Automated Instructor Assistant for Ship Damage Control

Vadim V. Bulitko & David C. Wilkins

Beckman Institute
University of Illinois at Urbana-Champaign
405 North Mathews Avenue
Urbana, IL 61801
{bulitko,dcw}@uiuc.edu

Abstract

The decision making task of ship damage control includes addressing problems such as fire spread, flooding, smoke, equipment failures, and personnel casualties. It is a challenging and highly stressful domain with a limited provision for real-life training. In response to this need, a multimedia interactive damage control simulator system, called DC-Train 2.0 was recently deployed at a Navy officer training school; it provides officers with an immersive environment for damage control training. This paper describes a component of the DC-Train 2.0 system that provides feedback to the user, called the automated instructor assistant. This assistant is based on a blackboard-based expert system called Minerva-DCA, which is capable of solving damage control scenarios at the “expert” level. Its innovative blackboard architecture facilitates various forms of user assistance, including interactive explanation, advising, and critiquing. In a large exercise involving approximately 500 ship crises scenarios, Minerva-DCA showed a 76% improvement over Navy officers by saving 89 more ships.

The Domain of Ship Damage Control

The tasks of ship damage control are vital to ship survivability, human life, and operational readiness. Most crises on military and civilian ships could be successfully addressed if handled promptly and properly. Typically the crisis management efforts on a ship are coordinated by a single person called the Damage Control Assistant (DCA). This person is in charge of maintaining situational awareness, directing crisis management crews, and managing other resources. Naturally, crisis management tasks are challenging even for seasoned Navy officers due to the inherent complexity of physical damage, limited resources, information overload, uncertainty, infrequent opportunities for realistic practice, and tremendous psychological stress. Studies have shown that the performance could be significantly improved by providing more opportunities for realistic practice (Ericsson 1993, Baumann et al. 1996). As in many other military domains, real-life training in often infeasible or inadequate due to the high cost and a limited number of possible scenarios. The Navy has been involved in supporting the creation of various damage control simulators to compliment textbook training (Jones et al. 1998, Bulitko 1998a, Fuller 1993, Johnson 1994). One of these projects resulted in creation of DC-Train 2.0, an immersive multimedia simulator (Bulitko 1998a). The system is capable of involved simulation including physical phenomena (fire, flooding, smoke spread, equipment failures) and personnel modeling (crisis management activities, casualties, standard procedures, and communications). The physical aspects of the scenarios are simulated from first principles starting with a sophisticated scenario specification tool mapping training objectives to primary damage specifications (Grois et al. 1998). A wide range of realistic scenarios are modeled.

However, the system, as described above, still needs a human instructor to (1) demonstrate a successful scenario solution, (2) provide the student with instructional advice, (3) observe the student’s problem-solving and provide a comprehensive critique, and (4) score performance on various scenarios for progress evaluation and comparative analysis purposes. While the simulator itself is implemented with numerical and knowledge-based simulation techniques, requirements of an automated instructor include, first, achievement of the level of expertise sufficient to solve arbitrary scenarios in real-time; and, second, an ability to observe the student in real-time, communicate with the student, and present intelligible feedback in a natural language format. Such functions clearly present an interesting challenge for modern AI technology.

In this paper we present an automated instructor assistant, called Minerva-DCA, that is capable of doing the aforementioned four instructor functions. Minerva-DCA has been fielded at the Navy’s Surface Warfare Officer School (SWOS) in Newport, Rhode Island and has shown impressive performance.

Minerva-DCA: The Automated Instructor Assistant

AI Technology

As outlined above Minerva-DCA is a real-time problem-solver, capable of explanation, advising, critiquing, and scoring thus serving as an automated instructor for the damage control environment. These abilities result from utilizing an innovative combination of AI technology as highlighted below:

1. Minerva-DCA is based on a blackboard architecture with exhaustive deliberation on all available domain data posted on the blackboard. This approach delivers a comprehensive set of feasible actions that address the current problem state.

2. A separation and explicit representation of domain and strategy knowledge layers allows for the output of the blackboard deliberation step to be a set of dynamically constructed set of strategy networks. These facilitate explanation, advising, and critiquing.

3. An Extended Petri Net envisionment-based blackboard scheduler allows for sophisticated critiquing that accounts for creativity on the trainee's side.

In the following sections, we will go into the key details of Minerva-DCA design and implementation.

