Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Security and Defense Applications (CISDA 2007)

Improved Missile Route Planning and Targeting using Game-Based Computational Intelligence

Ken Doris
Applied Visions, Inc.
Northport, NY 11768
631-754-4920
kend@avi.com

David Silvia
Naval Undersea Warfare Center DivNpt
Newport, RI 02841
401-832-2869
silviada@npt.nuwc.navy.mil

Abstract: This paper discusses a research project that employs Computational Intelligence (CI) to improve the ability of military planners to route sensors and weapons to effectively engage mobile targets. Future target motion is predicted through the use of multiple software agents employing Goal Oriented Action Planning (GOAP). Derived from the Stanford Research Institute Planning System (STRIPS), GOAP is a relatively new class of CI that is ideally suited to dynamic real-time environments such as military operations. The project is unusual in its adaptation of computer gaming industry technology for use in real-time, tactical military applications.

I. Introduction

The efforts described within this paper are the results of an on-going Small Business Innovation Research (SBIR) grant between the Navy and Applied Visions, Inc. of Northport, NY. The SBIR Topic, now entering its 2nd year of Phase II funding, is called *Display and Visualization of Movement Predictions for Ground Vehicles* and is managed by the Naval Undersea Warfare Center (NUWC) Newport, RI. Its primary focus is the Tactical Tomahawk missile and the associated control system, though the research and its application must be extensible to future weapon systems.

The Tomahawk missile is a long range, subsonic cruise missile, launched from U. S. Tomahawk missiles, used for land attack warfare, are designed to fly at extremely low altitudes at high subsonic speeds and are piloted over an evasive route by several mission-tailored guidance systems. There are two variants of the Tomahawk Missile currently deployed, the Block III and the Block IV. Both feature an Inertial Navigation System (INS) aided by Terrain Contour Matching (TERCOM) for missile navigation and a Digital Scene Matching Area Correlation (DSMAC) and Global Positioning Satellite (GPS) System, which are coupled to the guidance systems to provide precision navigation.

The Tomahawk missile has become the weapon of choice for the U.S. Department of Defense because of its long range, lethality, and extreme accuracy. They are used against high-priority, long-dwell targets whose priority does not change during the missile's transit time [1]. However, the Tomahawk has limited effectiveness against short-dwell or Time Sensitive Targets (TST), and has never been used against mobile, high-value targets such as mobile missile launchers. These targets present special challenges for the weapon system because the missile cannot be retargeted quickly. Furthermore, the Tomahawk missile has limited endurance, increasing the likelihood that it will run out of fuel before new target solutions can be determined [2].The Block IV, or Tactical Tomahawk, is the latest in the evolution of the Tomahawk weapon system. It adds the capability to reprogram the missile in-flight to either strike preprogrammed alternate targets or to redirect the missile to newly found targets of opportunity. The missile is also able to loiter over a given area, either using its on-board camera to assess the target, or operating in a stand-off mode, waiting for a target to enter a kill window, perhaps some distance away. Figure 1 below illustrates some of these capabilities.

The ability to reprogram the missile in-flight is provided by the Tactical Tomahawk Weapon Control System (TTWCS). The success of TTWCS will depend on how well a weapons officer can decide *which* missile should be assigned to *what* target and *when* it should carry out its attack. To make such decisions the warfighter requires timely, actionable information that enables effective use of existing weapon and sensor systems, as well as provides an understanding of the spatial relationships of the weapon, sensor, target, and the battlespace. The goal of this research project is to provide the TTWCS operator with the technology to achieve that understanding

