1993-07

Aiding teachers in constructing virtual-reality tutors

Rowe, Neil C.; Suwono, Francius

Monterey, California. Naval Postgraduate School

NPSCS-93-008

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Aiding Teachers in Constructing Virtual-Reality Tutors

Neil C. Rowe & Francius Suwono

July 1993

TECHNICAL REPORT

October 1, 1992 to July 1993

Approved for public release; distribution is unlimited.

Prepared for:

Naval Postgraduate School
Monterey, California 93943

NAVAL POSTGRADUATE SCHOOL

Monterey, California

REAR ADMIRAL T. A. MERCER
Superintendent

HARRISON SHULL
Provost

This report was prepared with research funded by the Naval Research Funds provided by the Naval Postgraduate School.

This report was prepared by:

YUTAKA KANAYAMA
Associate Chairman for
Research

PAUL MARTO
Dean of Research

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY 2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
4. PERFORMING ORGANIZATION REPORT NUMBER(S) NPSCS-93-008	5. MONITORING ORGANIZATION REPORT NUMBER(S) Naval Postgraduate School
6a. NAME OF PERFORMING ORGANIZATION Computer Science Dept. Naval Postgraduate School 6b. OFFICE SYMBOL *(if applicable)* CS	7a. NAME OF MONITORING ORGANIZATION ONR
6c. ADDRESS *(City, State, and ZIP Code)* Monterey, CA 93943	7b. ADDRESS *(City, State, and ZIP Code)* San Diego, CA
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Naval Postgraduate School 8b. OFFICE SYMBOL *(if applicable)* NPS	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER DARPA 13 Project under AO 8939
8c. ADDRESS *(City, State, and ZIP Code)* Monterey, CA 93943	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE *(Include Security Classification)*
Aiding Teachers in Constructing Virtual-Reality Tutors

12. PERSONAL AUTHOR(S)
Neil C. Rowe & Francius Suwono

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT *(Year, Month, Day)*	15. PAGE COUNT
Interim	FROM 9210 TO 9307	930719	11

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
FIELD	GROUP	SUB-GROUP	tutoring, computer-aided instruction, means-ends analysis, heuristic search, virtual reality, Prolog, reactive environments, declarative specification

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

Teachers need different tools for constructing virtual realities than do professional programmers. Teachers building tutoring environments need only and should only provide declarative and nonquantitative specification of the application, as such information is sufficient to build powerful prototypes or even products when exploited properly. We describe our METUTOR tutor-generation system for sequential-action skills, which uses means-ends analysis on a teacher's declarative specification of a set of actions. METUTOR asks the teacher to specify conditions for recommending actions, preconditions of actions, and expected and random consequences of actions. METUTOR also asks the teacher to associate pictorial and/or aural representations with facts, and to specify how and when to use them. METUTOR provides facilities for automatic resolution of interactions and conflicts between media objects. We show examples from a firefighting tutor and a pilot's emergency tutor.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION	
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL Neil C. Rowe	22b. TELEPHONE *(Include Area Code)* (408) 656-2462	22c. OFFICE SYMBOL CSRp

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

Aiding Teachers in Constructing Virtual-Reality Tutors

Neil C. Rowe and Francius Suwono
Department of Computer Science
Code CS/Rp, U. S. Naval Postgraduate School
Monterey, CA USA 93943
rowe@cs.nps.navy.mil

Abstract

Teachers need different tools for constructing virtual realities than do professional programmers. Teachers building tutoring environments need only and should only provide declarative and nonquantitative specification of the application, as such information is sufficient to build powerful prototypes or even products when exploited properly. We describe our METUTOR tutor-generation system for sequential-action skills, which uses means-ends analysis on a teachers's declarative specification of a set of actions. METUTOR asks the teacher to specify conditions for recommending actions, preconditions of actions, and expected and random consequences of actions. METUTOR also asks the teacher to associate pictorial and/or aural representations with facts, and to specify how and when to use them. METUTOR provides facilities for automatic resolution of interactions and conflicts between media objects. We show examples from a firefighting tutor and a pilot's emergency tutor.

