along the path. The soldiers were queried about their positions with respect to various objectives (e.g., targets, way points). The probe question technique was used as a measure of situation awareness, operationally defined as a correct response. Each question was phrased to elicit either a "yes" or "no" response. Mistakes made in response to the probe questions were the errors that were modeled for this effort.

## 2 METHODOLOGY

Each participant navigated a densely wooded path while wearing an HMD. The total length of the path was 3 kilometers and consisted of four segments (or legs) of different lengths that intersected five way points: The lengths of the path legs were 550, 700, 850, and 900 meters. The terrain was flat with elevation contours of 2 to 3 feet. The ground was covered with fallen trees and branches which, in some areas, were concealed by grass approximately 8 inches tall. There were some small streams along the path and marshy areas with standing water. Except for a few short, muddy sections of path that lacked ground cover, the hardy grasses and vegetation that grow in this area tend to recover quickly from footsteps, revealing little evidence of previous subjects.

A total of 20 probe questions was asked during the entire course. The first five probe questions are shown in the first column of Table 1. All the information necessary to answer the questions was available on the HMD. There were four different HMD screens (target, way point, enemy, and path), each of which could be accessed by the soldier via a small, belt-mounted keyboard. Participants were able to access any HMD screen at any time, except that immediately after a probe question, the HMD was blanked until the question was answered. For the overall study, subjects traversed the course in both HMD and non-HMD conditions; however, only the HMD data were used for the modeling effort. Data were collected by use of a computer linked to the HMD. The data collected were time stamped to indicate, among other things, the time when any of the four screens were selected for viewing and the time when each probe question was presented. In this way, the amount of time between the presentation of a given, relevant screen and the probe question could be calculated, as well as the number and timing of any other, irrelevant screens selected for viewing in the interim. (Test participants, of course, had no way of knowing what the next probe question would be, and therefore had no way of knowing which screens would be relevant and which irrelevant. They had been instructed to access the four screens at will in order to help them with their navigation-related tasks.) These were important variables that were used to build the ACT-R model.

Table 1.

First Five Probe Questions Given to Each Soldier During the Scenario and the Hypothesized Memory Retrievals Needed for Each Question as Represented in the Final Models

| **Probe Question** | **Memory Retrievals** |
|---|---|
| 1) Are you within 50 meters of your next target? | Target Memory Chunk<br>Pace Count Memory Chunk |
| 2) Are there friendly units only to the left of your path? | Friendly Memory Chunk<br>Direction Memory Chunk<br>Unit Memory Chunk |
| 3) Are you within 100 meters of the next way point? | Way point Memory Chunk<br>Pace Count Memory Chunk |
| 4) Is there an enemy unit within 200 meters of your left? | Enemy Memory Chunk<br>Direction Memory Chunk<br>Unit Memory Chunk |
| 5) Are you within 50 meters of your next target? | Target Memory Chunk<br>Pace Count Memory Chunk |

## 2.1 Cognitive Model Concept

The goal of the model was to predict when errors would be made by soldiers in response to probe questions. The model development was predicated on the development of an hypothesis of why errors were made. The hypothesis was relatively simple: memory decay for the memory chunks needed to answer the probe questions correctly would lead

to errors. The more time available for memory decay and the more competing or similar memory chunks, the more likely an error. The probe questions all pertained to information that could be displayed on the HMD. The HMD was blanked during the presentation of the probe questions; thus, information was not immediately available and had to be recalled from memory. Also, various other events, both external (e.g., objects along the path, an irrelevant HMD screen) and internal (e.g., rehearsing or checking information that was not related to the probe question) may have occurred between the viewing of the relevant HMD and when the probe question was asked. Finally, participants may or may not have actually viewed an HMD screen that was relevant to answering a given probe question, since the participant had to choose which HMD screen to view at any given time.

In summary, our hypothesis regarding error generation was based on the decay of a soldier's memory for the information that had been displayed on the HMD. More specifically, the likelihood of a soldier producing an error on a probe question would depend on the amount of decay for each separate memory chunk required to answer the probe question correctly. ACT-R assigns memory chunks an activation level, and this level can be lowered in the model through various means, one of them being time-based decay. In the end, this was a good hypothesis for ACT-R to evaluate, since one of the strengths of ACT-R is its computational simulation of memory decay.

For this study, seven subjects with similar experimental conditions were modeled, and five subjects with dissimilar experimental conditions were dropped. Because of the complexity and overall duration of the navigational task to be modeled, each subject and each question were modeled separately. This reduced the complexity of the modeling effort; however, it also reduced the amount of predictive power that the models would be able to achieve. For example, it eliminated the possibility of evaluating confusion across probe questions.

Each model (which consisted of one subject answering one probe question) was generated from a model template. The model template incorporated general aspects of the navigational task, but not the specific sequence of events that a given subject experienced during his scenario; this information was inserted later. The general model template included walking, checking the display, listening to messages, and responding to probe questions. One note here is that the model assumes nearly perfect perception of incoming information. In other words, if the model presents a screen, it assumes that all the displayed information was viewed and entered into memory perfectly, with no errors in the encoding process. The difference in each model was the precise sequence of events modeled prior to each probe question. In other words, even with the probe questions held constant, depending on their strategies or even small variations in the outside world, different subjects went through slightly different sequences of events prior to answering the question (e.g., precisely when each HMD screen was accessed).

A Macintosh HyperCard $^{TM}$ stack was created to read the raw data files and then generate the unique LISP code that represented each soldier's unique experience or sequence of events. The unique LISP code was then inserted with ACT-R productions.

