PREFACE

The Warfighter's Associate (WA) is an implementation of the Observe-Orient-Decide-Act (OODA) loop, a process soldiers and organizations perform continuously as they develop situation awareness and situational understanding, decide on action, and re-assess the situation to determine the next steps. The WA's knowledge base captures how military experts use the OODA loop in combat situations. The WA uses this knowledge to provide context-sensitive tips and cues, enabling inexperienced and / or stressed warfighters to function closer to the level of their more experienced counterparts. Its development is consistent with Artificial Intelligence software being developed in commercial settings.

If we examine applications such as PAL – the predecessor of Apple's SIRI – we would find not one application but rather an assemblage of components tailored for functions such as task, time and workflow management, machine learning, and document classification. IBM used a similar strategy for implementing Watson, the question-answering system that gained fame on Jeopardy. Based on work performed for Watson, IBM released an open-source version of its Unstructured Information Management Architecture (UIMA). The Apache foundation describes UIMA not as a single application but rather as an "industrial-strength, scale-able, and extensible platform."[1] UIMA is a *framework* for building applications like Watson.

This paper describes Velox, the *framework* Veloxiti has developed with CERDEC funding for implementing intelligent systems. This framework – the platform used for implementing the Warfighter's Associate – is *uniquely suited* for building intelligent systems for military use. Just as UIMA is a framework for building Watson-like applications, Velox is a framework for building applications that leverage the OODA-loop.

Some parts of the following discussion are high-level; other parts are technically detailed. Here is a summary of key points:

1. Our military heritage includes large programs such as the Pilot's Associate and efforts in the rotorcraft and UAV domains[2].
2. Our approach has been strongly influenced by John Boyd (the OODA loop), Michael Bratman (the originator of the Belief-Desires-Intention software model), and AI pioneers such as John McCarthy.
3. Velox is **not** a rule-based system and has little in common with rule-based systems.
4. Velox is also **not** a machine-learning system, but integrating with nearly any external AI component is possible – almost trivial, in fact. Thus, Velox can serve as the integration platform for many different AI tools and techniques.
5. Velox is a Java-based framework which includes an Eclipse-based development environment and a high-performance cognitive engine for executing knowledge.

[1] "UIMA Overview & SDK Setup." Version 2.6.0, p. 17. See https://uima.apache.org/documentation.html

[2] Both the Pilot's Associate and Rotorcraft Pilot's Associate were funded for hundreds of millions of dollars.

6. The Data Analysis Tool (DAT) is an application for gathering human performance data during manned system experiments.

HISTORY: TECHNOLOGY EVOLUTION

The history and evolution of the technology development dates back 27 years. The Pilot's Associate (PA) Program (1987-1992) was a $200 million effort to create a new model for human-centered intelligent systems, built from the start to cooperate and support their human user. All such systems are now known as "Associates". The program was sponsored by DARPA under the Strategic Computing Program. The goal of this program was to investigate the applicability of emerging Artificial Intelligence capabilities in military applications. The PA was administered by the USAF Avionics Laboratory at Wright-Patterson Air Force Base in Dayton, Ohio, and was originally conceived as a set of four collaborating expert systems: Systems Status to report the ability of the aircraft to perform its mission; Situation Assessment to determine the external environment; Mission Planning to provide a long-term strategic view of the mission; and Tactics Planning to give a short-term, reactive view. After consultation with the customer, three more expert systems were added: Pilot-Vehicle Interface (PVI) to manage the presentation of information to the pilot and interpret pilot actions; Mission Management to maintain a consistent view of the mission parameters and organize the communications between systems; and Mission Support Tool to allow pilots to tailor the behavior of the systems before each mission. ASI led the integration and test of the system and contributed to the knowledge design and implementation as a sub-contractor to Lockheed Aeronautical Systems.

When the program ended, Dr. Norman Geddes gathered a number of PA artifacts and founded Applied Systems Intelligence (known now as Veloxiti) to continue refining his concept of an Associate System. Dr. Geddes holds a Doctoral degree in Systems Engineering from Georgia Institute of Technology, with a specialty in Human-Machine Systems. He is a leading researcher on the creation and validation of real-time computer models of human beliefs and intentions, both as individuals and as groups. His ground-breaking use of real-time models of human beliefs and intentions has been recognized by numerous awards, including the Grover Bell Award for research from the American Helicopter Society for his work on the Army Rotorcraft Pilots Associate, and a 2005 National Group Achievement Award from NASA for his contributions to modeling of human beliefs and intentions in collaborative air traffic management. Dr. Geddes also holds a Master of Science in Aeronautical Systems and a Bachelor of Science in Physics. From 1971 to 1977, he served as a Naval Aviator, completing tours as a jet flight instructor and as a project test pilot for tactical jet fighter projects at the Naval Air Development Center. After making several technical contributions to human interaction with complex systems and simulations, Dr. Geddes turned his main research interests to the creation of real-time intelligent systems that model the complex behaviors of skilled humans, as individuals and as groups. His work in this area has been instrumental in such recent programs as the DARPA/Boeing X-45 Joint Unmanned Aircraft System and the Army Future Combat Systems Warfighter Machine Interface System. Dr. Geddes holds patents in the use of real-time dynamic models of intentions as applied to intelligent network management and dynamic business process management.

Associate system technology was further developed under the Rotorcraft Pilots Associate (1994-1998). For this $250 million effort, ASI participated in the system design, software development, and testing of the RPA and supplied the Crew Intent Model for use within the intelligent Crew Interface Manager. The Rotorcraft Pilot's Associate (RPA) was an advanced mission/cockpit management system developed by the Army and Boeing Mesa (then McDonnell Douglas Helicopter) to assist the pilot with complex combat tasks. The architecture and performance of the RPA was based directly on the earlier DARPA Lockheed Pilot's Associate (1987-1992). The target mission for the RPA was Scout-Attack, and it was developed on the Apache Longbow platform. Although RPA was designed to initiate time critical activities such as actions on contact, it always kept the pilot in charge of the aircraft. In battle situations, RPA identified and prioritized targets, selected battle positions, coordinated target handoffs among available teammates and provided safety areas for the pilot.

Beginning in 2003, ASI began working on an Intelligent Presentation Services component for Future Combat Systems / BCTM (2003-2011). Though Future Combat Systems was an unsuccessful program, some of the technical ideas developed in that program have been applied to the Warfighter's Associate. ASI was a key developer in FCS working on elements of the Warfighter Machine Interface that supplies real time, context sensitive information to the warfighters based on each warfighter's role. One of ASI's major contributions to this program is developing an Intelligent Presentation Services framework through which disparate domain services publish information that is then synthesized to provide the user with a comprehensive view of the situation. Intelligent Presentation Services are used on both the handheld Centralized Controller (CC) and the vehicle based Network Integration Kit (NIK). The other major contribution made by ASI is role based information management, which helps to mediate and orchestrate the flow of information through Intelligent Presentation Services to reduce information overload and help warfighters to better manage workload.

