(NASA-CR-199536) RULE BASED DESIGN OF CONCEPTUAL MODELS FOR FORMATIVE EVALUATION (Auburn Univ.) 9 p

N96-13370

Unclas

G3/60 0073749

RULE BASED DESIGN OF CONCEPTUAL MODELS FOR FORMATIVE EVALUATION

Loretta A. Moore[+], Kai Chang[+], Joseph P. Hale[*], Terri Bester[+], Thomas Rix[+], and Yaowen Wang[+]

[+]Computer Science and Engineering
Auburn University
Auburn, AL 36849
(205) 844 - 6330
moore@eng.auburn.edu

[*]Mission Operations Laboratory
NASA Marshall Space Flight Center
MSFC, AL 35812
(205) 544-2193
joe.hale@msfc.nasa.gov

ABSTRACT

A Human-Computer Interface (HCI) Prototyping Environment with embedded evaluation capability has been investigated. This environment will be valuable in developing and refining HCI standards and evaluating program/project interface development, especially Space Station Freedom on-board displays for payload operations. This environment, which allows for rapid prototyping and evaluation of graphical interfaces, includes the following four components: (1) a HCI development tool, (2) a low fidelity simulator development tool, (3) a dynamic, interactive interface between the HCI and the simulator, and (4) an embedded evaluator that evaluates the adequacy of a HCI based on a user's performance. The embedded evaluation tool collects data while the user is interacting with the system and evaluates the adequacy of an interface based on a user's performance. This paper describes the design of conceptual models for the embedded evaluation system using a rule-based approach.

INTRODUCTION

Formative evaluation is conducted through usability studies. Given a functional prototype and tasks that can be accomplished on that prototype, the designer observes how users interact with the prototype to accomplish those tasks in order to identify improvements for the next design iteration. Evaluation of the interaction is measured in terms of specific parameters including: time to learn to use the system, speed of task performance, rates and types of errors made by users, retention over time, and subjective satisfaction [Shneiderman, 1992]. Analysis of this information will assist in redesign of the system.

The conceptual model of a designer is a description of the system and how the user should interact with it in terms of completing a set of tasks [Mayhew, 1992]. The user's mental model is a model formed by the user of how the system works, and it guides the user's actions [Eberts, 1994]. Most interaction problems occur when the user has an inaccurate model of the system or when the user's model of a system does not correspond with the designer's conceptual model of the system. The evaluation approach which will be discussed in this paper evaluates the user's mental model of the system against the developer's conceptual model.

A rule-based evaluation approach, implemented using CLIPS, is used to develop the conceptual model. The model outlines the specific actions that the user must take in order to complete a task. Evaluation criteria which is embedded in the rules include the existence of certain actions, the sequencing of actions, and the time in which actions should be completed. Throughout the evaluation process, user actions are continuously associated with a set of possibly changing goals. Once a goal has been identified, the user's action in response to that goal are evaluated to determine if a user has performed a task correctly. Tasks may be performed at three levels: expert, intermediate, and novice.

This research is supported in part by the Mission Operations Laboratory, NASA, Marshall Space Flight Center, MSFC, AL 35812 under Contract NAS8-39131, Delivery Order No. 25. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressly or implied, of NASA.

The dynamic relationship between the evaluation tool and the user environment allows the simulation director to constantly introduce new goals that need to be responded to. This paper will discuss the approach of rule-based conceptual modeling and will provide an example of how this approach is used in the evaluation of a graphical interface of an automobile.

