Causal Models and Learning for Robust Human Behavior Models

Dr. Paul E. Nielsen, Jonathan T. Beard, Sean A. Lisse
Soar Technology, Inc.
Ann Arbor, MI 48105
nielsen@soartech.com, beard@soartech.com, lisse@soartech.com

ABSTRACT

Human behavior representations (HBRs) are an essential component of simulation-based training. Historically, these HBRs involve significant cost to encode expert knowledge for specific uses. One of the largest development costs is re-engineering due to failure, where failures stem from incorrect, incomplete, obsolete, or inconsistent information.

Previous work has demonstrated that it is possible for HBRs to overcome a significant number of their own failures by providing them with limited meta-awareness through self-monitoring, error-detection, and failure recovery. A limitation of those approaches has been their reliance on random selection among possible actions. Although self-correcting systems utilizing random selection are theoretically capable of eventually searching the entire recovery space, this is undesirable in real-time applications. Pruning the search space to decrease the number of options that must be attempted before achieving success can reduce the impact on real-time applications as well as reducing the incidence of recovery-induced failure. However, it can be developmentally intensive to manually annotate world states with suggested recovery actions.

This paper describes a complementary methodology that improves the quality of recovery actions without significantly increasing development cost. Specifically, this paper examines constraints on failure recovery through the use of causal models and learning. These constraints drastically reduce the search possibilities over exhaustive recovery techniques. First, HBRs are enhanced with internal representations of causal models describing recovery domain spaces. These models guide the reasoning process to the selection of appropriate, relevant recovery actions. Once a recovery method is attempted, the self-monitoring capability of the agents is utilized to measure the efficacy of the selected recovery actions. Finally, a learning mechanism, which will be described in detail, is used to prefer future selection of recovery actions in similar circumstances.

ABOUT THE AUTHORS

Paul E. Nielsen is vice-president and co-founder of Soar Technology, Inc. an intelligent, simulation software company based in Ann Arbor, Michigan. He received his Ph.D. in computer science from the University of Illinois in 1988. Prior to joining the Soar Technology he worked at the GE Corporate Research and Development Center and the University of Michigan Artificial Intelligence Laboratory. Dr. Nielsen's primary research interest lies in creating intelligent autonomous entities that generate human-like behavior.

Jonathan T. Beard is a Software Engineer and the TacAir-Soar Product Manager at Soar Technology, Inc. He has attended Hope College, the Kyrgyz State National University, and Northern Michigan University. Prior to joining Soar Technology, he was a Systems Architect at DaimlerChrysler, where he was the technical lead and designer of the EMIS-3 enterprise change-management system. Mr. Beard's primary research interests are in human behavior modeling, wearable and embedded computing, and cooperative autonomous agents.

Sean A. Lisse is a Software Engineer at Soar Technology, Inc. He has a Bachelor of Arts in Computer Science from Rice University. Prior to joining Soar Technology he worked at Enron Broadband Services, where he was a member of the EnFiber bandwidth trading software development team. Mr. Lisse's primary research interests are in knowledge representation, knowledge engineering, and the combination of traditional software engineering with advanced artificial intelligence techniques.

Causal Models and Learning for Robust Human Behavior Models

Dr. Paul E. Nielsen, Jonathan T. Beard, Sean A. Lisse
Soar Technology, Inc.
Ann Arbor, MI 48105
nielsen@soartech.com, beard@soartech.com, lisse@soartech.com

INTRODUCTION

Errors are inevitable in any non-trivial system. All representations that lack knowledge, including humans, will fail. The difference between humans and human behavior models is that humans can recognize their failures, resolve them, and move on, while models are generally incapable of recovery and tend to become "stuck" or crash.

We believe that it is not possible to eliminate the sources of failure in any non-trivial system. Legacy approaches have attempted to eliminate failure, but empirical results demonstrate that they have been unsuccessful (Ginsberg, 1989). While this may be attributed to sloppy programming, theoretical results indicate that it is impossible to test all possible scenarios of a complex system (Gödel, 1931). Thus, failure will occur in any non-trivial system. Our approach assumes that failure will occur and embraces it, allowing the system to recover before failure becomes catastrophic.

This paper presents an approach to error detection in complex human behavior models that is based on abstract causal models and learning. The approach is tractable for computation yet complete in its ability to represent all relevant influences upon the system. The type of knowledge presented here is fundamentally different from the "how to" knowledge normally associated with carrying out a task. Instead it describes high-level, general characteristics of the desired world state and makes explicit methods that are acceptable or unacceptable. Because there could be an enormous number of behaviors that are unacceptable, the knowledge must be specific to the goals the agent is trying to achieve. Rather than attempt to anticipate all possible behaviors a priori, there is additional knowledge in the system that will recognize discrepancies between allowable world states and unacceptable world states. Once the agent realizes that something is wrong, additional repair knowledge may be brought to bear that will alter the behavior until it is once again acceptable.

