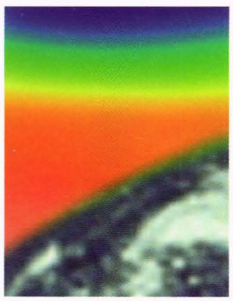


Distributed Vertical Model Integration

Robert R. Lutz

Persistent financial pressures within the Department of Defense have sparked increasing interest in the promise of software reuse as a mechanism to achieve greater efficiency in developing and applying models and simulations to support the systems acquisition process. A perpetual obstacle to the widespread proliferation of reusable software code within DoD is that most existing, older-technology ("legacy") models are extremely difficult to reuse within more modern software architectures without significant redesign. This article reports results of an experiment demonstrating the utility of distributed computing environments as a means to reuse detailed software representations of system- or subsystem-level hardware components through direct access from less detailed (higher-order) models.

INTRODUCTION

Applying modeling and simulation (M&S) techniques to gain information about the behavior of specific systems in a complex operating environment requires that systems be represented at varying degrees of detail, or levels of resolution. An application's objectives will drive the degree to which simulated representations of systems (and the environment in which they operate) are aggregated. For example, during the engineering development of a system acquisition program, high-resolution computerized models of actual hardware systems are frequently used to study, evaluate, and test the system's sensitivity to simulated representations of physical stimuli. In contrast, determining system requirements and exploring early system concepts normally require highly aggregated abstractions of the conceptual designs to be "played out" in an operational context where numerous systems interact.

In his early work, B. P. Zeigler,[1] who was a pioneer in addressing the need for integrated software environments to support multiple levels of model resolution, promoted the concept of a "model hierarchy." The hierarchy was based on the premise that practical simulation tools can be abstracted from highly complex models, called base models, that account for all possible behaviors of actual systems. These tools, called lumped models, are constructed from base models by grouping model components, aggregating variables, and simplifying interactions among components. In later works, Zeigler[2,3] expanded his model hierarchy paradigm to include an entire hierarchy of levels at which models can be constructed, ranging from the purely behavioral, in which the model claims to represent only the observed input and output behavior of the system, to highly detailed structural models. Integral to the

paradigm is the support of an M&S environment, in which the model hierarchy resides, to facilitate the composition of new models from existing reusable components. These components can then be reassembled into numerous new combinations for various study objectives. Note that modular, hierarchical model construction does not result directly in a lumped model. Features to facilitate the aggregation of lower-level models are required within such an environment to support transitions between levels of the model hierarchy.

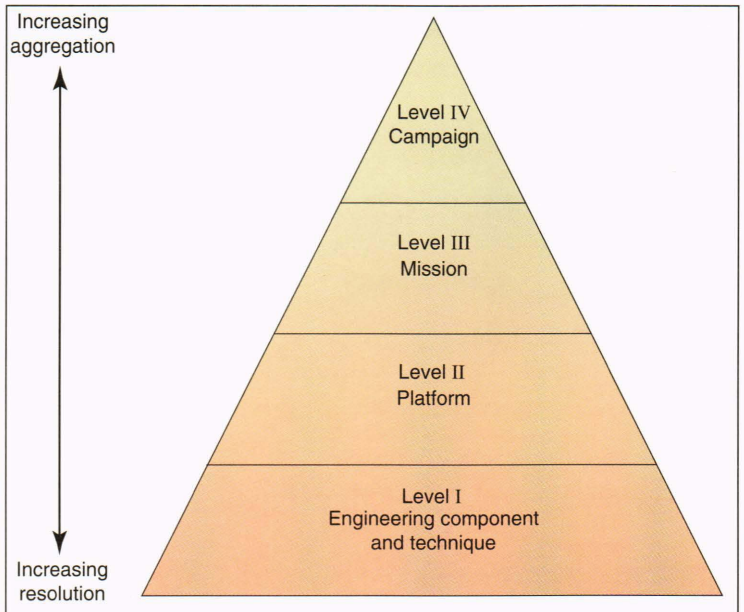


Figure 1. Hierarchy for modeling and simulation analysis commonly used in DoD. Because of the relationship between models at sequential levels, it is generally feasible to aggregate output data from models at one level to provide input for models at the next higher level.

THE HIERARCHICAL MODEL PARADIGM

Today, several alternative definitions are used to specify categories for different levels of modeling hierarchies. The most common paradigm in DoD is the four-level modeling hierarchy. Figure 1 shows the conventional modeling pyramid for categorizing levels of system analysis. Each level of the pyramid is defined as follows:[4]

Level I—Engineering Component and Technique. Level I models support analysis of individual subsystems or techniques in the presence of a single-threat weapon system element. Examples include models of sensors, countermeasures, weapon delivery systems, or other detailed engineering subsystems or components.