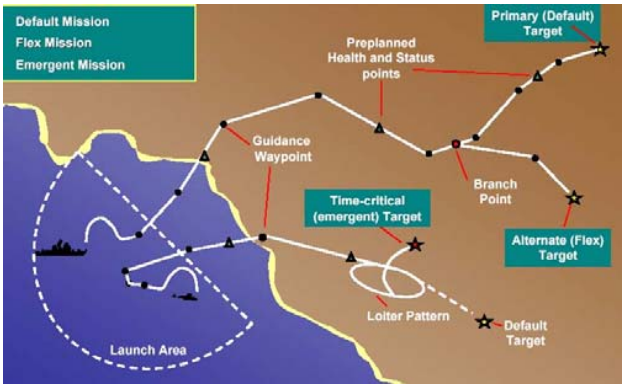

Figure 1 – Tactical Tomahawk Missions

III. PROBLEM DOMAIN

Currently, the TTWCS operator does not have the ability to fully utilize the features of the Tactical Tomahawk against an expanded target set that includes TSTs and high-value mobile assets. This is because the current level of Situational Assessment (SA) is insufficient to support the operator's ability to respond to these emerging targets in a timely fashion. This lack of SA results in longer response times to requests for weapon strikes; the sequence of events from request to response is known as the "kill chain". The inability to strike a target in a timely manner is not unique to the Tomahawk missile. Therefore, it is important that solutions to address this deficiency be extensible to route sensors, Unmanned Aerial Vehicles (UAV), Unmanned Combat Aerial Vehicles (UCAV), and future high-speed weapons, as well as any future weapon whose objective is to engage TSTs and mobile targets.

In order to attack TSTs, it is necessary to shorten the kill chain time. To reduce the response times it is necessary to provide the operator with the correct level of actionable information so that weapon/sensor selection and route planning decisions can be made in a timely manner. To address this problem a capability is required that supports the real-time comprehension of the battlespace in order to engage TSTs and moving targets. The operator must have the ability to visualize the spatial relationships between the sensors used to detect the target, the weapon employed against the target, threats to the weapon, the movement of the target, and the battlespace itself, which includes the terrain and weather conditions. Therefore, the proposed solution must provide the ability to conflate target movement and visualization knowledge and associate this information to land-based targets and sensor/weapon requirements in real-time. The solution must consider the target's capabilities and operating terrain, the position and capabilities of the sensors, and apply motion analysis techniques to quickly locate/relocate mobile targets, as well as predict the future movements in a rapidly changing battlespace.

IV. APPROACH

The goal of this project is to develop a cost-effective, yet high-performance capability that provides the warfighter the correct level of SA necessary to engage mobile and TSTs. While developing an entirely new system for this purpose is certainly possible, it would also represent a high-risk, high-cost solution. The most efficient approach is to adapt existing technology from the military or commercial marketplace. Since the technology found in existing military systems typically lags the commercial market by a decade or more, our research centered on the latter. We were somewhat surprised to discover that the *computer game* industry could provide what we were looking for in the form of a "Game Engine". A Game Engine is the core software component of a computer game that uses real-time graphics. The Game Engine itself is a middleware that provides a level of abstraction between the hardware and the application. It provides the underlying technologies for commercially produced computer games, simplifies development, and includes a rendering engine for graphics, a physics engine for vehicle dynamics and collision detection, a computational intelligence subsystem to control the non-player characters, a sound engine for aural effects, scripting, animation, and networking capabilities. Game Engine technology is driven by a huge market of consumers and the technology continues to improve each year. Commercially available Game Engines are well-documented, open, modular products that combine high-performance visual rendering, sophisticated real-world physics and vehicle dynamics, as well as the reliability that our product requires. In addition, due to its modular design, it is possible to easily upgrade the Game Engine, taking advantage of new features as they are introduced. Although, Game Engines are not typically used in tactical applications, their capabilities and maturity are a natural fit for the requirements of this problem domain,

The problem of predicting vehicle movement can be broken down logically into three essential parts: *prior history, current state,* and *future goals*.. These correspond to the following technology areas:

- ***Computational Intelligence*** – "memory" of prior events, "sensing" current status and events, and decision making, all resulting in evaluation of holding position or moving to a new destination.
- ***Pathfinding*** – given a decision to move to a new destination, what routes would the ground vehicle take and what are the relative benefits and risks of each path?
- ***Vehicle Physics*** – how quickly can the vehicle move along each route to reach the potential destinations?
- ***Visualization*** – provide the operator with a user interface (UI) that allows intuitive interaction and rapidly increases situational awareness.