TERMINOLOGY

Brittleness

Brittleness is a lack of reasonable behavior due to unanticipated external conditions. It is exacerbated both by complexity and the desire to use software in domains for which it was not originally designed. Brittleness severely limits the effectiveness of intelligent behavior systems and their application for military endeavors. Though the system may exhibit satisfactory performance within a narrowly defined area of expertise, once the situation becomes sufficiently complex the possibility of failure approaches one hundred percent.

Brittleness arises when knowledge about how to handle the current situation is:

1. Inaccurate: knowledge is untrue
2. Incomplete: knowledge is lacking
3. Obsolete: knowledge is no longer true
4. Inconsistent: knowledge contradicts itself
5. Ill-defined: knowledge is too vague to use

Brittleness significantly increases the life-cycle cost since software maintenance effort must be expended to diagnose and fix problems. It can result in lost person time and hold up other expensive resources resulting in loss beyond the system itself. If used for mission critical applications the results of failure can have catastrophic consequences. Finally, the results of any such system may be unsuitable because of lack of faith in the results.

Robustness

In contrast, *robustness* is the ability for a system to exhibit reasonable behavior in response to unanticipated events.

BACKGROUND

Soar Technology, Inc. undertook the task of improving the robustness of behavioral models for computer-generated forces with the idea that the problem of brittleness is fundamentally a knowledge problem arising out of ignorance (incomplete knowledge), not representation (architecture). Our claim is that

brittleness is caused by a lack of knowledge about what should be expected.

Soar Technology, Inc. develops human cognitive models for use in military simulations. These models account for a wide range of behaviors including air-to-air combat, air-to-ground attack, reconnaissance, control, and support. These representations are created in the *Soar* computational model of human cognition (Laird, et. al., 1991), which is used extensively as a basis for human performance modeling in cognitive science and artificial intelligence. Models developed with Soar can provide insight into such aspects of human performance modeling as task timing, cognitive system load, focus of attention, and novice vs. expert level behaviors (Newell, 1990; Rosenbloom, et al., 1993).

The specific application we are using to demonstrate this methodology is *TacAir-Soar*, (Jones, et. al., 1999) which is a rule-based implementation of human cognitive models for pilot and controller behaviors.

TacAir-Soar generates entity level behaviors for most aircraft and missions in the military. It is a highly complex application consisting of approximately 8000 rules describing navigation, coordination, communication, and tactics. It allows unsupervised agents to perform missions autonomously in simulation exercises.

CONTEXT

Scaling is problematic for human behavior representations. What works in a niche domain may be problematic for real-world applications. Thus, there have been large volumes of work reported about approaches that solve narrowly constrained problems but very few applications that attempt to replicate a significant portion of human behavior to a believable level of fidelity. The implications for robustness are that trivial systems may be proven correct exhaustively, but large-scale human behavior representations may fail because of complexity.

This ability to recognize failure is critical to developing intelligent systems. Before they can adapt their behaviors to new circumstances, they must first realize the futility of their current situation. Conventional programming techniques that do not do this will either continue their aberrant behavior or simply fail. Intelligent systems should embrace failure as an opportunity for improvement.

Our investigation is grounded in a specific application because a general solution to all commonsense knowledge would be too large in scope to produce tangible results. Without context, the problem explodes. Previous attempts at commonsense modeling have been largely unsuccessful because there is too much general-purpose commonsense to model it all completely (Copeland, 1997; Lenat, 2001).

APPROACH

To reduce brittleness in software systems, there are two primary things that must be done. First, there needs to be feedback from the environment. When directed to perform an action, the agent needs to confirm that the action was actually performed. For example, when told to turn, the entity needs to ensure that a turn actually is taking place, heading is changing, and the new heading is the indicated heading. This problem most often arises in missed communication, but also arises from actuator failure.

Second, the agents need a deep understanding of the task they are trying to perform. This project would develop a representation of the first principles of flight in a format usable by intelligent computer generated forces to provide them with a common sense awareness of expectations about their world. For example, a CGF should know that if speed is non-zero, position should be changing; maneuvering should result in a change of heading; etc.

