Concept-Level Anticipatory Planning

Kevin Huggins, Rusl Flowers, John Hill, John Surdu*
Department of Electrical Engineering and Computer Science
*Information Technology Operations Center
United States Military Academy
West Point, NY 10996, USA
{ kevin-huggins | thomas-flowers | john-hill | john-surdu } @usma.edu

KEYWORDS

Military, Planning Aids, Distributed Planning

ABSTRACT

The U.S. Military is investigating ways to take advantage of modern information systems, simulation techniques, and artificial intelligence methodologies to gain information dominance over potential adversaries. One of the research areas being explored is the implementation of a new approach to planning called "anticipatory planning and adaptive execution" that merges planning, execution, and monitoring into one continuous process. The first research prototype in this area, called OpSim, verified the utility of operationally focused simulations in detecting and reporting deviations from the plan. A follow on system, called APSS, proved the concept of combining execution monitoring, dynamic replanning, and adaptive execution to focus planning effort ahead of likely outcomes. The APSS system also demonstrated the requirement for two improved capabilities. The first is the ability to represent the plan at the conceptual level. The second is the necessity of maintaining the plan representation in a distributed environment.

BACKGROUND

Brigadier General (retired) Wass de Czege has proposed a new approach to military planning called "anticipatory planning and adaptive execution." [1] This approach One of the key pieces of this approach is continuous monitoring of execution and comparison of the "actual state" with the original plan. The OpSim prototype applied operationally focused simulations to the monitoring process and used artificial intelligence reasoning techniques to determine the significance of any detected deviations [2]. The Anticipatory Planning Support System (APSS) prototype used software agents called the planning executive, execution monitors, and planners to prioritize planning, monitor execution and compare it to the plan, and to conduct replanning, respectively. All of the agents operate on a common plan description. The key difference between the methodology employed in APSS and other planners is the persistence of the agents and their ability to continuously monitor execution and replan, as opposed to an interleaved sensing, planning, executing cycle. Figure 1 shows a test run of the APSS system driven by a stimulation system (background). The system shows the tree-like structure of the plan description, the actual state received from the stimulator, and the planned state in a future node of the plan description.

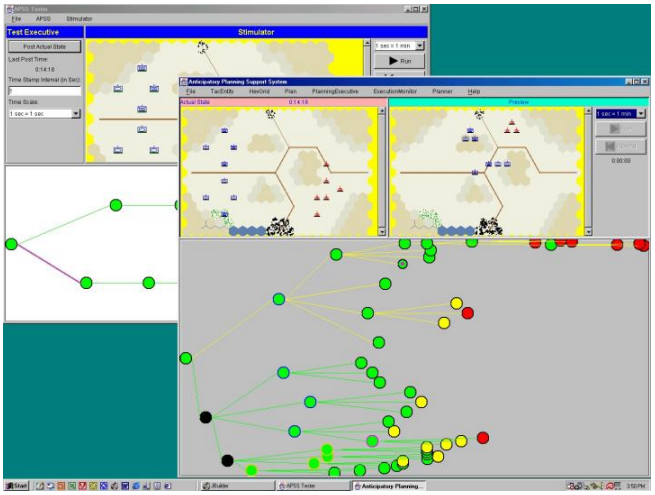

Figure 1: The APSS Prototype

APSS served as a good "proof of concept" but needs to address two weaknesses: First, the plan description at the entity level made "forward simulation" by the execution monitors degrade rapidly for small variations in the plan. Second, it didn't actually use a distributed database for the plan representation. In fact, over-reliance on local storage in specific format limits scalability and collaborative work. The solution is to plan at the concept level, and represent the plan in a distributed database implementation.