All relevant elements of the battlespace and their relationships must be presented by the system and understood by the operator. In its simplest form, the battlespace will contain a target, a weapon, a sensor, the terrain, and any threats to the weapon and/or sensor; all of which interacting with each other at some level. That is, it must be possible to predict the movement of a target within a specified terrain, optimize the use of sensors (most likely UAV) by employing efficient search routes, develop efficient weapon mission plans, incorporate threat avoidance into sensor and weapon routes, and determine the time-on-target distance vs. time relationships, all of which needs to be presented to operator in a comprehensible visual package. These tasks and relationships, though complex, are within the capabilities of modern game engines. In particular, the ability extend the computational intelligence incorporated in the game engine makes it an ideal solution for target movement prediction requirements.

In the Phase I of the project, our research looked at several types of game-based AI as potential candidates for the final product. We finally selected a product called *AI.implant* as our Phase I choice as it combined excellent functionality with the

availability of a free development license, good documentation and an active user community.

AI.implant's development environment (AI.DE) includes an editor that allows creation and modification of agents, including a full definition of the "brain" of the agent. This brain consists of the following elements:

- Knowledge of one's own capabilities and performance parameters
- Knowledge of prior events (world and self history)
- Knowledge of current state
  - Internal status, such as fuel, speed, etc.
  - External status, such as proximity to threats, gained through the use of user-defined sensors
- Decision Logic – given the current state and prior history, what action, if any, to take next.

All of these elements are completely customizable without writing any code, thus putting the design of the "brain" directly into the hands of an *Analyst* without needing the support of a *Programmer*. Figure 4 contains a screenshot taken from the *AI.mplant* development environment showing the development of an agent.

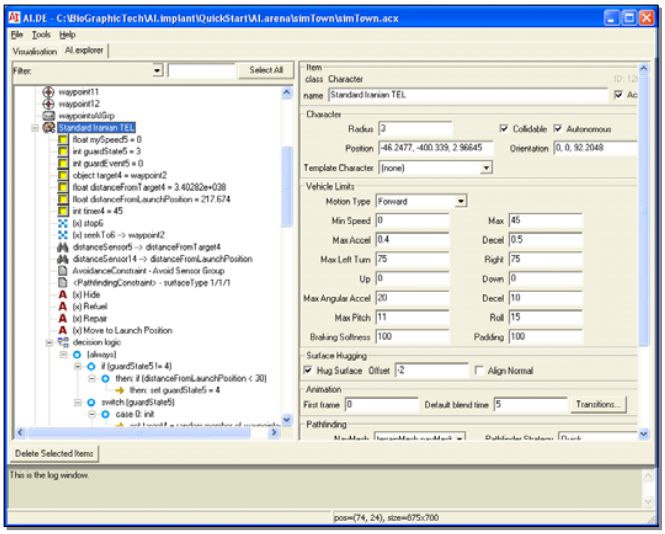

Figure 1 – Designing a "brain" in AI.DE

The version of *AI.implant* we used allowed two types of decision logic: *Decision Trees* and *Finite State Machines (FSMs).* In Phase I we experimented with Finite State Machines as the primary methodology. While this approach was certainly sufficient for the Phase I Prototype, we realized that further exploration in Phase II was necessary

In Phase II of the project, we entered into a Cooperative Research and Development Agreement (CRADA) with the Naval Postgraduate School (NPS), located in Monterrey, CA, to assist us in surveying the current state of AI as it applied to the computer gaming market. After reviewing the latest methodologies, NPS suggested we evaluate Goal Oriented Action Planning (GOAP) [3][4] for our application. GOAP, because it was developed to handle real-time game action, is well suited to dynamic environments such as military operations. It evolved from earlier cognitive systems, such as GOMS (Goals, Operators, Methods and Selections) [5], with major influence from work done at Stanford University on the Stanford Research Institute Problem Solver (STRIPS) [6]. STRIPS consists of goals and actions, where goals describe some desired state of the world, and actions are defined in terms of preconditions and effects. An action may only execute if all of its preconditions are met, and each action changes the state of the world in some way.