HISTORY: CORE IDEAS

Veloxiti's approach to building intelligent systems draws on ideas from three sources: a fighter pilot, a philosopher, and a computer scientist. What follows is a discussion first about the ideas that drive Veloxiti's approach to intelligent decision making followed by a discussion of our implementation approach.

JOHN BOYD: THE FIGHTER PILOT AND MILITARY STRATEGIST

Colonel John Boyd (U.S. Air Force) made two significant contributions as a fighter pilot and military strategist. They were Energy-Maneuverability theory and the OODA Loop. Col Boyd, a skilled U.S. jet fighter pilot in the Korean War, began developing the Energy-maneuverability theory in the early 1960s. He teamed with mathematician Thomas Christie at Eglin Air Force Base to use the base's high-speed computer to compare the performance envelopes of U.S. and Soviet aircraft from the Korean and Vietnam Wars. They completed a two-volume report on their studies in 1964. Energy Maneuverability came to be accepted within the U.S. Air Force and brought about improvements in the requirements for the F-15 Eagle and later the F-16 Fighting Falcon fighters.

Energy–maneuverability theory is a model of aircraft performance. It is useful in describing an aircraft's performance as the total of kinetic and potential energies or aircraft specific energy. It relates the thrust, weight, drag, wing area, and other flight characteristics of an aircraft into a quantitative model. This allows combat capabilities of various aircraft or prospective design trade-offs to be predicted and compared.

Based on Boyd's previous success he was kept on as a key military air-to-air combat strategist. Boyd's next key concept was that of the decision cycle or OODA Loop, the process by which an entity (either an individual or an organization) reacts to an event. According to this idea, the key to victory is to be able to create situations wherein one can make appropriate decisions more quickly than one's opponent. The construct was originally a theory of achieving success in air-to-air combat, developed out of Boyd's earlier Energy-maneuverability theory and his observations on air combat between MiG-15 and North American F-86 Sabre aircraft in Korea. Harry Hillaker (chief designer of the F-16) said of the OODA theory, "Time is the dominant parameter. The pilot who goes through the OODA cycle in the shortest time prevails because his opponent is caught responding to situations that have already changed."

Boyd hypothesized that all intelligent organisms and organizations undergo a continuous cycle of interaction with their environment. Boyd breaks this OODA cycle (Figure 1) down to four interrelated and overlapping processes through which one cycles continuously:

- Observation: the collection of data by means of the senses
- Orientation: the analysis and synthesis of data to form one's current mental perspective
- Decision: the determination of a course of action based on one's current mental perspective
- Action: the physical playing-out of decisions

Of course, while this is taking place, the situation may be changing. It is sometimes necessary to cancel a planned action in order to meet the changes. This decision cycle is thus known as the OODA loop. Boyd emphasized that this decision cycle is the central mechanism enabling adaptation (apart from natural selection) and is therefore critical to survival.

Boyd theorized that large organizations such as corporations, governments, or militaries possessed a hierarchy of OODA loops at tactical, grand-tactical (operational art), and strategic levels. In addition, he stated that most effective organizations have a highly decentralized chain of command that utilizes objective-driven orders, or directive control, rather than method-driven orders in order to harness the mental capacity and creative abilities of individual commanders at each level. In 2003, this power to the edge concept took the form of a DOD publication "Power to the Edge: Command ... Control ... in the Information Age" by Dr. David S. Alberts and Richard E. Hayes. Boyd argued that such a structure creates a flexible "organic whole" that is quicker to adapt to rapidly changing situations.

Veloxiti's approach to artificial intelligence has been heavily influenced by Boyd's problem solving approach, so much so that our development toolkit explicitly models the OODA loop.

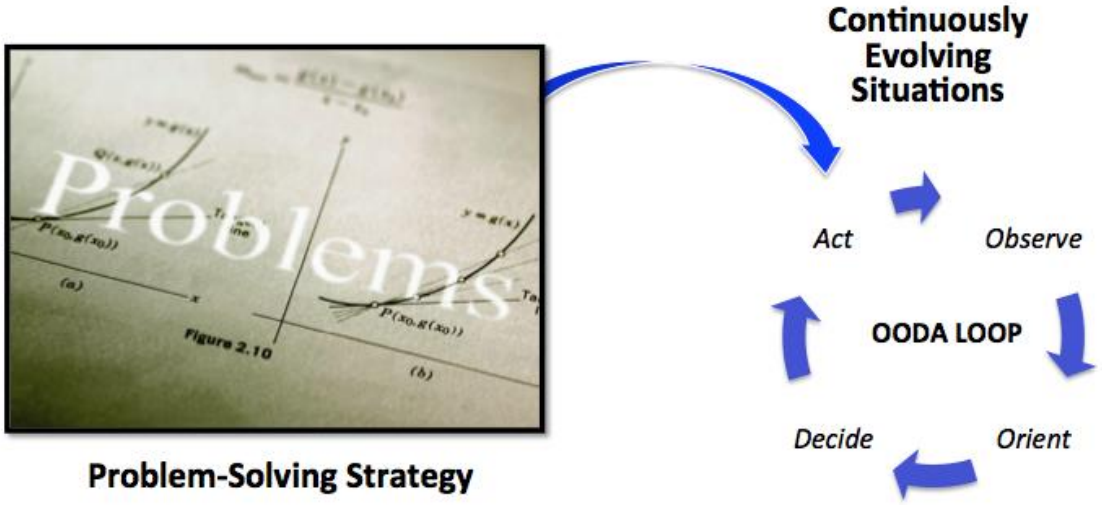

Figure 1: The OODA Loop Enables Continuous Adaptation to Continuously Changing Situations

MICHAEL BRATMAN: THE PHILOSOPHER

Michael Bratman is a Professor of Philosophy at Stanford University who has developed a theory of human reasoning known as the Belief-Desire-Intention (BDI) model. Broadly speaking Bratman is concerned with human action: how is it that humans can act rationally? Bratman argues that reasoning is fundamentally concerned with **beliefs**, **desires**, and **intentions**.

- *Beliefs* characterize what an agent (i.e., a human or computer system) considers to be true about the world.
- *Desires* represent an agent's goals or preferred end states and are a necessary element in planning and deciding. We need to know what we want to accomplish in order to develop a plan.
- *Intentions* are expressed in the plans developed for achieving *desires*.

Bratman's ideas are surprisingly consistent with John Boyd's view of the world. Consider the case of a fighter pilot flying a Combat Air Patrol (CAP). The CAP mission focuses on defeating or destroying all enemy aircraft within a combat area. This mission end-state is a high-level goal – i.e., a *Desire* using Bratman's terminology. Bratman's *Beliefs* represent what we know about the situation and accordingly are aspects of Observing and Orienting. Bratman's *Intentions* are plans and are components of Deciding and Acting.

