SELF-AWARE SYNTHETIC FORCES: IMPROVED ROBUSTNESS THROUGH QUALITATIVE REASONING

Jonathan Beard
Paul Nielsen
Jennifer Kiessel
Soar Technology, Inc.
3600 Green Court, Suite 600
Ann Arbor, MI 48105
734-327-8000
{ beard, nielsen, jkiessel }@soartech.com

ABSTRACT

Much work has been done in the modeling and simulation community to capture expert knowledge as human behavior representations (HBRs) in developing behavior models for synthetic forces. A pervasive problem in the use of synthetic forces by the modeling and simulation community is their brittleness when exposed to massively complex simulation systems. Although synthetic-force behavior models may accurately encode a great deal of expert knowledge, the lack of broader commonsense knowledge frequently leads to suboptimal performance. When used for training purposes, these failures can lead to negative training and re-engineering downtime. When deployed in real-world situations, such as Uninhabited Aerial Vehicle (UAV) control, these failures can have far more serious and costly consequences.

This paper describes a methodology for imbuing synthetic forces with a capacity for commonsense reasoning through diagnosis of error symptoms detected during continuous self-monitoring. This methodology is part of the broader 'Recourse' architecture of robustness in behavior modeling. Behavior models are instrumented with a self-monitoring capability by using qualitative reasoning-based tests of domain-specific parameters. Failed tests are categorized as potential symptoms which can be diagnosed to suggest atomic effector-based recovery actions. This paper also describes some of the trade-offs and issues encountered during the design of the methodology and provides insight into how the methodology could be applied to behavior models in general. A prototype implementation of this methodology was applied to TacAir-Soar, a very large rule-based system that produces cognitively plausible behaviors of military aviators.

ABOUT THE AUTHORS

JONATHAN T. BEARD is a Software Engineer and the TacAir-Soar Product Manager at Soar Technology, Inc. He is currently involved in behavior-model and systems-interface research & development. Prior to his work at 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. He has over ten years of experience in the computer industry, including software development, systems and network administration, database design and administration, and professional consulting. His software development background includes enterprise-scale change-management systems, e-commerce solutions, research software prototyping and computer game development. His research interests are in artificial intelligence, wearable and embedded computing, linguistics, and cooperative autonomous agents. He has attended Hope College, the Kyrgyz State National University, and Northern Michigan University.

PAUL NIELSEN is a Vice President and one of the co-founders of Soar Technology, Inc. He received his Ph.D. in computer science from the University of Illinois in 1988. Prior to joining Soar Technology, he worked at the GE Corporate Research and Development Center and the University of Michigan Artificial Intelligence Laboratory.

JENNIFER KIESSEL is a Software Engineer at Soar Technology, Inc. She is a Soar behavior developer on the Robustness in Behavior Modeling (RBM), F/A-18 Part-Task Trainer, and Country Chaos projects. She holds a BSE in Computer Engineering (2000) and a MSE in Computer Science and Engineering, concentrating in artificial intelligence (2001), from the University of Michigan. Prior to joining Soar Technology, Inc., she worked for Ford Motor Company, National Instruments, and Microsoft. Her research interests include compilers, machine learning, and natural language processing.

SELF-AWARE SYNTHETIC FORCES: IMPROVED ROBUSTNESS THROUGH QUALITATIVE REASONING

Jonathan Beard
Paul Nielsen
Jennifer Kiessel
Soar Technology, Inc.
3600 Green Court, Suite 600
Ann Arbor, MI 48105
734-327-8000
{ beard, nielsen, jkiessel }@soartech.com

1. INTRODUCTION

1.1 Problem Statement

Errors are inevitable in any non-trivial software system. They arise from knowledge that is incomplete, inconsistent, incorrect, obsolete, or poorly defined. Their consequences are lost time, money, usability, and productivity. In critical applications, they may even lead to loss of life.

This paper presents an approach to error detection in complex human behavior models that is based on an abstraction of a world model. This approach is tractable for computation yet complete in its ability to represent all *relevant* world states.

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 world states that are 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,* our approach relies on a generic set of additional knowledge in the system that will detect errors. Once the agent realizes what it should not do, additional knowledge may be brought to bear that will alter the behavior until it is once again acceptable.