Level II—Platform. Level II models support analysis of integrated weapons systems, including associated tactics and doctrine when the system is engaged with one or more threats. This level introduces a thorough platform simulation with a complete representation of all pertinent onboard guidance, communications, sensor, and other avionics systems.

Level III—Mission. Level III models support mission effectiveness and force survivability analysis of friendly forces composed of multiple platforms opposing threat forces (called "many-on-many" simulations). This level expands the scope of associated tactics and doctrine to two-sided, action–reaction analysis.

Level IV—Campaign. Level IV models support the incorporation of command, control, communications, and intelligence (C^3I) contributions of multimission joint-service operations against a combined threat force.

Fundamental to this hierarchical modeling paradigm is that each level of the hierarchy above Level I amalgamates the capabilities represented by the systems and platforms at the next lower level. For example, integrated weapons systems represented at the resolution of Level II consist of individual subsystem components represented by Level I models. Similarly, military campaigns (Level IV) combine several missions (Level III) across the joint services. This structure is similar to Zeigler's paradigm. However, true modular, hierarchical modeling environments, in which models at higher levels in the pyramid are constructed from reusable components at lower levels of the pyramid, are rare within DoD. A notable exception is the Air Force–sponsored Joint Modeling and Simulation System (J-MASS) Program, in which automated computer-aided systems engineering (CASE) tools are provided within an integrated M&S framework to control the construction of higher-level entities from lower-level components. Even so, the present J-MASS environment does not directly facilitate the transition from one level of model resolution to another.

Because of the relationship between models at sequential levels of a model hierarchy, it is generally feasible to aggregate output data from models at one level to provide input for models at the next higher level. For instance, the engineering-level subsystem performance parameters obtained during Level I analysis can be aggregated to provide input characteristics for a Level II integrated weapons system. The platform-level effectiveness data obtained during a Level II analysis can then provide input for a Level III multiplatform, mission-level analysis. Consequently, the relative effects of low-level engineering modifications on individual subsystems or components can be mapped to much higher-level operational effectiveness issues through proper application of this methodology. The process of capturing the results of analysis performed with low-level, high-resolution models in more aggregated higher-level models is known as vertical model integration.

Vertical Model Integration

Two primary methods are used to integrate models vertically within a model hierarchy. In the first method, the output of a model at one level is passed to the input of a higher-level model. Although widely practiced, this approach presents several potential problems. First, since data formats are rarely consistent between models at different levels, data transfer generally requires manual translation, which introduces a potential source of error. The process of aggregating data upward through the model hierarchy is also largely manual. Since it depends heavily on the technical expertise and judgment of the analyst, the potential for error is high. Finally, because this method is highly resource-intensive, particularly in terms of personnel, the associated costs are substantial. Efforts to minimize costs by using shortcuts can introduce additional potential sources of error. The common thread through all of these problems is a general lack of software tools to automate the transfer of data throughout the model hierarchy.

In the second method for vertical model integration, called a "software zoom," lower-level models are accessed directly from higher-level models.[5] Narrowly interpreted, the technique refers to the ability to incorporate high-resolution submodels as subroutines within a higher-level model. The technique is much broader, however; it refers to selective injection of detailed system representations into a simulation at the phases of a predefined scenario considered most sensitive to individual system performance. The software zoom technique exploits the coexistence of both coarse and detailed representations of simulated entities or systems within the same simulation environment. On the basis of either predefined scripts or direct intervention by the operator, the simulation controller "toggles" between high- and low-resolution modes for specific systems of interest. The user can focus computing resources on generating or collecting high-resolution performance data about systems of interest while allowing coarser representations of less critical systems or of the same system at less crucial times. The result is large increases in computing efficiency, allowing the analyst to capture the effects of low-level modifications to existing subsystems (or introduction of new subsystem concepts) on large-scale, force-on-force encounters within the same simulation.

Legacy Model Integration

Most existing high-resolution system models are stand-alone simulations funded by individual sponsors to support specific applications. These older models are known as legacy models. Ongoing proof-of-concept exercises on reuse and integration of these models show that the major disadvantage of the software zoom technique is the extensive amount of labor required to integrate existing high-resolution representations of existing systems into the framework of higher-level engagement- or mission-level models. Because legacy models were extensively designed and implemented according to older (in some cases obsolete) structural standards, their integration into existing higher-level simulations is generally quite complicated. As a result, the resources required to rehost legacy models written in different languages and resident on different types of hardware can rival the cost of starting from scratch.