Bratman's philosophical theory of reasoning has had a significant impact in the AI community and has led to development of a broad field of study referred to as BDI software architectures.

JOHN MCCARTHY: THE COMPUTER SCIENTIST

Like Michael Bratman, John McCarthy taught at Stanford Univerdity. Not only did McCarthy invent the term "artificial intelligence", he also created the LISP programming language, created one of the earliest chess playing programs, and influenced the development of artificial intelligence in innumerable ways. In his essay "What is Artificial Intelligence?"[3] McCarthy provides the following definition of AI and intelligence:

> "It *[artificial intelligence]* is the science and engineering of making intelligent machines, especially intelligent computer programs…Intelligence is the computational part of the ability to achieve goals in the world. Varying kinds and degrees of intelligence occur in people, many animals, and some machines."

John Boyd's OODA loop provides an approach for adapting continuously to continuously changing situations. Michael Bratman's BDI model describes rational action in terms of *beliefs*, *desires*, and *intentions*. Taken together, Boyd and Bratman offer a practical model for making *Intelligent Decisions*. John McCarthy identifies the kinds of problems that must be addressed in order to build *Thinking Software*.

McCarthy's divides the discipline of Artificial Intelligence into 12 branches, as shown in Table 1. Veloxiti's knowledge-based systems cannot be mapped directly to any single branch of McCarthy's AI tree. However, we can certainly describe Veloxiti's approach in terms of knowledge *Representation* technique, its *Inferencing* methods, and its *Planning* methodology.

Table 1: McCarthy's Branches of AI

AI Branch	Quick Definition
Logical AI	Uses mathematically-based logical languages to reason about the world
Search	AI programs often examine large numbers of possible solutions, and search techniques enable the discovery of good solutions
Pattern Recognition	AI techniques which involve comparing currently available data of some sort with patterns. Pattern recognition is commonly used not only in vision systems but in many other applications such as problem diagnosis systems.
Representation	Techniques for representing information about the world in software
Inference	Techniques for inferring new information from existing facts. Rule-based systems use inference as do many other techniques.
Common sense reasoning	Techniques for capturing and reasoning about the common sense world
Learning from experience	Techniques for learning about the world based on observation
Planning	Techniques for determining how to reach a goal.

3

Epistemology	The study of the kinds of knowledge required for solving problems in the world.
Ontology	The study of things that exist in the world and their relationships.
Heuristics	Rules of thumb for making decisions
Genetic programming	Techniques for problem solving that involve selecting the best of many – possibly millions -- of generated candidate solutions.

VELOXITI'S IMPLEMENTATION

Veloxiti builds intelligent systems based on the principles of Boyd's OODA loop, Bratman's BDI model, and classical Artificial Intelligence, as expressed by McCarthy. This section describes how Veloxiti's cognitive engine is implemented. To be consistent with McCarthy's view of AI, we will discuss the topics of *knowledge representation*, *inference*, and *planning* within VELOX, Veloxiti's knowledge engineering framework.

KNOWLEDGE REPRESENTATION

A software engineer staring at a blank editor screen and a great American novelist staring at a blank sheet of paper share the same problem: How should I begin? The novelist begins by imagining a world populated with characters driven by a plot line. The software engineer begins by imagining how to create in software a model of the world that is good enough to solve the problem at hand. Of course, instead of characters and a plot line, software engineers think in terms of data sources, user interfaces, business rules, and algorithms. Knowledge representation deals with how we represent the elements of a problem in software.

Due in great part to AI research, engineers have a lot of representation options. To cite two examples, we might

1. Use IF-THEN rules to *explicitly* represent knowledge in terms of antecedents (the "IF" part of the rule) and consequences (the "THEN" part of the rule).
2. Use a neural net to *implicitly* represent knowledge within the hidden layers of the network.

We say that rules are *explicit* because it is possible to examine a rule and understand what elements of knowledge are being expressed. We say that neural nets are *implicit* because it is typically not possible to examine a network's inner workings. In addition to distinguishing between explicit and implicit knowledge representation, we can also distinguish between representation formats that require *knowledge acquisition* vs. formats that provide for *machine learning*. IF-THEN systems typically require knowledge acquisition while neural net systems can be trained.

Knowledge representation is an enormous topic. We could expand the discussion by talking about additional kinds of neural techniques (e.g., associative memories), by considering approaches such as case-based reasoning, by discussing statistically based methods for both supervised and unsupervised learning and so on. All of these approaches are driven by underlying knowledge representation strategies – and each approach has strengths and weaknesses depending upon the problem to be solved.

Veloxiti's approach to knowledge representation can be described in just a couple of sentences. First, **we represent knowledge explicitly using knowledge graphs** not by using IF-THEN rules or cases. Secondly, **we use a knowledge acquisition approach** for determining how these graphs are structured, and hence Velox is not a machine learning system. (However, later in this paper we explain how Velox can be *integrated with* many different kinds of AI components, including IF-THEN systems, Case Based Reasoners, neural nets, and so on.)

Graphs are widely used in a broad range of software contexts, particularly in AI applications. In a typical AI system the set of possible solutions to a problem can be represented as a graph, which can be searched to determine the "best" solution. Much of the early work in AI focused on development of efficient graph search algorithms.

Consistent with our bias for OODA loop problem solving, Velox represents information about the state of the world in an Observe-Orient graph and information about plans in a Decide-Act graph (Figure 2). Consistent with our bias towards BDI models, the Observe-Orient graph represents *beliefs*, and the Decide-Act represents goals (i.e., desires) and plans (i.e., intentions). Though the graphs differ with respect to their detailed implementation, both the OO and DA graphs are acyclic, directed graphs