Blackboard Framework

Minerva-DCA is an extended blackboard architecture system (Hayes-Roth 1985, Carver and Lesser 1994, Bulitko and Wilkins 1998b, Nii 1989, Larsson et al. 1996, Park et al. 1994, Najem 1993, Park et al. 1991). The blackboard is accessed by a number of domain knowledge sources that constitute the domain knowledge layer. Domain knowledge sources, however, do not operate on their own but rather are used by the strategy knowledge sources. The reasoning is done via the strategy networks.

```
ccf(r1012,1,1,[alarm,fire,Where,Time],800, [fire, Where,
FireClass, discovered, Time],0.6,[]).
ccb(r1012,1,1,[alarm,fire,Where,TimeAlarm],800,[fire, Where,
FireClass, Status,Time],0.6).
```

Meaning: finding alarm indicates hypothesis fire with a confidence of 0.6.

Figure 1. Example of a domain rule that provides evidence for fire hypothesis

In the next few sections we will describe the domain and strategy level representation. This is fairly conventional in terms of second-generation expert systems (Clancey 1985, Clancey 1987, Chandrasekaran 1986); the innovative part is the way the output of the domain and strategy level unification is processed by the envisionment-based scheduler.

Domain Knowledge Layer

Each domain knowledge source contains several Horn-clause style domain rules related to a particular domain topic (e.g. handling a certain type of fire). An annotated example of an actual domain rule for the Navy damage control domain is shown in Figure 1.

Domain knowledge and domain vocabulary are conceptually organized in the domain graph (Bulitko 1998a) which is a graph with domain findings, hypotheses, and actions in the vertices. Domain rules in the presented format comprise the edges of the domain graph and thus allow movement from one domain datum to another. The entire decision process is roughly represented as traversing the domain graph starting at the vertices with known findings and eventually arriving at the vertices with domain actions.

In order to handcode the domain knowledge layer we have conducted a knowledge acquisition process involving: (1) researching the Navy damage control manuals and damage control plates (analogous to ship blueprints); (2) consulting Navy domain experts; (3) attending classes at the Surface Warfare Officer School (SWOS) in Newport, Rhode Island.

Strategy Knowledge Layer

Minerva-DCA's deliberation mechanism uses a declarative and domain-independent representation of strategy knowledge. The strategy layer of Minerva-DCA knowledge is used to drive the deliberation process. Depending on the new data and other blackboard contents, Minerva performs backward or forward chaining. This makes Minerva-DCA flexible in its reasoning process. While domain knowledge sources reason over the lexicon of domain findings, hypotheses, and actions the rule-based strategy knowledge sources reason over the lexicon of domain-independent strategy goals. Strategy goals come at

Top-level goals:
- process_hypothesis(Hypothesis)
- process_finding(Finding)
- explore_hypothesis(Hypothesis)
- remove_datum(Datum)

Intermediate-level goals:
- applyrule_backward(Rule)
- applyrule_forward(Rule)
- findout(Datum)
- pursue_hypothesis(Hypothesis)
- test_hypothesis(Hypothesis)

Bottom-level goals:
- perform(Action)
- lookup(Finding)
- remove(Datum)
- conclude(Hypothesis)

Figure 2. Strategy goals used as nodes of the strategy networks in Minerva-DCA

different levels as summarized in Figure 2.

The goals are used in building the strategy networks. A strategy network is a directed acyclic graph consisting of strategy chains. Each chain starts with a top-level goal, goes through the intermediate and top-level goals, and ends with a bottom level goal that carries a domain-level action to take. During the deliberation process the strategy knowledge sources rules are triggered by important findings and active hypotheses on the blackboard. The

```
mr(pf1,process_finding(F),applyrule_forward(Rule,Hyp)) :-
        finding(F,_),
        red-flag(F),
        ccf(Rule,N,M,F,CF,Hyp,CFC,UL),
        satisfied(F,CF).
```

Meaning: strategy goal process_finding(F) could be reduced to strategy goal applyrule_forward(Rule,Hyp) if there is a domain rule Rule with F as its condition clause and the clause is satisfied. So if we are processing finding F and there is a rule conditioned on F then try to apply that rule.

Figure 3. An example of strategy rule used for forward-chaining

triggering data is considered within the context of top-level goals. For example: fire alarm finding(fire-alarm) would lead to the top-level goal process_finding(fire-alarm). In turn, this goal will trigger other strategy rules and thus entail intermediate-level goals. This process eventually results in the strategy chain network (a strategy chain example is shown in Figure 4). Each edge in the network is labeled with the corresponding strategy operator identifier (e.g. pf5). Figure 3 shows an example of actual strategy rule. Strategy operators are also implemented as Prolog clauses. The first argument of mr is the strategy operator identifier (in this case pf1). The second argument is the higher level goal the operator applies to. Finally, the third argument is the lower-level goal. So, in a way, a strategy operator is nothing but a transition from a higher to a lower level goals. The pre-conditions of mr are various predicates that have to hold in order for the strategy operator to fire. In the example shown in Figure 3 we should apply Rule in a backward manner if:

(a) we are trying to process finding F;
(b) F is important (a "red-flag");
(c) there is (ccf) a domain rule (Rule) such that F is one of its preconditions (with the required confidence factor of CF) and Hyp is its conclusion;
(d) and finally F is known to hold with confidence factor of at least CF.