RELATED WORK

Much of the work in artificial intelligence planning has centered on creating plans for agents (even before the term 'agent' was fashionable) to execute in solving some problem. Some of the earliest works in this area include the now-classic systems STRIPS [3] (totally ordered planning) and NONLIN [4] (partially ordered planning). Partially ordered planning involves adding plan steps that satisfy previously unsatisfied preconditions without threatening already satisfied preconditions; however, the decision about the ordering of the plan steps is delayed as long as possible [5]. These early planning systems (and many planning systems based on them) involved sensing all relevant attributes of the environment and then making a plan. These systems generally assumed that only agents affected the world and that actions were instantaneous [6]. These planners also focused on created detailed, immediately executable plans. They did not behave well in dynamic environments.

The Open Planning Architecture addresses some of these issues by taking into account duration of plan steps and by allowing hierarchical planning [7]. Hierarchical planning involves the creation of higher-level, abstract plans that will accomplish the goal. Each step of the high-level plan is then decomposed to less-abstract plan steps. Hierarchical planning was used in the Capture the Flag project to build detailed plans for entities in simulation [8]. Eventually, a detailed plan that achieves the desired goal is built. Hierarchical planning can reduce the computational and time complexity of creating plans[9]. Hierarchical planning, however, still does not by itself address the issues of a dynamic environment.

Eller-Meshreki, Saunders, and Meshrieki assert that most planning systems suffer from the need to have vast amounts of domain knowledge prior to planning, much of which may never be used [9]. They assert that agents can begin executing parts of plans while other details of the plan can be filled in later. For instance, a person can get on a train bound for Chicago without knowing all the details about how to get from the train station to their hotel. In many domains the details needed to make a plan may not be available until after execution has begun. When getting on the train, the person would have no way of knowing that the road from the train station to the hotel will be blocked by a traffic accident by the time the person gets off the train in Chicago. In most planning systems, this would require re-planning, building a new plan from the point at which the current plan failed [10].

Another common method of accounting for a dynamic environment is the interleaving of planning and execution steps. In these systems, such as ONCOCIN [6], a plan is executed for some amount of time, the world state representation is updated, a new plan step is created, and execution resumes. Often these detailed plan steps are generated based on an abstract, higher-level plan.

Agre and Chapman argue that no planner can create "completely detailed plans in domains of realistic complexity" [11]. They argue that plans should merely be one source of input to the executing agent, a guide so to speak. They also indicate that hierarchical planning in many complex domains may not work, as it is unclear whether man realistic problems can be easily decomposed.

In response to the notion that traditional planning theories are incapable of providing useful plans in realistic environments, researchers such as Brooks [12] have proposed reactive agents. Brooks' subsumption architecture builds intelligent behaviors from layers of lower-level, simple behaviors. The theory behind subsumption is that by being physically grounded in the real world agents can more easily operate in dynamic environments. Robots built on this architecture have shown great abilities to perform interesting tasks in crowded offices and buildings.

One source of impetus for automatic planning has been the creation of more intelligent simulation entities, ones that do not require as much human control and decision-making. Hayes, et al., demonstrated the automatic generation of courses of action in simulation [13], a necessary precondition to creation of plans. Porto, Fogle, and Fogle used genetic algorithms to combine plan segments into plans for entities in the ModSAF tactical battlefield simulation [14]. Gelenbe, Seref, and Xu proposed Learning Agents which use experience gained in previous simulations to guide future actions [15].

Simulations have been used to make agents behave more intelligently. West, et al., used simulations to help create strike plans for aircraft [16]. Lee and Fishwick used simulation to conduct route planning for mobile robots [17]. Interestingly, they also assert that the simulation model "can be used to serve as a reference model to track the state of the execution in order to monitor [the robot's] progress towards the goal." Davis used simulations running in parallel with the real operation to predict the outcome of possible decisions in order to make a better choice [18, 19]. A similar approach was used in OpSim to predict when the execution of an operation was diverging from the plan [2].

Gilmer and Sullivan used a concept called Multitrajectory Simulation to predict the outcome of decisions [20]. The operation was simulated, and each time a decision point in the plan was reached, the simulation

forked multiple copies of itself to explore the results of each possible decision. This branching continued until success or failure was reached. Starting closest to the end state of the operation, the results of each branch were analyzed to determine the correct choice at each branch point. By combining the various correct decisions, a plan was built. By combining Davis' approach with Gilmer's and Sullivan's approach a planner might be built which is responsive to a dynamic environment.