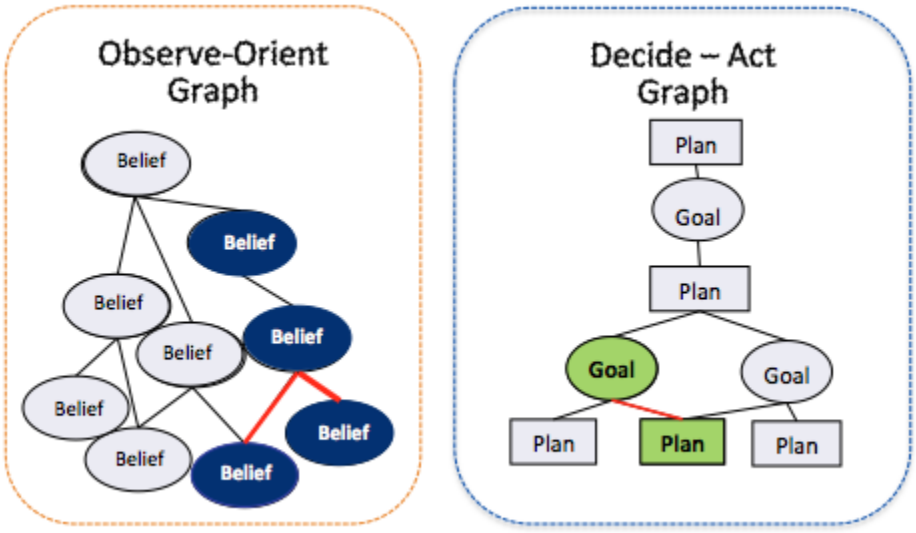

Figure 3: The Observe-Orient and Decide-Act Graphs

Note that knowledge graphs are not the same as rule sets. As is done in all software, Velox applications do use If-Then statements: there is no escaping IF statements in programming. However, Velox does not represent domain knowledge as simply a collection of IF-THEN rules – we represent knowledge as beliefs, desires (goals), and intentions (plans) in a graph. Furthermore, the "engine" Velox uses for executing knowledge is not at all like the Rete algorithm typically used in rule-based systems.

However, there is a deeper distinction between Velox and IF-THEN systems. Conventional rule-based systems boil human reasoning down to antecedents and consequents – nothing more. We believe that human reasoning is too complex to be represented solely as IF-THEN rules. In fact, we believe that Boyd's OODA loop and Bratman's Belief-Desires-Intentions model provide

a much richer context for problem solving. For example, the Velox Decide-Act graph provides developers with methods for controlling when a plan is proposed to a user, who has authority to execute a plan (i.e., the user or the system), the conditions under which plan execution should begin, and what should be done if a plan fails. Strictly speaking, it is *possible to* build IF-THEN systems with these capabilities but doing so is typically awkward and difficult.

In addition, the Velox framework is not based on machine learning. Unlike machine-learning approaches like neural nets, case-based reasoners, and statistical classifiers, the knowledge in a Velox application is more-or-less "fixed" during construction of the system's Observe-Orient and Decide-Act graph. Velox applications are not trained – they are designed and constructed.

Unfortunately, this last statement implies that Velox knowledge is not only "fixed" during development but "set in stone". This is an exaggeration since various techniques exist for parameterizing knowledge or adapting system behavior during execution. For example, the Army C-CAT application includes knowledge for selecting IR assets to be used for monitoring critical battlefield events. While the underlying logic is embedded within a knowledge graph, C-CAT provides a user interface for adding new kinds of IR platforms to the system's table of assets. Innovative engineering enables development of knowledge graphs that provide a surprising amount of flexibility in the field.

Furthermore, Velox was designed from the outset to facilitate integration with external procedures and programs, which might well leverage machine learning. For example, it is surprisingly easy to incorporate Bayesian reasoning within the OO graph to facilitate probabilistic assessment of conditions in the world. Integrating with other kinds of AI is equally easy, making it possible to combine the benefits of OODA-loop problem solving with external components implemented using rules, neural nets, case-based reasoners, statistical classifiers, and so on.

INFERENCING

An *inference* is a conclusion that can be drawn from the facts at hand. In an intelligent system, inference techniques enable a system to draw conclusions about the state of the world and to determine what should be done to deal with conditions of interest. Knowledge representation and inference techniques are inseparable topics. For example, conventional rule-based systems capture knowledge in rules. For large systems, the core challenge is to decide which rule(s) should be executed during each processing cycle. The Rete algorithm commonly used in rule bases provides an efficient means for selecting and "firing" rules. Because Velox represents knowledge in graphs, its inference component is essentially a graph traversal engine. Explaining how this engine works requires discussion of several detailed concepts.

Concept 1: Pattern and instance graphs

The discussion above describes the two knowledge graphs in a Velox application – the Observe-Orient graph and the Decide-Act graph. This description is a simplification; in fact, an executing Velox application actually has 4 graphs. In addition to distinguishing between a system's OO and DA graphs, Velox makes a distinction between **pattern** and **instance** graphs.

Programmers who work with object-oriented languages such as C++ and Java are familiar with the distinction between a class definition and a class instance. Consider an application whose

function is to determine whether an enemy air track is a threat to my aircraft. During development, we might define a class AirTrack that has attributes such as an ID, a Friend-Foe flag, an aircraft type, a maximum weapons range, a speed and a heading. Using these attributes, we could write a method for determining whether the enemy aircraft is – or will soon be – in range to be a threat. During execution, the system must track many air tracks, and thus the AirTrack class definition serves as a template (i.e., a pattern) for creating many AirTrack instances, one instance for each aircraft detected. Figure 4 illustrates this basic idea with a class definition on the left of the diagram and a class instance – which represents an actual track – on the right.

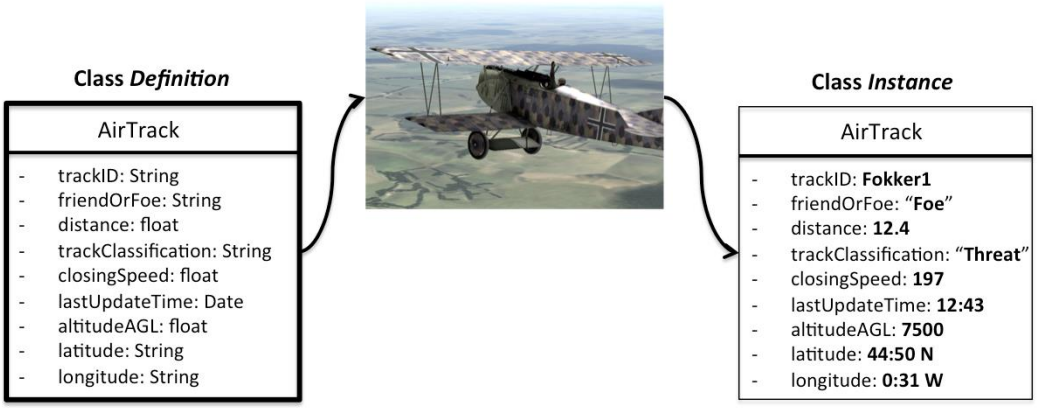

Figure 4: Class Definitions vs. Instances

This distinction between patterns and instances is fundamental in Velox. During application implementation, Velox developers create two **pattern** knowledge graphs, a pattern OO graph and a pattern DA graph. These graphs are like class definitions in standard object-oriented programming. We can think of the pattern OO graph as a specification of all the beliefs that the system is capable of reasoning about. For example, an agent designed for use in air-to-air combat must be able to reason about the enemy's weapons systems (air-to-air, ground-to-air, hand-held), sensors (radar, IR), and threat platforms. The OO pattern graph must represent the knowledge required for determining whether any particular platform like an aircraft or SAM site is a threat. Hence the pattern graph must represent all of the beliefs that are included in the scope of the system.

