

Dynamic Decision Support in the Advanced Tactical Architecture for Combat Knowledge System

Faisal Momen
Jerzy W. Rozenblit

University of Arizona
Department of Electrical and Computer Engineering
Tucson, AZ 85721
[\[momen,jr\]@ece.arizona.edu](mailto:[momen,jr]@ece.arizona.edu)

As modern military systems demand faster reactions and become more mobile, the difference between planning and execution will fade until the planning process appears to merge with the battle management process. Continuous planning systems must be fast, intuitive, and accurate. In particular, the amount of information will be overwhelming and the number of options unmanageable for many future tactical environments. The Advanced Tactical Architecture for Combat Knowledge System (ATACKS) has been designed to incorporate both visualization tools and intelligent algorithms to allow for rapid visualization and decision making in these military environments.

The work presented in this paper demonstrates how an external, intelligent system, in this case a system based on the Discrete Event System Specification (DEVS) framework, was adapted and successfully integrated with ATACKS to produce dynamic decision support for battlefield visualization in a distributed environment. The DEVS decision support application was designed to provide recommendations on the feasibility of proposed courses of action enacted in ATACKS. By querying the appropriate unit models, it derives the state of the current battle. Subsequently, DEVS uses its wargaming rules to formulate a Go or No-Go decision, which is communicated back to the commander working with ATACKS.

Keywords: Discrete event simulation, distributed computing, decision support systems

1. Introduction

The Advanced Tactical Architecture for Combat Knowledge System (ATACKS) [1] was developed as a commander's decision support tool designed to provide an abstract visualization of the battlespace environment and to allow the user to quickly create and execute major theatre of war and Stability and Support Operation (SASO) scenarios based on its library of 3-D elements. Developed in Java using the Java 3D library, ATACKS seeks to provide a simple yet potent user interface from which 3-D elements can be loaded and placed anywhere in the battlefield. ATACKS is geared toward increasing portability by modularizing and isolating the GUI and graphics rendering portions of the application, while improving the object-oriented class hierarchy and taking advantage of powerful

external support tools. In addition, the simulation engine has been improved and many new features including configural displays [2] have been added. Further details on these new features can be found in section 2.

The visualization engine, which is at the core of ATACKS, however, can only be extended to include a limited set of functionality. During its development, the war game rule base and the inference engine have been continually updated to handle new types of scenarios that mark a drastic departure from the major theatre of war operations that ATACKS was initially designed to accommodate. While it is difficult enough to keep pace with the rapid developments in the domains of military doctrine and warfighting, newer advances in the computer and cognitive sciences such as battlefield reasoning under uncertainty push technology requirements even further. To survive as a useful tool in this rapidly evolving environment, ATACKS must be able to interface with the various latest commercial

off-the-shelf products, leveraging its flexible object-oriented visualization base to display the intelligence derived from these external sources.

A majority of military simulation systems adopt the goal of providing a highly realistic representation of the battlefield. They facilitate decision making by virtually recreating the area of operations and providing a more natural view of the battlespace than would be possible using 2-D maps and overlays. Although these systems have loosened their dependence somewhat on underlying high-performance hardware (for graphics rendering or database services), they still tend to require extensive support in terms of time and effort to set up detailed and highly accurate scenarios. As a result, these fine-grained visualization systems are typically confined to specialized decision support applications such as battlestaff training or geospatial analysis, where the immediate evaluation of proposed planning options is not the primary consideration (for example, see <http://www.stk.com>). A number of the industry sponsored battlefield decision support tools that have emerged are more focused on providing an open architecture for integrating various low-level planning and execution tools, collaborating with commercial and national data sources (such as geographic, satellite, meteorological) and incorporating various data and visualization formats (www.webtas.com, www.viewcore.com). Although they are capable of providing decision support based on massive data sets containing information from a variety of high-fidelity sources, the need for rapid and effective evaluation of a scenario has become overshadowed.

ATACKS attempts to address this deficiency by providing a decision support tool that is tailored for rapid evaluation, response, and analysis of courses of action. From a visualization perspective, rather than attempting to provide the user with life-like or realistic rendering of the battlefield, ATACKS seeks to relieve the burden on the processing and information bandwidth by conveying only the most important aspects of the unfolding battle process. Raw data is compacted into abstractions and represented in such a way so as to be more meaningful to and easily assimilated by the commander. While human computer interaction (HCI) studies and cognitive experiments are required to arrive at such representations, the architecture of ATACKS was designed with the initial goal in mind of facilitating the substitution of various types of visual representation at run-time [3]. From a decision support standpoint, the ATACKS architecture has been designed to allow easy integration with external tools that facilitate a quick, simple, and high-level evaluation of a given scenario.

This paper describes the evolution of ATACKS from a tool that aids the commander in quickly generating and visualizing abstract 3-D battlespaces into a (multi-tiered) distributed scenario execution environment with decision support capabilities. Section 2 gives background on the history and evolution of ATACKS, goes over some of the current features, and discusses the redesigned components and some of the new technologies added. Section 3 introduces the area of decision support systems (DSS) and describes how an application based on the Discrete Event System Specification (DEVS) framework can be used to provide feedback to the commander on the validity of applied courses of action. Section 4 forms the most crucial part of this paper and discusses the challenges involved integrating a DSS with ATACKS. It provides the implementation details on how such a coupling was accomplished. Section 5 provides an example scenario where the commander is given a chance to test the DEVS-based DSS through a SASO-type scenario. Finally, section 6 provides some concluding remarks and directions for further research.

2. The Advanced Tactical Architecture for Combat Knowledge System (ATACKS)

ATACKS began as a “framework for testing various display strategies” [3] with the goal of facilitating understanding of the process of the battle as opposed to merely displaying events as they occur on a screen. This implies that the display of battlefield events undergoes some transformation to make the presentation more meaningful and closer to the user’s mental picture of battlespace processes. The architecture allowed display strategies to be easily switched, in order for researchers to test, through experiments, which representations were most effective in conveying the underlying battlespace process. The architecture was required to be flexible and extensible, which an object-oriented design naturally supported.

The prototype that was created based on the architecture was a 2-D visualization system built around the concept of a process-centered display [2]. Written in C++, the prototype successfully demonstrated all of the goals of the architecture using a simulator to provide the layers of data amalgamation and intelligence production while using the visualization layer prototype that was built to evaluate the various display strategies. This prototype was later extended to a 3-D interface and ported from the Silicon Graphics C++/Open Inventor platform to Java/Java3D. The advantages of a 3-D environment are that it is closer to the user’s mental picture of battlespace events and allows the user to view the events as they are being

presented in a number of perspectives that includes the traditional 2-D view. The 3-D environment also allows for a richer database or library of battlefield elements, allowing each element access to another plane in which to represent various properties about itself. An example can be found in the evolution of the basic friendly or enemy "Unit" whose 3-D structure allows it to represent at least four times the information, using only the faces of its cube, than a 2-D Unit symbol could present on its single face.

The driving forces behind the move toward the Java/Java3D platform were to leave behind performance and portability limitations of hardware and allow incorporation of the latest off-the-shelf 3-D development tools that would allow rapid production of robust object-oriented applications and scenes. SGI workstations initially provided the most feasible solution and the required development tools, but with the release of a 3-D library for Java, it has become possible to continue development of ATACKS on the Windows PC environment. The combination of Windows and Java currently offers better potential for cross-platform integration with external applications in addition to a greater variety of economical hardware configurations for both development and testing. The availability of Java and Java3D on all major platforms also ensures that the application will not be limited in the future by the current choice of implementation language and platform.