Modern simulation architectures routinely support the premise of a software zoom by providing features to configure simulations from entities represented at several different levels of resolution. This structure is strongly supported by the object-oriented paradigm, since specific objects can be defined at several levels of resolution and accessed within a simulation according to a common message interface (polymorphism). However, experience with treating legacy models as low-level objects within a simulation architecture is much more limited. Apart from the obvious (but expensive) option of completely redesigning the software of the legacy model into an object-oriented format, efforts to date on integration techniques for legacy models have focused largely on the use of interface shells ("wrappers") to encapsulate legacy models in an object-like structure. The wrapper provides a mechanism to connect the legacy model to other objects in the simulation system.

One of the best examples of the use of wrappers is the Navy-sponsored Capabilities Assessment, Simulation, and Evaluation System (CASES).[6] An automated decision-support system, CASES supports the development and analysis of exercise, contingency, and war plans for the Pacific Fleet Command Center. An overview of the CASES architecture is shown in Fig. 2. Evaluation of campaign-level issues in CASES is based on four warfare-area-specific legacy models (strike, ship defense, antisubmarine warfare, and logistics) and a specialized simulation controller for scheduling and invoking simulation events. The foundation of the CASES architecture is the Cronus distributed computing environment (DCE) developed by BBN Systems and Technologies of Cambridge, Massachusetts.[7] Cronus is a network-transparent, host-independent (heterogeneous) interprocess communication facility for distributed computing based on an object-oriented design paradigm. It uses a client–server implementation technique to emulate active software objects by means of the traditional process and memory models of computer systems. A server (also called an object manager) defines and manages objects of one or more distinct data types. The Cronus kernel locates objects and transmits operation invocations from clients to servers (Fig. 3). The Cronus DCE also contains automated tools for rapidly integrating existing legacy code. These tools greatly simplify the process of wrapping legacy