At any point in time, an executing application's OO **instance** graph will be a snapshot of threats believed to exist in the world. Consider Figure 5. A simple OO pattern graph is shown on the left hand side of the illustration. This instance graph will be structured in a manner that is consistent with the structure of the pattern graph. Hence, we might have 0, 1 or many active

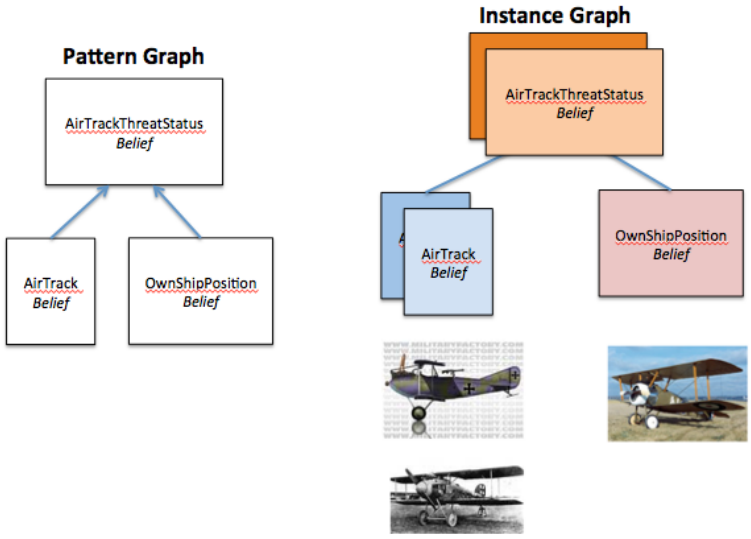

Figure 5: OO Pattern vs. OO Instance Graphs

threats in the OO instance graph, depending upon what is happening in the world. Engineers create the pattern graphs at design time. The Velox cognitive engine creates the instance graphs when data are provided to the system at runtime.

Concept 2: Graph Nodes and Links

Like any graph, OO and DA graphs are composed of **nodes** and **links**. Figure 6 illustrates a notional OO graph in the air combat domain. In this case, the objective is to determine whether an air track is a threat given its position and the own ship position.

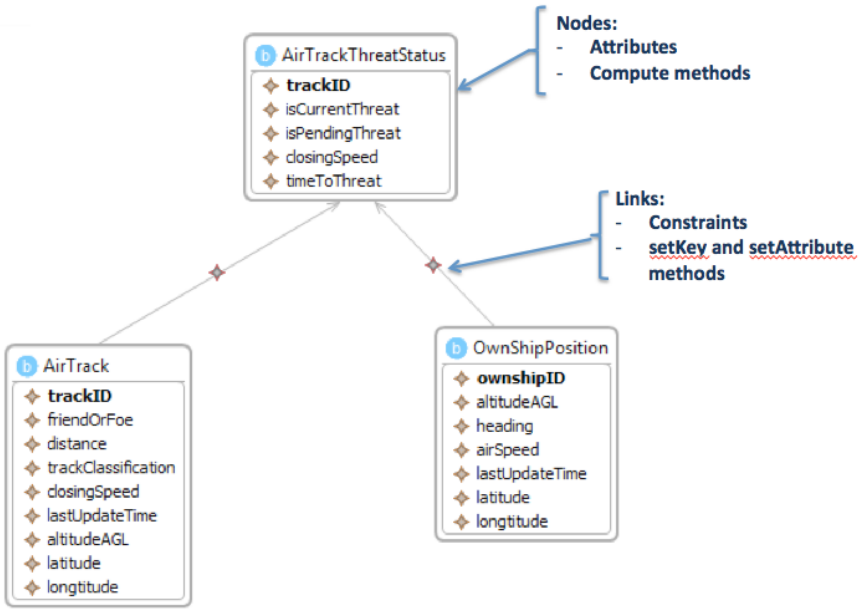

Figure 6: OO Graph Nodes and Links

Without going into elaborate detail, we can make several generalizations about nodes and links.

- Graph **nodes** are primarily containers for keys and attributes – i.e., the properties or variables that describe an object.
- **Nodes** may contain compute methods for setting the values of attributes.
- **Links** contain methods known as constraints for determining under what conditions we can cross from a parent link to a child link
- **Links** also contain logic for setting the keys and attributes of child nodes.

Recall the distinction between a pattern graph and an instance graph. Remember that the pattern graph is designed and implemented when a system is developed. Instance graphs, though, come into existence only when a system is running. The OO instance graph captures what we know about the world, and the DA instance graph captures what we are doing to deal with conditions of interest in the world.

So, we can think of the template graph as a set of instructions for creating the OO and DA instance graphs during system execution. Links are important because they contain the control logic for determining when nodes in the instance graph should be created. Links are also important because they contain the logic for propagating data values from parents to children in the graph.

Though both are composed of nodes and links, the OO and DA graphs have some important differences. We refer to nodes in the OO graph as Beliefs, a convention that is consistent with the BDI software model. The DA graph is composed of Goal nodes and Plan nodes. A Goal corresponds to a BDI desire; a Plan corresponds to a BDI intention. As was done in the PA programs Plan-Goal Graphs (Figure 2 above), goals and plans in a DA graph are organized as an And-Or graph governed by the following rules:

1. Goal and Plan nodes alternate in the graph. In other words, Goals may have child plans and Plans may have child Goals.
2. All of a Plan's sub-Goals need to be accomplished in order for a Plan to succeed.
3. Any one of a Goal's sub-Plans should be sufficient to satisfy the parent Goal.

Concept 3: Graph Traversal

Recall that McCarthy identifies *inference* as one of AI's key branches. The Veloxi cognitive engine is an inference engine that performs a very basic function during system execution: using the instructions encoded in the OO and DA template graphs, it uses currently available data to incrementally create the OO and DA instance graphs.

Referring to the Air Track example shown above, assume that own ship's radar system has just detected an air track for the first time. When provided with this data, the Velox cognitive engine will either create a new node instance or update an existing instance of the AirTrack belief. Then the engine will use the instructions in the OO **pattern** graph to instantiate additional instance graph nodes. Starting with new AirTrack node, the engine will evaluate the constraint logic contained in the link between AirTrack and AirTrackThreatStatus. If the constraint returns TRUE, then the engine will create an **instance** of AirTrackThreatStatus and set key and attribute values in AirTrackThreatStatus using the link's *setKey* and *setAttribute* methods.

Concept 4: Monitors

The cognitive engine's core function is to incrementally create the OO and DA **instance** graphs using the logic coded in the OO and DA **template** graphs. That being said, there are some details that need to be considered.