ARCHITECTURE OF THE HCI PROTOTYPING ENVIRONMENT

The Human-Computer Interface Prototyping Environment with Embedded Evaluation capability is designed to allow a developer to create a rapid prototype of a system and to specify correct procedures for operating the system [Moore, 1993]. The first component of the architecture is the Graphical User Interface (GUI) development tool. This tool allows the designer to graphically create the interface of the system and specify a data source for each object within the display. The simulator tool provides the capability to create a low-fidelity simulation of the system to drive the interface. The embedded evaluation tool allows the designer to specify which actions need to be taken to complete a task, what actions should be taken in response to certain events (e.g., malfunctions), the sequences the user actions should take, and the time frames in which these actions should be taken. Each of these components is a separate process which communicates with its peers through the network server. Figure 1 shows the architecture of the HCI Prototyping Environment

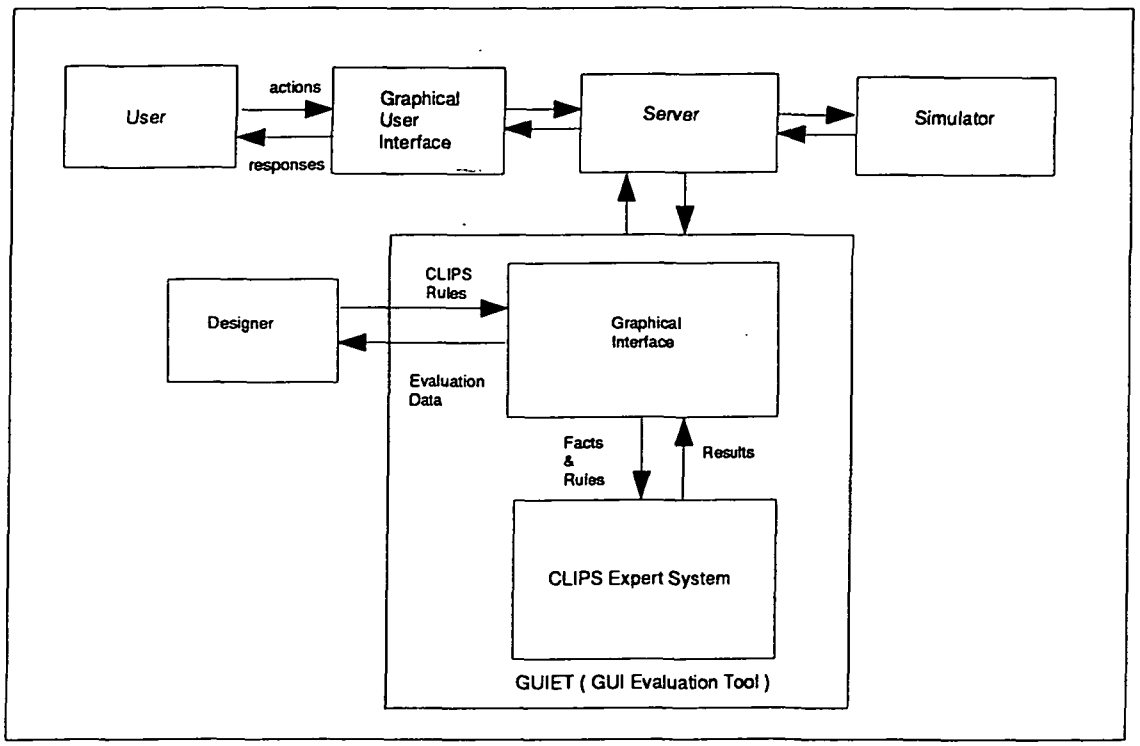

Figure 1 - Architecture of the HCI Prototyping Environment

During execution of the system, the interface objects send and receive data and commands to the simulator by way of the data server and the simulator provides realistic feedback to the interface based on user inputs. The server sends the embedded evaluation tool the actions which the user has taken, all events and activities which have occurred, and the times associated with these items. The embedded evaluation tool analyzes the actions which have been performed by the user, that is, the user's model of the system, against the predefined conceptual model of the designer. The system identifies which tasks were completed correctly, or not, and provides data to the designer as to the points in the interaction in which the user's model of the system did not correspond to the designer's conceptual model of the system.