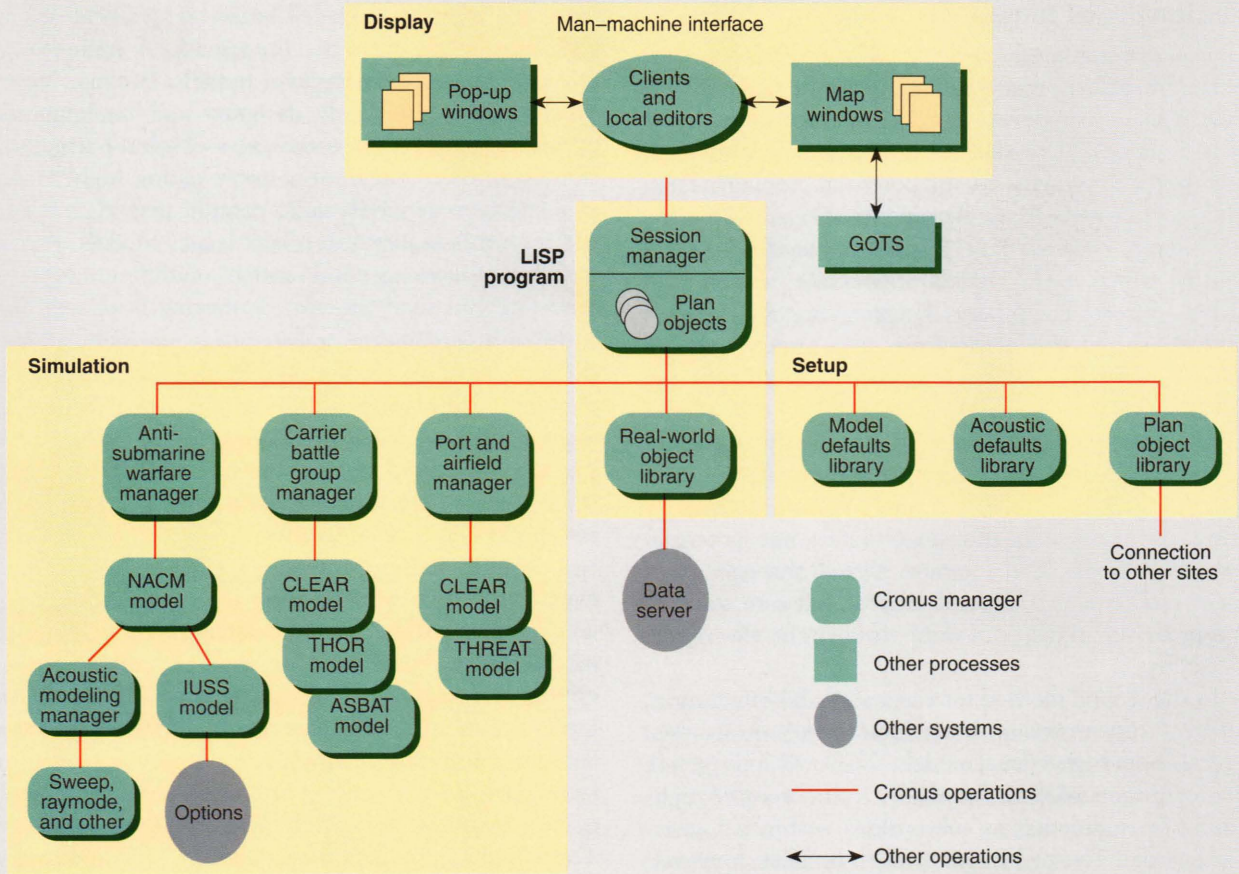


Figure 2. Software architecture of the Capabilities, Assessment, Simulation, and Evaluation System (CASES), an automated decision support system that provides interface shells (wrappers) to encapsulate existing legacy models in an object-like structure. (GOTS is government off-the-shelf software; LISP is the programming language used by the session manager; and NACM, CLEAR, IUSS, THOR, THREAT, and ASBAT are legacy models wrapped for incorporation in the higher-level simulation.)

models. Legacy models do not have to be rehosted, because models written in different languages and running on different machine types and operating systems can be accessed and invoked directly from a remote simulation controller. This approach represents the state of the art in legacy model integration within the Navy.

The CASES program proved that Cronus represents a viable technology for integrating legacy models, but the application of distributed computing technology to vertical model integration remains largely unexplored. Although CASES successfully demonstrated the feasibility of linking, invoking, and controlling disparate legacy models within an integrated simulation environment, no attempt was made to invoke lower-level models directly from higher-level models. The lack of this capability for capturing the impact of individual low-level subsystem modifications on large-scale, force-on-force encounters is considered a significant deficiency in the use of M&S within the DoD acquisition community. This article investigates an approach to the deficiency, extending lessons learned from experiments in legacy model integration to the concept of distributed vertical model integration.

EXPERIMENTAL OBJECTIVE: VERTICAL LEGACY MODEL INTEGRATION

The purpose of our experiment was to enhance the distributed computing technology demonstrated in the CASES program (Cronus DCE) to capture the effects of low-level model results within a higher-level model. The higher-level model chosen for this experiment was Suppressor (version 5.2), an event-stepped digital simulation model used to model multisided conflicts involving air, ground, naval, and space-based systems at the mission level of detail (Level III).[8] Suppressor is an established, relatively mature tool for examining the integrated effects of various weapon systems, sensors, tactics, command and control structures, susceptibilities, and countermeasures. Configuration control of the Suppressor model is currently supplied by Science Applications International Corporation personnel at the Electronic Combat Simulation Research Laboratory, Wright-Patterson Air Force Base, Dayton, Ohio. The modular structure of the Suppressor software architecture made this high-level model an especially attractive

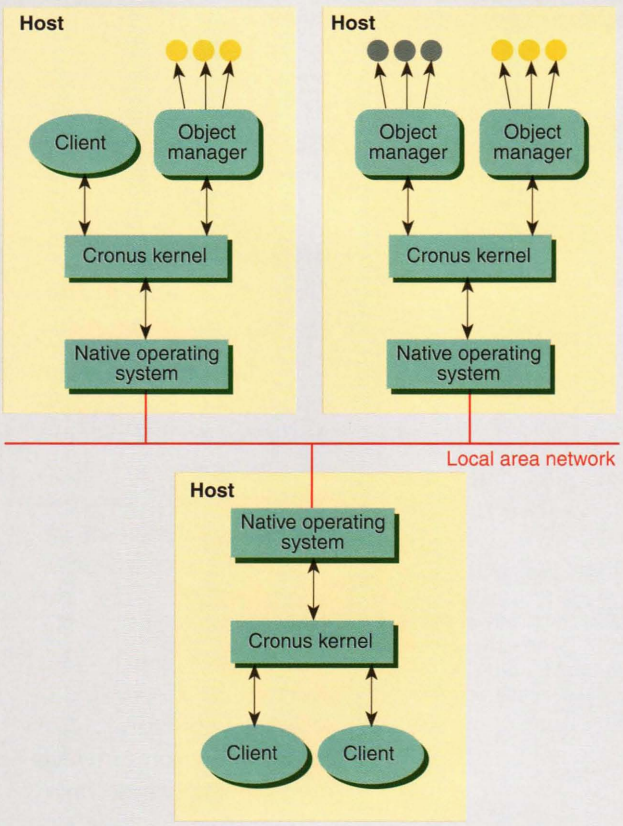


Figure 3. Architecture of the Cronus distributed computing environment. Cronus is a network-transparent, host-independent interprocess communication facility for distributed computing based on an object-oriented paradigm.

choice to demonstrate the vertical integration of lower-level model effects.