These assertions provide a mechanism for performing a "reality check." The recovery procedure will be situation specific, but may include such recommendations as abandoning the current course of action, re-attempting the action, or requesting assistance. The resultant corrective action represents an opportunity for learning in the system so it will avoid making such mistakes in the future.

Our approach to software recovery is structured around an eight-step recovery plan, termed "*Recourse*", as follows:

1. Reduce failure
2. Represent the environment.
3. Recognize progress toward goals.
4. Allocate resources to the problem.
5. Diagnose the possible reasons.
6. Create alternative responses.
7. Initiate recovery actions.
8. Remember successful recovery actions.

This paper focuses on tractable methods of representing the environment and remembering successful recovery actions. Details of other aspects of this approach may be found in (Nielsen, et. al., 2002; Beard, et. al., 2002; and Kiessel, et. al., 2002)

CAUSAL MODEL

Although the effector-based recovery mechanism that is part of the Recourse approach is theoretically sufficient to eventually search the entire realm of

possible recovery actions, it is possible that the consequences of continuing failure might be terminal before recovery was successful. To reduce the amount of recovery action search necessary, we determined that giving agents the ability to reason about cause-and-effect relationships could help to guide selection of recovery actions – thus increasing the possibility of successful recovery before terminal failure in a real-time system.

For an agent to reason about cause-and-effect within a simulation environment, it is necessary for the agent to have an internal representation of that world to reason about. Providing a mental model of the environment as detailed as the actual environment is cognitively and computationally implausible, effectively requiring a recursive copy of the external world. Instead, we have given the agents an abstracted representation of the environment using qualitative models. A qualitative representation uses intervals, rather than actual numeric values, exhaustively covering the domain to be represented. Although this representation may suffer from ambiguity and lack the detail necessary to predict which of multiple outcomes will actually happen, it is useful as a guide to suggest possibilities and actions for recovery that can be tested in the physical environment and then further refined.

For the purposes of this research, we have only implemented a causal model representation for flight dynamics within TacAir-Soar, but there is no reason that causal models could not be created to cover every domain in which an agent may act. In our implementation, there are qualitative metrics for each evaluation in the flight domain. Using these qualitative representations, we can develop a computational model of causal actions, which may be predictive.

The causal model illustrated here (see Figure 1) shows how the relationships between the environmental parameters of the entity can be affected by the entity's actions. This illustration is a prototype visualization of a causal model indicating the cause and effect of various effectors in the flight domain. In developing recovery actions, an enhanced version of this causal model was used to describe the flight domain. Recovery actions are constrained to the effectors of the system; the agent can only use the controls it has the capability to manipulate. Therefore, the causal model defines the relationships between effectors of the system. The causal model itself was implemented within the agent as production rules encoding sets of cause-and-effect beliefs.

Agent reasoning which backchains through the causal model shows how different effectors can influence various environmental parameters of the agent. For example, if an agent detects an error in the execution of its goals, it may diagnose that the error is due to flying too fast. With that diagnosis, the agent could try to use the recovery system originally implemented as part of the Recourse methodology, and begin attempting recovery actions within the flight domain. On the other hand, using the newly implemented system of causal-model relationships encoded as beliefs, the agent would be able to evaluate the causal model encoded as beliefs to determine the selection of an appropriate effector action. In the model used, changes in acceleration and altitude both influence speed, so if a problem with the entity's speed is detected, it is possible to change the acceleration to influence the speed. The causal model beliefs describe the relationship of each of the environmental parameters of the entity, and the nature of the influence, i.e. whether it is direct or inverse. The variables described in this causal model are: pitch, roll, yaw, delta-altitude, altitude, acceleration, speed, position, thrust, heading, and delta-heading.

The agents have beliefs about how these various variables influence each other – both which environmental parameters affect which other parameters, and whether the effect is direct or inverse. The agents use that knowledge in their choice of recovery actions. The agents can identify flight-domain symptoms that correspond to the causal model, e.g., "going too slow". They translate that into a desire to go faster, or increase speed. Then the causal model tells them that to increase speed, one can, for instance, increase acceleration. Therefore, the agent could choose "increase acceleration" as a viable recovery action.