In order to evaluate the architecture, an automobile system was prototyped in the environment. An automobile was chosen because it has sufficient complexity and subsystems' interdependencies to provide a moderate level of operational workload. Further, potential subjects in the empirical studies would have a working understanding of an automobile's functionality, thus minimizing pre-experiment training requirements. The conceptual model which will be described in this paper is that for the automobile prototype. Before describing the rule based conceptual model, we will first discuss changes made to integrate CLIPS into the HCI Prototyping Environment.

THE GRAPHICAL USER INTERFACE EVALUATION TOOL (GUIET)

The goal of GUIET is to provide for dynamic evaluation of user actions within the HCI Prototyping Environment. Using GUIET, the process of formative evaluation has more flexibility and takes less time for analysis. The main feature is that the evaluation of most of the participant's actions are automated. The evaluation is performed at runtime by an expert system. The knowledge base of the system contains the developer's conceptual model, of how he/she thinks the user should interact with the prototyped system. Because the knowledge base is not hard coded into the application, the knowledge base can be dynamically changed according to the needs of the evaluator. This provides the flexibility to evaluate different interfaces with the same evaluation criteria or one interface with different evaluation criteria. This design saves time because the data is automatically collected and analyzed based on the rule based conceptual model. If a new interface is prototyped, the only change that needs to be made with GUIET is changing the knowledge base.

In addition to the rule-based modeling design, GUIET provides a graphical interface and communication capabilities to CLIPS. In order to be integrated with the existing architecture, GUIET needed to receive information from both the interface and the simulator. The server sends the messages that are passed between the GUI tool and the simulator tool to GUIET. This is done using datagram sockets for the interprocess communication. The messages are in the form:

(newfact 193.0 I>S Message SetVariable gear 2)

where newfact is a string used for pattern matching for CLIPS rules, 193.0 is the time stamp, I>S states that communication is from the interface to the simulator, Message is the type of communication, SetVariable represents that the value of a variable is being set by the user, gear is the variable name, and 2 is the variable value.

Although this is the natural way to assert facts into CLIPS, the state for the car is stored not as a set of facts, but as one fact with many variable/value pairs. Before the new fact is evaluated it is translated into the form of a fact. This translation is done by a set of translation rules. An example of a translation rule is:

```
; GEARS
:
(defrule trans_gear
        ?newfact <- (newfact ?time ?direction ?type ?action gear ?value)
        ?state <- (car_state)
=>
        (modify ?state (gear ?value))
        (retract ?new_fact)
        (bind ?*current_tag* (+ ?*current_tag* 1))
        (assert (action (type gear) (value ?value) (time ?time)
                        (clock_tag ?*current_tag*) ))
```

Another set of rules perform the evaluation of the user's actions. The next section describes the the rule format for evaluation of the user's actions. The last section describes graphical interface which has been created for CLIPS.

CONCEPTUAL MODEL FOR THE AUTOMOBILE PROTOTYPE

The tasks which the user are asked to perform with the prototype can be divided into two categories: driving the car (i.e., using the controls) and responding to events (e.g., environmental and maintenance). The tasks measured include:

- Starting the car
- Driving forward (including changing gears)
- Driving backward
- Turning
- Stopping (at stop signs, lights, etc.)
- Parking the car
- Increasing and decreasing speed [Responding to speed limit changes]
- Driving uphill and downhill [Responding to hill events]
- Performing maintenance [Responding to maintenance events]
- Responding to environmental conditions

The events which can occur while the user is driving include environmental condition events (e.g., rain, snow, fog, and clear weather), time of day events (e.g., day and night), terrain changes (uphill and downhill), speed limit changes, and maintenance problems (e.g., gas, oil, battery, alternator, and engine). In addition to the events, the participant is given a set of instructions that must be followed. These are in the form of driving directions (e.g., drive 5 miles north and park the car).