The lower-level model chosen for this experiment was the Advanced Low Altitude Radar Model (ALARM91). ALARM91 is a high-resolution digital computer simulation designed to evaluate the performance of a ground-based radar system against a single airborne target vehicle (a one-on-one simulation).[9] This model supports radar subsystem analysis at resolution Level I. Target detection calculations within ALARM91 are based on common signal-to-noise radar range equations derived by established experts in radar theory. The model also captures effects of atmosphere, terrain masking, clutter, multipath radar returns, and jamming. It is one of several models under the control of the Air Force Survivability/Vulnerability Information Analysis Center (SURVIAC) for configuration control and dissemination.

The processing of a radar sensing event in the Suppressor model consists of several steps. Factors such as target status (dead or alive), radar Doppler limits, terrain masking, sensor azimuth/elevation limitations, and signal-to-noise level are all considered in determining target detectability. The inherent Suppressor methodology for determining the signal-to-noise level of a target is similar to, but less detailed than, the procedures used in ALARM91. Since this methodology is largely contained within a single Suppressor module, our objective was to develop the capability to selectively toggle between the current Suppressor radar module and the more detailed ALARM91 model based on changing conditions during execution. Specifying these toggle conditions, calibrating the individual radar system databases, and implementing the resources required to remotely access the ALARM91 model over a heterogeneous network were the main technical challenges of the project.

RESOURCES

The Laboratory's Strike Warfare Laboratory was chosen to conduct the experiment. Preparations involved defining and establishing a hardware and software configuration that could adequately demonstrate the utility of heterogeneous DCEs in vertical legacy model integration. The main requirement of the selected hardware and software configuration was that Suppressor and ALARM91 be hosted on dissimilar hardware and operating systems on different local area networks. The configuration chosen is shown in Fig. 4. A VAXStation 3100 (node 1) running the VAX/VMS operating system was the ALARM91 host. An Apollo DN10000 (node 2) utilizing the Berkeley UNIX (BSD 4.3) environment under the Apollo Domain operating system was the host for Suppressor. Each machine shown in Fig. 4 was a node on one of two different local area networks interconnected through a gateway (node 3). Also required in this configuration were the Cronus DCE, installed on both host computers, and Wollongong Pathway software, installed on the VAX computers, to support transmission control protocol/Internet protocol (TCP/IP) communication.

METHODS

Several steps were required to meet experimental objectives. A Cronus object manager for the ALARM91 model and a Cronus client routine for the Suppressor model were developed. We also calibrated the static radar data in the two models to minimize discontinuities during transitions between high- and low-resolution modes, devised control logic to manage the high–low resolution transitions, and tested parallel processing capability to enhance execution speed. The following paragraphs describe each step in detail.

Step 1: Develop Object Manager for ALARM91 Model

An object manager in Cronus is a process that defines and manages objects of one or more distinct