CONCEPT-LEVEL PLANNING WITH APSS

In much of the literature on the subject of planning, the goal is to derive detailed, executable plans. In the military planning domain that is the focus of this research, it is appropriate to *keep the planning at the abstract level*. As experiments with the prototype APSS system showed, detailed plans that focus on entity-level locations and activities are too sensitive to differences between the actual states of execution and the planned states.

What is needed instead are abstract plans, called *concepts*. These concepts would not be executable by entities, simulated or real, in the traditional way. Rather, they shift the focus of the planning system to accomplishing the *goal* of the concept. Armed with the concept and the goal, a planning module can postulate different ways of assigning tasks to available entities that will lead to accomplishment of the goal.

The change to concept-level planning necessitates many changes in the APSS prototype, the most fundamental of which is the plan description. In addition, the agents in APSS use simulations, evolutionary algorithms, fuzzy rules, and other AI technologies to conduct planning and execution monitoring. All of these will have to be modified to operate on the concept-based plan description.

Concepts

As currently envisioned, a concept has two major components: a course of action (COA) for the Red Force, and one for the Blue Force. The desired end-state, force structure, and set of tactical tasks for the Red Force are assumed, and may come from initial analysis, or be driven by current observations. With the Red COA in mind, the Blue Force COA is set up to lead to the *global* desired end-state. An example of a complete concept appears in Figure 2. Note that the Red and Blue forces are broken up into sub-forces that are assigned tasks. The actual allocation of tactical entities to those sub-forces does not happen until it is time to implement the concept.

A tactical task is the activity to be performed by a sub-force. Typically, a task will include an assigned area on the battlefield in which the sub-force operates. Depending on the type of task, it may target a specific enemy sub-force or a piece of terrain. The task will have its own desired end-state in terms of the locations and strengths of the Red and Blue sub-forces it encompasses. Some example tactical tasks are **Destroy**, **Block**, and **Penetrate**.

RED (assumed)	BLUE
Red *Desired* End State	**Blue *Desired* End State**
Time: 10171400Z	Time: 10171500Z
Red Location: an area	Blue Location: an area
Red Strength: a percent	Blue Strength: a percent
Blue Location: an area	Red Location: an area
Blue Strength: a percent	Red Strength: a percent
Red Force Structure	**Blue Force Structure**
SubForce 1 (RSF1)	Blue SubForce 1 (BSF1)
SubForce 2 (RSF2)	Blue SubForce 2 (BSF2)
	Blue SubForce 3 (BSF3)
Tactical Tasks	**Tactical Tasks**
RSF1: Penetrate(…)	BSF1: Block(…)
RSF2: Assume(RSF1)	BSF2: Destroy(RSF1)
	BSF3: Destroy(RSF2)

Figure 2: Concept Example

An end state describes the conditions that should hold following the execution of a tactical task or a concept. It includes a time stamp, friendly force location, friendly force strength, enemy force location, and enemy force strength. End states for tactical tasks support accomplishment of the end states for concepts, which in turn are designed to lead to accomplishment of the global desired end state.

Plan Description

The plan description contains all of the information relevant to the plan, including a representation of the tarrain and the tactical entities that are participating. As the operation progresses, the plan description receives and stores actual states, each of which reflects a snapshot of the location and status of the entities. As planning progresses new branches are added to the tree-like structure of the plan, with nodes in the tree holding planned states. See Figure 3 for a visualization of the plan description.