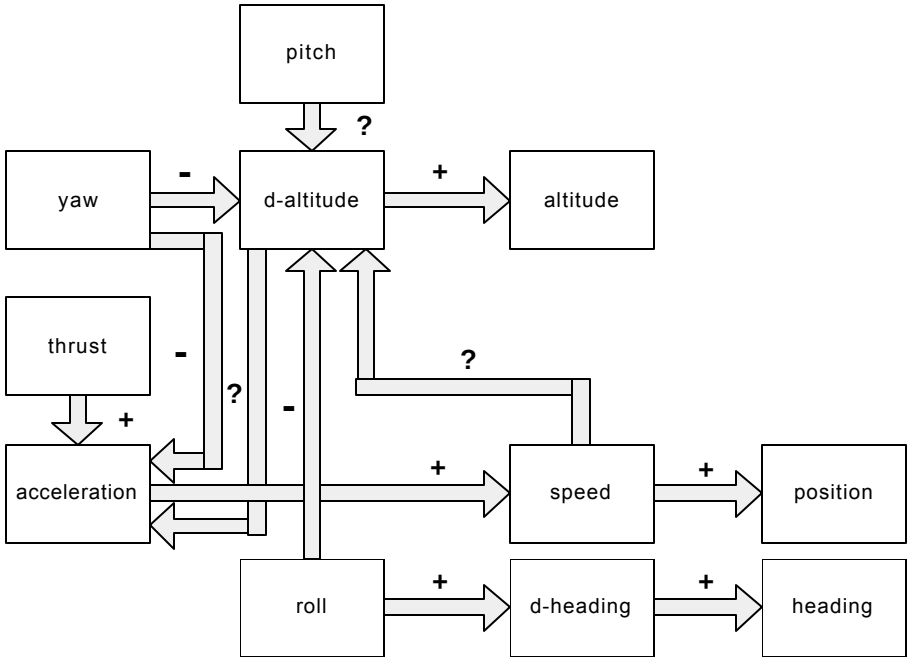

Figure 1. Flight domain causal back-chain model

Arrows indicate an influence of some sort between quantities. Those with a plus (+) indicate a positive influence between the quantities; those with a minus (-) indicate a negative influence; and those with a question mark (?) indicate that there is a relationship between the quantities, but it is indeterminate or requires further knowledge to determine the direction of the influence**.**

LEARNING

Learning is used principally at the action selection stage - if an action has been successful in resolving a particular problem in the past, that action should be remembered and tried first when the problem next occurs. Our first implementation attempt was to create a recovery state with learning enabled and chunk the effects of tried actions.

All learning in Soar occurs through the "chunking" mechanism. Chunking is a way of converting goal-based problem solving into accessible long-term memory. Whenever problem solving provides a result, a new production is created, whose actions are the new results and whose conditions are the working-memory elements that were involved in producing the results. This production is called a “chunk.” (Newell, 1990)

Several problems were experienced while experimenting with this method:

1) Soar immediately chunks the effects of any actions that effect parts of the agent outside the scope of the recovery action (e.g., putting something on the output link), making it very difficult to not learn actions that fail.
2) If success or failure can only be determined by examining elements outside of the learning state (e.g., checking input link parameters), Soar includes those conditions in the conditions of the chunk, making the chunk useless (i.e., it will only fire in situations in which the problem has already been solved).
3) Chunking does not deal well with time sensitive actions. It is very difficult to learn time constraints on actions (perform an arbitrary action and wait for ten seconds to see if it worked) and sequences of actions.

The current learning mechanism, as implemented, essentially collects counts of successes and failures to determine a historical success rate for each action suggested for a given diagnosis. These success rates are used to set preferences for choosing recovery actions in the recover state.

A separate recover state is created to try each action. An action is proposed and applied, and a timer is set. If the conditions for success are not met before the timer expires, the action has failed. It is marked as tried, and a new action is selected.

In both cases (success and failure), learning is briefly turned on, and a count of the success or failure is recorded on the "experience" node for the action. A new state must be created, so that trial activity can proceed without learning enabled. The counts are later used to determine a success rate for the action.

This method effectively works around the first and third problems described above. Setting flags for both success and failure to true before recovery starts can overcome the second problem described above. That will allow chunks to fire and contribute their knowledge, setting them to false at the start of recovery, and then setting the appropriate flag to signal success or failure during recovery.

This method has the added advantage of recovering gracefully from serendipitous successes in early trials, while maintaining the desirable property that an agent will persevere with an action as long as it is working.

Unfortunately, this method also seems to subvert the design intentions of chunking more than working with it. Whether this is due to my unforeseen applications or essential properties of chunking is an open question.