1. Introduction

Virtual reality can enhance a wide range of educational simulations. For instance, military training includes many specialized procedural skills for emergencies, e.g. firefighting. Broad educational application of virtual-reality ideas has been limited by the necessary complexity of their implementation. Our METUTOR project tries to address this problem by providing special tools for computer-inexperienced teachers and instructors to develop their own virtual-reality tutors for procedural skills. An example of our target audience is an instructor who trains Navy firefighting-team leaders.

Computer-inexperienced teachers often have trouble giving algorithms for procedural skills, although they can specify correct sequences of actions in particular circumstances. Furthermore, they usually cannot describe numerically or mathematically the components of the virtual reality, as this often requires sophisticated mathematics. This rules out all but minimal three-dimensional modeling, and usually requires a discrete representation of time. But if teachers can indeed perform the skill they wish to teach, they must know the best thing to do next in any situation. This "local" knowledge is close to the concept of declarative specification in languages like Prolog, in which the programmer specifies what things must be accomplished but not entirely their order [4].

So our METUTOR project has devised an approach that combines a simple language for specifying state descriptions and discrete state changes, a simple media-object (bitmap and audio) construction facility, and a way to relate state descriptions to media objects. Our specification syntax is that of Prolog, and our implementation is in Quintus Prolog. Our state-change semantics [4] is based on means-ends analysis with several of our own additions, and our handling of media objects is suggested by techniques of cartoon animation with several additional features.

Means-ends analysis is a problem-solving technique useful for a broad range of human problems, and people often seem to use something like it in everyday activity. To automate it, we must identify a set of possible discrete actions, and preconditions, postconditions, and recommendation circumstances for each action. Mean-ends specification of actions

permits a tutor to comment on the appropriateness of student actions in solving a problem, since it knows the possible reasons for them. We extend basic means-ends analysis with context dependency (so the same action has different preconditions or results depending on other facts that are true when it is applied), and most importantly, constrained random changes to the states to provide an element of unanticipated challenge to the student. All this teacher-supplied action specification is local information, and for most applications will be intuitive; see section 3 for examples. What teacher errors in specification do occur, as we have observed in forty tutors written using the METUTOR system, are usually errors of omission not commission, for which we have special debugging tools.

Means-ends analysis works by repeatedly comparing the current state and the goal objectives, and then selecting the highest-priority action recommended for at least some of the differences. If the highest-priority action cannot be done immediately, a recursive subproblem ("precondition recursion") is created with goal the preconditions of the desired action. If further differences between current state and goal objectives remain after the desired action has been performed, a new recursive subproblem ("postcondition recursion") is created for the current state. Although more sophisticated methods of planning have been developed in artificial intelligence, means-ends is surprisingly broad in applicability and robust. For instance, random occurrences can be taken in stride because means-ends constantly recomputes what to do. Means-ends can exploit well the automatic backtracking feature of software like Prolog, so even if the teacher's priorities for actions are poor, analysis will eventually back up and try something else. So means-ends is a good choice for tutoring software that computer-inexperienced teachers must write.

Using means-ends analysis means that virtual realities constructed with METUTOR are not just passive, but can advise the student. While the student thinks, the tutor reasons hypothetically to find the best sequence of actions to solve the problem from the current state. Then if the student picks a suboptimal action, the tutor can use 22 domain-independent mal-rules (of which more than one could apply to the same situation), exploiting this hypothetical reasoning plus automatic backtracking, to understand what the student is doing. Appropriate tutoring is tied to each mal-rule. For instance:

--If the student's action does not change anything, say so.
--If the original goal is unsolvable, say so and stop.
--If the student has five times avoided a certain best action, give a hint.
--If the student's action's name is similar to the best action, give a hint.
--If the student's action is the exact opposite of the best action, give a hint.
--If the student's action only makes sense by misreading the current state slightly, point this out.
--If the student's action is unnecessary to solve the problem, point this out.
--If the student has returned from a digression, point out where and how they digressed. (A digression is signalled by an action that, while perhaps eventually needed, does not make the problem easier to solve.)
--If the student picks a recommended action, but that action is not of the highest priority, analyze the justifications for both actions in terms of differences in precondition chains in the simplest terms.