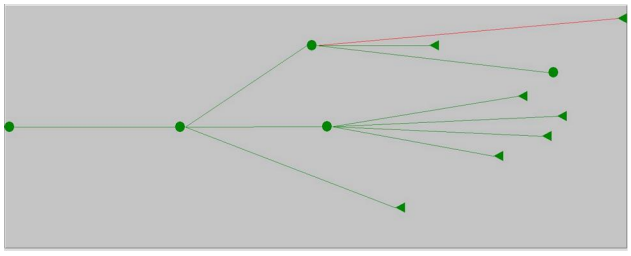

Figure 3: Visualization of the Plan Description

Branches in the plan description are created by the planner agents to implement the concepts. From a particular node, several concepts may be explored, and multiple branches may be created for each concept. Branches generated for the same concept will be differentiated by movement paths and interactions. Each node contains a planned state containing a time stamp, the areas occupied by the forces, and the status of the forces.

Agents

Planners are software agents that are aware of the global desired end state. They are placed on a node (which contains a planned state) by the planning executive and tasked to develop concepts for accomplishment of the desired end state from that planned state. Once the concepts are determined, the planner invokes branch generators to produce some number of representative branches from that node based on each concept.

Execution monitors are software agents that are placed on a node by the planning executive. Their purpose is to periodically compare the actual state of the operation with the planned state contained in the node. By simulating forward from the actual state, guided by the concepts represented by the branches, the execution monitor can assign a metric for how well the operation is progressing with regards to the planned state held in the node. If significant deviations are detected, the execution monitor initiates a process (through the planning executive) of "walking back" towards the actual state until the particular branch causing the problem is determined. The planning executive can then place planners at the latest "good" node and initiate new planning efforts ahead of that node.

DISTRIBUTED IMPLEMENTATION

Earlier versions of APSS used a hard coded plan description. This made it difficult to disseminate a plan description to subordinate or adjacent units. By using a relational database, plan descriptions are easier to share between different units and different echelons of the same units because databases (and the distributed middleware above it) are designed for concurrent multi-user access. Furthermore, referential and domain integrity comes automatically with most relational database management systems. These characteristics all provide support dynamic representations to plan descriptions.

Data management in a distribute environment is becoming a common requirement, especially with the advent of the Internet. This applies to the military also. The US Army's mission dictates an ability to function in a highly distributed environment with parts of units possibly spread around the globe. The Army's software applications that support tactical missions are organized around Battle Field Operating Systems (BOS). The BOS's include Infantry, Armor, Intelligence, Communications and others.

A key goal for the Army in this distributed environment is maintaining a common operational picture in which all commanders have a common digital representation of the battlefield. The state of the battlefield is maintained in the Joint Common Database (JCDB). Each BOS software application assesses the JCDB to obtain information from or update the JCDB. The problem with this approach that there is no decoupling from the data access level tasks and the application logic tasks. As a result, all applications must be cognizant of the details of the database. This tight coupling makes the system quite brittle and difficult to manage and change [21].

APSS is also an inherently distributed environment. However, our approach decouples the data access logic from the application logic. Hence, software modules that implement the application logic are completely unaware of an underlying database. For example, if the database needs to be changed or converted to a distributed database, the application logic software goes unchanged. This decoupling enhances location irrelevance, improves maintainability and increases scalability.

THE PLANNING DATABASE AND DISTRIBUTED ENVIRONMENT

APSS as a Distributed Planning System

The APSS is conceptually a distributed system. Therefore, the architecture and the infrastructure that hosts the system must support and facilitate a distributed computing environment. Components of the APSS, such as data, services, business objects, and clients can be and will be distributed. The architecture must also ensure that these distributed components are loosely coupled and that standard specifications, such as, TCP/IP, CORBA, and XML are supported. Loose couplings, interfaces, and standards facilitate a pluggable environment, where components can be swapped in or out without affecting other components. For example, a planner module should not need to know how data is stored in the data layer. In such a case, the database structure can be changed or another database vendor adopted and the planner would not need to be modified.

APSS is designed for organizations that are in a fluid situation, such as on the battlefield. Components of the organization that are responsible for decision-making or for contributing to the decision-making process will be moving

or they must be able to move with short notice. When not moving, these components are rarely collocated for any extended period of time. Therefore, the decision-making components are distributed on the battlefield, and realistically can be anywhere in the world. The system that supports the decision-making process in a distributed environment should itself be distributed. The system should conform to the nature of the business. Unless there is a compelling reason, organizations may discard a system that does not fit the mission and their way of doing business.