1.2 Background

There is a perception that human performance models are too brittle for a wide range of real world applications. They are difficult to test, opaque to inspection, and too often built to replicate a sample scenario rather than provide broad domain coverage. The scope of military endeavors is perceived as too vast, the domains too complex, and the environments too variable.

The Robustness in Behavior Modeling (RBM) Project was undertaken to address this situation and develop methodologies which demonstrate that a complex, real-world application can be made impervious to a range of error conditions. The techniques presented here will be applicable to any blackboard-like system that posts desired goals and waits for results.

The specific application we are using to demonstrate this methodology is TacAir-Soar, which generates entity level behaviors for most military aircraft and their combat missions [10]. TacAir-Soar is a highly complex application consisting of over 7500 rules describing navigation, coordination, communication, and tactics. It allows unsupervised agents to perform missions autonomously in simulation exercises.

1.3 Previous Work

In exploration of the brittleness problem, we have drawn on research and results of several different schools of thought. Some of these influences are software engineering best practices, such as methods of creating dependable and fault-tolerant systems described by J.C. Laprie [16]. We have relied on his research for a definition of the terms "dependability", "failure", "error", and "fault":

- Dependability: "Dependability is defined as that property of a computer system such that reliance can justifiably be placed on the service it delivers. (The service delivered by a system is its behavior as it is perceptible by its user(s); a user is another system (human or physical) which interacts with the former.)" Also includes attributes of dependability such as "availability,

reliability, safety, confidentiality, integrity, etc".

- Failure: "A system failure occurs when the delivered service deviates from fulfilling the system function, the latter being what the system is aimed at."

- Error: "An error is that part of the system state which is liable to lead to subsequent failure: an error affecting the service is an indication that a failure occurs or has occurred. The adjudged or hypothesized cause of an error is a fault."

- Fault: "The result of a programmer's error is a (dormant) fault in the written software (faulty instruction(s) or data); upon activation (invoking the component where the fault resides and triggering the faulty instruction, instruction sequence or data by an appropriate input pattern) the fault becomes active and produces an error; if and when the erroneous data affect the delivered service (in value and/or in timing of their delivery), a failure occurs."

Although the problem of brittleness can be reduced through the application of sound software engineering practices, for the bulk of our approach, however, we rely on imbuing software agents with a capacity for self-awareness and both pro- and re-active applications of that awareness.

Key to cultivating this capability for self-awareness is a means for the agent to internally represent its perception. We have chosen to use qualitative physics and qualitative reasoning as our means to that end. The survey efforts of Forbus and Falkenhainer [7] were instrumental in providing both introduction and guidance to the body of research.

Amongst the first to propose researching software engineering approaches to post-failure robustness in the context of artificial intelligence were Toyama and Hager at Yale University [21]. They both observed that there are many examples in nature of robust recovery by organisms of varying complexity.

Inspiration for our approach is drawn from the ability of biological systems to recover and adapt when faced with problems. Psychological research on executive dysfunction, a problem wherein the human sufferer is unable to attempt alternative corrective action in the face of failure and continues to perseverate in unsuccessful task execution, serves to shed light on similar behaviors in software systems [19]. By recognizing similar faults in software, we intend to apply the non-dysfunctional human mechanisms for problem resolution to software agents.

Subsequent to the detection of error, our approach emphasizes recovery as a robust solution to brittleness. Michael Cox explored this topic in his dissertation on recovery from reasoning failure and introspective machine learning techniques [5].

The University of Michigan AI Lab has also investigated the problem of detecting and responding to error. Their system, the Cooperative Intelligent Real-time Control Architecture (CIRCA), is designed to handle certain specific classes of unplanned-for world states [2]. CIRCA is dependent upon an external re-planning system to handle any detected errors. Because the re-planning system is only able to create plans for known problems, it remains primarily a preventative measure against failure. The approach described in this paper encompasses error detection, correction, effector-based novel solutions, and recovery.

The Information Sciences Institute at the University of Southern California has studied methods of error-detection and recovery for cooperative multi-agent systems, culminating in their STEAM architecture [11][12]. The STEAM architecture concentrates exclusively on problems relating to belief validity checking in the domain of social communications and cooperative action. In contrast, our work to date has focused on the internal workings of a single agent.