OODA loop problem solving is typically driven by changes in the situation. So too is processing in a Velox agent. A cognitive processing cycle begins when the situation changes. When data describing the situation are provided, the engine updates the OO instance graph according to the instruction in the pattern knowledge. This may result in belief instances being created, updated, or revoked.

Some changes in the situation are important; other changes are not. For example, it is *useful* to know that an enemy aircraft is far away and on a heading that will take it even farther away. However, it is **critical** to know that an aircraft has just approached close enough to be capable of attacking own ship. In the first case, the changed situation likely does not require a change in tactics. In the second case, survival depends on the pilot's response. The OO graph handles these high priority conditions of interest by way of **monitors**.

A *monitor* is similar in principle to a database trigger. A monitor "fires" when a condition of interest is detected. Monitors serve to bridge the gap between OO graph processing and DA graph processing. The firing of a monitor triggers processing in the DA graph that focuses on addressing the challenge posed by a condition of interest. For example, the monitor that fires when an enemy aircraft gets to close might be linked to a DA goal node labeled "Threats Neutralized."

The cognitive engine's processing cycle begins when new data become available in the OO instance graph. The engine incrementally updates the OO instance graph. If a condition of interest is detected, the engine fires a monitor and turns its attention to the DA graph. Because monitors point from a Belief node to a Goal / Plan node, the cognitive engine knows where to begin processing when a monitor fires. As is done in the OO graph, the engine uses knowledge encoded in the DA pattern graph to incrementally construct a DA instance graph.

The cognitive engine's underlying algorithm is "greedy", meaning that the engine aggressively tries to instantiate OO and DA nodes. When the engine can do no more processing in both the OO and DA graphs, it pauses and waits for new situational data to be provided to the OO graph.

PLANNING

The Decide-Act components of the OODA loop correspond to McCarthy's AI branch of planning. In the discussion above we described the Decide-Act (DA) graph as an And-Or graph composed of alternating Goal and Plan nodes. We can think of the Decide-Act graph as a way of organizing important goals to be achieved and the plans, which can be used for achieving these goals. Using BDI terms, goals are *desires* and plans are *intents*. If we think of the DA graph as simply a means of organizing knowledge, then we can treat *implementation* as a separate topic. In other words, we could use any kind of technology for building the Decide-Act portion of an OODA-loop system. In fact, during the Pilot's Associate program, an And-Or graph was used primarily for *organizing and structuring* air combat domain knowledge.

Now, it turns out that Veloxiti's approach uses DA graphs in a very direct manner. The Velox planner is a *skeletal* planner. Specifically, we generate plans by decomposing high-level goals into lower-level plans. Using AI-speak, the Velox cognitive engine performs top-down, decompositional planning. This approach differs from a number of different planning approaches such as case-based reasoning, generative planning, or methods that rely on machine learning techniques.

Note also that the Velox planner is a *reactive* planner, meaning that it is designed to respond to changes in the situation. The planner selects plans that are capable of satisfying parent goals, but these plans may not be optimal. This feature enables the planner to be quick. When a plan is instantiated, the planner continuously monitors the plan to ensure that it remains viable given the changing situation. If the situation has changed sufficiently that the plan is no longer capable of satisfying its parent goal, the plan is revoked and re-planning occurs.

As is done for the OO graph, the DA graph created during development is a *pattern* graph. During execution, the Velox cognitive engine uses this knowledge graph as a template for creating an instance graph, just as is done in the OO graph. When new data become available, the cognitive engine traverses the DA pattern graph and incrementally constructs a DA instance graph. In addition, though, there are a number of subtleties that need to be addressed during planning that do not apply to situation assessment. For example, when should plan execution begin and how do we know when a plan either completes successfully or fails? Furthermore, who should be responsible for executing the plan – the user or the system acting on behalf of the user? Velox addresses these issues by tagging plans and goals with a lifecycle state to provide fine-grained control over plan execution.

ARE OTHER AI TECHNIQUES COMPATIBLE WITH VELOX?

Veloxiti builds intelligent systems that leverage the OODA loop and the BDI model. Does this mean that Velox precludes other AI techniques such as machine learning, statistical algorithms, generative planning, pattern recognition and so on? *Actually NO – our approach to intelligent systems is compatible with many different AI technologies.*

Recall that a Velox knowledge graph is composed of nodes and links. Nodes in the OO graph are *beliefs;* nodes in the DA graph are *goals* and *plans.* Links in either graph are just *links.* Recall also that the Velox cognitive engine traverses the OO and DA *pattern* graphs to create an *instance* graph that represents our beliefs about the state of the world plus goals to be achieved and plans for achieving goals.

Broadly speaking, the Velox cognitive engine has two fundamental functions:

1. As it traverses the template graph, the cognitive engine evaluates logic contained in links to determine which paths to follow in the graph. This control logic is embedded in the link's *constraint* method.
2. When it creates a node (i.e., a belief, goal, or plan), the engine sets the values of attributes in the node using a *compute* method.

Stripped down to bare-bones basics, the Velox cognitive engine evaluates link constraints to determine how to traverse the OO and DA graphs, and it uses computes to set values of attributes in nodes. That's essentially all it does.

In most cases, constraint logic and compute logic can be encapsulated within links and nodes using plain Java code. However, we need not think in such limited terms. Velox is enormously flexible and it is perfectly feasible – almost trivial – to invoke external algorithms and programs, from link constraints or node computes.

For example, Veloxiti used a statistical algorithm known as Continuous Value at Risk (CVAR) to identify vehicles on a road segment whose speeds were significantly slower than average based on MTI (moving target indicator) data. Though originally developed for use in financial modeling, the CVAR algorithm was adopted for use with MTI (moving target indicator) radar data. For this application, the CVAR algorithm was invoked from a node compute statement to identify vehicle speed outliers on road segments. In this case, the external algorithm was a statistical technique, but a neural net, an associative memory or some other AI technique could perhaps serve the same function.

Recall the discussion above dealing with the PAL framework and the Universal Information Management Architecture (UIMA). Both PAL and UIMA are specifically designed for integrating many kinds of AI. We see a similar role for Velox. In our case, the OODA loop and BDI principles provide a model for organizing and controlling higher-level thinking and decision-making. Other AI techniques such as rules, pattern recognizers, case-based reasoners, and machine learning components can provide lower level services that can be invoked as needed. This layered view of problem solving reinforces the value of integrating Velox systems with external AI systems.

THE VELOX TOOLKIT

Not too many years ago, true AI researchers wrote code in Lisp, a programming language invented in the mid-1950's. Though elegant and powerful, Lisp is also difficult to learn and infrequently used today. Velox is a development toolkit for engineers who may not know Lisp and may not have graduate training in AI. Velox leverages Java and the Eclipse framework, two of today's most commonly used development tools. Figure 7 illustrates how Velox works.