In handcoding the strategy knowledge layer we started with a strategy layer of Minerva-3 medical diagnosis expert system (Park et al. 1991, Park et al. 1994) which was a refinement of Neomycin (Clancey 1987, Wilkins 1990). The layer has then been significantly modified to address the following major differences between damage control decision-making and medical diagnosis domains: (1) multiple simultaneous crises; (2) action generation; (3) real-time autonomous operation. Details can be found in (Bulitko 1998a).

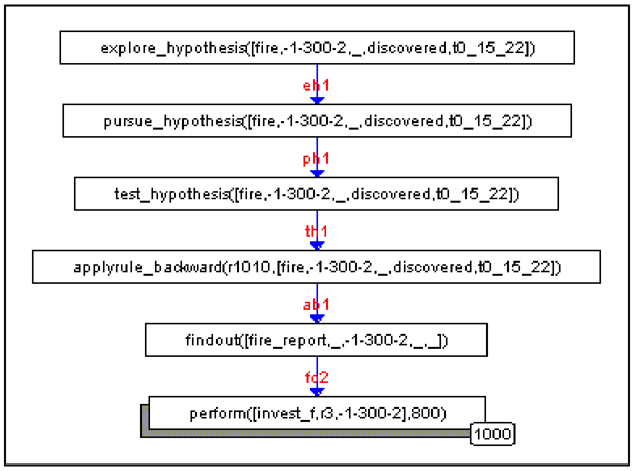

explore_hypothesis([fire,-1-300-2,_,discovered,t0_15_22])
eh1
pursue_hypothesis([fire,-1-300-2,_,discovered,t0_15_22])
ph1
test_hypothesis([fire,-1-300-2,_,discovered,t0_15_22])
th1
applyrule_backward(r1010,[fire,-1-300-2,_,discovered,t0_15_22])
ab1
findout([fire_report,_,-1-300-2,_,_])
fc2
perform([invest_f,r3,-1-300-2],800)
1000

Figure 4. A Strategy Chain Example: Order Repair-3 station to investigate compartment 01-300-2 since there is a suspected fire in the compartment; a fire report would confirm/disconfirm the fire hypothesis; and a fire report could be obtained through investigation.

As we have said, for the domain and strategy knowledge layers the described knowledge representation and inference method are fairly conventional with respect to second-generation expert systems (Clancey 1987, Wilkins 1990). The innovative aspect of Minerva-DCA relates to the way the scheduler processes the output of the deliberation level created by unification of domain and strategy knowledge layers.

Scheduler Knowledge Layer

Minerva-DCA uses an innovative envisionment-based scheduler with the environment model implemented as an extended Petri Net (Bulitko 1998a). The scheduler consists of the following main parts:

1. *An Extended Petri Net (EPN) environment model* is capable of a coarse-level environment envisionment. The model takes into account the current state of the ship as well as the actions under consideration. It then predicts a sequence of ship states within the 10-minute range. Using an EPN model has a number of advantages including the following: (1) it models the ship at a coarse level of detail hence allowing for a high-speed operation (up to 600 times faster than real-time) and better noisy data handling; (2) it can model physical phenomena (fire, smoke, flooding, etc.) as well as equipment operation (overheating, failures, etc.) and personnel activities (occupation, casualties, etc.); (3) it can take proposed DCA actions as an input; (4) Petri Nets naturally support concurrency and have a convenient

graphical representation; (5) Petri Nets have a sound mathematical theory and allow for various analysis and proof methods.

2. *A static state evaluator* is used to evaluate the envisioned states with regard to the state severity from the damage control standpoint. While the EPN model describes ship states by markings of the Extended Petri Network, the state evaluator uses a vector form where each value summarizes the important parameters such as the status (engulfed, ignited, intact, etc.) and duration of the status about vital ship components. The evaluator assesses the severity of the state and outputs a single value: the estimated time to ship loss (i.e. a major disaster such as a missile magazine explosion) (Bulitko 1998a). Naturally, lower values of the time-to-ship-lost correspond to more severe states.