The Cognitive Science and Integrative Behavior Biology research groups at Michigan State University have done extensive study of insect navigation [6]. Their discoveries, particularly with regard to redundant and hierarchical systems, have helped to guide the design of our general recovery handlers and novel solutions system.

Finally, we feel that behavior moderators have a useful role to play in filtering information for both the error detection and failure recovery processes. Of particular interest in that regard was research by the Institute for Theoretical Psychology at the University of Bamberg. They developed a theory of emotional responses to meta-cognition and self-monitoring called PSI [3]. However, the PSI theory is focused almost exclusively on self-monitoring, without any formal system of error detection or recovery.

2. MOTIVATION

Brittleness is aptness to break. It increases with complexity and as the program is applied in domains for which it was not originally intended. We claim that brittleness in an intelligent system is a knowledge problem that arises out of ignorance (knowledge) rather than representation (architecture). There are many reasons to believe this claim, both theoretical [9] and practical [14], but one of the most compelling arguments is that biological systems make mistakes all the time. They eventually recognize these mistakes, recover, and learn from them, so that their performance improves continuously over time.

Biological systems recognize mistakes by exploiting additional deep knowledge about the domain. One form of this knowledge is an internal world model that is not as complex as the real world, but allows answering questions such as "What happens next?" "Is this consistent with my goals?", and "Am I making progress?"

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 apparent 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.

The benefits of our approach include increased reliability, preservation of investment, reduced life cycle cost, increased life cycle, and not holding up expensive assets. The benefits of human emulation are believability of the behavior, avoiding negative training, and comprehensibility.

3. APPROACH

Our approach to software recovery is structured as follows:

1. Represent the environment.
2. Recognize error.
3. Allocate resources to the problem.
4. Diagnose the possible error reasons.
5. Create alternative responses to error.
6. Initiate recovery actions.
7. Remember successful recovery actions.

This section will discuss qualitative representations, error detection methods, and recovery mechanisms of this *Recourse* [18] methodology.

3.1 Qualitative models for representation

For our approach to work, it is necessary to create internal abstract representations of the agent's state, which the agent then monitors to evaluate its own performance. Additionally, these representations are defined qualitatively to provide broad, complete domain coverage.

For example, for an agent concerned with spatial relationships, such as flight, the space is represented as regions, intervals, and ranges, rather than discrete points or values. These computational models are useful for both the characterization of agent states, and the derivation of appropriate responses to erroneous states. In that capacity, the models may also be regarded as predictive in that they permit causal-chain reasoning.

3.2 Monitor goal progress

Before recovery can occur, failure must be recognized. Furthermore, to recognize failure requires a persistent self monitoring, which constantly compares some representation of the agent's state with the agent's intended goals.

Using the qualitative models described above, we have implemented production rules that compare the state of those models with the current intended goals of the agent, and react to apparent discrepancies. This self monitoring is effected by qualitative evaluation functions based on the problem space representation which indicate failure, success, lack of success, or a continuing lack of success. Success or failure is measured in terms of whether or not goal conditions are met, while lack of success is the state of neither failure nor success. A measured continuing lack of success will, depending on operator-specific circumstances, eventually translate into a necessity for recovery action in a fashion almost identical to an evaluation of failure.

3.3 Detect failure and impending failure

3.3.1 Qualitative common-sense reasoning

The evaluation functions described above are based on a mapping of the problem space into a comprehensive qualitative representation. An example of this would be the "fly-racetrack" problem space in TacAir-Soar. A racetrack is a type of orbital flight pattern, focused on a particular flight waypoint. The aircraft flies an elongated oval, with one focus at the waypoint, and the other some distance away, in a direction towards the aircraft's base. The qualitative representation of this goal set is composed of multiple nested layers. The first of these is a binary

division of space into "close" to the point and "far" from it. As long as the aircraft is within a range from a point that is considered "close" enough, the evaluation will register non-failure. When the aircraft is clearly "far" from the point, the evaluation will register failure.

At another layer of evaluation, the aircraft should be shuttling between the two foci of the oval pattern. To evaluate this behavior, the state representation of the aircraft is compared over time. If the aircraft has a goal of flying the outbound leg of the racetrack flight pattern, the evaluation will test if the aircraft is, in fact, approaching that point over time. If it is, that will register as non-failure; if it is not, then the evaluation function will note failure and initiate recovery.