Hypothetical reasoning and mal-rules have been used in other procedure tutors, e.g. [2], but means-ends permits especially general mal-rules. (Not to be confused with the student mal-rules, 30 additional rules check for teacher errors like misspelled words and impossible preconditions.)

2. Specification of graphics

A central objective of METUTOR is to help the teacher set up a many-to-many mapping between facts describing the virtual reality and media objects depicting it. That is, a fact or facts will correspond to media objects or objects, as in the more general Prolog-based approach of [3]. Teachers must obtain or construct images and sounds, and we provide tools to facilitate this. For audio, the teacher will specify a set of frequencies and amplitudes of those frequencies as a function of time within a time period, which permits a variety of simple noises. For images we already provide a simple facility for freehand line-drawing using the mouse, plus the capability of reusing bitmap pictures from a library [5]. Regions can be drawn, moved, scaled, clipped, filled, among other standard graphics operations. When finished, the screen location of the bitmap and its associated fact or facts are associated. Realistic images are not needed in many tutors for adult students, since such students can understand metaphoric symbols; otherwise, a professional program-

mer could later improve an teacher's images and sounds. Figures 1 and 2 show example metaphorical displays.

The mapping between internal facts and media objects can be many-to-many when the semantics of the two is sufficiently different. For instance, a big fire in a compartment on a Navy ship can be shown by bitmaps of flames scattered across the picture. Conversely, multiple facts may correspond to a single image if some facts are necessary to explain others. For instance, the facts that the oxygen has been tested and the oxygen is safe can be displayed in a single image, a dial reading "OK".

An object-oriented system, with inheritance and defaults, is the obvious way to assign properties to media objects. For instance for a firefighting tutor, the instances of flames displayed on the screen can inherit shape and color from an object representing a single prototypical flame, but could override its default size and screen-position attributes when the fire is almost out. In general, a media object can have these inheritable properties (some of which are not yet implemented):

(1) a pointer to its display representation (image or audio);
(2) a set of immediate generalizations (superconcepts);
(3) size if an image or duration if audio;
(4) brightness if an image or loudness if audio (maximum brightness in our implementation);
(5) color if an image (each has a single color in our implementation);
(6) location on the screen if an image or starting time if audio;
(7) orientation on the screen if an image;
(8) periodicity information if an intermittent image or sound;
(9) contextual conditions under which it is to be displayed (see below);
(10) overlay plane if an image (see below);
(11) a set of pointers to its subparts.

And pointers to these objects are stored in a table whose retrieval keys are single facts or sets of facts.

METUTOR provides two ways to specify interactions between media objects. First, as a simple method for handling image overlap, the teacher can specify occlusion order for image pairs. For instance, the image of the medical corpman could always be displayed occluding ("in front of") the image of their patient, as in Figure 1. Typically, movable objects should occlude setting-depicting images, and people should occlude movable objects. On the other hand, flames and smoke are more transparent, and should not occlude one another even if their images overlap, so no occlusion-order specification is needed for them. Images can have holes in which to place other images, like an outline of a tank with room to draw a water level within it. (Simultaneous audio objects just have their signals added, since sounds do not occlude.)

The second kind of interaction between media objects that we provide for is contextual variation in object graphical properties. This is common. For instance, only when the firefighters are at the fire can they see flames; and then they are shown at the center of the screen, otherwise to the right side. As another example, if a fire team member gets injured, that should reduce by one the number of fire team members shown normally. This context dependency differs from the oxygen-display example: The context facts will be displayed normally, but their presence causes other facts to be displayed differently. In general, the presence or absence of any fact could affect the display of another.

Surprisingly, we discovered when we began implementing the preceding ideas that some natural-language output was essential. To some extent, we can avoid it with metaphorical images and sounds, like radiating lines over a person's head to indicate that they are angry, or sounds of people yelling. But this can hurt clarity: Lines over a person's head could mean saintliness or a bad haircut. So we always display a text box that completely describes the current state. This is essential for currently-undisplayed facts (like what is happening at the fire when you are not there), orders that are currently in effect, descriptions of mental states, history, and so on, as well as clarifying displayed information. To facilitate natural-language output, we provide a modifiable rule-based system for paraphrasing internal representations of state descriptors and actions. We also let text to be drawn directly on the picture, which allows labeling. A fact can be represented by multiple pictures and/or multiple texts, like a picture of a switch with labels on its positions.