Driving the car consists of manipulating graphical objects on the screen. For each of the tasks described above, the designer has determined a set of correct actions that must be made to complete the task. For example, the actions which must be taken for starting the car include:

1. Lock the seatbelt
2. Release emergency brake
3. Depress the brake
4. Depress the clutch
5. Put the gear in neutral
6. Turn the key on

Task correctness is evaluated based mainly on three evaluation criteria: the existence of certain actions, the sequencing of actions, and the time associated with the completions of the actions or task. An integer clock counter is used to indicate the action or event sequence. In the beginning of evaluation, the clock is reset to zero. Every subsequent action taken by the driver would increment the clock by one. Action sequence is important for many driving maneuvers. For example, clutch must be engaged before shifting gears. The evaluation process evaluates the correctness and effectiveness of a driver's interactions with the graphical user interface. User performance can be classified into three levels for most tasks - expert, normal, and novice. There may also be no response to a task. A counter is designated for each performance level. Every time a sequence of user actions is classified at a particular level, the associated counter will be incremented by one. The purpose of the evaluation is not to classify or evaluate users, but to evaluate the interface. The classification of users into categories is done to identify the level at which the users are interacting with the system at. The goal is to have most if not all interacting at what the designer would consider the expert level. If users are not interacting at this level, it is the interface which is enhanced to improve user performance.

```
        ; This part of the rule ensures that all of the actions have occurred.

        ; the first action putting the seatbelt on
        ?f1 <- (action (type seatbelt) (value 1) (clock_tag ?t))
        ; next action is about the emergency brake
        ?f2 <- (action (type emer_brake_on) (value 0) (clock_tag ?t1))
        ; next action is brake on
        ?f3 <- (action (type brake) (value 1) (clock_tag ?t2))
        ; next action is clutch depressed
        ?f4 <- (action (type clutch) (value 1) (clock_tag ?t3))
        ; next action is gear neutral
        ?f5 <- (action (type gear) (value 0) (clock_tag ?t4))
        ; next action is key on
        ?f6 <- (action (type key) (value 1) (clock_tag ?t5))

        ; The following test is to see what sequence the events occurred in.
        ; The proper sequence is: key turned after the clutch is off, clutch off
        ; after the brake is on, and seatbelt is on before the key is turned.
        ; Additional actions may be done in between the needed actions.
        ; For example, turning on the fog lights.

        (test (> ?t1 ?t))
        (test (> ?t2 ?t1))
        (test (> ?t3 ?t2))
        (test (> ?t4 ?t3))
        (test (> ?t5 ?t4))
=>
        ; The following assertion aids in keeping track of whether a response
        ; was made or not
        (assert (response engine))

        ; effects of actions considered, therefore removed from the fact base
        (retract ?f1 ?f2 ?f3 ?f4 ?f5 ?f6)

        ; increment intermediate level count
        (bind ?*intermediate_count* (+ ?*intermediate_count* 1))

        (printout evaluation "TASK: starting the engine" crlf)
        (printout evaluation "TIME:  " (- (integer (time)) ?*start_time*) " seconds" crlf)
        (printout evaluation "# ERRORS: 1 or more" crlf)
        (printout evaluation "EXPLANATION: Extra events occurred in the sequence." crlf )

) ;normal_start_engine

(defrule novice_start_engine "Novice level starting engine"

        ; salience value smaller than intermediate level, but higher than no response level
        (declare (salience 1))
```

In the CLIPS implementation a fact template is used to represent the car states and driving scenarios. The following shows the template for car-states:

```
(deftemplate car_states "Variables used in the car interface"*new*
        (slot turnsignal)
        (slot brake)
        (slot emer_brake_on)
        (slot clutch)
        (slot key)
        (slot gear)
        (slot throttle)
        (slot speed)
        (slot seatbelt)
        (slot wipers)
        (slot lights)
        (slot fog_lights)
        (slot oil)
        (slot gasoline)
        (slot engine_temp)
        (slot battery)
        (slot alternator_ok)
        (slot rpm)
        (slot terrain)
        (slot day)
        (slot weather)
        (slot speed_limit)
) ;car template
```