The Velox toolkit is a set of Eclipse plug-ins. Eclipse is an open source software development framework. A "plug-in" is an extension to Eclipse's core capabilities. One of our key objectives was to simplify development so that ordinary Java engineers – rather than AI graduate students – can build intelligent systems. To that end, we created two graphical editors, one for creating OO graphs and one for creating DA graphs. These graphical editors allow users to draw, connect, and annotate beliefs, links, goals, and plans. Because it would be tedious to graphically create all of the code for a program, the graphical editors are synchronized with text editors that function much like the text editors developers typically use. The graphical and text editors are synchronized, meaning that changes in a graphical editor are reflected in the text editors and vice versa. Inside the text editor, knowledge is represented in the Velox Domain Specific Language (DSL), a simple dialect of Java specialized for building knowledge graphs that are composed of beliefs, monitors, goals, and plans. Developers write code using this DSL.

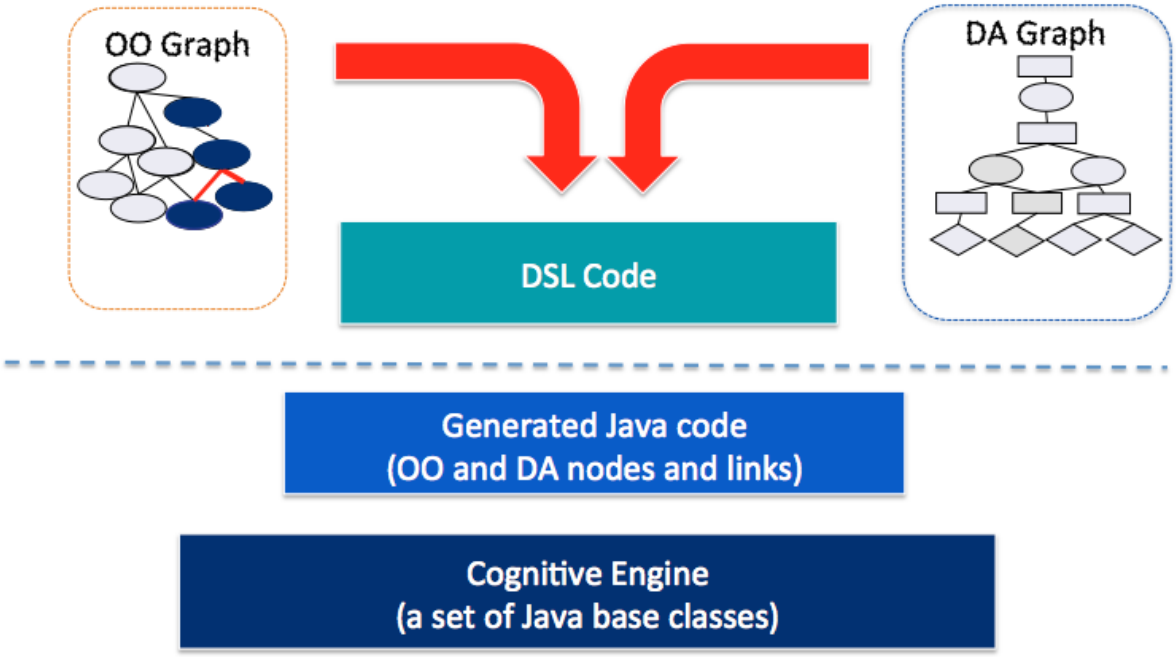

Figure 7: The Velox Implementation "Stack"

Returning to Figure 7, note the Cognitive Engine at the base of the diagram. The base classes defined at this layer contain the core logic used for setting attribute values, traversing graphs, firing monitors, and so on. Base classes exist for beliefs, monitors, goals, plans, and links. The Cognitive Engine classes are provided in a library that developers need not modify.

Sitting between the DSL Code layer and the Cognitive Engine are a set of Generated Java classes. When a developer creates a node or link using the graphical and text editors, the Velox toolkit *automatically generates* a Java class definition that derives from a cognitive engine class and that has specialized behavior based on the DSL written by the developer. Code generation is absolutely transparent – developers need not do anything special to generate code.

Figure 8 illustrates the Velox Designer. The top pane in the figure is a text editor displaying a DSL snippet; the bottom pane is a graphical view of an OO graph. Velox has a look and feel similar to the numerous Integrated Development Markets available via Eclipse. The Velox learning curve for a reasonably capable Java programmer is minimal.

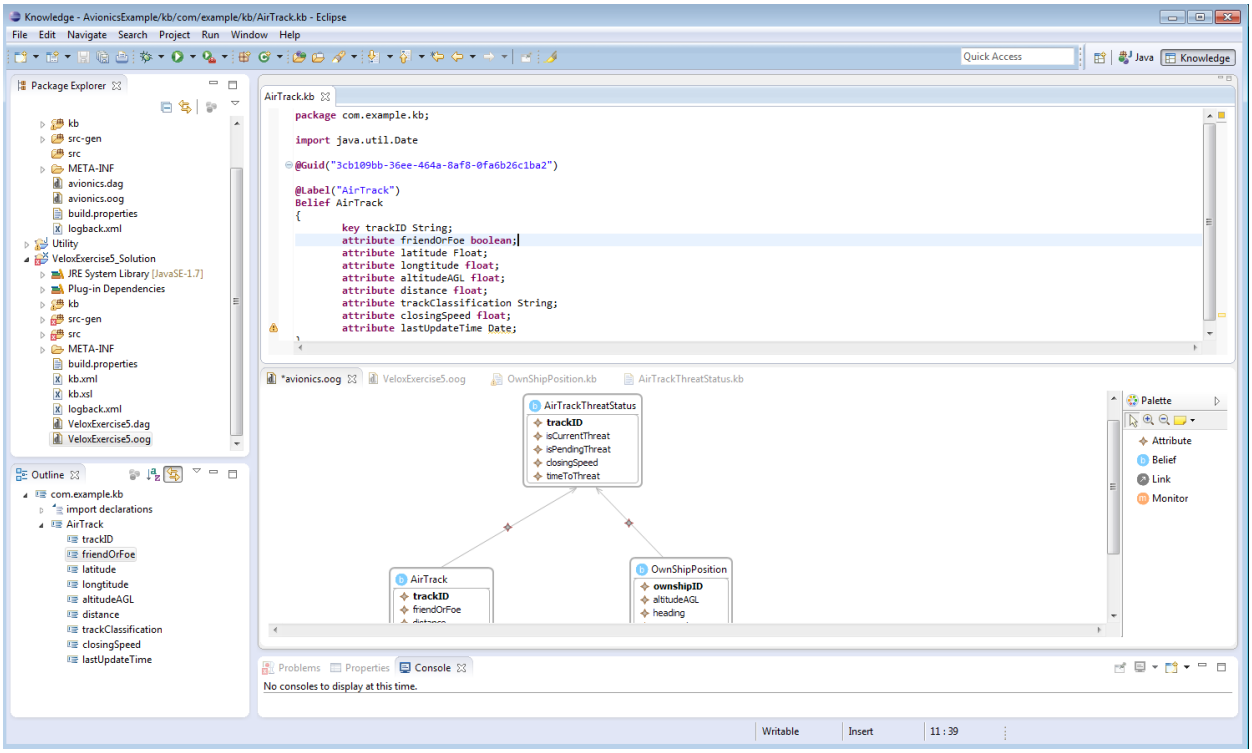

Figure 8: The Velox Development Tool in Eclipse

Because Velox generates plain Java code, it's possible to integrate a Velox application with external data bases, programs, user interfaces, web services, and so on using any of the mechanisms supported by Java. Furthermore, the toolkit includes modules for choreographing the behavior of multiple Velox agents in a system of systems.