Having elicited domain knowledge from various sources, we handcoded the Extended Petri Net model. Automated generation of such a model is an area of current research. The state evaluator, however, has been automatically generated from the past scenarios in the following fashion. While running simulated damage control scenarios we had regularly recorded the states of the ship and time-stamped them. Complete scenario timeline available at the end of scenario was used to annotate the sequences of ship states with the time-to-ship-lost. Thus, each state annotated with

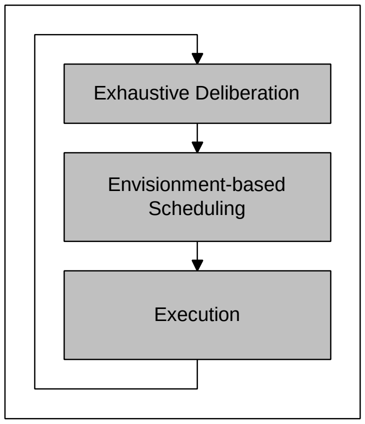


Figure 5. Minerva-DCA Operation: the blackboard problem-solving cycle stages

time-to-ship-lost constituted a single training example. A collection of such training samples has been used with well-known machine learning packages such as NeuroSolutions' back propagation learning engine (Haykin 1994) and C5.0 decision-tree/rule learner (Quinlan 1993). The output of these packages comprised the state evaluator. In the case of backpropagation learning it was an artificial neural network while C5.0 produced a decision ruleset. The decision ruleset has attained cross-validation accuracy of 80-90% (Bulitko 1998a).

The Blackboard Cycle

Minerva-DCA works in cycles (Figure 5). Each cycle consists of the following three stages:

1) Exhaustive deliberation;
2) Envisionment-Based Scheduling - qualitative prediction, state evaluation, utilities computation;
3) Execution.

The following subsections will go into details on each of the stages.

Deliberation. At the deliberation stage, the important data from the blackboard are used to trigger domain rules and build a strategy network that represents exhaustive deliberation. Each of the strategy chains starts with a top-level goal (e.g. process_hypothesis(fire)) and ends with a feasible action (e.g. perform(fight_fire)). Different networks can share different nodes. The size of the strategy network is, at most, linear in the number of important (red-flag) findings and hypotheses. Since the network is built in a depth-first manner, the deliberation time is linear in the total number of findings as well (Bulitko 1998a).

Scheduling. At the scheduling stage prediction, evaluation, and utility computation are carried out. As mentioned above, each strategy chain ends with a feasible action (either internal to the system (e.g. conclude(hypothesis)) or external or domain-level (e.g. perform(action)). While most internal actions could be executed at once, domain-level actions require scheduling due to the limited resources, different priorities, and possible inconsistencies. Scheduling stage has three substages:

(a) Envisionment-Based Scheduling: An Extended Petri Net (EPN) predictor models the environment to predict effects of a particular action or consequences of a particular finding.
(b) State evaluator evaluates the predicted states and assesses their severity levels.
(c) Utility computation module combines the outputs of the both stages above and ranks the actions by assessing their utility.

Execution. The last stage of the blackboard operating cycle is execution. There are three different execution modes: problem-solving, advising, and critiquing.

Operation Modes

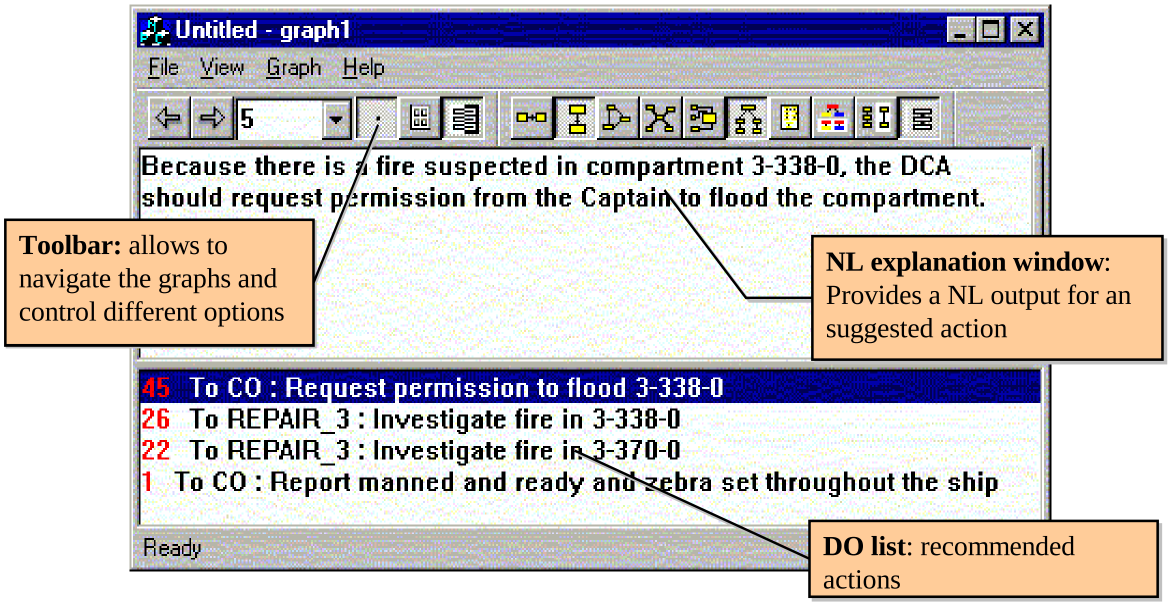
The three execution modes will now be covered in detail.