We have been developing generic evaluations with which any given operator can be easily augmented. These evaluations are dependent on abstract qualitative representations of each problem space, but can be applied to any problem space within their intended domain.

For example, we have implemented the following non-exhaustive set of qualitative tests in the spatial domain:

- within-range / beyond-range
- facing-toward / facing-away
- above / below
- moving-toward / moving-away
- turning-toward / turning-away

Each evaluation of the form described above assumes both a subject and an object and can be used as a test for truth or falsehood (i.e., "should" be within-range, or "should not" be within-range). In the case of TacAir-Soar, the subject is typically the aircraft itself, while the object varies among such things as waypoints, friendly or enemy aircraft, etc. In our implementation, as long as a problem space has a qualitative representation built, an operator need only be annotated with a single augmentation for the general evaluations described above to compare the intended goal to the qualitative model state.

Similar qualitative tests are made for each domain for which we have built models (communications, weapons, etc). In non-spatial domains, it is still possible to define possible ranges in values as falling into particular sets. For example, in the communications domain, the temporal dimension is the cognate of spatial dimensions for the flight domain. Qualitative tests take the form of "expect-reply", "do-not-expect-reply", etc.

3.3.2 Temporal evaluation

This is a direct outgrowth of the "continuing non-failure" evaluation function outcome. Although the non-failure registration of a particular evaluation function may generally indicate that the situation remains nominal, if the condition of non-failure persists for an unreasonable period of time, it can be determined that non-failure has become failure.

In that case, it is necessary to initiate recovery in a fashion similar to the actual detection of failure. The amount of elapsed time necessary to change an evaluation of non-failure into failure is problem space dependent. This essentially translates to additional derived qualitative tests for such things as "before", "after", etc. While the spatial tests in the flight domain may indicate that the agent is oriented in a spatially appropriate fashion for the particular problem space, the elapsed time may eventually indicate that a previously acceptable state of non-failure has become failure.

An example of this might be an aircraft engaged in the racetrack maneuver described above. Although it may well be within the range required to be "close" and appropriately "facing-toward" and "moving-toward" one of the foci of the racetrack, if enough time passes and the agent still has not achieved "success" in the racetrack, then the "before" temporal evaluation can indicate failure – at which point recovery is initiated.

3.4 Diagnosing the problem

Once failure has been detected, recovery is initiated. Although our effector-based recovery mechanism includes the capability for random search through all possible actions, we believe that a directed search can reach appropriate recovery actions faster than trial and error alone. In any case, if the directed search options themselves fail, trial-and-error search for recovery actions is initiated with full knowledge of the already attempted actions.

To give direction to our recovery mechanism, the set of actions to try first can be primed with a set of suggested actions. These suggested actions are proposed by a diagnosis. In our implementation of this approach, a diagnosis is the collection of a set of failed evaluation functions in conjunction with a set of possible recovery actions. An example might be an "off-course" diagnosis of the racetrack problem space. If a set of evaluations show that the aircraft is still airborne, but not within the "close" region of the racetrack waypoint, it may be diagnosed as being "off-course". In that case, the recovery action

suggested might be to "turn-toward" the waypoint, or to "give-up" on the goal.

3.4.1 Multiple diagnoses & specificity

Although the qualitative evaluations of status are local to each particular problem space, the detection of failure is registered globally. Diagnoses are made on the basis of these globally detected failures. For any particular set of failures, there can be an arbitrary set of diagnoses, of varying degrees of specificity.

In the above example, one diagnosis might be that the aircraft is "off-course". While that diagnosis is true, it might not take into account that the aircraft is also in pursuit of an enemy aircraft. Another diagnosis might account for that and explicitly suggest "postpone goal" as a recovery action for flying the racetrack, so that the interception can be completed.

Although multiple diagnoses of existing failures can co-exist, only one process of recovery can be in progress at any given time. In the case of multiple diagnoses, the most specific of the diagnoses proposed is the initiator of recovery.

3.4.2 Suggested Recovery Actions

When recovery actions are suggested by a diagnosis, they should be contextually appropriate. Much of this can be handled automatically by the types of qualitative evaluation being made. A corresponding recovery action for a failed test of "should be facing-toward" would be "turn-toward". Sometimes, it might be appropriate to suggest a non-obvious recovery action for a particular diagnosis. For example, in the above example, it might be more appropriate to "radio for help" given a particular diagnosis.