Associated with each action taken by the user, a template is defined as:

```
(deftemplate action "Action taken by the user on the interface"
        (slot type)                     ;action type
        (slot value)                    ;action value
        (slot time)                     ;action time
        (slot clock_tag)                ;action sequence
) ;action template
```

An evaluation rule is designed for each performance level. After a sequence of actions is completed, it will be evaluated based on the rules for the three performance levels. However, only one of the rules would succeed. The rules are organized in a way that the expert level would be tried first, then the intermediate level, and then the novice level. Once a rule has been successfully fired, this sequence of actions will be discarded. The prioritization of these rule is achieved through the *salience* values of Clips.

The following examples describe a portion of the evaluation process for *starting the engine.*

```
; Rules for Starting the Engine

(defrule expert_start_engine "Expert level starting engine"
```

```
        ; salience value highest among the four performance evaluation rules
        (declare (salience 3))

        ; This part of the rule ensures that all of the actions have occurred.
        ; the first action putting the seatbelt on
        ?f1 <- (action (type seatbelt) (value 1) (clock_tag ?t))
        ; next action is about the emergency brake
        ?f2 <- (action (type emer_brake_on) (value 0) (clock_tag ?t1))
        ; next action is brake on
        ?f3 <- (action (type brake) (value 1) (clock_tag ?t2))
        ; next action is clutch depressed
        ?f4 <- (action (type clutch) (value 1) (clock_tag ?t3))
        ; next action is gear neutral
        ?f5 <- (action (type gear) (value 0) (clock_tag ?t4))
        ; next action is key on
        ?f6 <- (action (type key) (value 1) (clock_tag ?t5))

        ; To ensure that the actions occurred in the proper order the following
        ; test is performed. The proper sequence is lock seatbelt, release
        ; emergency brake, depress brake, depress clutch, select neutral gear,
        ; turn key, release brakes and release clutch.

        (test (= (+ ?t 1) ?t1))
        (test (= (+ ?t 2) ?t2))
        (test (= (+ ?t 3) ?t3))
        (test (= (+ ?t 4) ?t4))
        (test (= (+ ?t 5) ?t5))
=>
        ; The following assertion aids in keeping track of whether a response
        ; was made or not.
        (assert (response engine))

        ; effects of actions considered, therefore removed from the fact base
        (retract ?f1 ?f2 ?f3 ?f4 ?f5 ?f6)

        ; increment expert level count
        (bind ?*expert_count* (+ ?*expert_count* 1))

        (printout evaluation "TASK: starting the engine" crlf)
        (printout evaluation "TIME:  " (- (integer (time)) ?*start_time*) " seconds" crlf)
        (printout evaluation "# ERRORS: 0" crlf crlf)

) ; expert_start_engine

(defrule normal_start_engine "Normal level starting engine"

        ; salience value smaller than expert level, but higher than novice level
        (declare (salience 2))
```

```
        ; This part of the rule ensures that all of the actions have occurred.

        ; the first action putting the seatbelt on
        ?f1 <- (action (type seatbelt) (value 1) (clock_tag ?t))
        ; next action is about the emergency brake
        ?f2 <- (action (type emer_brake_on) (value 0) (clock_tag ?t1))
        ; next action is brake on
        ?f3 <- (action (type brake) (value 1) (clock_tag ?t2))
        ; next action is clutch depressed
        ?f4 <- (action (type clutch) (value 1) (clock_tag ?t3))
        ; next action is gear neutral
        ?f5 <- (action (type gear) (value 0) (clock_tag ?t4))
        ; next action is key on
        ?f6 <- (action (type key) (value 1) (clock_tag ?t5))

        ; The events do not have to occur in any particular order. They just
        ; all have to occur.
=>
        ; The following assertion aids in keeping track of whether a response
        ; was made or not
        (assert (response engine))

        ; effects of actions considered, therefore removed from the fact base
        (retract ?f1 ?f2 ?f3 ?f4 ?f5 ?f6)

        ; increment novice level count
        (bind ?*novice_count* (+ ?*novice_count* 1))

        (printout evaluation "TASK: starting the engine" crlf)
        (printout evaluation "TIME:  " (- (integer (time)) ?*start_time*) " seconds" crlf)
        (printout evaluation "# ERRORS: 1 or more" crlf)
        (printout evaluation "EXPLANATION: Events occurred out of sequence." crlf)

) ;novice_start_engine

(defrule no_response_start_engine "No response to starting engine"
        ; No salience value is assigned therefore it has the default value of 0

        ; Check to see if there was an attempt to start the engine
        :
        :

        ; Check to see if the time limit has exceeded for starting the engine
        (test (>  (integer (time)) (+ ?*timeout* ?*start_time*)))
=>
        ; increment no response count
        (bind ?*no_response_count* (+ ?*no_response_count* 1))

        (printout evaluation "No response to starting the engine" crlf)
```

```
(printout evaluation "TIME:  " (- (integer (time)) ?*start_time*) "seconds" crlf crlf)

) ;no_response_start_engine
```