The modeling effort here was very extensive, and in many ways, atypical from more traditional cognitive modeling. In all, 2 million lines of ACT-R code were written, compiled, run, and analyzed. Additionally, the modeling effort was challenging because prediction of behavior required a standard set of default parameters to be chosen that

would hopefully be representative of the end behavior. Specifically, ACT-R has several parameters that are typically set by their being mapped through iteration to real-world data. Because our models were predictive, we had to make estimates for the default values of many ACT-R parameters. In some cases, this was relatively easy, since a few ACT-R parameters have "unofficial" default values; however, this was difficult for other parameters where default parameters, even "unofficial" ones, do not exist.

## 2.2 Model Development

In order to begin building the predictive models, first the experimental protocol and the HMD displays were reviewed. The information available to help answer each one of the 20 probe questions was examined. The time when each HMD screen was accessed and each probe question was asked were entered into individual subject's models. The initial assessment, then, was to determine which was the single most important memory chunk for each question and to return or retrieve that single chunk from the model's declarative memory. The average activation levels of the single memory chunk associated with each question from this initial set of ACT-R models were compared to the experimental data; however, the results were not significant.

It was noted, even in the beginning, however, that answering probe questions could involve the retrieval of multiple memory chunks, each with its own decay, and therefore potentially multiple conflicts or partial matches with existing memory chunks (see the second column in Table 1). As a verification, a correlation between errors and these estimates of the raw number of memory chunk retrievals needed for each probe question was computed, and that was significant $r(19) = .43$, $p < .05$. In other words, a probe question that we had hypothesized would need four memory chunks was more likely to be answered incorrectly than a question needing only one memory chunk.

Following these initial analyses, the models were revised to accommodate multiple memory chunks being retrieved. Also, the initial analysis revealed that we had a restriction of range problem with the activation levels of the memory chunks. To counter this limitation, certain ACT-R parameters were changed in order to increase the variance among activation levels. Third, following a consultation with a military subject matter expert (SME), it became clear that an important variable had been omitted from the initial models. Infantry soldiers typically keep track of something called a "pace count." This is a count of the number of steps they have taken while walking which then enables a soldier to calculate the total distance traveled in meters. Infantry soldiers can then use this information to estimate how far certain distances are by determining how far they have walked. This memory chunk, which is constantly revised, was then added to the ACT-R models. The final models contained memory retrieval productions for multiple memory elements and the pace count.

In general, there were memory chunks for targets and way points. Targets could be either friendly or enemy and they were designed so that they could be confused with each other by via ACT-R's partial matching feature. The direction memory chunk was treated as a separate memory chunk and not as a slot belonging to another memory chunk. This allowed for retrieval of the direction chunk separately from targets or way points. The pace count memory chunk was constantly updated by the simulation; this was

designed to simulate the soldier keeping track of his approximate distance traveled as a function of his footsteps.

## 3 RESULTS

The ACT-R models of each question for each subject (a total of 140 models) were each run 40 times. For each subject, the activation values of each memory chunk across the 40 runs was averaged. Then the activation values of all the memory chunks retrieved for each question were averaged. In other words, the activations from the 40 runs were averaged across runs, across memory chunks, and across subjects to yield a single activation level for each question. In this way, the resulting single activation level per question could be correlated with the percent errors for each question. This analysis yielded a significant correlation. Our hypothesis was that lower activation levels would lead to a higher likelihood of making an error. This enabled use of a one-tailed Pearson's Product Moment correlation coefficient between the average activation value and the percentage of errors for each question $r(19) = -.43$, $p < .03$.

## 4 CONCLUSIONS

Error prediction using cognitive architectures is a new and exciting application of computational cognitive modeling, but it requires further application and refinement to show its viability to the system design process. In this study, we were able to make general predictions concerning the error rates of probe questions and the subsequent effects on situation awareness using cognitive modeling. The models had to be “tweaked” during the development process, albeit only slightly. It was an adjustment nonetheless. It was also unclear precisely how much benefit the cognitive modeling effort added over the initial (non-modeling) estimates of error generation. One could only speculate that real benefits of modeling would be incurred during the system modification phase, which would allow for errors to be tested on future designs by the use of previously developed models. The benefits of such an iterative error modeling process to system design could be significant; therefore, such efforts should continue to be pursued by the human factors community.

## REFERENCES

Anderson, J. (1993). Rules of the Mind. Hillsdale, NJ: Earlbaum.

Anderson, J. R., and Lebiere, C. (1998). Atomic components of Thought. Hillsdale, NJ: Earlbaum.

Grant, S. (1996). SimulACRUM: A cognitive architecture for modeling complex task performance and error. In: Proc. 8th European Conference on Cognitive Ergonomics. (ECCE 8) Granada, Spain, 97-102.

Glumm, M. M., Marshak, W. P., Branscome, T. A., McWesler, M., Patton, D.J., Mullins, L.L., (1998). A Comparison of Soldier Performance Using Current Land Navigation

Equipment with Information Integrated on a Helmet Mounted Display. (Army Research Laboratory publication No. ARL-TR-1604). Aberdeen Proving Ground, Aberdeen, MD.

Newell, A. (1990). Unified Theories of Cognition. Cambridge: University Press.

Norman, D.A. (1981). Categorization of Action Slips. Psychological Review, 88 (1), 1-15

Rasmussen, J. (1982). Human Errors – A Taxonomy for Describing Human Malfunction in Industrial
Installations. Journal of Occupational Accidents, 4 (2-4), 311-335.

Reason, J. (1990). Human Error. Cambridge: University Press.

**Author Note:**
The author would like to acknowledge the contribution of Nancy Grugle ( ngrugle@vt.edu ) to this effort. Correspondence concerning this article should be addressed to Troy Kelley, Army Research Laboratory, Human Research and Engineering Directorate, AMSRL-HR-SE, APG, MD, 21005-5425. tkelley@arl.army.mil