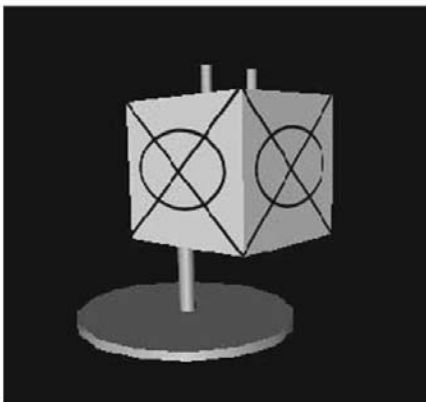


Figure 1. 3-D Unit representation in ATACKS

2.1 ATACKS Software Architecture

The requirements specifications for a visualization tool such as ATACKS tend to experience virtually boundless growth. Abstract symbologies, visualization concepts, decision support tools, and the types of scenarios are all continually being updated, fueling the expansion of ATACKS into new and different areas. The recent interest in stability and

support operations has spearheaded the development of multiple types of configural displays (CDs)—where previously there was only one—and has led to the integration of an independently developed SASO wargaming simulator [4]. The addition of new features can strain a software system that was not designed with reusability and flexibility in mind early in its design life cycle. With ATACKS, we have tried to rely on the sound object-oriented (OO) principles that have been successfully used in industrial software projects to reduce the need for spurious system redesign. The resulting OO architecture of ATACKS allows us to more easily integrate new concepts and functionality.

Hierarchies occur at many levels in ATACKS. All 3-D scene objects in ATACKS are derived from a common base class. The Object/Actor class provides the basic interface for all 3-D objects and implements the commonly required methods for building, transforming, selecting, deselecting, and editing an object's scenegraph. In addition, the interface for a container of Objects/Actors and all of its subclasses is also defined in the Object class. Subclasses of Object include Units, terrain elements, lines of defense, brigade and battalion boundaries, paths, the terrain, the grid and the composite class ActorGroup. The composite pattern allows a group of objects to be treated the same as a single object. The ActorGroup class, which inherits the interface of the Actor, implements the basic composite methods to add or remove Actors (and its subclasses, which are also Actors) from the group or to search and retrieve an Actor from the group by name. Using this strategy, we can have an overall Unit group comprised of two ActorGroups, enemies, and friends; and under each of those ActorGroups we can have the individual friendly and enemy Units. Based on this hierarchy, behaviors can be assigned to all the Units on the battlefield, just the friendly Units, all the enemy Units of type Infantry, or to any single enemy or friendly Unit. Design patterns such as the composite pattern are recurring OO software constructs that have been identified by experienced software engineers as useful, reusable solutions to common design problems [5]. They are used extensively throughout ATACKS.

The ATACKS class hierarchy has also been designed with an eye toward maximizing the modularity among the functional groups. Figure 2 shows the major classes that make up the ATACKS software system.

The classes to the right of the AttacksDirector (Java 3D scenegraph) represent new packages in ATACKS that allow us to incorporate various decision support mechanisms into the core visualization architecture. The Configural Display Manager collaborates with the Scenegraph which maintains all the Unit objects, to

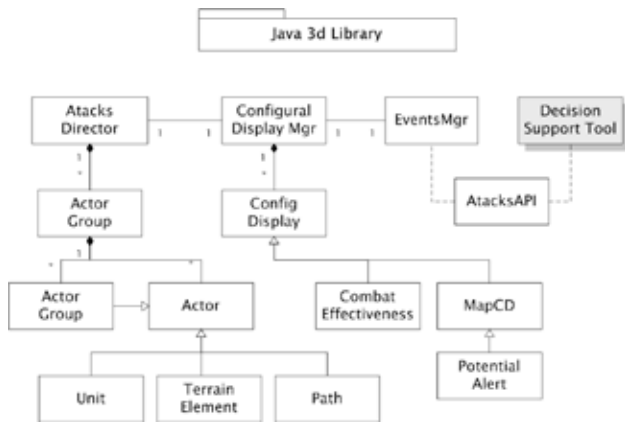


Figure 2. ATACKS architecture

coordinate the set of CDs for each Unit in a particular scenario. A discussion of CDs follows in the next section. Next to the Configural Display Manager is the Events Manager class, which was added to consolidate scenario event collection, generation, and propagation facilities. Through the ATACKS Application Programming Interface (API), the Events Manager can communicate events that occur within ATACKS to an external intelligent evaluator, which can investigate the impact of the event on the current course of action or provide any other meaningful recommendations to the commander using ATACKS. With a flexible API design, ATACKS becomes an open architecture to which a broad range of decision aids can be coupled, provided that the appropriate information translation layers are in place between the applications. Section 4 describes how an external intelligent dynamic decision support application was successfully integrated with ATACKS to provide feedback based on the commander's proposed courses of action.

2.2 Graphical User Interface and Configural Displays

The present version of ATACKS has fulfilled many of the requirements and suggestions put forth in the earlier designs [3]. Most of the enhancements can easily be recognized as changes in the ATACKS graphical user interface (GUI) and improvements in how the user interacts with the system. CDs of various types have been introduced to alert the commander to certain consequential events as they are played out in the scenario. Collectively, the CDs portray the process of the battle, while the battlespace window depicts the actual events underlying the battle process. Additionally, many important abilities from the perspective of using ATACKS as an experimental tool, for example, timing and storing user responses

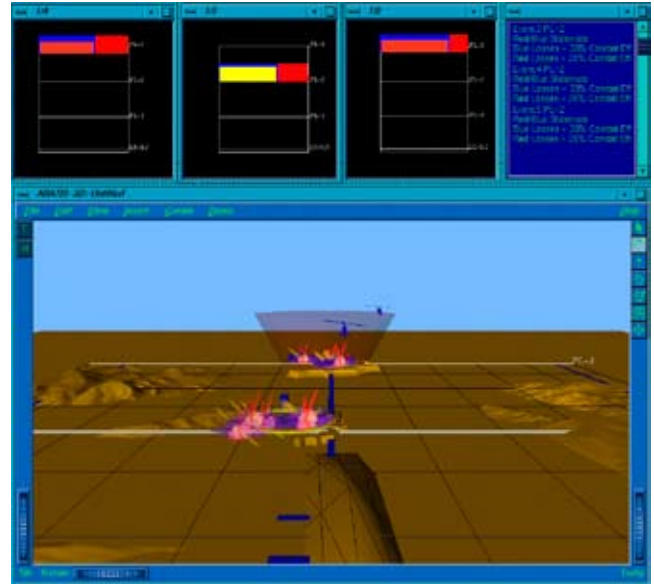


Figure 3. Combat effectiveness configural display

to scenario queries, have been incorporated into the design.

Configurable displays present the user with abstract representations of key events as they occur in the battle. Different types of CDs were designed to display various aspects of the war-gaming and battle process. The basic CD designed for use in major theatre of war scenarios is shown in Figure 3. The chalked rectangular outline delineates the battle grid whose dimensions can be adjusted by the commander through the ATACKS GUI. Also represented in white are the Phase Lines (PL), the Line of Advance (LOA), the Forward Edge of the Battle Arena (FEBA) and other similar command and control features. The purpose of these outlines is to provide the viewer of the CD with references as to the position and progress of the Units along the battlefield. The position of the multicolored bar is tied to the location of the Units on the battlefield. In addition to position, the CD also portrays the combat effectiveness of the blue force and the red-blue combat ratio in the case where the friendly Unit encounters an enemy. As the Unit comes into contact with and engages enemy forces, its combat effectiveness diminishes and the green bar will change to yellow and finally red, indicating the Unit is no longer combat effective. Since a CD is created for each friendly Unit before the scenario begins execution, the combined CDs provide an at-a-glance indication of the status and progress of the Units according to the battle plan. If the commander is interested in the CD of a particular Unit, a click of the mouse on that Unit will cause the CD for that Unit to become highlighted.

The ATACKS visualization window and CDs were designed to serve as useful tools to allow a military user to quickly gain situational awareness and understanding of a scenario. Many times, however, a commander may wish to adjust certain elements of a course of action to quickly investigate their impact on the execution of a scenario before accepting a particular solution. Decision support tools help users make informed, objective decisions on strategic or operational issues, such as picking one course of action over another. By modeling the dynamics of the battlespace as discrete events, and defining an interface through which other programs can learn or be informed about the events, ATACKS can accommodate a large array of decision support tools while maintaining its simple and modular design philosophy. Users will then be able to interact with the system through the visualization interface as if it were an analysis tool, rather than simply be observers of the executing scenario.