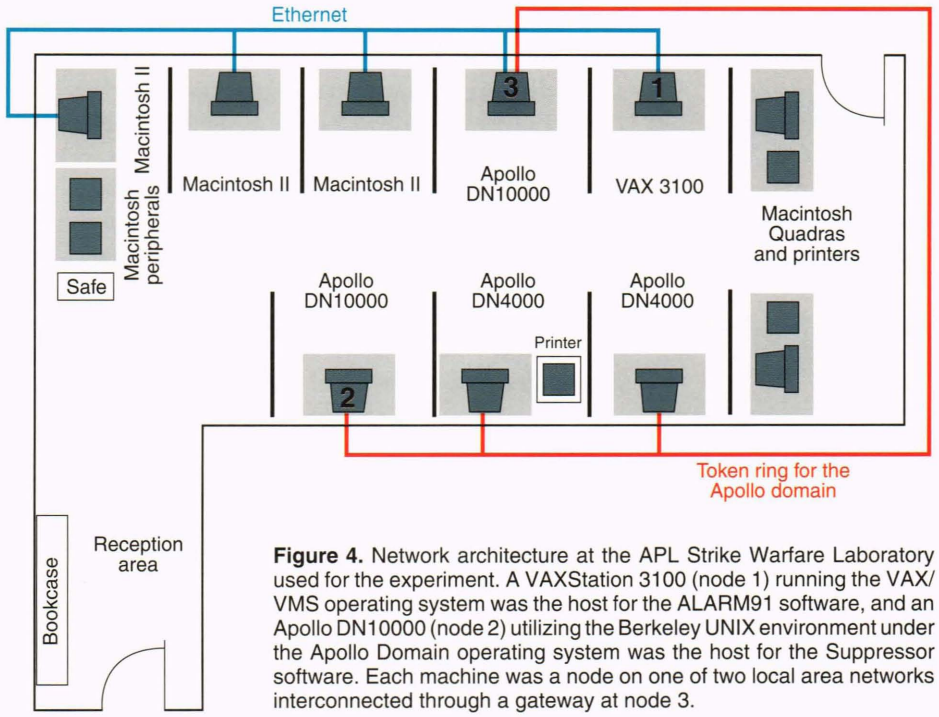


Figure 4. Network architecture at the APL Strike Warfare Laboratory used for the experiment. A VAXStation 3100 (node 1) running the VAX/VMS operating system was the host for the ALARM91 software, and an Apollo DN10000 (node 2) utilizing the Berkeley UNIX environment under the Apollo Domain operating system was the host for the Suppressor software. Each machine was a node on one of two local area networks interconnected through a gateway at node 3.

data types. The object data type in this case was the radar system represented by the ALARM91 model. The specification of the ALARM91 data type consisted of three distinct operations: a CREATE operation to create the ALARM91 object during Suppressor initialization, a RUN operation to invoke the ALARM91 object directly from the Suppressor model, and a DELETE operation to remove the ALARM91 object at the end of a Suppressor run. Included within the specification of the RUN operation was the dynamic interface information required by the ALARM91 object to define the sensor–target geometry at the time of the sensing opportunity. The interface information included target latitude, longitude, altitude, velocity, pitch, roll, and heading. The Suppressor–ALARM91 interface also required the specification of the data file that the ALARM91 object should use (on its host machine) to define and configure the radar system of interest. The data file contained static data: numerous radar parameters, such as beamwidth, transmitter power, and pulse width, which did not change during a simulation run.

Once the ALARM91 data type was defined, tools embedded within the Cronus environment automatically generated the ALARM91 object manager shell. Finally, the ALARM91 code was manually inserted into the RUN operation shell to produce the desired functionality for that operation. This task required us to restructure the ALARM91 top-level (main) routine into a subroutine and also to resolve cross-language communication issues between the ALARM91 applications code (written in Fortran77) and the ALARM91 object manager (generated in C).

Step 2: Develop Cronus Client Routine for Suppressor Model

A client in the Cronus environment is an application program that requests services from an object manager. The construction of a client program was required to create an interface "bridge" from the Suppressor model to the ALARM91 model. This program, developed as a C language subroutine called from Suppressor, provided the proper operation invocations to both create and delete ALARM91 objects and to invoke the ALARM91 model with interface information provided by the core Suppressor model. During creation of the object manager shell, Cronus automatically provided the required formats for the operation invocation requests in a machine-generated file. This routine was compiled separately and linked with the Suppressor object code prior to execution.

Step 3: Calibrate Databases

To avoid or minimize discontinuities in simulated radar performance during transition between low- and high-resolution modes, an effort was made to calibrate the static radar data in the ALARM91 and Suppressor models. Generally, databases defined at different resolution levels can best be calibrated through the development of a disciplined "bottom-up" mapping methodology to build aggregated data sets for high-level models directly from the high-resolution, engineering-level data of lower-level models. Although defining and implementing this methodology would be critical for most practical vertical integration applications, the procedure was applied in a limited fashion for this experiment. We simplified calibration by use of an existing parameter in the Suppressor radar data set to automatically calibrate simulated radar systems to a specific detection range. Suppressor uses the value of this parameter to adjust the radar receiver noise value (defined in the input) to correspond to the desired range

performance when evaluating the radar range equation. Thus, in addition to "aligning" other radar database parameters wherever possible, adequate correlation between Suppressor radar performance and the ALARM91 model was readily achieved through off-line execution of the ALARM91 model to obtain the appropriate value definition for this parameter.

Step 4: Develop Required Control Logic for High–Low Resolution Transitions

An important element of the software zoom concept is the capability to selectively interject high-resolution system representations into a simulation only during critical phases of the mission. In this experiment, critical mission phases were specified by introducing three new parameters in the radar receiver capabilities section of the Suppressor type database. These parameters stipulated the times at which high-resolution mode should begin and end during the simulation and indicated a range beyond which high-resolution mode was not required. Although most practical applications will require a much broader set of transition criteria, the intent here was simply to establish a baseline logic framework that could easily be expanded to include the requirements of diverse users.

Besides the required database modifications already described, a new software routine was developed within the Suppressor model specifically to control transitions between high- and low-resolution modes. The purpose of the routine was to set a high–low resolution toggle switch based on the current simulation time and the range from the sensor to the target at the time of the Suppressor sensing opportunity. If the outcome of this routine specified that the sensing opportunity required no high-resolution data, the control logic for processing radar sensing events within Suppressor was invoked. If the sensing opportunity did require high-resolution data, control was passed to a specialized submodule to construct the required ALARM91 interface information for the Cronus client routine. The client program then assumed control of the invocation of the ALARM91 model over the distributed network (via Cronus) and routed the result of the high-resolution radar sensing event back to the Suppressor model for standard (resolution-independent) processing.