The overall logical execution flow of a GOAP system is relatively simple. At a given point in time, each agent has a set of goals that need to be achieved, and tries to satisfy the goal or goals that are most relevant for the current situation. For a given goal, the logic regressively searches for actions that have an effect that matches the goal. For each matching action, the logic looks at its preconditions, determines if they are already satisfied, and if not, performs another regressive search to find actions that have effects that that match the precondition of the previously selected action. Using the classic A* search technique, the regressive searching continues until it finds a path from the end goal all the way back to the current world state. The agent then begins executing the actions, rechecking the validity of the path and end goal at each step to accommodate for dynamic changes in the environment. Several interesting and useful concepts are embodied in this architecture which we feel make it a prime candidate for our research:

- For a given goal, there may be multiple paths. The system can be designed to either simply pick the first path that works, or a weighting system can be applied to individual actions, thus providing the mechanism for searching for the lowest cost path. This can be employed to find the lowest risk plans for achieving military goals.
- There is no explicit mapping between goals and actions, thus allowing for dynamic resolution of unexpected conditions such as weather changes.
- The GOAP architecture lends itself to a separation of implementation and data. This separation of the coding from the data allows non-programmers the ability to create or modify behaviors, an extremely important attribute for the future deployment of this application.
- Regressively searching for plans in real-time affords opportunities to *learn* and find multiple solutions to problems
- Atomic goals and actions of a GOAP system are easy to read and maintain, and can be sequenced and layered to create complex behaviors.
- A GOAP system imposes a modular architecture that facilitates sharing behaviors among agents and even across software projects.

- The GOAP architecture can be defined using the Planning Domain Definition Language (PDDL), a machine-parsable standardized syntax used widely throughout the AI planning community. Also, translators exist between PDDL and DARPA Agent Markup Language (DAML) [7].

V. IMPLEMENTATION

Having chosen GOAP as the preferred AI methodology, we are now pursuing two parallel efforts for incorporating it into our project. The first effort is that of implementing the TTWCS "GOAP Engine" - our own version of the planner - and integrating it into the overall game engine architecture. For that portion of the work, we have entered into a Cooperative Research and Development Agreement (CRADA) with the Naval Postgraduate School (NPS), located in Monterrey, CA. That effort is now underway at NPS with an initial version expected to be available by the end of 2006.

The second effort is that of developing the planning logic that will be processed by the GOAP Engine. The planning logic needs to encompass a fairly large number of scenarios, such as the following

Scenario: Target Vehicle to Missile Launch at Remote Location Scenario

Goal: Launch missile from "launch basket" – an area at some distance from current position that is within range of one or more selected targets for the missile.

Preconditions for launch:

- Within "launch basket"
- Weapon system ready
- Sufficient fuel to escape after launch
- Weather conditions limit probable detection (e.g. night, cloud cover, etc)

In the simplest case, with the assumption that no opposing action will be taken against the target vehicle, the following steps must be followed to achieve the goal:

1. Call a *pathfinding* function to determine the best route to the launch basket.
2. Traverse that route, segment by segment, until the launch basket is reached.
3. Ready the weapon
4. Launch the weapon.