3. Dynamic Battlespace Modeling

The decision support system (DSS) introduced in this paper is based on the DEVS framework [6, 7]. In the DEVS framework, objects—e.g., enemy and friendly Units (battalions, platoons, etc.)—are represented by models. The models are characterized by their input, output, and state sets, and a state transition function. The input set defines all the messages the model is able to receive and the output set defines the response messages the model may signal to the outside world. Inputs that arrive from external sources, i.e., other models, may trigger a change of state for the model receiving the input. In addition, an internal transition function can be defined that regulates the change of states in the absence of external inputs. These simple atomic models can then be coupled with other models to create more complex coupled models, which themselves can be coupled with other atomic or coupled models. This property of closure under coupling also allows DEVS simulation models to be readily mapped to high level architecture (HLA)-compliant modeling and simulation environments [8, 9], opening the way for interaction with a range of decision support tools that adhere to the Defense Modeling and Simulation Office (DMSO) standard.

Models in the DEVS environment are event driven. Most of the interactions between the models occur because events are exchanged, with the exception of those events generated by internal transitions. For example, in the ATACKS DEVS simulation, the Engine model sends *move* messages to the WarGamer model, which interprets the message as an event that it needs to respond to and responds accordingly.

Conversely, an internal event would be when a Unit model discovers it is low on fuel and places itself in a *not_ready* state. In order to use ATACKS with the DEVS simulation environment, the events that need to be exchanged between the two applications need to be defined. Currently, the progress (position, combat effectiveness) of friendly Units on a particular course of action (COA) is tracked by the CDs. Whenever a friendly Unit encounters an enemy within the vicinity of a phase line or line of defensible terrain, the war game rules in ATACKS are triggered and a message is sent to the consequences display to output the results of the encounter. For DEVS to be able to provide decision support, it needs to be notified of these events as they occur during the course of the simulation.

Figure 4 is a representation of how the various tools, ATACKS, DEVS, and FOX, a two-sided course of action generator and war gamer [10], fit together in a distributed architecture. FOX uses its genetic algorithm and COA domain expertise to generate multiple friendly and enemy COAs. The COA is output as an XML file and transferred to the system running ATACKS. ATACKS then uses its XML parser to translate the FOX COA into a local scenario representation. While ATACKS is simulating the COA, the user is free to inject into the simulation external events that FOX either did not or cannot evaluate using its fitness function. Political events such as demonstrations and surprise elements such as booby traps or ambushes, with which FOX was not set up to interact, can be inserted into the simulation by a user to further test the resiliency of the generated COA. Since FOX has completed its job by providing a COA for visualization in ATACKS, decision support for

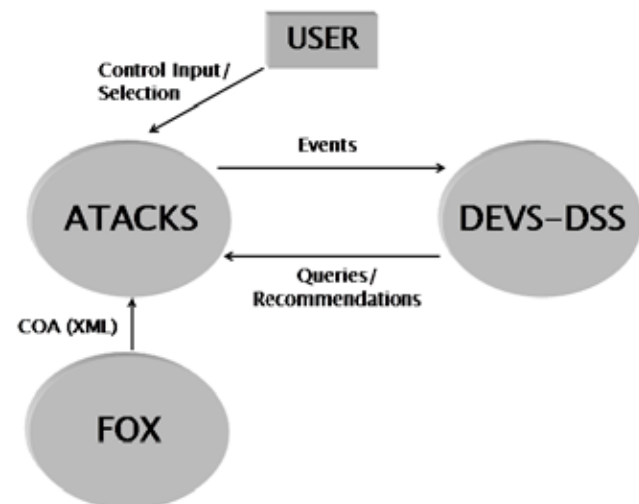


Figure 4. Distributed architecture overview

these new user-generated events needs to be obtained from an external war gamer and inference engine. The DEVS decision support tool described in this section was designed for precisely that purpose.

3.1 DEVS DSS Models

Within the DSS, collaboration takes place between the WarGamer and the Unit models. There is one Unit model for each friendly or enemy Unit in ATACKS. In addition, an Engine model coordinates the actions of the DSS, and interfaces with the outside world. The following sections provide a description of these models and how they were coupled to produce an abstract DSS for use with ATACKS.

3.1.1 DSS Interface Engine

The Engine primarily serves one purpose: it interfaces with ATACKS to receive notification of events on a periodic basis or request further information that is required by the WarGamer model for evaluation. The WarGamer queries the Units to find out if the grid location representing the destination of the move is occupied by an enemy. If so, damages are calculated and broadcast to the Units—only the matching Units subtract the damage amount from their strengths. Based on the results of this attrition to both sides, a recommendation for the move can be deduced that will be sent back to ATACKS to alert the commander. The Engine model initializes to the ready state and times out in `delta_1` to the waiting state. As it transitions to the waiting state, it begins waiting for a *move* event to arrive from ATACKS. Once the move event is received, a message is sent to the WarGamer containing the name of the Unit that the commander wishes to relocate as well as the change desired in the X/Y-direction for that Unit.

After sending the move to the WarGamer, the Engine waits for a response. If none is received an error message is printed, and the simulation continues with the next move event that is received. In most cases however, the WarGamer responds to the *move* message by returning the name of a Unit and the recommendation for the change in its position. The Unit for which the WarGamer sends back the move response does not necessarily have to be the same as the Unit whose move the Engine asked it to evaluate. The specifics of which Unit should be moved is a function of the `ComputeBattleResults` method of the WarGamer. Once the Engine receives the move event from the WarGamer, it goes into its sending phase to broadcast the move to all the Units, leaving it to the Unit to decide if the message applies to it. (This is accomplished by passing the `UnitName` as the second

field in the message, the first field being the command itself.) This is done primarily so that the Unit models have an updated picture of what is transpiring on the battlefield. The recommendation is then sent back to ATACKS through the interface mechanism.

3.1.2 WarGamer

The WarGamer receives moves from the Engine for evaluation. Since it is only given the name of the Unit and the desired change in its position, it broadcasts the `UnitName` and waits in state *waitingPosn* for the matching Unit to reply with its x and y coordinates. Once it receives the response from a Unit, it uses the received position values and the desired change given to it by the Engine to calculate the destination of the move. Now all that remains is to check if there is an enemy at the destination location. The calculated destination moves are sent to all the Units and only those Units which are of type “enemy” and whose position matches the advertised destination coordinates respond. The strength of the responding Units is sent in the reply and used by the WarGamer to calculate the friend/enemy force ratios. The WarGamer contains a simplified rule base that assigns attrition to both sides based on the force ratio. If the ratio is unfavorable to the friends, an alternate move should be calculated and returned to the Engine. However, in this simplified model, the WarGamer simply chooses to ignore the response step, causing the Engine to assume that an error occurred in the WarGamer and to proceed with the next move in the scenario. This achieves the same function as returning a move recommending the Unit to stay where it is.

If there happen to be no enemies at the destination location, the recommended move is calculated and, in this case, it is the same as the original sent from the Engine. Since there is no engagement and consequently no attrition, the `sendDamage` phase is bypassed advancing directly to the `sendMove` phase, which sends the recommended move response to the Engine.

3.1.3 Units

The Unit is the most basic and simple of the models. It waits in the ready state until it receives an external event. The first field of the message contains the command to be performed on the Unit; for example:

- If the command is *move*, the Unit first checks to see if it is alive. (A Unit is dead if its strength drops below a minimum threshold, `minimumStrengthToLive`.) If alive, the Unit updates its position variables to reflect the changes dictated by the requested move.

- If the command is *sendPosn*, the Unit checks to see if the name contained in the second field of the message matches its name. Once again it checks to see if the strength is greater than zero, i.e., if there is still life left in the Unit to perform move operations. If it has any strength, the Unit responds by sending its position and strength. If the strength is less than zero, a message is printed informing the system that the request befell a deactivated Unit.
- If the command is *sendEnemy*, a quick check is performed to see if the Unit is of type enemy and if its position variables match the received location coordinates. If these checks are satisfied and the Unit is alive, i.e., capable of engaging in battle, then it responds by sending out its strength to the WarGamer. The WarGamer will wait to accumulate the strengths of all enemies located in the vicinity of coordinates it broadcast in order to determine the overall enemy force ratio, if more than one enemy responds. In addition to the strengths, the names of the responding enemies are also stored in an array by the WarGamer for use in the next step, which is the dispensation of damage to all committed enemy and friend Units.
- Finally, if the command is *attrition* and the name sent matches the name of the Unit, the sent damage is subtracted from the Units strength. If the strength dips below the minimum life threshold, the state of the Unit is changed to dead.