Our current implementation is in Prolog with Prowindows and uses bitmaps to hold all stored images. The hardware is Sun Sparc workstations, and this does impose performance limitations because these machines are not designed for real-time graphics. So we use bitmaps for all images rather than drawing even simple shapes in real time.

Even bitmaps can take a second to load. So a technique for additional efficiency that we have explored is to recompute certain combinations of bitmaps that are commonly shown together, "compiling" them into one big bitmap which can be loaded more quickly than the individuals.

The METUTOR framework extends easily to multi-student virtual realities, as in replenishment at sea, a skill which requires the cooperation of two ship commanders [6]. In our implementation of this tutor, each student runs a separate program copy with its own goals, and each program writes into and reads from a shared state file. Each program repeatedly checks the state file before changing it; whenever the file is changed, the other program throws away its previous reasoning and begins reanalyzing the situation with the new state.

3. Examples

Figure 1 shows an example from the tutor previously mentioned for fire team leaders on Navy ships, based on the analysis of [7]. The text box is on top, the graphical representation of the current state is below, and the menu of actions to select is in the lower right. Figure 1 shows a tutoring session during which the student ordered watching of the fire area for reignition of the now-extinguished fire, but he ordered it before emptying the compartment of smoke, and has thereby caused the reflash watchman to collapse from smoke inhalation. So the student then directed the medical corpman to give first aid to the casualty. In the default color implementation, the lower-left shapes are light blue, the water is green, the smoke is yellow, the medic on the right has a red cross, the other people are dark blue, and the other shapes are black. Note the oxygen canister occludes the oxygen tester and the medical corpman occludes the patient. The very bottom of the text box (the description of the current state, almost identical to that of the previous state described) has been obscured to permit seeing some of the previous tutoring. Note that the tutor lets students do foolish things like this sending unequipped people into smoke-filled compartments because the visual consequences are clearer than words, although the tutor does hint. Note also that the tutor noticed that the student has returned from a digression on their previous action.

Figure 2 shows an example from the tutor for emergency fuel problems on F-4 aircraft [1]. It shows a session in which the student (a pilot) is doing something useful, but not the most important action at the moment, so the tutor confines itself to hints. Since the name of the best action is similar to that the student chose, a lexical confusion may have occurred. Just four bitmaps, besides the text, were used to create the entire picture; this was deliberate, because real cockpits have easily-confusable switches. Simple subroutines permitted separating the shape and position specifications.

To illustrate declarative teacher specifications, here are examples from the firefighting tutor as they actually appear.

```
start_state([location(repair,locker),
  raging(fire),smokey]).
goal([verified(out(fire)),safe(gases),
  safe(oxygen),not(equipped(team)),
  not(smokey),not(watery),
  not(watched(reflashing)),
  not(unreplaced(casualty)),
  not(treated(casualty)),
  not(dead(casualty)),
  debriefed(team),
  deenergized(fire,area)]).
```

The first two lines above specify that in the starting state the fire team is in their repair locker, the fire is raging, and the fire area is smokey. The remaining lines specify that the student must achieve the following: the fire is verified to be out, the gases and oxygen in the fire area are safe, the fire team in unequipped, the fire area is neither smokey nor full of water, no reflash watch is on, no unreplaced or untreated or dead casualties are present, and the fire area is deenergized.

```
recommended([out(fire)],extinguish).
recommended([not(present(casualty))],
  [present(medical,corpman)],
  direct_medical_corpman).
```

The first says that if you want the fire to be out, you should extinguish it. The second says that if you want a casualty to no longer be present, then if there is a medical corpman present, you should direct the corpman to handle the casualty.

```
precondition(extinguish,
  [location(fire),raging(fire),
  equipped(team),set(boundaries),
  confronted(fire)]).
```