In a more realistic scenario, the Scud launcher will need to account for possible opposition. It may have to alter its route to the launch basket based on dynamic conditions affecting its visibility/vulnerability to enemy sensors and weapon platforms. To check for these conditions, an additional function is called while traversing the route to check for the safety of the current position. The logic now expands to the following

1. Call a Pathfinding function to determine the best route to the launch basket.
2. Traverse that route, segment by segment until launch basket is reached. In each iteration perform the following
    a. Call a *Visibility* function to check if it is safe to continue on the current route.
        i. If Safe, then continue with current route and destination
        ii. If Unsafe, then call *Pathfinding* to find route to closest hiding area and continue to call *Visibility* function until no longer visible, then call Pathfinding to calculate new route to launch basket
    b. Repeat 2.a.
3. Ready the weapon
4. Launch the weapon.

When other factors, such as the launcher's fuel consumption and weather conditions are added to the scenario, the complexity of the problem increases significantly. We are exploring ways to express the problem space in a format that can be used by system analysts and subject matter experts (SME's). We began by looking at the potential use of the Program Domain Descriptor Language (PDDL) [8]. A PDDL definition consists of two parts: The *domain* and the *problem* definition. A *domain* definition contains the properties of objects in the world of interest (*predicates*) and operators (a.k.a. *actions*). A simple domain definition is formatted as follows:

```
define (domain DOMAIN_NAME)
  (:requirements     [:strips]     [:equality]
[:typing])
  (:predicates (PREDICATE_1_NAME [?A1 ?A2 ...
?AN])
               (PREDICATE_2_NAME [?A1 ?A2 ...
?AN])
         ...)
  (:action ACTION_1_NAME
    [:parameters (?P1 ?P2 ... ?PN)]
    [:precondition PRECOND_FORMULA]
    [:effect EFFECT_FORMULA]
   )
  (:action ACTION_2_NAME
    ...)
  ...)
```

A *problem* definition contains an initial world state and a desired state of the world in terms of a *goal*. The PDDL for a simple problem is formatted as follows::

```
(define (problem PROBLEM_NAME)
  (:domain DOMAIN_NAME)
  (:objects OBJ1 OBJ2 ... OBJ_N)
  (:init ATOM1 ATOM2 ... ATOM_N)
  (:goal CONDITION_FORMULA)
  )
```

Using our sample scenario, we began the translation into PDDL, as illustrated in Figures 2 and 3. While the PDDL was fairly manageable at first, we soon realized that we would need to extend the PDDL to handle complex pathfinding, fuel consumption and dynamic weather conditions. While other researchers had done comparable extensions [9], the specific nature of our problem space would undoubtedly require significant additional work. This, coupled with the desire to put the construction of the logic into the hands of non-programmers has led us to abandon the PDDL approach. Our plans now include the development of a planning definition tool (PDT) that will provide a deployable environment to de-

fine the world state variables, available actions, goals and associations.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Simple  Missile Launcher domain
;;;

(define (domain Mobile_Missile_Launcher)
      (:requirements :strips :equality :typing)
      (types (vehicle weapon place road fuelstation )
      (:predicates       (road ?from - place ?to - place)
                         (at ?veh - vehicle ?p - place)
                         (loaded ?wpn - weapon ?veh - vehicle)
                         (launchsite ?ls - place)
                         (missilebase ?base - place)
                         (fueldepot ?fd - fuelstation)
                         (observable ?obsv - vehicle)
      );end predicates
      ; Need to update this to include fuel usage
      (:action Travel
           :parameters (?veh ?from ?to)
           :precondition (and (road ?from ?to)
                          (at ?veh ?from)
                          (not (= ?from ?to))
                          (not(?obsv))
           :effect (and (at ?veh ?to) (not (at ?veh ?from)))
      );end Travel
      (:action LoadWeapon
           :parameters (?weapon ?place ?vehicle)
           :precondition (and (weapon ?weapon)
                              (vehicle ?vehicle)
```

Figure 2 - PDDL Domain Definition Code

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Simple Missile Launcher Problem Defintion
;;; Launch a missile by travelling from Base to
;;; Launchsite position
;;;
(define (problem Launch_Missile)
      (:domain Mobile_Missile_Launcher)
      (:objects a b c d e f g Culvert MM _Launcher Missile)
      (:goal (weaponlaunched Missile))
      (:init
            (at MM_Launcher Culvert)
            (at Missile Base)
            (road Culvert b) (road b Culvert)
            (road a e) (road e a)
            (road e b) (road b e)
            (road a c) (road c a)
            (road c b) (road b c)
            (road d f) (road f d)
            (road f g) (road g f)
            (road d Launch_Site) (road Launch_Site d)
```