In a simulation, the models of the Units should behave as described above. Units only respond to commands given by a superior, in this case, the Engine model, which could be considered the equivalent of a commander in the field. The WarGamer could be thought of as the battlestaff, analyzing the suggested moves of the Engine (commander) and recommending alterations and calculating damages accrued. The ATACKS simulator would be responsible for the visualization of the battlespace and coordination of the enemy and friendly forces in a scenario, while a rule-based inference model would validate suggested moves and calculate any necessary attrition to the blue and red forces.

3.2 Coupling

The coupling between the atomic models discussed above is shown in Figure 5. The Engine and WarGamer communicate with the Units via broadcasting—usually some information (e.g., the name of a Unit) is sent in the message that allows a Unit to decide if the message is intended for itself and, if so, to respond accordingly.

The coupling basically represents the information shown, namely, which output port of which model is attached to the input port of another model. For example, one can see that the “UnitsOut” output port of the Engine is connected to the “in” input port of model Friend1. This coupling is then repeated for all Units.

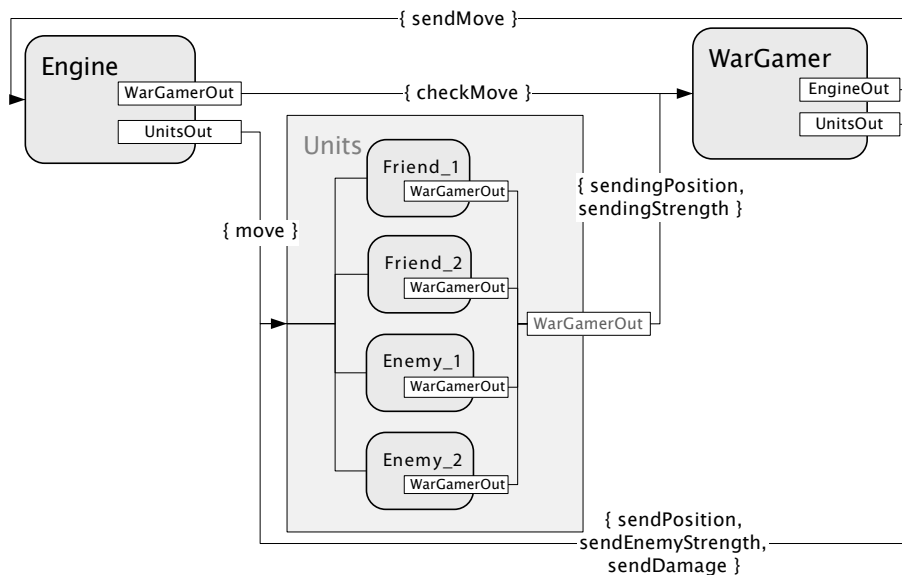


Figure 5. Coupling between DEVS-DSS models

4. Integration of Dynamic Models with ATACKS

As described in the previous section, the WarGamer in DEVS DSS queries the state of the Unit models and, based on its war game rule base, reaches a decision regarding the move in question. In reality, the Unit models that are queried need to supply the WarGamer with an accurate account of their status based on what is happening in the simulation within ATACKS. In other words, the Unit models in DEVS DSS need to query the status of the actual ATACKS Unit elements that they represent so that the WarGamer's recommendation is based on the most recent factual data. Although the DEVS DSS is written in JAVA and runs in the Windows environment, the same cannot always be said for other third-party tools that we may wish to integrate with ATACKS. Such tools need to communicate and obtain information from ATACKS that can potentially run on another JAVA platform. The DEVS DSS outlined above, implemented on the DEVS/HLA platform, can be used as an intermediary to interconnect other HLA-compliant support tools (i.e., other tools that run on the HLA runtime infrastructure) with ATACKS. If greater control over the modeling and simulation environment or interoperability with non-HLA-compliant systems is desired, a DEVS middleware based distributed simulation environment implementation can be used [11, 12]. The following sections describe how communication between two applications was achieved using the Common Object Request Broker Architecture (CORBA) and how, in this instance, CORBA Interface Definition Language (IDL) was used to define the interface between ATACKS and the DEVS DSS.

4.1 Middleware Selection

Decision support tools provide a very useful resource for commanders in helping to improve the quality of their decision-making process. Battlestaff make heavy use of overlays in their terrain maps to help them plan COAs. Many of these functions are now being taken over or enhanced through the use of computer-based tools. For example, FOX uses a steady-state genetic algorithm to generate thousands of COAs and then narrows the choices down to the few best while ensuring that the selected options are sufficiently different from each other. It presents the choices to the user who ultimately decides which COA to select for execution. Incorporating such tools directly into ATACKS may be a worthwhile undertaking, but FOX is only one example of the kinds of decision support tools that are available. Attempting to incorporate

every interesting DSS that is encountered into ATACKS would be extremely difficult if not impossible to achieve. In any case, there is no guarantee that the tools would even support one another, for instance, if they use differing thresholds for engagement of an enemy Unit. The solution demonstrated here would be to use the different tools but, rather than code them to be modules within ATACKS, to define an API that any external tool could use to communicate with ATACKS.

Moreover, even though ATACKS runs on a generalized platform, interoperability issues invariably arise when trying to interface with programs written for different target environments. CORBA is a communication medium that is language independent, as well as platform neutral, while being available for a wide range of programming languages and platforms. The Object Management Group (OMG) adopted CORBA as the standard infrastructure for applications that need to work together over a network [13]. The specification is vendor neutral and independent of any implementation language since all products based on the specification must support or use the standard Internet Inter-Orb Protocol (IIOP) to inter-operate with each other.

CORBA enforces adherence to three defined standards: the OMG Interface Definition Language (OMG IDL), the Object Request Broker (ORB), and the standard IIOP. The entire architecture itself is object oriented, and the interface for each CORBA object is defined using IDL. The IDL interface describes what operations an object can perform as well as what the parameters for those operations are. The same IDL file can be compiled on two different machines into any of the currently supported high-level languages including JAVA, C, or C++ through standardized mappings.

The files generated by the CORBA IDL compiler include client stubs and server skeletons. These classes define the operations that need to be implemented by the actual application client and server as specified in the original IDL file. As long as these basic operations are implemented, any CORBA-compliant client will be able to invoke the IDL defined operations on any CORBA-compliant server. The generated stub and skeleton then serve as proxies for the local client and server, respectively, taking care of whatever is needed to get a local method invocation through the network to an object that can correctly handle the request.

4.2 Interface Design

Designing the interface between ATACKS and DEVS using CORBA requires careful consideration in determining what information needs to be shared

between the applications. At the least, some information regarding a Unit, e.g., its designation, position, and status, would be required by the WarGamer. In ATACKS, every Unit that has been assigned to a path automatically becomes associated with a sensor that detects intersections of that Unit with other enemy and friendly forces whenever the Unit's translation is updated. Consequently, each friendly Unit maintains a list of the close-by or engaged enemy and friendly forces. The WarGamer in the DEVS DSS uses the friendly to enemy combat ratio in its rules to determine attrition and arrive at a recommendation, so the intersecting enemy and friendly Unit information would also need to be communicated across the applications. Finally, a method is provided to retrieve the current status of all the active Units in ATACKS in order to refresh the information contained in the DEVS Unit models so that they are up-to-date before any inferences are formed. This final operation can be used in lieu of multiple calls for individual Unit information if the bandwidth availability is a primary concern or if the data needs to be packaged in a structured format such as XML for processing in the decision support layer.

In addition to the types of information described above that are paramount to any tool designed for analyzing and providing feedback for events triggered in the battlefield, scenario generators such as FOX require a whole new set of interfaces, e.g., to populate the battlefield with terrain or Units, assign behaviors to objects, and so on. The preliminary Application Programming Interface for ATACKS (ATAKKS API) was developed with methods that interact directly with the ATACKS Scenegraph class and provide a means to fulfill this need. The methods that make up the API are listed in Figure 6.