In a worst case scenario, a diagnosis might not make any specific suggestions for recovery action, but merely direct the trial-and-error process to begin within a particular domain of effector action (i.e., "flight"). In such a case, although the diagnosis might be at a loss to direct appropriate action, it has at least identified the recovery problem as falling within a particular domain.

3.5 Recovery

When a diagnosis is formed, it may provide recovery actions and it always provides a domain for the problem. Any recovery actions provided by the diagnosis are based on awareness of the context of the situation, that is, which symptoms are present in the system. These recovery actions provided by the diagnosis are called "known solutions", since they are associated with the diagnoses.

If all known solutions to a problem have been tried and the symptoms have not been alleviated, the agent will try "generic solutions". These recovery actions are general within a specific domain. For example, if the problem is determined by the diagnosis to have occurred in the radar domain, the agent may try such actions as changing the radar elevation, changing the radar azimuth, or switching radar modes. These general solutions allow the agent to experiment in its world and try new things, which may fix the problem. If all higher-priority generic solutions fail to alleviate the symptoms, the most high-level generic solutions will eventually be tried, which are to return to home base or to simply land the plane.

When designing recovery actions, we kept several core methodologies in mind: awareness of context, awareness of time, and the ability to manage multiple actions [13].

When a problem develops, an awareness of the context of the situation is key to an agent's recovery. Consider the example of a pilot flying a mission who encounters a threat ahead of him. If the threat is a surface-to-air (SAM) missile site, the pilot should turn away or he is at risk of being shot down. If the threat is an enemy fighter plane, the pilot should NOT turn his back; he should keep the enemy on his radar and face him to fire weapons. Therefore, the same or similar inputs to an agent (a blip on radar indicating a threat) could lead to very different behaviors as output, depending on context.

An awareness of time is also crucial for agent recovery. The agent must monitor the results of a recovery action and apply an alternative strategy if it is not making progress toward solving the problem. A key element of this is the knowledge of when to give up on a recovery action and try a new one, which necessitates an awareness of the passing of time.

Agents must also have the ability to try multiple recovery actions. We can specify a partial order on the recovery actions such that the most specific recovery action is performed before the most general.

4. APPROACH VALIDATION METHODS

Given that we chose to implement our proof-of-concept within TacAir-Soar, it was necessary to test the performance of that implementation to demonstrate empirical results. The particular version of TacAir-Soar used for our implementation and

testing was the 9.0 version, integrated with the Maritime Battle Center version of JointSAF 5.0 (JSAF). JSAF is a simulation system that generates entity level platforms, interactions, and behaviors in a synthetic natural environment [1].

Although there are a wide variety of mission types and parameters available with the versions of TacAir-Soar and JSAF used, our prototype implementation of the seven-step robust behavior modeling architecture currently implements only error detection and recovery for portions of the flight and weapons domains. To demonstrate the performance of the prototype would require large amounts of data generated under controlled conditions, with varying initial parameters to compare outcomes. Because there are a combinatorially large number of possible configurations to test, we created a reusable test harness for the TacAir-Soar/JSAF system.

An important innovation in the development of our reusable test harness was the creation of partial mission specifications. Each of these partial mission specifications could be parametrically combined at test runtime to create a fully functional JSAF exercise. By creating these modular mission definitions corresponding to major exercise design decision points, we drastically reduced the amount of time it would otherwise have taken to create all the JSAF exercises necessary to test current and future versions of our architecture. Additionally, the generality offered by the mission specification modules provided a framework broad enough to serve as a set of reproducible baselines for any regression testing in the foreseeable future.

To provide enough data for statistically significant results, it was resolved to run each test configuration for a total of fifty trials. On average, each of these 50 trial test-sets would take about 48 hours to run on an Athlon 1200MHz processor running RedHat 6.2 Linux. Several preliminary tests were run to discover if running the simulation at faster than real-time speed at all contaminated the results. At multiplication factors of up to four times, there was no appreciable degradation in performance. For all tests reported in this paper, the tests were run using a time multiplication factor of four, reducing each test's run time to 12 hours.