In problem-solving mode, the highest-ranked domain-level (external) action is executed. All internal actions are also executed. Examples of internal actions include asserting hypotheses and removing obsolete data from the blackboard. Executing domain-level actions involves passing the appropriate messages to the actuators (either simulated or real). It is worth noticing that domain-level actions could be inconsistent with each other (e.g. one of the reasons is the existence of multiple solution paths). To avoid potential conflicts, Minerva-DCA executes the single top-ranked domain-level action at every cycle.

In advising mode, the system's external actions are not passed to the domain actuators but instead used for advising. Specifically, we present *n* top-ranked actions to the student through an Advisory Graphical User Interface (GUI). For each action the following information is available upon request:

a) Reasoning behind the action could be shown by displaying the appropriate strategy chain(s) in both graphical and textual forms. The textual form involves generating natural language output for the action and the top nodes of the chain (the short form) or possibly all nodes of the chain (the long form). Indeed, the advantage of the explicit strategy knowledge representation and reasoning is that we can easily explain why a certain action was suggested by simply traversing the strategy chain and translating the nodes into a natural language. Figure 6 shows a screenshot of the actual GUI displaying several suggested actions and a short-form explanation for one of them.

b) Reasoning behind the action's rank could be shown by displaying the environment states predicted by the EPN prediction module and their scores computed by the state evaluator. Further details could be provided by tracking down state evaluator's reasoning if it allows for that (e.g. in case with decision trees).



Because there is a fire suspected in compartment 3-338-0, the DCA should request permission from the Captain to flood the compartment.

45 To CO : Request permission to flood 3-338-0
26 To REPAIR_3 : Investigate fire in 3-338-0
22 To REPAIR_3 : Investigate fire in 3-370-0
1 To CO : Report manned and ready and zebra set throughout the ship

Toolbar: allows to navigate the graphs and control different options

NL explanation window: Provides a NL output for an suggested action

DO list: recommended actions

Figure 6. Minerva-DCA Advisory Graphical User Interface features natural language explanations of the advised actions

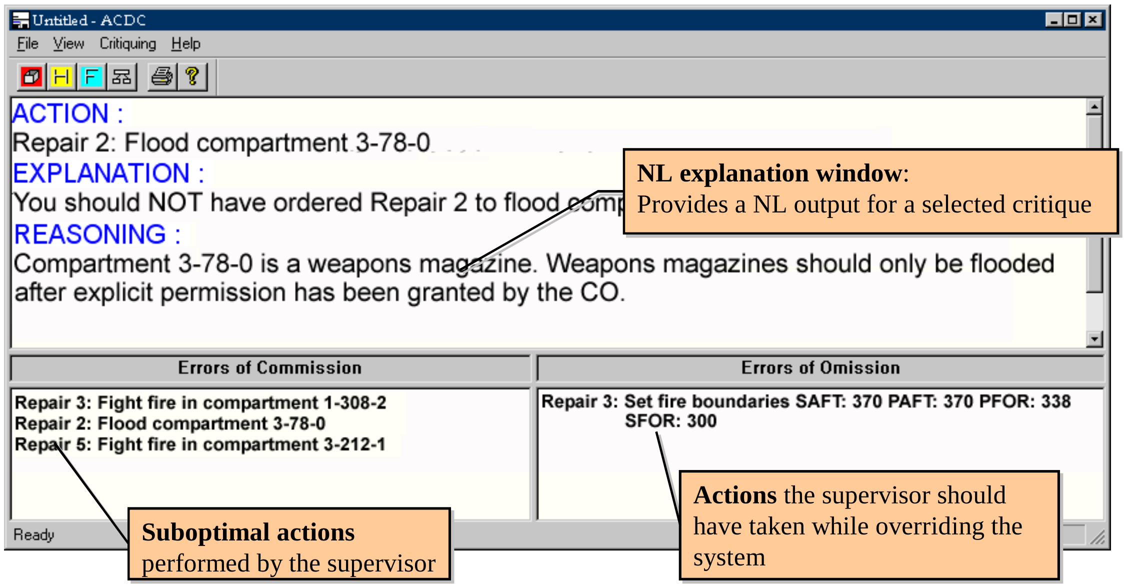
In the critiquing mode the scheduled actions are not passed to the actuators for execution but used to match against actions of the subject being critiqued. The intricate details of the approach are presented in (Bulitko 1998a) while in this section we will limit ourselves to a brief overview.