4.3 Integration Revisited

The collaboration framework for ATACKS and DEVS DSS is shown in Figure 7. On either extreme lie the



Figure 7. Integration overview

applications that we are trying to integrate. The ATACKS-side CORBA server is linked to ATACKS on one side through interprocess communication (IPC), and to the DEVS-side server on the other through the ORB. IPC allows one process to exchange messages with a second process on the same machine. In its most basic form, a parent process spawns a child process and they are able to communicate back and forth through a one-way and sometimes two-way stream [14].

The ATACKS-side server, which is executed independently of the ATACKS visualization component, listens for messages in the IPC message queue after registering its interface objects with the ORB and binding with the DEVS-side server. The DEVS-side server must already be running at this point on any Windows PC machine or the ATACKS-side server will fail to bind with it and exit, raising a CORBA_NO_IMPLEMENT exception. Whenever an ATACKS event message is received, the ATACKS-side server exits the IPC loop—that is, it stops actively listening for messages from ATACKS on the message queue—and forwards the event notice to the DEVS DSS using the DEVS object reference that is acquired earlier during the binding state.

Once the message has been sent to DEVS DSS, it is possible that further information may be required from ATACKS before the DSS can determine a recommendation. As a result, immediately after forwarding the event to DEVS, the ATACKS-side server invokes the `impl_is_ready()` method on the ATACKS-

```

void createUnit (String shapefile, String unitName, float[] position)
void createTerrainElement (String shapefile, String name, float[] position)
void addBehaviorToUnit (String unitName, String behavior)
void createPath (String pathName, float[][] coordinates)
void moveUnitOnPath (String unitName, String pathName, float[] speedProfile)
String getAllUnitInfo ()
String getUnitInfo (String unitName)
String getEnemies (String unitName)
String getFriends (String unitName)
  
```

Figure 6. ATACKS API

side CORBA object to put it into the CORBA event loop or waiting state. Then, whenever a message is received from the DEVS-side server, whether it is the final recommendation for the decision or just a request for further information, the ATACKS-side object will be able to receive and respond to the call. Since the ATACKS CORBA server runs as a separate process, it does not interfere with the execution of the scenario in ATACKS, which is free to handle further user inputs. Any events that are generated in the meantime—that is, while the ATACKS-side server is listening for requests—simply become queued and wait there until the server is free to read and dequeue them for processing.

Once the final recommendation is received from the DSS, the ATACKS-side server again repeats the process of reading the next message off the queue, forwarding it to DEVS DSS, and so on. However, when it receives a request for further information, the type of the request is decoded and a second message queue to ATACKS is

opened. The request type is queued onto the message queue and the server begins listening for a response on the first message queue. A callback function installed with the ATACKS Events Manager class is responsible for periodically checking this second message queue for requests. When it finds one, it makes the necessary calls to the ATACKS API, which returns the desired information. This information is again queued into the first message stream between ATACKS and the ATACKS-side CORBA server. When the information is received by the ATACKS-side server, it is forwarded again to the DEVS-side server that requested it, and the ATACKS-side server returns again to the wait loop state. The sequence diagram in Figure 8 shows some of the typical steps that arise in the interaction of the various components.

In some preliminary testing of the distributed DSS architecture, a predefined scenario was executed and various events were generated to test whether there were any conflicts (e.g., where messages are sent out

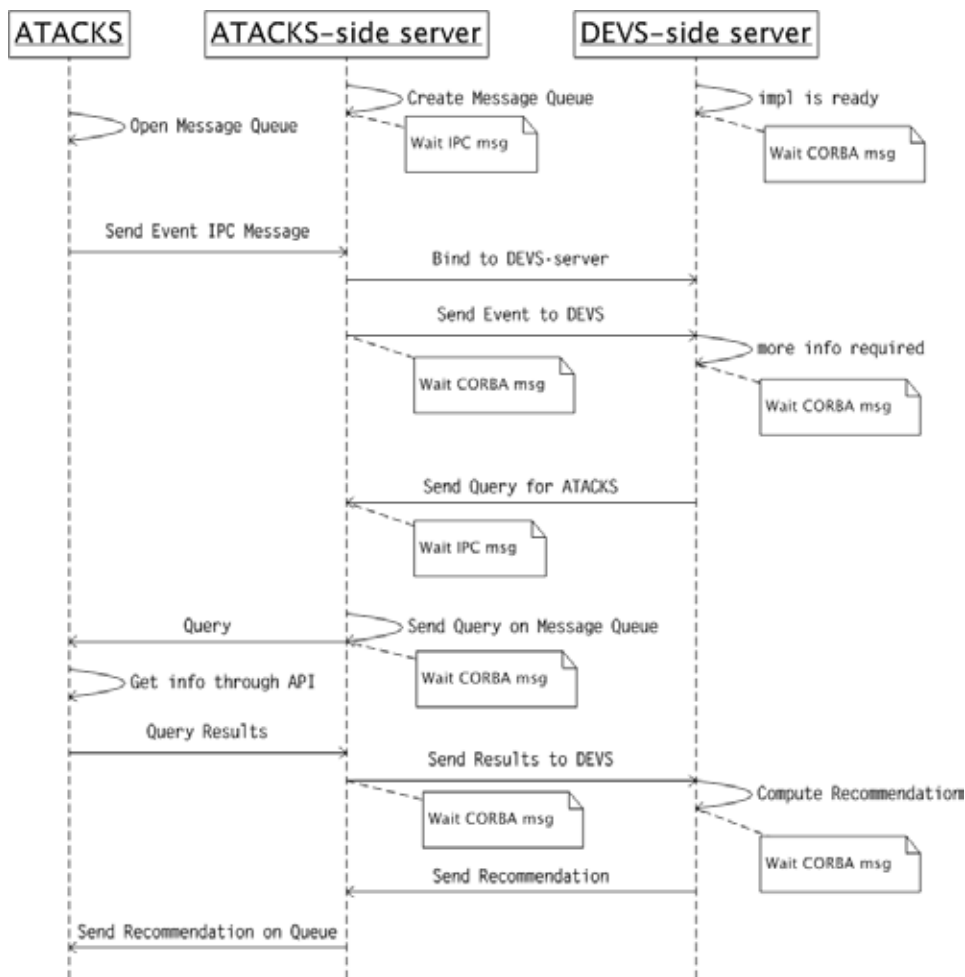


Figure 8. Interaction and sequence diagram

of sync) or bottlenecks. The results from the case study that were used to demonstrate the efficacy of our solution are described in the next section.

5. Case Study

In order to demonstrate how the DEVS DSS that has been integrated with ATACKS provides added value to the commander as a tool that enables him or her to evaluate various maneuver options, we will consider a common hypothetical scenario based on a SASO that was previously designed with the aid of military domain experts. ATACKS and DEVS DSS and the CORBA servers are executed on their respective hosts on two different computers connected via LAN.

5.1 Brigade Combat Team in SASO

The tactical scenario that will be used in ATACKS is brigade-level SASO set in a terrain environment that is intended to approximate a region in southwestern Asia. Terrain and man-made features, selected from the ATACKS icon library, are added to the grid at their appropriate locations. Mountains, roads, and an airfield are placed within the grid at their exact geographic locations. Buildings of various types and roads may be added to the urban site from the ATACKS icon library.

The terrain shown in this demonstration depicts the generally open vicinity of a notional provincial capital. The area of operations extends from the vicinity of the capital to an international boundary. Mountains in the area begin to converge toward the highway bridge that sits astride the international boundary. Command and control features are also added from the library. The scene that has been built for this demonstration includes brigade-, battalion-, and company-level symbology for Units ranging from mechanized infantry companies to an aviation battalion.

Paramilitary platoons are cited in the operations area using appropriate red symbols. A U.S. Army brigade combat team with a divisional support package is used to complete the assigned initial mission of securing the airfield near a provincial capital adjacent to the threatening neighbor. The ATACKS demonstration simulates a mechanized infantry company securing the airfield allowing for the insertion of the brigade command post, the remainder of the brigade combat team, and support Units. When successfully inserted and the airfield secured, the provincial capital is to be secured by one battalion while another is to separate the contending local factions, one loyalist, the other an opposition faction supplied and funded by the hostile neighbor.