A distributed system enables distributed development. The database resides in the data layer of the system. Services, such as the plan description, can be decoupled from the database and reside in the services layer of the system. Furthermore, modules (like planners, the planning executive, and node evaluators) can be decoupled from both the services and data layers and can access data via the services layer over a network. Thus, the modules only need access to the services layer to effectively be a part of the APSS system.

Platform Independence

Ideally, the system should also be platform independent. Platform independence will enable developers to use the environment of their choice. Network protocols, such as TCP/IP, and mark-up languages, particularly XML are well-suited to developing interdependent heterogeneous systems that can function and share information.

Common Object Request Broker Architecture (CORBA) is another technology that facilitates object-oriented design and development that is relatively platform independent. CORBA is an open, vendor-independent architecture and infrastructure that computer programs use to interoperate over a network. A CORBA-based program, on almost any computer, operating system, programming language, and network, can interoperate with another CORBA-based program, on almost any other computer, operating system, programming language, and network.

Component Transaction Monitoring

The database sits at the core of the system, and all components of the system rely upon the integrity and reliability of the database. System components must be able to access the data and conduct transactions between one another and the database. A relational database management system provides services for managing data. However, the same type of services that relational databases provide for data must also be provided system-wide for components that interoperate with one another.

Component transaction monitors provide these services by managing the environment and providing components with the entire infrastructure required to support concurrency, transactions, and load balancing. The transaction monitor for APSS must support CORBA so that components need only to have a CORBA mapping defined by the Object Management Group to participate in the system [22]. Mappings have been defined for C, C++, Java, COBOL, Lisp, Ada, Python, and Smalltalk.

Proposed Architecture and Technologies

CORBA appears to be the most attractive technological approach to solving the system requirements discussed above, but there is no specification for a CORBA component transaction monitor [23]. Thus, there is no guarantee against vendor lock-in. An attractive solution is the Java Enterprise Platform (J2EE) [24], which specifies a server-side component model called Enterprise JavaBeans (EJB) [25]. The EJB specification enables implementation of the APSS using an EJB-compliant component transaction monitor. A different component transaction monitor can be used later if a different vendor is used. The EJB architecture can also incorporate CORBA objects and services. Some vendors, however, may not incorporate CORBA into their EJB implementation, so this vendor-specific feature must be considered when switching vendors if CORBA is required.

The EJB platform is designed so that components of the APSS system will be available for large numbers of users in mission-critical situations. The application server that hosts an enterprise system that uses EJB can automatically manage resources, concurrency, transaction, security, persistence, load balancing, and distribution of components. This means that the development of the APSS system can focus on the business logic, rather than low-level infrastructure issues.

CONCLUSIONS

The OpSim and APSS prototype systems have proven to be very valuable tools for evaluating the anticipatory planning and adaptive execution idea. The concept-level approach to plan representation and manipulation promises to make the system more robust and flexible. The distributed implementation of the plan description will enable many desirable capabilities, perhaps the most important of which is collaborative planning. Ultimately, this type of planning support system will allow commanders in the field to evaluate plans and decide on execution faster than the enemy.

REFERENCES

[1] Wass de Czege, H., Jr., Personal Communication (regarding Anticipatory Planning), October, 1999.

[2] Surdu, J. R., et al. 2000. “Simulation Technologies in the Mission Operational Environment.” *Simulation*, No. 74(3), March: 138 - 160.

[3] Fikes, R. E., et al. 1971. “STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving.” *Artificial Intelligence*, No. 2(3 (Winter 1971)): 189-208.

[4] Tate, A. 1977. “Generating Project Networks.” In *Proceedings of the 5th International Joint Converence on Artificial Intelligence* (Boston, MA, 22-25 August). 888-893.

[5] Russell, S., et al. 1995. *Artificial Intelligence: A Modern Approach*, Prentice-Hall, Englewood Cliffs, NJ.