Basically, the critiques provided are of two kinds: errors of omission and errors of commission.

An *error of omission* occurs when the subject fails to take an action associated with a critical goal. This corresponds to a high-level goal remaining unaddressed on the blackboard. When displaying such an error it is easy to invoke the explanatory facilities of the shell and generate a NL explanation on why a certain action should be taken and the goal it would address.

Errors of commission are handled in two steps. First, the system tries to match a subject's action against the actions the system has recently deliberated and scheduled for execution. If the matching succeeds and the matching action found was ranked low by the scheduler then we can critique the subject by supplying the scheduler's reasoning ("poor action" critique). If the matching action is not one of the top-ranked actions then we can critique the subject by showing a better action ("suboptimal action" critique). Finally, if there is no matching action on the system's side we feed the action to the envisionment-based scheduler. The scheduler produces a rating of the action based on the envisionment process. It may well be that the action is better than that of the Minerva-DCA problem solver, in which case no critique needs to be displayed. If the action is judged by the envisionment-based scheduler to be a poor one, then a critique is appropriate. The envisionment process outputs the deleterious consequences of the subjects action.

Figure 7 presents the Minerva's Critiquing GUI showing an error of commission critique explained. Just like with advising, the explicit strategy knowledge representation and reasoning process allow the critiquing facility to be implemented naturally without an extra dedicated critiquing expert system shell like it would with alternative critiquing approaches such as the use of a bug library.


Untitled - ACDC
File View Critiquing Help
NL explanation window: Provides a NL output for a selected critique
Errors of Commission
Errors of Omission
Suboptimal actions performed by the supervisor
Actions the supervisor should have taken while overriding the system
Ready

ACTION :
Repair 2: Flood compartment 3-78-0
EXPLANATION :
You should NOT have ordered Repair 2 to flood comp
REASONING :
Compartment 3-78-0 is a weapons magazine. Weapons magazines should only be flooded after explicit permission has been granted by the CO.

Repair 3: Fight fire in compartment 1-308-2
Repair 2: Flood compartment 3-78-0
Repair 5: Fight fire in compartment 3-212-1

Repair 3: Set fire boundaries SAFT: 370 PAFT: 370 PFOR: 338 SFOR: 300

Figure 7. Minerva-DCA Critiquing Graphical User Interface features natural language explanations of the critiques

Hardware and software implementation details

Minerva-DCA is implemented in LPA Prolog and runs under Windows NT. An ODBC interface integrates it into the DC-Train simulated environment which uses Microsoft Access databases to represent the state of the ship and other dynamic and static domain information. The graphical user interfaces are implemented in Visual C++ and run as separate tasks under Windows NT. The graphical user interfaces and Minerva-DCA exchange information through ODBC interface and dedicated Access files.

In our tests Minerva-DCA has been running at 50-800 blackboard cycles per second speed on a dual-Pentium II-400MHz workstation.

Minerva-DCA Experimental Evaluation

To evaluate the performance we have run approximately 500 scenarios of the DC-Train ship damage control simulator at the Navy officer training school in Newport, Rhode Island and at our home laboratory. We have compared the problem-solving performance of Minerva-DCA to Navy officers. The results are presented in Figure 8.

Any scenario could have one of the three outcomes: "ship lost" meaning that a major disaster such as a missile magazine compartment explosion has occurred; "ship possibly saved" meaning that at 25 minutes scenario time the ship was still alive yet there were active crises; and "ship saved" means there were no active crises at the 25-minute mark.

In the experiments Minerva-DCA has lost 21 ships. This is a 46% improvement over Navy officers where 39 ships were lost. Likewise, Minerva-DCA has shown a 76% improvement in the number of ships being saved (117 vs. 28).

Analysis of the scenarios where Minerva-DCA has failed to save the ship helped locating suboptimal components of Minerva-DCA domain knowledge layer as well as certain modeling problems of the DC-Train damage control simulator. A large-scale evaluation of the improved versions of Minerva-DCA and DC-Train is pending.