Figure 3 PDDL Problem Definition Code

SUMMARY

Computational Intelligence, adapted from the computer gaming industry, will be used to assist U.S Navy personnel in SA of the battlespace and to optimize route planning for the Tomahawk missile and associated sensor platforms. The technology represents a leap-ahead in SA over the levels currently provided to operators. The approach not only supports the engagement of TSTs and mobile targets through complete comprehension of the relationships of the entities contained in the battlespace, it also provides the ability to predict and visualize future changes in the relationship of the entities. An operator can not only comprehend the current battlespace, he can also visualize what that battlespace will look like when the weapon arrives.

AUTHORS

Ken Doris is the Vice President of Engineering at Applied Visions, Inc. in Northport, NY. He serves as the Principal Investigator on the Navy SBIR project described in this paper as well an on-going Army SBIR that also uses gaming technology for battlefield analysis and visualization. One of the authors of IEEE 1278, Ken has published numerous technical papers on subjects such as network traffic simulation, 3D visualization, Command and Control, and computer game technology. He received his Bachelor of Electrical Engineering degree from Rensselaer Polytechnic Institute.

David Silvia is an Engineer with the Naval Undersea Warfare Center (NUWC) in Newport, Rhode Island. He currently serves as a liaison for the Tomahawk Weapon System (TWS) Advanced Concept Working Group (ACWG) participating in technical exchanges between Naval Surface Warfare Center, Johns Hopkins Applied Physics Laboratory, Lockheed Martin Corporation, and NUWC. In addition, Mr. Silvia conducts technical evaluations, surveys new candidate technologies, and develops prototypes to address future requirements for the TWS. His past work for the Navy has included the development of applications in speech recognition, distributed database design, and peer-to-peer architectures. Mr. Silvia has over 20 years of experience in software development and engineering. He has a Bachelors Degree in Engineering from Roger Williams University in Bristol, Rhode Island. He has worked for the Navy for six years and lives in Massachusetts.

REFERENCES

[1] *United States Navy Fact File: Tomahawk Cruise Missile.* 11 Aug. 2003. Office of U.S. Navy Information. 4 Feb. 2004 <http://www.chinfo.navy.mil/navpalib/factfile/missiles/wep-toma.html >.

[2] Morrow, Capt. Steve. "What Comes After Tomahawk*?"* Naval Institute Proceedings, July, 2003.

[3]Orkin, J. (2004), "Applying Goal-Oriented Action Planning to Games*",* AI Game Programming Wisdom, 2nd Edition, Hingham, MA, Charles River Media, Inc.

[4] Orkin, J. (2005), "Agent Architecture Considerations for Real-Time Planning in Games", 2005 Artificial Intelligence and Interactive Digital Entertainment (AIIDE) Conference Proceedings.

[5] Card, S., Moran, T., and Newell, A. (1983). "The psychology of human-computer interaction". Hillsdale, NJ: L. Erlbaum.

[6] Nilsson, J. (1998), *STRIPS Planning Systems,* Artificial Intelligence: A New Synthesis, Pg 373-400, Morgan Kaufmann Publishers, Inc.

[7] See www.daml.org

[8] An excellent description of PDDL can be found at http://www.ida.liu.se/~TDDA13/labbar/planning/2003/writing.html

[9] llghami, O. and Murdock, J., "An Extension to PDDL: Actions with Embedded Code Calls", paper presented at the International Conference on Automated Planning and Scheduling (ICAPS), June, 2005