THE DATA ANALYSIS TOOL (DAT)

Demonstrating the value of a technology is no trivial task, particularly complex technologies such as the Warfighter's Associate. To facilitate system evaluation, Veloxiti developed a Data Analysis Tool (DAT) for collecting performance data during man-in-the-loop experiments. ARL's Human Research and Engineering Directorate has used the DAT for evaluating both individual and team performance with – and without – the Warfighter's Associate.[4]

Recall that the Warfighter's Associate was constructed in a BDI framework, meaning that the system, as it executes, records beliefs about the world, goals to be achieved, and plans to be executed. The essential idea behind the DAT is rather elegant. The Warfighter Associate's instance graphs capture what is happening in the world (the OO graph) and what soldiers intend to do (the DA graph). By measuring what is happening in the graphs, we can make

[4] Norbou Buchler, et al. "The Warfighter Associate: Objective and Automated Metrics for Mission Command." 18th International Command and Control Research and Technology Symposium, 10-21 June 2013, Alexandria, VA.

inferences about soldier and team performance. In a paper that describes experiments performed by ARL's Human Research & Engineering Directorate, Norbou Buchler writes:

> The underlying activation of the knowledge structures in the Warfighter Associate can provide a dynamic real-time model of human intention that has the potential to support the analytical community with novel classes of metrics…Examples include:
>
> - Cognitive workload (number of concurrently active goals across time)
> - Currently active plans and goals in the D-A graph
> - Timing to complete tasks
> - Necessary collaborations (shared plans & goals)
> - Force synchronization (timeliness of distributed sub-goal satisfaction)[5]

The Warfighter Associate can be configured to capture the changing states of the OO and DA instance graphs in a database. Either in real-time or after-the-fact, these data can be used to calculate metrics like those Buchler describes. Figure 9 below illustrates one of the DAT's displays, a "Heat Map" that visually portrays nodes in the WA's knowledge graphs that are active.

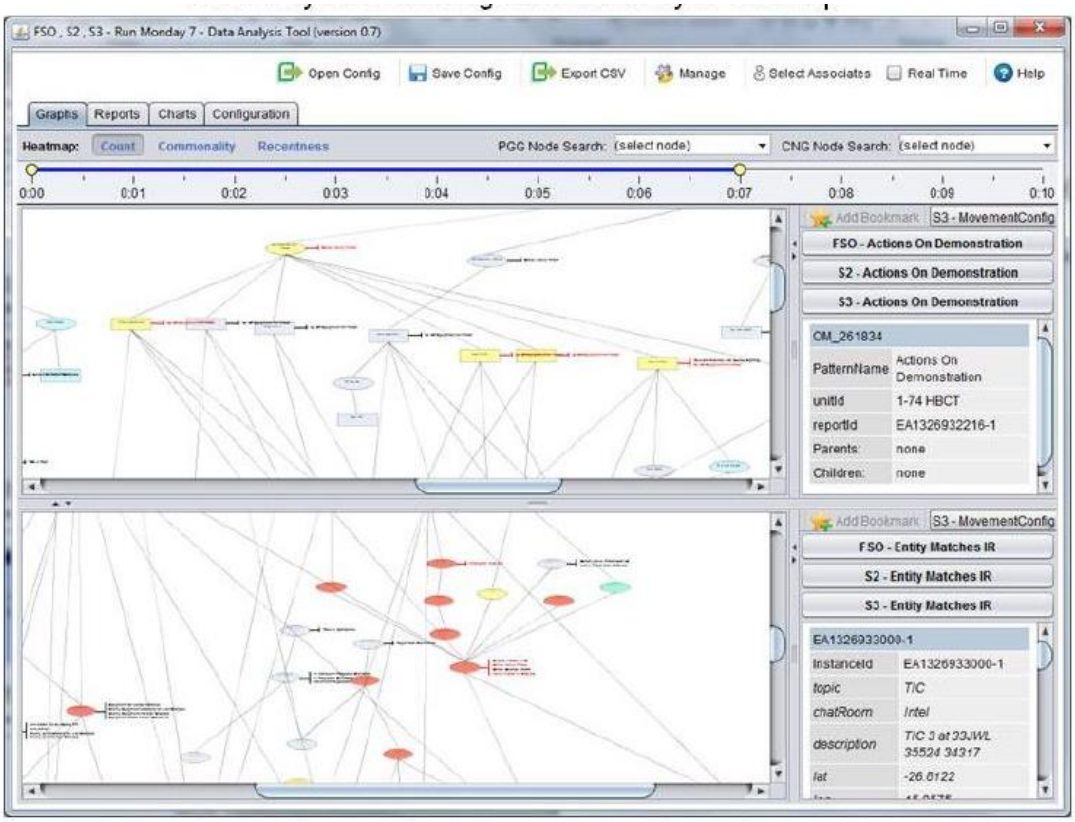

Figure 9: DAT Heat Map

[5] Buchler, p. 9.

A QUICK SUMMARY

The Warfighter's Associate is an instance of an intelligent decision support system implemented using Velox, Veloxiti's framework for building AI systems. Velox is a uniquely capable tool for building military domain applications because it leverages the OODA loop and the BDI software architecture, a well-respected theory of human action.

Velox is not a rule-based system, nor is it a machine learning system. Velox represents knowledge using two graphs – one for Observing and Orienting and one for Deciding and Acting. The Velox uses the OO and DA pattern graphs during system execution to build a model of the situation in an OO instance graph and a model of actions to be taken in a DA instance graph. Like commercially developed frameworks such as IBM's Unstructured Information Management Architecture, Velox is designed for integrating many kinds of AI.

The Velox design tool is an Eclipse-based set of graphical and text editors for creating and integrating intelligent systems. It has been explicitly designed to simplify the implementation of complex OODA-loop applications.

The Data Analysis Tool (DAT) is an application for measuring both individual and team performance. It has been used successfully in a number of human-in-the-loop experiments performed by ARL's Human Research & Engineering Directorate.