Application Use and Pay-Off

A proof-of-principle prototype, called DC-Train 1.0, was first used at the Navy officer training school in Newport, Rhode Island, in March 1997. A refinement of this system, called DC-Train 2.0, was permanently deployed at SWOS in December 1998. It is to be regularly used in the six-week damage control course, which has approximately 30-50 Navy officers every six weeks. It provides the damage control course with its first tool that can generate arbitrarily many damage control scenarios and thereby provide extensive and immersive whole-task training.

To date, the controlled evaluation has been limited to each person solving four simulated scenarios at most, and has not involved giving the subjects feedback using the Automated Instructor Assistant. The initial limitations on the numbers of scenarios solved, and the withholding of automated feedback are necessary to establish a performance baseline. Experts have, however, validated the accuracy of the feedback generated by the Automated Instructor Assistant.

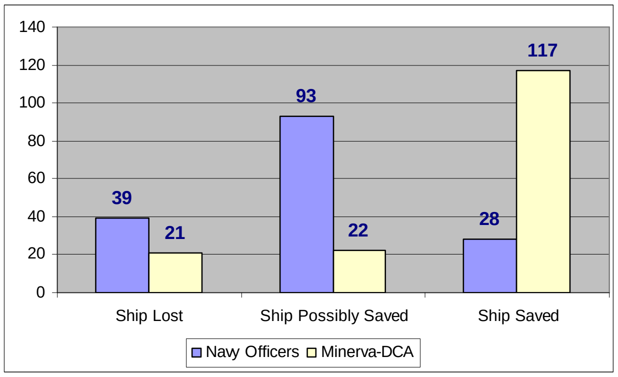


Figure 8. Minerva-DCA vs. Navy officers experimental comparison

The primary benefits of the deployed immersive simulator and trainer described in this paper include (1) decreased load on the damage control course instructors, (2) the opportunity for the students to get a comprehensive feedback on their performance anytime, and (3) uniform and standardized scoring with a graphical feedback. This is in contrast to the current practice where an instructor is always present when a student solves a scenario, which places a very large time-demand on the instructors.

Minerva-DCA deployment at SWOS

Minerva-DCA was developed in the Knowledge Based Systems Group (KBS) of Beckman Institute, University of Illinois at Urbana-Champaign. After several years of field tests the system has been permanently installed at the Navy officer training facility (SWOS) in Newport, Rhode Island. An internet link to the University of Illinois allows for real-time automated data collection and analysis.

Since the deployment of the system at SWOS, the KBS group has maintained DC-Train 2.0 and Minerva-DCA packages via a close co-operation with SWOS personnel. Maintenance has included: (1) collecting feedback from the instructors and students; (2) sending in updates and patches; and (3) discussing the new features and extensions to be implemented in the up-coming versions.

Future Work

During the remainder of 1999, a group of research psychologists will be conducting controlled experiments to measure the training effectiveness of solving large numbers of scenarios. They will also measure the impact of the feedback generated by the Automated Instructor Assistant in terms of scores, advice, and critiques. This will provide essential information for refinement of the system, and quantification of its utility.

Another near-term project goal is to install the DC-Train system aboard ships for Afloat Training. All seasoned damage control assistants aboard ships receive weeks of refresher training every year, and DC-Train will allow them for the first time to receive extensive whole-task training with an computer-based damage control simulator. This is of interest because it will be possible to explore the efficacy of a single person solving more scenarios than is possible within the context of a relatively short course.

Finally, it is planned to use Minerva-DCA as a component of a project sponsored by the Naval Research Lab on automated intelligent control for the next generation of ships to be built. It is called DC-ARM – (Damage Control: Automation for Reduced Manning). This project seeks to automated many aspects of the damage control process (Wilkins and Sniezek 1997). We will investigate the extent that Minerva-DCA can assist with the automation process, and the Intelligent Assistant can assist with providing a damage control supervisor with real-time situation awareness.

Acknowledgements

Scott Borton, Adam Boyko, Dr. Valeriy K. Bulitko, Tony Czupryna, Joel Hegg, Sebastian Magda, Arthur Menaker, and Tamar Shinar have helped with Minerva-DCA implementation and have contributed a number of valuable suggestions. We gratefully acknowledge the support of the SWOS personnel, in particular: CAPT Wittcamp, CDR Shikada, CDR Kunnert, LCDR Morrill, LT Anderson, LT Cisneros, LT Gianelli, LT Kenderick-Holmes, DCCS(SW) Ciesielczyk, and Dave Monroe. SWOS students and KBS members have been very cooperative. The research has been supported in part by ONR Grant N00014-95-1-0749, ARL Grant DAAL01-96-2-0003, and NRL Contract N00014-97-C-2061.

References

Baumann, M.R.; Sniezek, J.A.; Donovan, M.A.; Wilkins, D.C. 1996. Training effectiveness of an immersive multimedia trainer for acute stress domains: Ship damage control. University of Illinois technical report, UIUC-BI-KBS-96008.