Step 5: Implement Parallel Processing Capability to Enhance Execution Speed

Although the first four steps established the desired Suppressor/ALARM91 distributed processing capability, the application resulted in inefficient utilization of computing resources. The software structure of the Suppressor model was not designed to take advantage of current parallel processing technology. Because the flow of control during Suppressor (sensor) event processing was entirely serial, the Suppressor model always had to wait for the ALARM91 model to return the outcome of a high-resolution sensing opportunity before any further processing within the Suppressor model could take place. The purely serial approach to sensor processing always kept one of the two host machines idle during a simulation run.

Although the inherent computing inefficiency was not significant for this experiment, other potential applications may require at least rudimentary parallel processing capability if the technology is to be considered practical. Fortunately, Cronus provides a mechanism, called a "Future," to support concurrent operations. When a remote operation is called using Futures, the interface to the operation is largely identical to the standard invocation. However, instead of waiting for the remote operation to finish, a value (called the Future) is immediately returned to the calling client as a pointer to the results of this operation. At an appropriate time later in the calling program, a "Claim" operation uses this pointer value to retrieve the results of the remote operation. Thus, the calling program is free to process other routines in parallel with the remote call until the results of that operation are needed.[7]

The simplified diagram in Fig. 5 illustrates the application of Futures in our experiment. The top-level logic branch is based on whether this is the initial sensing opportunity for the particular sensor–target pair. If so, the decision of whether to invoke the current Suppressor radar routine or the ALARM91 model (via Cronus) over the distributed network depends on the outcome of the high–low resolution transition routine described in Step 4. After sensor processing, the decision logic determines whether to start a Future for the next sensing opportunity for this sensor–target pair. The decision is based both on simulation time and on a spatial projection of the target position at the next scheduled sensing opportunity. If the decision is to start a Future (that is, high-resolution mode is required), then the required interface information is constructed, the Cronus client executes the request, and the Future pointer value is stored in the Suppressor sensor–target buffer. This step preprocesses the next high-resolution sensing event for this sensor–target pair on a different machine so that the outcome of the sensing event is immediately available to the Suppressor model when simulation time "catches up" to the time of the projected sensing opportunity.

If this is not the first sensing opportunity for the sensor–target pair, a check is immediately performed to determine if a Future was started for the pair during the last sensing opportunity. If not, the implication is that high-resolution mode is not required. In this case, the