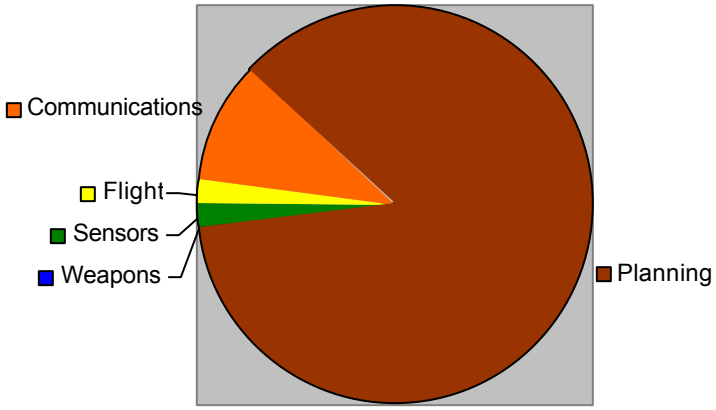

Figure 2. Percentages of errors induced by domain through rule removal

TEST METHODOLOGY

Platform setup

The tests were run on two platforms, an AMD Athlon XP 1700+ running at 1.4 GHz with 512 MB of RAM and an AMD Athlon at 1GHz and 512MB of RAM.

Each of the platforms had an identical default copy of Red Hat 7.3 installed. Each was furnished with an identical copy of JSAFv5 (MBCJSAF). To prevent outside interference, the test platforms were isolated from the local area network during test runs.

A set of test scripts were developed and committed to a version control system (CVS). As the tests were developed, each change to the scripts was committed into version control. Before each set of test runs, cvs update was run upon both platforms, and then the results of cvs diff on each of the platforms was used to identify any discrepancy between the CVS versions of the Tcl test scripts and the versions on the platforms.

Automation

The automation system was based upon the Android user interaction scripting language for XWindows. Android is built upon the Tcl language. It was modified for ease of installation, and a custom C program was coded to add the ability to detect the current window's title. Several Tcl libraries were developed to aid in JSAF automation and log parsing.

A set of modular scenarios was created. The scenarios created were as follows:

- 1S - A single Blue TAS agent in an F/A-18C
- 1D - A single Red task frame agent in a MIG-29 Fulcrum flying a sweep
- 1RS - A single Red TAS agent in a MIG-29 Fulcrum
- 4S - A four-ship of Blue TAS agents in F/A-18Cs
- 4D - Four single Red task frame agents in MIG-29 Fulcrums
- 4SR - A four-ship of Red TAS agents in MIG-29 Fulcrums

These modular scenarios were composed into the testing scenarios. For example, 1S vs. 1RS is an air engagement between an F/A-18C and a MIG-29. The complete set of testing scenarios was 1S, 1Sv1D, 1Sv1RS, 4S, 4Sv4D, and 4Sv4SR. Each testing scenario was run with one of three different levels of artificially induced failures: no productions removed, 5 productions removed (roughly 1% of original rulebase), or 55 productions removed (roughly 10% of original base). Two test runs were performed for each set of rule removals: with robustness error-correction code, and without. The productions to be removed were chosen randomly from the existing productions for each pair of test runs.

Pairs of test runs were performed in batches of 50 pairs each, once for each possible combination of testing scenario and production removal level. Each batch contained agent log files from 100 test runs. An example batch might use the 1Sv1RS testing scenario, and have a production removal level of 5 rules. Each batch was performed on both testing platforms.

Timing

Each test run lasted for 70 minutes of simulation time. The runs were performed at a ratio of 4:1 simtime to realtime. Each took roughly 19 minutes to perform from application run to scenario end. A total of 1800 test runs were performed on each testing platform (the platforms ran simultaneously). The total estimated time for all testing runs was 570 hours (if performed back-to-back 24 hours a day and 7 days a week, 3.39 weeks).