This says that to extinguish a fire, you must be at the fire, the fire must be raging, the fire team must be equipped, the boundaries of the fire must be set, and you must be facing the fire.

```
deletepostcondition(go(fire),
  [location(repair,locker)]).
addpostcondition(extinguish,
  [out(fire),watery,smokey]).
addpostcondition(extinguish,
  [not(deenergized(fire,area))],
  [present(casualty),dead(casualty),
   present(crater),raging(fire)],
  'There is a big explosion!').
```

The first definition says that if you go to the fire, you are no longer at the repair locker. The second says that when a fire is extinguished, the fire is out, the area is watery, and the area is smokey. The last definition says that in the special case where the student tries to extinguish when power to the fire area is on, a casualty is present and dead, a crater in the floor is present, the fire is still raging, and the message "There is a big explosion!" is printed. The last definition overrides the second when both apply.

```
randchange(extinguish,[],
  out(fire),raging(fire),0.3,
  'Fire is still raging.').
```

This says that 30% of the time when a student tries to extinguish a fire, they fail.

```
bmap(raging(fire),[location(fire)],
  fire2,308,135,red).
```

This says that if the fire is raging and the fire team is at the fire, the screen should show red flames (whose bitmap is in file "fire2") with upper left corner of the bitmap at (308,135). (The position was automatically computed by the shape-construction program when the flames were created and situated in a dummy box.)

```
opclick(380,6,320,175,
  [location(fire),smokey],desmoke).
```

This says that anytime the student clicks the left mouse button in the region 320 pixels wide and 175 pixel high whose upper left corner is at (380,6), while at the same time the simulation has the fire team at the fire and the fire area is smokey, that click means the student wants to desmoke.

```
draw_order(present_casualty,
   present_medical_corpman).
draw_order(treated_casualty,
   present_medical_corpman).
```

These say that the image of the medical corpman should occlude images of the casualty and the stretcher (the latter the "treatment").

4. References

[1] M. Kang, "Pilot emergency tutoring system for F-4 aircraft fuel system malfunction using means-ends analysis", M.S. thesis, Department of Computer Science, U.S. Naval Postgraduate School, Monterey CA, June 1990.

[2] R. Loftin, L. Wang, P. Baffes, and G. Hua, "An Intelligent Training System for Space Shuttle Flight Controllers", Conference on Innovative Applications of Artificial Intelligence, Stanford CA, March 1989, 105-110.

[3] S. Matsuoka, S. Takahashi, T. Kamada, and A. Yonezawa, "A general framework for bidirectional translation between abstract and pictorial data", *ACM TIS, 10*, 4 (October 1992), 408-437.