This demonstration shows three distinct phases of the SASO: phase 1 – insertion, phase 2 – separation, and phase 3 – support. The initial phase, insertion, is completed when the entire brigade combat team complement has arrived in the operational area.

The separation phase begins as friendly Units leave the airhead and begin to encounter both loyalist and opposition forces. A reconnaissance screen is established between the provincial capital and the international boundary. Reflecting activities required by the separations phase of the operations order, company-sized elements move to execute their assigned missions.

The brigade commander wishes to develop a simple idea for the support phase of the operation. The brigade has been tasked to position a force forward and secure the bridge along the international boundary. The commander wishes to assign the mission of securing the bridge to the first battalion. ATACKS allows the commander to simply select the desired battalion icon from the current operation and place it in the future operations scene at the bridge site. The desire to move that battalion is captured by the DSS and validated according to current operational data. The first battalion has apparently taken more than a few casualties and is low on fuel. The CD reflecting that battalion's weakness appears on the future operations panel allowing the brigade commander to use another battalion. By deleting the first battalion, the commander may wish to explore conducting an air-mobile operation to lift all or part of the third battalion to the bridge site. The commander simply drags the aviation battalion symbol to the area, as well as the third battalion symbol. Further, the commander may position an artillery Unit midway in the extended area of operations near a village along the highway. The decision aid also checks these projected moves, reporting if the designated Units are able to complete the proposed missions.

5.2 Scenario Execution

To begin execution of the particular SASO scenario under discussion, the user selects a previously created scenario file from the list of stored scenarios or creates a new scenario using ATACKS. To create a new scenario, the user may insert elements from the 3-D symbol library, draw command and control features such as boundaries and avenues of approach directly on the grid, and specify paths for the Units to move along (optionally specifying a speed profile). Units can be assigned a range of behaviors, such as color, symbol, designation or affiliation, speed, strength, etc.; and all symbols can be geometrically manipulated (translation, rotation, scaling, etc.) to conform to the

users requirements. In this example, when the scenario shown in Figure 9 is loaded, all the graphical elements such as terrain elements, enemy and friendly Units, lines of defense, and phase lines appear, covering the bare grid. Once a scenario has been designed or loaded, the user can execute the scenario through commands on the ATACKS GUI menu. An animation engine is then activated which moves Units along their paths and brings up CDs that summarize the battle process. As the Units progress through their assigned paths in the pre-defined COA, some engagements with the enemy forces are encountered. These engagements are represented by an abstract *chaos* symbol. The results of these elementary engagements are derived from

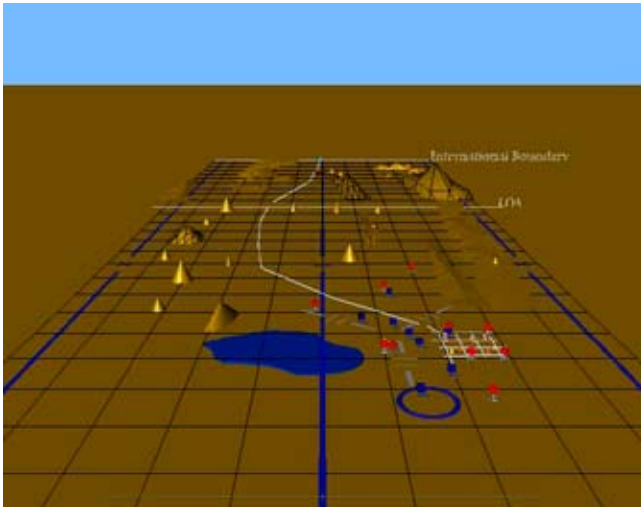


Figure 9. SASO scenario in ATACKS

the simple force ratio rules currently embedded into ATACKS and are reported in a separate CD. Figure 10a shows the first battalion engaged with an enemy force and suffering a loss in its combat effectiveness as a result. While the scenario is executing, any Unit that is not assigned to a path can be selected by the user and positioned anywhere in the battlefield. Manipulation of such a Unit by the commander constitutes an event that is collected, managed, and distributed by the ATACKS Event Manager and signifies that the user wishes to merge the ongoing execution activities with replanning.

In the first case, when the commander selects the first battalion, as shown in Figure 10b, an event is generated and sent out from ATACKS containing the name of the particular Unit as well as its current position. Again when the Unit is deselected, a second message containing the final coordinates is transmitted. These messages are first transmitted from ATACKS, where the event was generated, to the ATACKS-side CORBA server on the same machine, via message queues. Once both messages (Unit selection/deselection) denoting a change in the Unit's status are received, an appropriate CORBA message is sent across the network to the DEVS tool. The DEVS-side CORBA server decodes the message and, in this case, writes out the data it receives in the message—namely the name of the Unit that was updated by the commander in ATACKS and the coordinates to which the Unit was moved—to the DEVS input file. The Engine in the DEVS decision support tool, which searches the input file for new inputs, eventually picks up the new move and transmits

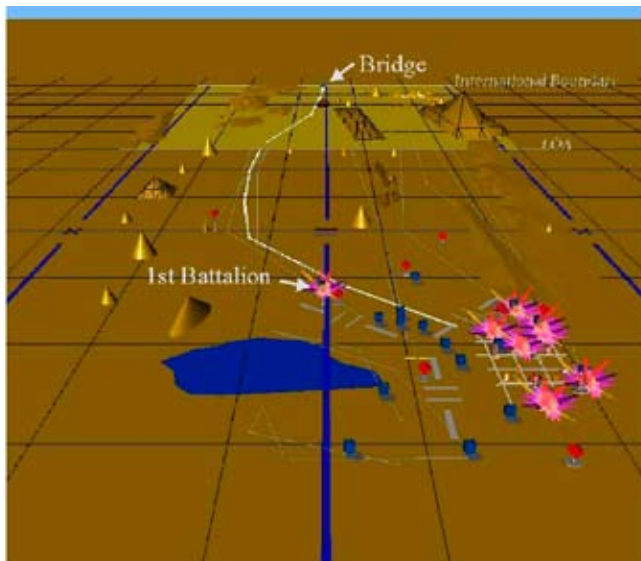


Figure 10a. ATACKS Event: First battalion engaged with opposition force

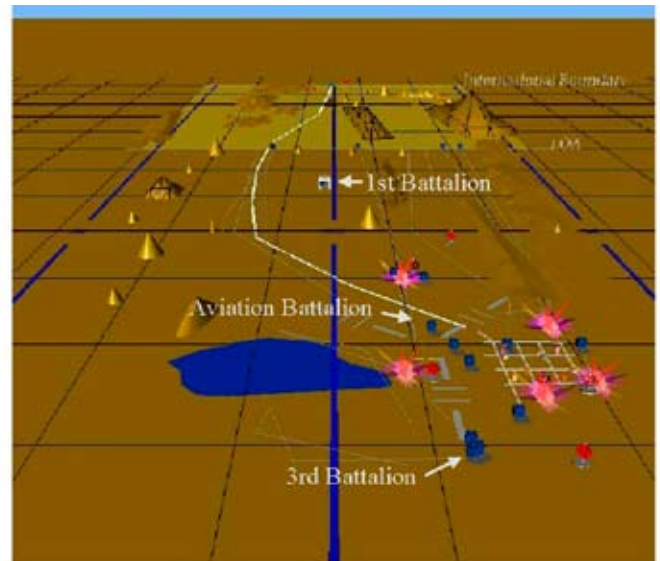


Figure 10b. ATACKS Event: First battalion being dragged to bridge

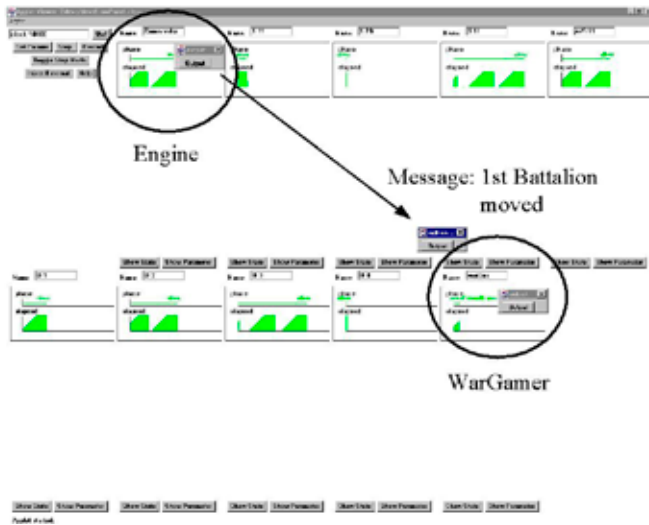


Figure 11. Notification of ATACKS event in the DEVS decision support tool

the move in a DEVS message to the WarGamer for evaluation, as shown in Figure 11.