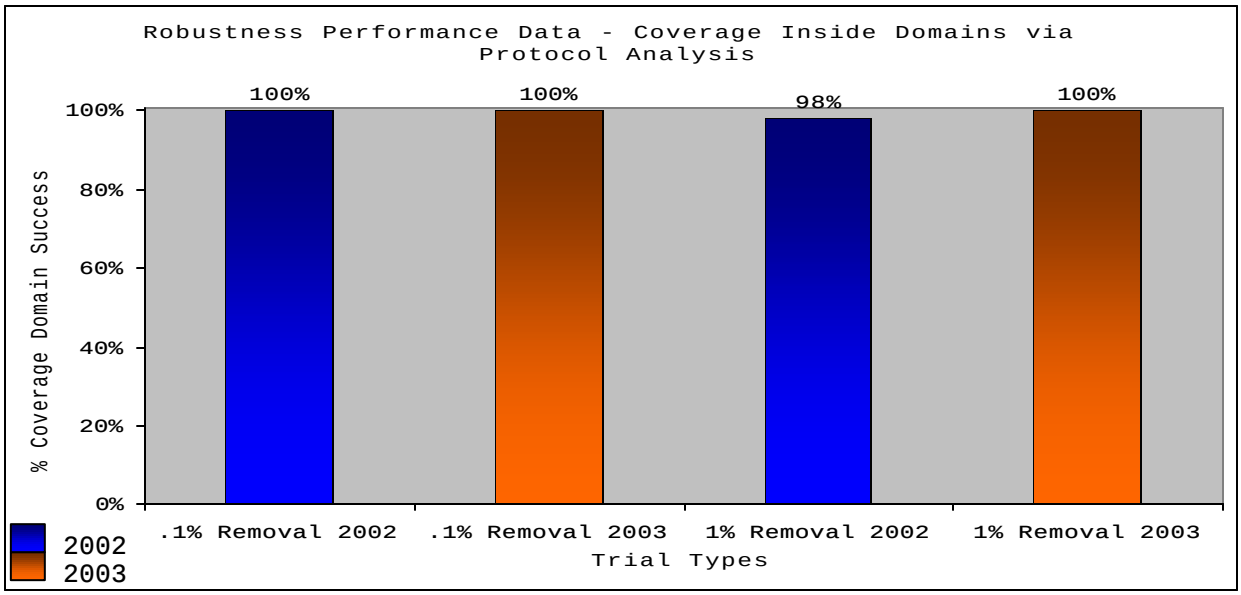

Figure 3. Comparison results of domain coverage via protocol analysis

Log Processing

Monitoring productions were inserted into each of the agents' rule bases, not considered for removal by the production-removal code. Each agent produced logs recording three separate types of events: production firings, operator selections, and milestone events. Tcl scripts invoking the Linux text-processing tools *sed* and *awk* were used to process the log scripts into comma-separated-value format.

Scripts also separated the events into logs of the different types of events, sorted them, and compared them quantitatively via *diff* and *lc* with selected golden logs containing logging information from known correct agent runs.

RESULTS

We derived some initially promising results with the tests described above in 2001. Specifically, for the 0.1% removal trials, on average, the Recourse-enabled TAS agents had about a 2.25% higher rate of mission success. For the 1.0% removal trials, the Recourse-enabled TAS agents had about a 5.25% higher rate of mission success. When compared to the amount of induced failure, this is an appreciable achievement. At the 0.1% rule removal rate, failure is at 12.25% and 10.0% respectively for non-Recourse and Recourse-enhanced TAS agents. That comes out to an 18.0% performance improvement from non-Recourse to Recourse-enhanced TAS agents. A similar comparison at the higher 1.0% rule removal rate shows a 6.9% improvement in performance.

When examining the log files generated during each of the failed trials and manually categorizing the cause of the failures, it was possible to further characterize the nature of the failures. In the case of the 1.0% rule removal rate, 86.5% of the categorized errors fell into either pre-mission planning or communications; neither of which had error detection or failure recovery implemented. A similar condition existed at the 0.1% removal rate, where 95% of the errors that occurred fell into either pre-mission planning or communications domains (see Figure 2).

Using the above-described categorization of errors, the performance improvement afforded by the initial Recourse enhancement very nearly covered the entire range of errors encountered by the agents subjected to random rule removal. The data generated by the first phase of this study also seemed to indicate that the areas of greatest potential failure appear to be in pre-mission planning and communications, which became our next area of research.

During 2002, we built upon the generic infrastructure that had been established in 2001 and were able to quickly develop domain-specific self-monitoring and

recovery actions for both the pre-mission planning and communications domains. Furthermore, we enhanced the generic solution recovery system with a generic causal model back-chaining system. To test the causal model back-chaining system, we implemented a simple causal model for the flight domain. Finally, we were able to experiment with cognitive learning mechanisms and build a functional prototype of a diagnosis and blame-assignment learning system.

In the late part of 2002 and first quarter of 2003, we re-worked the testing regimen established in the first phase to eliminate some areas of concern with respect to the statistical significance of our performance metric gathering methods. Amongst these improvements was a change to the random generation of rule removals so that in a given trial batch of 50 paired runs, the particular rule removed per run for the non-Recourse and Recourse-enhanced versions of TacAir-Soar was identical. We believe that this strengthened the significance of the results captured in the second round of performance metric gathering that took place in 2003.