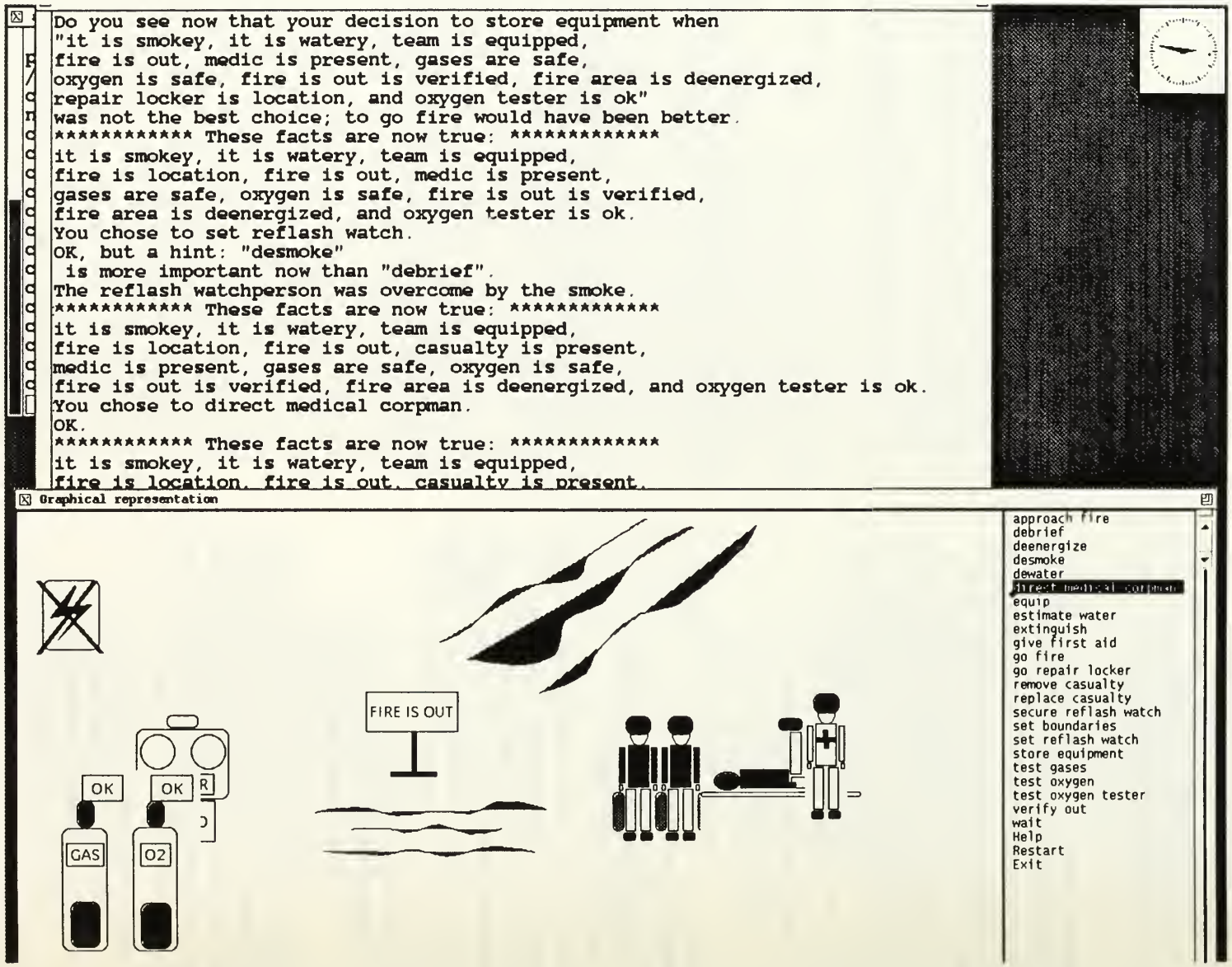
[4] N. Rowe, *Artificial Intelligence though Prolog*, Prentice-Hall, Englewood,Cliffs, New Jersey, 1988.