Rules for different tasks may contain different evaluation criteria. It depends on the designer's conceptual model of how he/she feels the task needs to be completed.

GRAPHICAL INTERFACE FOR CLIPS

A graphical interface was created for CLIPS using the Motif toolkit. The purpose of this interface is to provide the human evaluator a graphical means by which to use CLIPS. The main window of GUIET is composed of two areas. The top area is used for CLIPS standard input and output, and the bottom area displays any error or warning messages from CLIPS. These areas receive their their input from a series of CLIPS I/O router functions. The menu bar for this window provides the following options: System, Network, CLIPS, Rules, and Facts. The system pulldown provides functions for quitting the application and for bringing up a window for entering participant information. The network pulldown allows the evaluator to send a selection of messages to the server (e.g., connect and disconnect). The CLIPS pulldown supports functions that affect the entire expert system, such as loading the evaluation rule base, resetting the expert system to its initial settings, running, clearing, or accepting input. Functions that affect the rules, such as watching/unwatching the rules fire or viewing/editing existing rules, can be accessed through the rules pulldown. Functions that affect the facts in the expert system are provided under the facts pulldown.

The user information window accessed under the system pulldown allows the evaluator to enter data relevant to the current participant and system being evaluated. The top section of this window displays the expert system's evaluations. There is an I/O router which handles the display for the evaluation window and is accessed through the printout statement within the rules. The bottom area is a text area in which the evaluator can enter comments or observations about the participant's session. The window provides options for saving, printing, and cancelling information.

CONCLUSION

The rule-based design of conceptual models enables the iterative process of design and evaluation to proceed more efficiently. A designer can specify user performance criteria prior to evaluation, and the system can automatically evaluate the human computer interaction based on the criteria previously specified. In order to evaluate the system which has been designed, a study is being planned which will evaluate user performance (using the rule based system developed) using a good interface and a bad interface. The hypothesis is that the good interface will produce more user responses at the expert level, and the bad interface will produce less acceptable responses.

REFERENCES

Eberts, Ray, User Interface Design, Prentice Hall, Englewood Cliffs, New Jersey, 1994.

Mayhew, Deborah, Principles and Guidelines in Software User Interface Design, Prentice Hall, Englewood Cliffs, New Jersey, 1992.

Moore, Loretta, Assessment of a Human Computer Interface Prototyping Environment, Final Report, Delivery Order No. 16, Basic NASA Contract No. NAS8-39131, NASA Marshall Space Fight Center, Huntsville, AL, 1993.

Shneiderman, Ben, Designing the User Interface: Strategies for Effective Human-Computer Interaction, Second Edition, Addison-Wesley, Reading, Massachusetts, 1992.