Incorporating the improvements developed in 2002, we proceeded to gather performance metrics in scenarios identical to those tested in 2001 (with the exceptions noted above). Recovery within coverage areas (which had expanded from 2001) was 100% across the board (see Figure 3). Furthermore, there was a fourfold increase in error recovery effectiveness for the Recourse-enabled TacAir-Soar agent from 2001 to 2003 for both the 0.1% and 1.0% removal cases. The breakdown of failure by domain was nearly identical to that observed in 2001, but the expanded domain coverage exhibited recovery in areas that were invariably fatal before. Given the 100% recovery demonstrated in domains with coverage, we believe that this improvement can be attributed to the newly implemented causal model and learning systems.

RELATED WORK

Brittleness has been a perennial problem with intelligent systems. It can be reduced by good software engineering principles, but not eliminated. Previous work to solve this problem has fallen into several different schools of thought.

Forbus and Falkenhainer (1992) proposed methods for using qualitative physics in simulations to describe quantitative knowledge. Their work focused on self-explanatory simulations. These mechanisms served as an inspiration for our system of building domain-specific qualitative models and using them to detect potential errors. Although qualitative reasoning in our architecture is used to differentiate between nominal and error states, the error-detection tests are dependent on a system of self-monitoring and the models are human behaviors, not physics models.

Qualitative physics (Forbus, 1996) attempts to capture the kinds of representations and reasoning techniques that humans use in dealing with the physical world. Rather than precisely modeling physical interactions, people seem very good at figuring out what is happening around them, accounting for a broader range of alternatives, and working with far less data than would be required by traditional methods. Qualitative techniques create representations for continuous aspects of the world, such as space, time, and quantity, which support reasoning with very little information. Our primary interest in these techniques is using these interval representations to provide alternative outcomes that may not have been foreseen by the subject matter experts in describing their response to a situation.

Temporal database theory defines two dimensions of time: transaction time and valid time (Snodgrass and Ahn, 1983). In (Jensen and Snodgrass, 1996) it is shown that these two dimensions are sufficient for answering a broad range of questions about when an event occurred and when it was recorded that an event occurred. We borrow from this theory and define two dimensions for agent reasoning.

The qualitative time intervals (Always, Past, Distant Past, Recent Past, Now, Near Future, Distant Future, Future) aid in writing human understandable inferences. It has been shown (Trafton, et. al., 2003) that humans are effective at thinking about complex problems qualitatively. We borrow from the research done in Qualitative Process Theory (Forbus, 1984) to structure our formalism around qualitative terms and reasoning, thus making it more human understandable. The exact number of qualitative bins may vary as required. Furthermore, the exact boundaries of these intervals can be modified to reflect the temporal scope of the inferences and the task to be accomplished.

Using the above time representation, we can reason about what is happening now, what happened in the past, what will happen in the future, and what was previously thought about.

Some of the design decisions for our recovery handler system were based on Michael Cox's dissertation work on recovery from reasoning failure (Cox, 1996). Specifically, Cox describes a variety of mechanisms for the application of introspective machine learning techniques. These were principally of interest to us for the design and development of our general recovery handling system, and for future work on the integration of learning into recovery.

At Yale University, Toyama and Hager (1997) were among the first to propose researching post-failure

robustness in the context of artificial intelligence. The philosophy behind our approach closely matches the case made by Toyama and Hager that the most effective general-purpose solutions to the problem of brittleness are to be found in recovery, not exhaustive prevention. Their perspective and other similar views led us to rely on nature's examples for appropriate general recovery actions.

The Cognitive Science and Integrative Behavior Biology research groups at Michigan State University have done extensive study of insect navigation (Dyer, et. al., 1993). Their discoveries, particularly with regard to redundant and hierarchical systems, have helped to guide the design of our general recovery handlers and generic solutions system.

CONCLUSIONS

This approach is not a panacea. There will still be a wide range of ways for the system to fail. Incorrect target coordinates may not be caught, low-level flights may still crash into the terrain, planes may circle endlessly waiting to land on a dead airfield, and aircraft will still get shot down.

This approach cannot guarantee correctness of results. It will not notice subtle errors, but it will detect grossly incorrect behaviors. However, these grossly incorrect behaviors are the same types of error most people notice and that they typically describe as the system behaving “stupidly.”

What is considered “grossly incorrect behavior” is domain specific. For example, in paleontology a difference of a few hundred years may not be significant, while for computer hardware a difference of a few nanoseconds could render a chip defective. By tying error detection to the specific goals the system is trying to solve, the system is able to use context dependent information to resolve problems for which it was not specifically programmed.

The basic principles described here are applicable to all complex software systems based on a hierarchy of goals. The key point is the approach of working to correct failure in real-time when it occurs, rather than trying to prevent all failure.