To test performance in the domains of flight and weapons, it was resolved to use one of the mission specification modules for a single flight domain mission parametrically modified in subsequent runs to include further variations. The mission selected was a Barrier Combat Air Patrol (BarCAP); a relatively simple mission where the agent is tasked with flying to a specific point, orbiting the point while intercepting any enemy aircraft, and then returning to base. It was deemed that this would be sufficient to exercise a significant portion of the basic flight capabilities of the TAS agent. With the addition of runtime configurations including a single opponent agent, this mission would also test basic weapons capabilities of the TAS agent.

Because TacAir-Soar is routinely used for training exercises and is actively maintained, any obvious behavioral errors are corrected through normal software engineering processes soon after they are detected. The purpose of the seven-step robust behavior modeling architecture is to automatically detect and recover from errors in complex software systems. Without nontrivial *a priori* knowledge, we are unable to reliably and preemptively detect errors. To overcome this potential difficulty in testing the efficacy of our architecture, we emulated incorrect knowledge by purposely introducing errors through randomly removing portions of the code.

5. APPLICATIONS

The methodologies of Robust Behavior Modeling are appropriate to a wide variety of applications. Intelligent human behavior models have been applied in domains such as training, guidance systems, acquisition, mission rehearsal, strategic planning, deterrence, and vehicle control. Each of these domains requires behavior performance indistinguishable, at some level of fidelity, from human performance. For example, strategic planning requires that overall routing and timing be realistic while training requires interaction with the human trainee in close proximity.

Human behavior models with robust behaviors will exhibit failures less frequently, and thus reduce negative training and false results for military simulations. For live mission rehearsal where expensive assets such as aircraft carriers are used, agents that fix their own mistakes are a benefit because system downtime is lowered, resources are conserved, and cost is reduced. Finally, for vehicle applications such as UAV control, error detection is absolutely mandatory. UAVs interact with the real world, and the cost of these failures could result in loss of assets or loss of life.

6. LIMITATIONS

This approach cannot guarantee correctness of results. It will not notice subtle errors, but it will detect grossly incorrect behaviors. When the system

fails, this is the type of error most people notice and say that the system is behaving "stupidly."

What is considered "grossly incorrect behavior" is domain-specific. For example, in archaeology 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 handle these extremes.

Error detection may result in false positives. The corresponding human metaphor is a counter-intuitive process. One approach is to allow a recovery action that waits to see if the problem goes away.

7. FUTURE WORK

Currently we are working to automate the detection of many classes of failures. We have developed coding templates that allow a higher-level specification of error detection, diagnosis, and recovery for errors that occur independently.

Currently error detection has only been implemented in the flight and weapons domains. We will expand this coverage to the communications and mission-planning domains. Preliminary testing, using random rule removal, shows that over half of the errors that occur in planning cannot be corrected by later application of knowledge.

We plan to implement failure anticipation through a look-ahead search. Detecting errors that have already occurred may be too late. Instead, the system must anticipate impending failure and take action while it still can, but performing this look-ahead can lead to resource limitations, since it is an expensive process.

8. CONCLUSIONS

We have addressed why brittleness in software is a problem. We have provided a seven-step architecture for robustness in behavior modeling. Finally, we have shown that our implementation of this architecture in the TacAir-Soar system improved the performance of the system.

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.

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

8.1 Implementation Costs

TacAir-Soar is a large rule-based system [4], comprised of approximately 7500 individual production rules. Within those rules, there are approximately 750 operators. The current implementation of our approach could be generally described as annotating the operators or decision points of a system. The 750 operators in TacAir-Soar fall into about five general areas: pre-mission planning, flight, communications, weapons, and sensors.

For the version of our approach described in this paper, we have implemented coverage of the flight and weapons domains, with partial coverage of sensors and weapons. Implementation of pre-mission planning coverage is currently underway. As implemented, our system is presently on the order of 100 rules – it is projected that full coverage of TacAir-Soar would not more than double that count. Hence, using a 7500-rule total and a solution implementation of 200 rules, an approximation for estimating the cost of implementing this solution would seem to be an additional development overhead of 2.5%.

In actuality, we are working on generalizing the implementation of our approach, inclusive of language-independent first-order logic representations of our self-monitoring and response behaviors. Ideally, much would be re-usable in future implementations, particularly those with overlapping or closely related performance domains (mission planning, three or six "degree-of-freedom" (DOF) flight, communications, weapons, and sensors).