[6] Tu, S. W., et al. 1989. “Episodic Skeletal-Plan Refinement Based on Temporal Data.” *Communications of the ACM*, No. 32(12), December: 1439-1455.

[7] Currie, K. W., et al. 1991. “O-Plan: the Open Planning Architecture.” *Artificial Intelligence*, No. 52(1): 49-86.

[8] Atkin, M. S., et al. 2001. “Hierarchical Agent Control: A Framework for Defining Agent Behavior.” In *Proceedings of the 5th International Conference on Autonomous Agents* 425-432.

[9] Eller-Meshreki, R., et al. 1996. “An Architecture for Planning with External Information Points in a Real-Time System.” In *Proceedings of the ACM 24th Annual Conference on Computer Science* 58-66.

[10] Hall, M. R., et al. 1998. “A Mission Planning Architecture for an Autonomous Vehicle.” In *Proceedings of the First International Conference on Industrical and Engineering Applications of Artificial Intelligence and Expert Systems* ACM Press, 582-589.

[11] Agre, P. E., et al. 1991. “What Are Plans For?” Published in *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, P. Maes, ed., MIT Press, Cambridge, MA, 17-34.

[12] Brooks, R. A. 1991. “Intelligence Without Reason.” Published in *Proc. 12th Intl. Joint Conference on Artificial Intelligence*, R. M. a. J. Reiter, ed., Morgan Kaufmann, Sydney, Australia, 569-595.

[13] Hayes, C. C., et al. 1999. “CoRAVEN: Intelligent Tools to Provide Multi-Perspective Decision Support and Data Visualization.” Technical Report UMN-IE-99-001, University of Minnesota, Minneapolis, MN.

[14] Porto, V. W., et al. 1999. “Evolving Tactics in Computer-Generated Forces.” In *Proceedings of the Enabling Technologies for Simulation Science III* (Orlando, FL, 7-11 April). SPIE, 75-80.

[15] Gelenbe, E., et al. 2001. “Simulation with Learning Agents.” *Proceedings of the IEEE*, No. 89(2), February: 148-157.

[16] West, D., et al. 1995. “Infrastructure for Rapid Execution of Strike-Planning Systems.” In *Proceedings of the 1995 Winter Simulation Conference* 1207-1214.

[17] Lee, J. J., et al. 1997. “Simulation Based Planning in Support of Multi-Agent Scenarios.” Technical Report 97-001, Computer and Information Science and Engineering Department, University of Florida, Gainesville, Florida.

[18] Davis, W. J. 1998. “A Framework for the Distributed Intelligent Control of Advanced Manufacturing Systems.” Draft, University of Illinois at Urbana-Champaign, Urban, IL.

[19] Davis, W. J. 1998. “On-Line Simulation: Need and Evolving Research Requirements.” Published in *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice*, J. Banks, ed., John Wiley and Sons, Inc., New York, 465-516.

[20] Gilmer, J. B., et al. 2000. “Recursive Simulation to Aid Models of Decisionmaking.” In *Proceedings of the Winter Simulation Conference* (Orlando, FL, 10-13 December). IEEE, Piscataway, NY, 958-963.

[21] James, J. R. 2000. “AFATDS and the JDBC: An Assessment of Vulnerabilites Associated with Data Replication / Distribution Mechanisms.” Technical Report ITOC-TR-200-202, Information Technology and Operations Center, Department of Electrical Engineering and Computer Science, United States Military Academy, West Point, New York.

[22] The Object Management Group. "OMG Specifications." Available at http://www.omg.org. Last accessed on January 2, 2002.

[23] The Object Management Group. "CORBA FAQ." Available at http://www.omg.org/gettingstarted/corbafaq.htm. Last accessed on January 2, 2002.

[24] Sun Microsystems. 2002. "Java™ 2 Platform, Enterprise Edition." Available at http://java.sun.com/j2ee. Last accessed on January 2, 2002.

[25] Monson-Haefel, R. 1999. *Enterprise Java Beans*, O'Reilly, Cambridge, Massachusetts.