Failures cost time, money, and confidence in software. Other disadvantages associated with brittleness are increased cost spread over life-cycle, loss of time and training, expensive support, user frustration, lack of faith in results, and unacceptable results for mission-critical applications. The benefits of our approach are increased reliability, preservation of the investment, reduced life cycle cost, increased life cycle, and not holding up expensive assets. The benefits of human emulation are believability, avoiding negative training, recovery actions are comprehensible, and when it does break, it responds like a human. Applications include training, guidance systems, acquisition, mission rehearsal, strategic planning, deterrence, and UAV control.

We claim that our eight-step plan is sufficient to model recovery. These eight steps are plausible descriptions of human and biological recovery mechanisms. Our robust agents imitate this behavior.

ACKNOWLEDGEMENTS

This research was funded by the Office of Naval Research contract number N00014-00-C-0312 under the direction of Harold Hawkins. Thank you to Jennifer Marsman, Dr. Richard Frederiksen, Dr. James Beisaw, Richard Falk, Chris Kiekintveld, Nancy Shanley, and Glenn Taylor for their contributions to this research.

REFERENCES

Beard, J., Nielsen, P., Kiessel, J. Self-Aware Synthetic Forces: Improved Robustness Through Qualitative Reasoning. Proceedings of the I/ITSEC Conference, 2002.

Copeland, B. J.: "CYC: A Case Study in Ontological Engineering." http://ejap.louisiana.edu/EJAP/1997.spring/copeland97 6.2.html. 1997

Cox, M. Introspective Multistrategy Learning: Constructing a Learning Strategy Under Reasoning Failure. 1996.

Dyer, F.C., Berry, N.A., Richard, A.S. "Honey Bee Spatial Memory: Use of Route-Based Memories After Displacement." Animal Behaviour 45 : 1028-1030. 1993.

Forbus, K.D. "Qualitive Process Theory." In Artificial Intelligence, Vol 24. p.85-168. 1984.

Forbus, K.D., "Qualitative Reasoning." CRC Handbook of Computer Science and Engineering. CRC Press. 1996.

Forbus, K.D., Falkenhainer, B. "Self-explanatory Simulations: Integrating Qualitative and Quantitative Knowledge." Recent Advances in Qualitative Physics. B. Faltings and P. Struss eds. MIT Press. 1992.

Ginsberg, M. L. "Universal Planning: An (Almost) Universally Bad Idea." AI Magazine, vol. 10, no. 4, 1989.

Gödel, Kurt. "On Formally Undecidable Propositions of Principia Mathmatica and Related Systems", 1931. Available at: http://www.ddc.net/ygg/etext/godel/godel3.htm

Jensen, C. S. and Snodgrass R. T. "Semantics of Time-Varying Information" in Information System. 21:4. pp. 311-352. 1996

Jones, R. M., Laird, J. E., Nielsen, P. E., Coulter, K. J., Kenny, P., & Koss, F. V.: "Automated Intelligent Pilots for Combat Flight Simulation." AI Magazine, 20(1):27-41. 1999.

Kiessel, J., Nielsen, P., Beard, J. Failure Recovery: A Software Engineering Methodology for Robust Agents. Proceedings of the SELMAS Conference, 2002.

Laird, J.E., Newell, A., Rosenbloom, P.S.: "Soar: An Architecture for General Intelligence." Artificial Intelligence, 47, 289-325. 1991.

Lenat, D. B.: From 2001 to 2001: Common Sense and the Mind of HAL. http://www.cyc.com/halslegacy.html

Newell, A.: Unified Theories of Cognition, Harvard University Press, Cambridge, Massachusetts, 1990.

Nielsen, P., Beard, J., Kiessel, J., and Beisaw, J. Robust Behavior Modeling, Proceedings of the 11th CGF Conference, 2002.

Rosenbloom, P.S., Laird, J.E., Newell, A. (eds.). The Soar Papers: Research in Integrated Intelligence. The MIT Press: Cambridge, MA. 1993.

Snodgrass, R.T. and Ahn, I. "Temporal Databases" in IEEE Computer 19:9 pp. 35-42. 1986.

Toyama, K., Hager, G.D.: "If at First You Don't Succeed..." Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97), pp. 3-9. 1997.

Trafton, J., Kirschenbaum, S., Tsui, T., Miyamoto, R., Ballas, J., and Raymond, P. "Turning Pictures into Numbers: Extracting and Generating Information from Complex Visualizations" International Journal of Human Computer Studies. 2003.