8.2 Performance Considerations

Given that the solution is implemented as additional rules within the system, focused on dealing generically with unplanned-for knowledge problems, these new rules rely equally on the underlying architecture at runtime. The Soar language [15][17] implementation in our testing environment is Soar7. Ever since Soar6, the Rete matching algorithm has given the Soar language runtime performance independent of the number of rules in the system. This capability of the Rete algorithm was demonstrated by Charles Forgy at Carnegie Melon in 1979 [8].

Thus, the only performance drawback to additional rules in a system implementing the Rete algorithm is

the additional memory overhead. However, given the comparatively small number of rules necessary to implement our approach (currently 100 rules in an 7500-rule system, or 1.25%), the additional memory required is relatively insignificant.

8.3 Expected gains

The payoff of building human behavior systems that recognizes their own errors is realized in terms of reliability, time, understandability, and cost. High reliability is critical for fielded applications that aid in decision making affecting the lives of troops. Increased reliability means less down time so results are obtained quickly. When the system does fail, the reasons are clear and comprehensible, so it can be repaired easily or repair itself.

Finally, the cost savings are enormous. Software engineering estimates [20] indicate that errors in the field can cost up to one hundred times as much to repair as those detected during development. In this domain, the costs can be even greater, since many applications, once installed, are inaccessible.

9. ACKNOWLEDGEMENTS

The authors wish to thank all of their reviewers for invaluable assistance and guidance. Finally, we would like to gratefully acknowledge Harold Hawkins and the Office of Naval Research for funding our research under contract number N00014-00-C-0312.

10. REFERENCES

[1] "Maritime Battle Center Joint Semi Automated Forces Information Paper" MAGTF Staff Training Program, Modeling & Simulations. http://www.mstp.quantico.usmc.mil/modssm2/InfoPapers/INFOPAPER JSAF.htm

[2] Atkins, E. M., Durfee, E. H., and Shin, K. G. Detecting and Reacting to Unplanned-for World States, *Proceedings of the Fourteenth National Conference on Artificial Intelligence* (AAAI-97), 1997, pp. 571-576.

[3] Bartl, C.: "Metacognition in Complex Problem Solving." http://www.uni-bamberg.de/ppp/insttheopsy/psi-literatur.html

[4] Bickhard, M. H. Physical Symbol Systems, http://www.lehigh.edu/~mhb0/aiforcogs7.html

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

[6] 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.

[7] 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.

[8] Forgy, C.L. "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem", *Artificial Intelligence*, 19, pp 17-37, 1982.

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

[10] 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.

[11] Kaminka, G. A. and Tambe M. What is Wrong With Us? Improving Robustness Through Social Diagnosis, *Proceedings of the Fifteenth National Conference on Artificial Intelligence* (AAAI-98), 1998.

[12] Kaminka, G.A., Tambe, M.: "Social comparison for failure detection and recovery." Intelligent Agents IV, Lecture Notes in Artificial Intelligence, 127-141. Springer Verlag, 1998.

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

[14] Klein, M. and Dellarocas, C. Exception Handling in Agent Systems, *Proceedings of the Third International Conference on Autonomous Agents*, Seattle, WA, May 1-5, 1999.

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

[16] Laprie, J.C. (1995) Dependable computing: concepts, limits, challenges. In Proc. 25th IEEE Int. Symp. Fault-Tolerant Computing - Special Issue, Pasadena, CA, pp. 42-54. IEEE.

[17] Newell, A.: *Unified Theories of Cognition*, Harvard University Press, Cambridge, Massachusetts, 1990.

[18] Nielsen, P., Beard, J., Kiessel, J., and Beisaw, J. Robust Behavior Modeling, *Proceedings of the CGF Conference*, 2002.

[19] Shallice T., Burgess P.: "Higher-order cognitive impairments and frontal lobe lesions in man."

Frontal Lobe Function and Dysfunction, edited by Levin H, Eisenberg H, and Benton A. New York, NY: Oxford University Press; 1991.

[20] Tassey, G. "The Economic Impacts of Inadequate Infrastructure for Software Testing". NIST, Gaithersburg, Maryland, 2002.

[21] 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.