The DEVS-side CORBA interface knows that once the new input is processed by the decision support tool, the WarGamer may query the Unit models for their strength and position parameters. In order to keep the information in the DEVS models as up-to-date as possible, whenever the DEVS CORBA server receives an event from ATACKS, it automatically sends an invocation back requesting the current status of each Unit. The information that it receives from ATACKS is then written out to the appropriate data input file for each of the corresponding Unit models in the DEVS tool.

In this case, the WarGamer in the DSS first checks to see if the selected Unit has sufficient resources to complete its assigned task. The DEVS-side server is contacted and it requests ATACKS, through its CORBA interface, to provide the supply factors for the given Unit. The desired information is obtained through the appropriate call to the ATACKS API and returned to the decision support tool. Before the war-gaming rules are triggered, a quick check determines that the Unit's combat effectiveness has dropped below an acceptable threshold, most likely from a previous engagement. Therefore the DSS skips further evaluation using its war-gaming rules and returns a NO_OK insufficient combat effectiveness message, which is displayed on a popup window in ATACKS.

Once the commander is notified that his or her plan was rejected by the DSS, he or she can either leave the Unit where it is or proceed with other options. In this example, the commander removes the first battalion

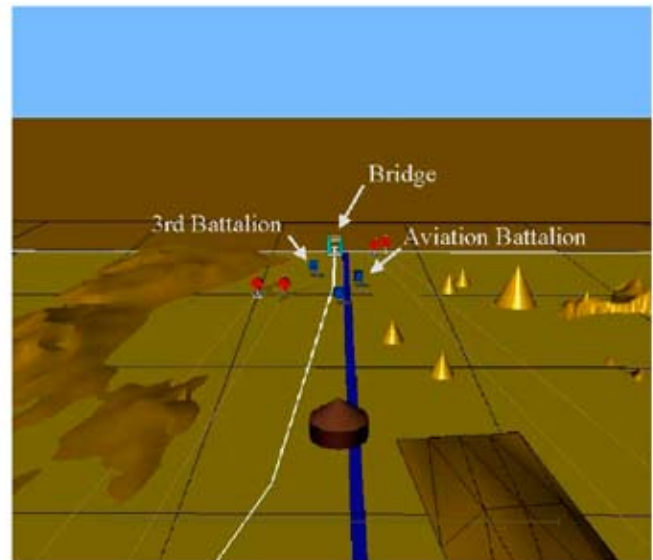


Figure 12. Selection of third and aviation battalions as an alternative action

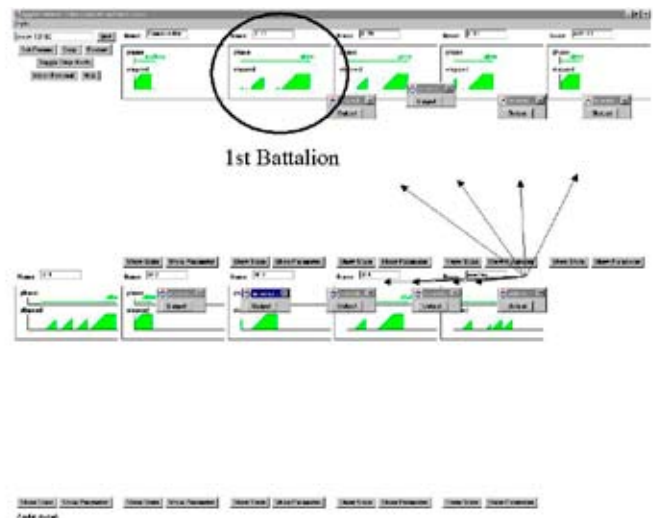


Figure 13. Broadcast query in the DEVS-DSS

and moves the third battalion and the aviation battalion to the future operations area as shown in Figure 12. Once again events are generated in ATACKS and relayed to the DSS through the CORBA interface. This time the selected Units have a sufficient combat effectiveness rating to go through with their assigned missions, so the WarGamer sends a broadcast message to retrieve the strengths and positions of all enemy and friendly Units in the vicinity of the bridge site; see Figure 13. The four opposition platoons located near the site respond with their strength parameters

allowing the WarGamer to carry out its rule-based inferencing. Since the friendly battalions outnumber the enemy platoons, the result is favorable to the friendly forces and this outcome is communicated back to the commander using ATACKS. At this point the commander has effectively evaluated two minor deviations to the COA that is being executed in the scenario and has received feedback from the decision support tool in both cases regarding the projected moves. The first option was rejected by the decision aid, so by creating paths that fulfill the objectives of the second option and assigning the selected Units to these paths, the commander can easily observe the consequences of his or her modification to the original battle plan.

5.3 Validation

The DSS for ATACKS demonstrates through the preceding scenario that it adequately meets the needs of the commander evaluating elementary deviations to proposed SASO maneuvers in real time. However, as mentioned, the war game rules that were used to evaluate the proposed changes in the courses of action are fairly primitive—relying solely on combat ratios of the opposing forces—and can be quickly computed in real time. Newer war gamers incorporate an extensive array of factors in producing and evaluating their courses of action. Current war-gaming systems such as Sheherazade, for example, take into account the presence of information operator units (such as media or refugees), the demographic makeup of the locale or region, and even the regional attitudes or outlook. Consequently, generating a few distinct options through its GA takes a few hours (as opposed to under a second with the current DSS); and thoroughly evaluating an option on the fly, as the ATACKS DSS attempts to do, would result in significant wait times for the user. The major bottlenecks in this case however, lie primarily in the algorithmic performance of the war gamer, and potentially in the bandwidth limitations of the technologies underlying the implementation. The first issue can be addressed by exploiting concurrency in the algorithms and through the use of parallel computing hardware, to speed up the generation (and evaluation) of courses of action. Indeed, high-performance parallel and distributed implementations of war gamers that complement today's highly time-sensitive military decision making environment have already been proposed [15]. A similar approach can be taken to hold off the bandwidth bottleneck, by having multiple threads perform the task of receiving queries (from the DSS) or forwarding the requested information (from the scenario executing in ATACKS). A thorough field test

of the current ATACKS DSS with battlestaff personnel should provide valuable feedback in helping identify some of these potential bottlenecks and should help discover any unexpected conditions within the architecture.

For the DEVS DSS used in this case study, an experimental frame setup provides a straightforward approach in the verification of the system. In an experimental frame, a generator (DEVS model) simulates correct and incorrect inputs to the system under test (e.g., scenario updates enacted in ATACKS) while an acceptor collects the results or outputs. A transducer analyzes the process, reporting any abnormalities in the output. The DEVS DSS was able to provide accurate feedback (go/no-go results) for user input at higher rates than typically observed or expected (slightly over one scenario update per minute, for instance). An experimental frame setup can be used in a similar fashion to evaluate future decision support tools and determine the limitations of such tools before they are integrated with the system. If they are unable to meet the level of responsiveness that would result in an acceptable level of added value to the decision makers, then it may not be worthwhile to undergo the effort of integrating that particular tool. On the other hand, external tools can be independently optimized, e.g., by utilizing multi-processor/programming techniques, until they meet the user's desired performance criteria as specified in the experimental frames.

Although the ATACKS DSS presented here has been designed with input from military domain experts to address the needs of current and future commanders and their battlestaff, the need for thorough system validation and performance evaluation by the intended end users cannot be underestimated. This process is ideally suited for research psychologists who have the background and expertise to set up and execute a comprehensive evaluation that exercises standard and unusual decision making scenarios in the current domain. To facilitate this practice, ATACKS provides many features such as the ability to load and save scenarios, employ scripted dialogs that prompt the user to test their situational awareness or understanding, and record and time user responses. The recommendations that result from experiments with subject-matter experts can subsequently be incorporated into the ATACKS DSS framework.