Bulitko, V. 1998a. Minerva-5: A Multifunctional Dynamic Expert System. MS Thesis. Department of Computer Science, University of Illinois at Urbana-Champaign.

Bulitko, V.; Wilkins, D.C. 1998b. Minerva: A Blackboard Expert System for Real-Time Problem-Solving and Critiquing. Tech. Report UIUC-BI-KBS-98-003, University of Illinois at Urbana-Champaign.

Chandrasekaran, B. 1986. Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design. *IEEE Expert*, 1(3):23-30.

Carver, N.; Lesser, V. 1994. The Evolution of Blackboard Control Architectures. In Expert Systems with Applications--Special Issue on the Blackboard Paradigm and Its Application, Volume 7, Number 1, pp. 1-30, Liebowitz. New York, Pergamon Press.

Clancey, W.J. 1985. Heuristic Classification. *Artificial Intelligence*, 27(3):289-350.

Clancey, W.J. 1987. Acquiring, representing, and evaluating a competence model of diagnostic strategy. In Chi, Glaser, and Farr, eds. *Contributions to the Nature of Expertise*. Lawrence Erlbaum Press.

Ericsson, K.A.; Krampe, R.T.; Tesch-Romer, C. 1993. The Role of Deliberate Practice in the Acquisition of Expert Performance. Psychological Review, 100(3): 363-407.

Feigenbaum, E.A. 1977. The Art of Artificial Intelligence: I. Themes and Case Studies of Knowledge Engineering. Proceedings of the Fifth International Joint Conference on Artificial Intelligence, 1014-1029. Cambridge, MA.

Fuller, J. V. 1993. Measuring Damage Control Assistant's (DCA) Decision-Making Proficiency in Integrated Damage Control Training Technology (IDCTT) Training Scenarios. Master's Thesis. Naval Postgraduate School: Monterey, CA.

Grois, E..; Hsu, W.H.; Voloshin, M.; Wilkins, D.C. 1998. Bayesian Network Model for Generation of Crisis Management Training Scenarios. *In The Proceedings of The Tenth IAAI conference*. 1113-1120. Menlo-Park, Calif.: AAAI Press.

Hayes-Roth, B. 1985. A blackboard architecture for control. *Artificial Intelligence*, 26(2):251-321.

Haykin, S. 1994. *Neural Networks: A Comprehensive Foundation*. Macmillan College Publishing Company.

Jones, R.M.; Laird, J.E.; Nielsen, P.E. 1998. Automated Intelligent Pilots for Combat Flight Simulation. *In The Proceedings of The Tenth IAAI conference*. 1047-1054. Menlo-Park, Calif.: AAAI Press.

Johnson, M. 1994. Validation of an active multimedia courseware package for the integrated damage control training technology (IDCTT) Trainer. Master’s thesis. Naval Postgraduate School, Monterey, California.

Larsson, E.; Hayes-Roth, B.; Gaba, D. 1996. Guardian: Final Evaluation. Knowledge Systems Lab, Stanford University. TechReport KSL-96-25.

Najem, Z.H. 1993. A Hierarchical Representation of Control Knowledge For A Heuristic Classification Shell. Ph.D. Thesis. The Department of Computer Science. University of Illinois at Urbana-Champaign.1993.

Nii, H.P. 1989. Blackboard Systems. In Barr, A.; Cohen, P.R.; Feigenbaum, E.A. eds. *The Handbook of Artificial Intelligence*. Volume IV, Chapter XVI. Addison-Wesley.

Park, Y.T; Tan, K.W.; Wilkins, D.C. 1991. Minerva 3.0: A Knowledge-based Expert System Shell with Declarative Representation and Flexible Control. Tech. Report. UIUC, Department of Computer Science.

Park, Y.T.; Donoho, S.; Wilkins, D.C. 1994. “Recursive Heuristic Classification”. *International Journal of Expert Systems*, Volume 7, Number 4, 329-357.

Quinlan, J.R. 1993. *C4.5: Programs for Machine Learning*, San Mateo, California: Morgan Kaufmann.

Wilkins, D.C. 1990. “Knowledge Base Refinement as Improving an Incorrect and Incomplete Domain Theory”, in *Machine Learning: An Artificial Intelligence Approach, Volume III*, Y. Kondratoff and R.Michalski (eds.), Morgan Kaunfmann, 493-513.

Wilkins, D.C.; Sniezek, J.A. 1997. An Approach to Automated Situation Awareness for Ship Damage Control. KBS Tech. Report UIUC-BI-KBS-97-012. University of Illinois at Urbana-Champaign.