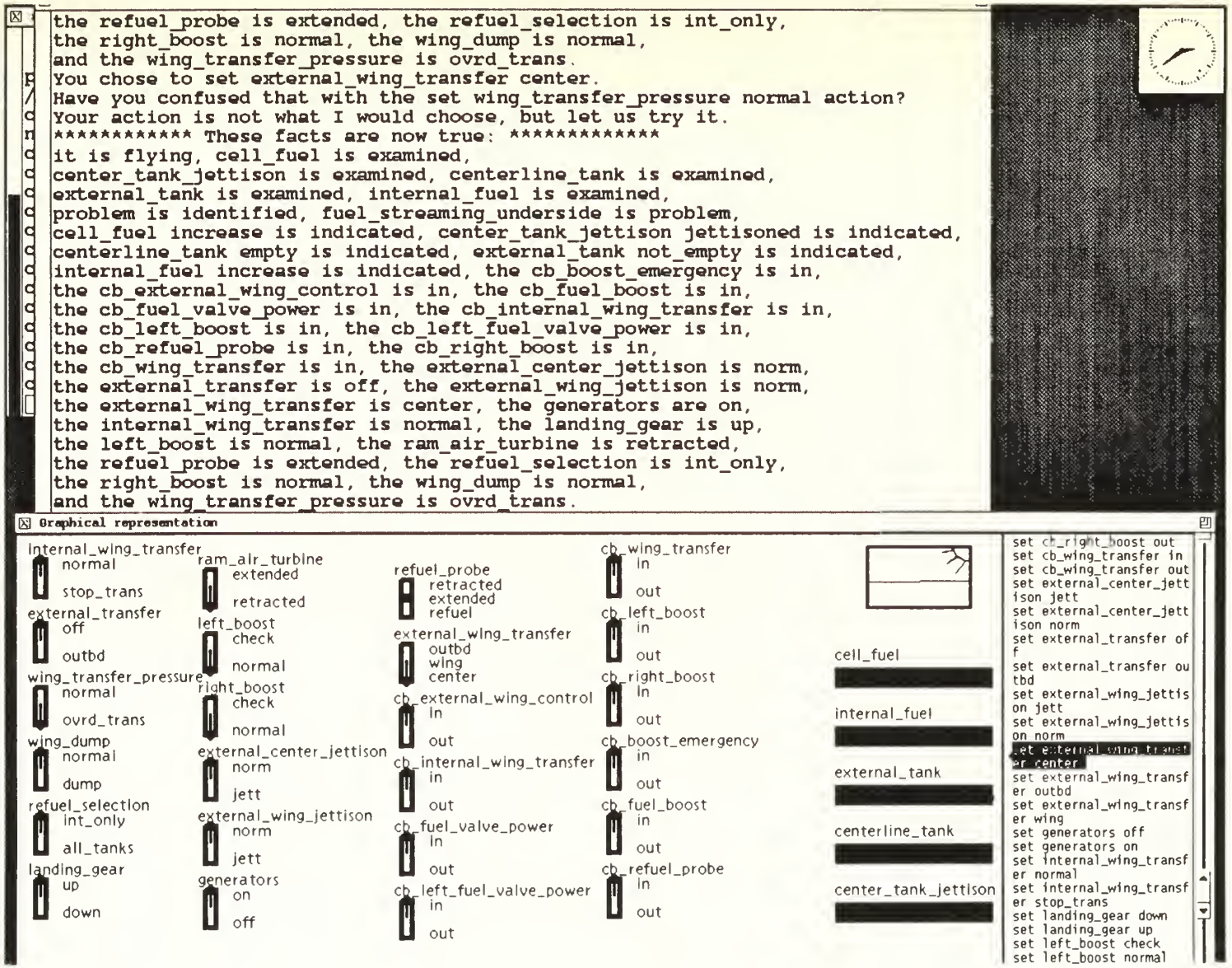
[5] F. Suwono, M.S. thesis, Dept. of Computer Science, Naval Postgraduate School, June 1993.

[6] P. Salgado, "An intelligent computer-assisted instruction system for underway replenishment", M.S. thesis, Department of Computer Science, U.S. Naval Postgraduate School, June 1989.

[7] S. G. Weingart, "Development of a shipboard damage control fire team leader intelligent computer aided instructional tutoring system," M.S. Thesis, Department of Computer Science, U. S. Naval Postgraduate School, June 1986.

This work was sponsored by the the U. S. Naval Postgraduate School under funds provided by the Chief for Naval Operations, and by the Defense Advanced Research Projects Agency as part of the I3 Project under AO 8939.


Do you see now that your decision to store equipment when
"it is smokey, it is watery, team is equipped,
fire is out, medic is present, gases are safe,
oxygen is safe, fire is out is verified, fire area is deenergized,
repair locker is location, and oxygen tester is ok"
was not the best choice; to go fire would have been better.
********** These facts are now true: **********
it is smokey, it is watery, team is equipped,
fire is location, fire is out, medic is present,
gases are safe, oxygen is safe, fire is out is verified,
fire area is deenergized, and oxygen tester is ok.
You chose to set reflash watch.
OK, but a hint: "desmoke"
is more important now than "debrief".
The reflash watchperson was overcome by the smoke.
********** These facts are now true: **********
it is smokey, it is watery, team is equipped,
fire is location, fire is out, casualty is present,
medic is present, gases are safe, oxygen is safe,
fire is out is verified, fire area is deenergized, and oxygen tester is ok.
You chose to direct medical corpman.
OK.
********** These facts are now true: **********
it is smokey, it is watery, team is equipped,
fire is location, fire is out, casualty is present,
Graphical representation
approach fire
debrief
deenergize
desmoke
dewater
direct medical corpman
equip
estimate water
extinguish
give first aid
go fire
go repair locker
remove casualty
replace casualty
secure reflash watch
set boundaries
set reflash watch
store equipment
test gases
test oxygen
test oxygen tester
verify out
wait
Help
Restart
Exit
FIRE IS OUT
OK
OK
GAS
O2