6. Conclusions

The integration of the DEVS-based DSS with ATACKS demonstrates the adaptability of the architecture to third-party tools that aim to aid the commander in decision-making tasks. DEVS is an ideal platform for

the construction of a distributed DSS. The discrete event-based methodology has been shown to be highly efficient in terms of representation and execution [16] and fits naturally with the domain of battlespace modeling. DEVS has also been extended to serve as a high-performance, advanced distributed simulation middleware and for use with real-time distributed simulation systems, which can enrich the application's user interface experience. With little additional work, the CORBA-based distributed architecture presented can accommodate external applications built around any of the supported platforms and environments. The modular design underlying DEVS and ATACKS allows us to keep the visualization layer simple and efficient while introducing complexity by expanding the responsibilities of the DEVS decision support tool.

The DEVS tool discussed here uses only a subset of the interface methods provided by the ATACKS API. It represents a passive system that requests information from ATACKS without seeking to directly influence the execution of the scenario. In the future, more active systems like Sheherazade or FOX-GA, which select and present the best COAs for a particular scenario could interface with ATACKS such that the COA output is translated and directly input to ATACKS as a complete scenario. FOX-GA has already defined XML schemas for the representation of battlefield COAs and, in the effort, defined the necessary vocabulary to describe key elements of the battlefield and battlefield processes including Unit compositions, terrain characteristics, and so on. By referencing the schemas defined by FOX-GA, any application can take advantage of the same element declarations and type definitions. The representation of the language in XML ensures both flexibility and adaptability to future requirements. Moreover, as a standard language for the exchange of information across the different applications emerges, the overhead of coding the data translation layer will be reduced through the use of a shared standard war-gaming vocabulary.

ATAACKS has the potential to extend in many directions from its current state. The war-gaming rules that are currently used by the DEVS-based decision support tool are fairly primitive, and support only a limited scenario base. As the need for higher resolution decision support evolves, the DEVS tool could also be enhanced to provide greater analysis and more specific feedback of the war-fighting situation. A good starting point would be the incorporation of a rule set specific to the SASO operations for which the 3-D library and suite of configural displays have recently been developed. The Sheherazade war gamer, developed by the Army Research Laboratories, which has also recently been interfaced with ATACKS, provides such a SASO simulation engine designed for modeling

multi-sided conflicts between groups that include terrorist, information operations, media units, etc. [17, 18].

The decision support tool presented in this paper is only one instance of the coupling of the visualization program and an underlying military intelligence tool. As ATACKS begins communicating with more than one external tool, it will become necessary to deal with the additional complexity of managing information from multiple sources. With interoperability based on a DEVS platform that supports HLA and CORBA, we anticipate that ATACKS is well suited for future development as a dynamic decision support tool.

7. References

- [1] Momen F., et al. Three layer architecture for continuous planning and execution. *Proceedings of the 2001 Army Research Laboratories Symposium*; 2001; College Park, MD.
- [2] Barnes MJ. Process centered displays and cognitive models for command applications. *IEEE International Conference and Workshop on Engineering of Computer Based Systems*; 1997. 129–135.
- [3] Keane JS, Rozenblit JW, Barnes MJ. The advanced battlefield architecture for tactical information selection. *IEEE Conference and Workshop on Engineering of Computer Based Systems*; 1997; Monterey, CA. 228–237.
- [4] Suantak L, et al. Intelligent decision support of Support and Stability Operations (SASO) through symbolic visualization. *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*; 2001. 5: 2927–2931.
- [5] Gamma E, et al. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley; 1995.
- [6] AI Simulation Research Group. Discrete event system specification. Available from: <http://www.acims.arizona.edu>
- [7] Zeigler BP, Kim TG, Praehofer H. *Theory of modeling and simulation*. 2nd ed. Academic Press; 2000.
- [8] Department of Defense. High level architecture interface specification version 1.0; 1996.
- [9] Zeigler BP, et al. Implementation of the DEVS formalism over the HLA/RTI: problems and solutions. *Proceedings of Simulation Interoperability Workshop*; 1999; Orlando, FL.
- [10] Hayes CC, Sclaback JL, Feibig CB. Fox-GA: An intelligent planning and decision support tool. *IEEE International Conference on Systems, Man, and Cybernetics*; 1998. 3: 2454–2459.
- [11] Zeigler BP, et al. DEVS modeling and simulation: a new layer of middleware. *IEEE Third Annual Workshop on Active Middleware Services*; 2002. 22–31.
- [12] Cho YK, Zeigler BP, Sarjoughian H. Design and implementation of distributed real-time DEVS/CORBA. *IEEE International Conference on Systems, Man, and Cybernetics*; 2001; Tucson, AZ. 5: 3081–3086.
- [13] Object Management Group. CORBA. Available from: <http://www.omg.org>
- [14] Stevens WR. *Advanced programming in the UNIX environment*. Addison Wesley Longman; 1993.
- [15] Momen F, et al. A distributed approach to genetic algorithm-based course of action generation. *CMS 2005: Conference on Conceptual Modeling and Simulation*; 2005; Marseilles, France.
- [16] Zeigler BP, Moon Y, Kim D, Ball G. The DEVS environment for high-performance modeling and simulation. *IEEE Computational Science and Engineering*; 1997. 4(3): 61–71.

- [17] Suantak L, et al. A coevolutionary approach to course of action generation and visualization in multi-sided conflicts. *IEEE International Conference on Systems, Man, and Cybernetics*; 2003. 1973–1978.
- [18] Suantak L, Momen F, Rozenblit JW, Barnes MJ, Fichtl T. Modeling and simulation of Stability and Support Operations (SASO). *IEEE International Conference and Workshop on the Engineering of Computer Based Systems*; 2004. 21–28.

Author Biographies

Jerzy Rozenblit, Ph.D., is Professor and Head of the Electrical and Computer Engineering Department at The University of Arizona. During his tenure, he has established the Engineering Design Laboratory with major projects in design and analysis of complex, computer-based systems, software engineering, embedded systems, and symbolic visualization. The projects have been funded by the National Science Foundation, U.S. Army, Siemens, Infineon Technologies, Rockwell, McDonnell Douglas, NASA, Raytheon, and Semiconductor Research Corporation. He has extensive teaching experience and conducts a vigorous graduate program as evidenced by many successful Ph.D. and M.Sc. students and best teacher awards. Dr. Rozenblit is active in professional service in capacities ranging from editorship of *ACM and Simulation: Transactions of the Society for Modeling and Simulation*, program and general chairmanship of major conferences, to participation in various university and departmental committees. Among several visiting assignments, he was a Fulbright Senior Scholar and Visiting Professor at the Institute of Systems Science, Johannes Kepler University, Austria, Research Fellow at the U.S. Army Research Laboratories, Visiting Professor at the Technical University of Munich, and Fulbright Senior Specialist in Cracow, Poland. Over the years, he has developed strong associations with the private sector and government entities. Dr. Rozenblit's management and project experience includes over \$8 million in externally funded research. He has served as a research scientist and visiting professor at Siemens AG and Infineon AG Central Research and Development Laboratories in Munich, where over the last decade he was instrumental in the development of design frameworks for complex, computer-based systems. For the last eleven years, he has led a vigorous research program at the University of Arizona in visualization, human-computer interaction, and artificial intelligence funded by the U.S. Army. Currently, jointly with the Arizona Simulation Technology and Education Center, he is developing virtually assisted surgical training methods and systems. Co-author of several edited monographs and over a hundred publications, Dr. Rozenblit holds the Ph.D. and MS degrees in Computer Science from Wayne State University, Michigan, and the M.Sc. degree in Computer Engineering from the Technical University of Wroclaw, Poland.

Faisal Momen is currently a Ph.D. student in the Electrical and Computer Engineering Department at the University of Arizona, Tucson. He received a BS and MS in Computer Engineering from the University of Arizona and has worked in industry as a Software Engineer with Motorola Computer Group, Tempe.