the refuel_probe is extended, the refuel_selection is int_only,
the right_boost is normal, the wing_dump is normal,
and the wing_transfer_pressure is ovrd_trans.
You chose to set external_wing_transfer center.
Have you confused that with the set wing_transfer_pressure normal action?
Your action is not what I would choose, but let us try it.
********** These facts are now true: **********
it is flying, cell_fuel is examined,
center_tank_jettison is examined, centerline_tank is examined,
external_tank is examined, internal_fuel is examined,
problem is identified, fuel_streaming_underside is problem,
cell_fuel increase is indicated, center_tank_jettison jettisoned is indicated,
centerline_tank empty is indicated, external_tank not_empty is indicated,
internal_fuel increase is indicated, the cb_boost_emergency is in,
the cb_external_wing_control is in, the cb_fuel_boost is in,
the cb_fuel_valve_power is in, the cb_internal_wing_transfer is in,
the cb_left_boost is in, the cb_left_fuel_valve_power is in,
the cb_refuel_probe is in, the cb_right_boost is in,
the cb_wing_transfer is in, the external_center_jettison is norm,
the external_transfer is off, the external_wing_jettison is norm,
the external_wing_transfer is center, the generators are on,
the internal_wing_transfer is normal, the landing_gear is up,
the left_boost is normal, the ram_air_turbine is retracted,
the refuel_probe is extended, the refuel_selection is int_only,
the right_boost is normal, the wing_dump is normal,
and the wing_transfer_pressure is ovrd_trans.
Graphical representation
internal_wing_transfer
normal
stop_trans
external_transfer
off
outbd
wing_transfer_pressure
normal
ovrd_trans
wing_dump
normal
dump
refuel_selection
int_only
all_tanks
landing_gear
up
down
ram_air_turbine
extended
retracted
left_boost
check
normal
right_boost
check
normal
external_center_jettison
norm
jett
external_wing_jettison
norm
jett
generators
on
off
refuel_probe
retracted
extended
refuel
external_wing_transfer
outbd
wing
center
cb_external_wing_control
in
out
cb_internal_wing_transfer
in
out
cb_fuel_valve_power
in
out
cb_left_fuel_valve_power
in
out
cb_wing_transfer
in
out
cb_left_boost
in
out
cb_right_boost
in
out
cb_boost_emergency
in
out
cb_fuel_boost
in
out
cb_refuel_probe
in
out
cell_fuel
internal_fuel
external_tank
centerline_tank
center_tank_jettison
set cb_right_boost out
set cb_wing_transfer in
set cb_wing_transfer out
set external_center_jett
ison jett
set external_center_jett
ison norm
set external_transfer of
f
set external_transfer ou
tbd
set external_wing_jettis
on jett
set external_wing_jettis
on norm
set external_wing_transf
er center
set external_wing_transf
er outbd
set external_wing_transf
er wing
set generators off
set generators on
set internal_wing_transf
er normal
set internal_wing_transf
er stop_trans
set landing_gear down
set landing_gear up
set left_boost check
set left_boost normal

Distribution List

Defense Technical Information Center Cameron Station Alexandria, VA 22314	2
Library, Code 52 Naval Postgraduate School Monterey, CA 93943	2
Center for Naval Analyses 2000 N. Beauregard Street Alexandria, VA 22311	1
Director of Research Administration Code 08 Naval Postgraduate School Monterey, CA 93943	1
Mr. Russell Davis HQ, USACDEC Attention: ATEC-1M Fort Ord, CA 93941	2
Dr. Neil C. Rowe, Code CSRp Naval Postgraduate School Computer Science Department Monterey, CA 93943	50
Prof. Ted Lewis, CS/Lt Naval Postgraduate School Computer Science Department Monterey, CA 93943	2
LTCOL Francius Suwono JL. Sengkuni No. 2, Dirgantara III Halim PK AFB, Jakarta 13510 Indonesia	2