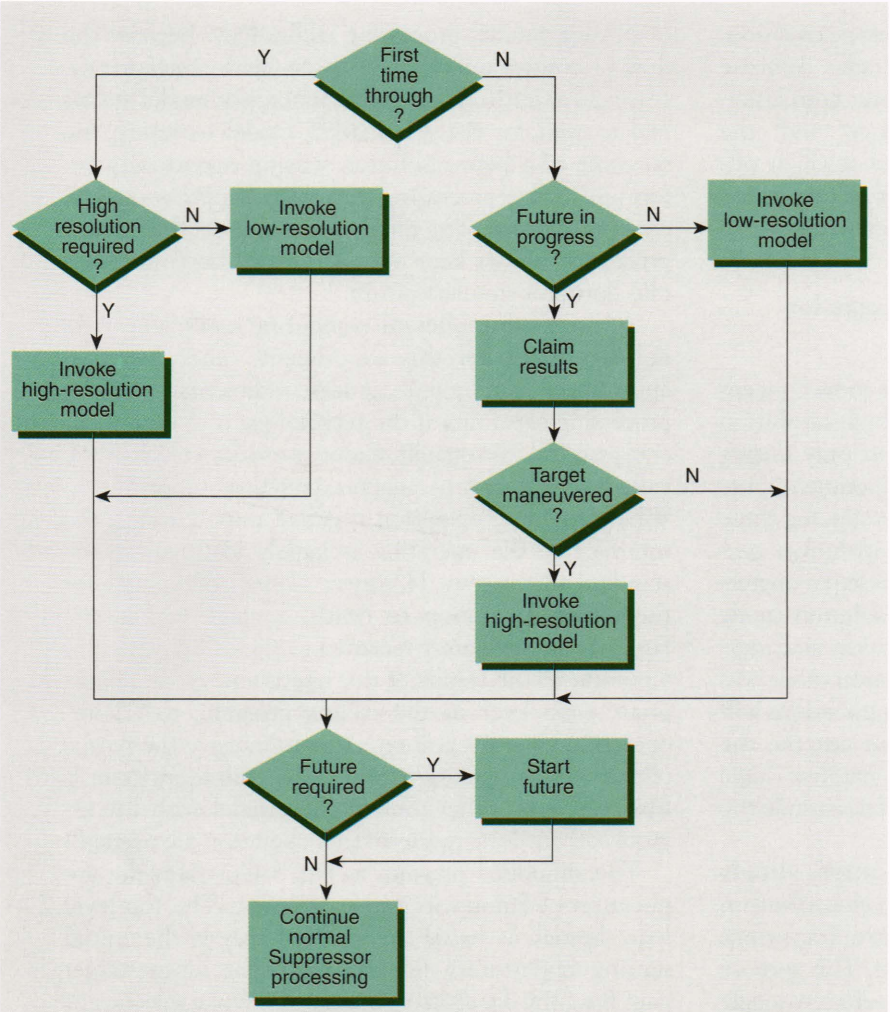


Figure 5. Simplified diagram of the logic for the transitions between high- and low-resolution analysis in the Suppressor program.

current Suppressor radar routine is invoked to determine the outcome of the sensing event, and the decision logic to determine whether to start a Future for the next sensing opportunity is executed. If a Future was started at the last sensing opportunity for this sensor–target pair, the pointer value in the sensor–target buffer is used to claim the results. A check (based on conditions within the simulated environment) is then performed to determine if the target has performed any dynamic maneuvers since the last sensing opportunity. If so, the results claimed from the prior Future are invalidated, because the spatial state projection of the target was built according to preexisting flight plans. In this case, the ALARM91 model must be reinvoked for the sensing opportunity based on the new target state information. If the target has not maneuvered dynamically since the last sensing event, the Future logic for the next scheduled sensing opportunity is executed, and standard processing of the detection results within Suppressor continues.

TESTING

Testing our implementation of the software zoom concept was supported by the definition and implementation of a limited surface-to-air engagement scenario. Several elements of the scenario, such as the size of the opposing forces and engagement geometries, were varied during the testing to exercise key elements of the methodology. The primary emphasis in the testing phase was to validate the high–low resolution transition software, assure the accuracy of the Suppressor–ALARM91 interface, and verify the correctness of the Cronus Future facility and supporting control logic. Although the testing was not as formal or exhaustive as would be required for more practical applications, all major facets of the methodology were successfully demonstrated during this phase.

An additional area of testing we performed involved comparing the detection results of the Suppressor and ALARM91 radar models at high–low resolution transition points. Although model results were found to be highly correlated, introducing additional higher-order effects in high-resolution mode (i.e., terrain, propagation) could readily produce potentially significant discontinuities during high–low resolution transitions. Such discontinuities could then invalidate previous reactive events in the simulation that were based on low-resolution results. This undesirable consequence of the software zoom technique needs additional study, and may eventually represent an entirely new aspect of the verification, validation, and accreditation (VV&A) process.

CONCLUSIONS

The distributed vertical model integration methodology described has considerable potential value to the DoD systems acquisition process. The technique offers a highly efficient method of leveraging the enormous financial investment in high-resolution system and subsystem software models to capture the effects of new systems (or modifications to existing systems) on large-scale, force-on-force encounters. This technique is particularly beneficial in the near term: the

integration of low-level system representations into higher-level models offers a technically sound interim solution until object-oriented simulation environments are developed that can directly support software zoom through multilayered object hierarchies.

Although this experiment was not a real-time application, the technique can be readily applied to support the throughput requirements of individual nodes in a distributed real-time exercise. For example, consider the fictitious real-time application depicted in Fig. 6. In this exercise, several distributed simulation nodes, each node representing unique players within the simulation, operate within a virtual battlefield via a predefined communication protocol (i.e., Distributed Interactive Simulation). Suppose the objective of the exercise is to evaluate the impact of a new F-18 radar warning receiver (RWR) system on operational effectiveness. Although a high-resolution software representation of the RWR system is required to adequately evaluate the relative utility of the RWR hardware in this environment, integrating the high-resolution RWR software representation into the F-18 simulator degrades performance of the F-18 host processor to less than real-time. The resolution requirements on the RWR representation can be relaxed, but an alternate approach would be to host the RWR software on a remote processor, accessed directly from the F-18 simulator (via Cronus) when requirements dictate. Besides the obvious benefit of reestablishing real-time performance through off-loading processor-intensive functions onto remote hardware resources, the insertion of high-resolution system representations into the simulation may be constrained to those critical mission phases in the scenario that drive the hardware performance evaluation. Although this example illustrates the utility of this technique for a single system or single node application, supporting hardware resources could be dedicated to any number of individual systems. Via the Cronus DCE, any node on the distributed network can then access these high-resolution system representations as needed.

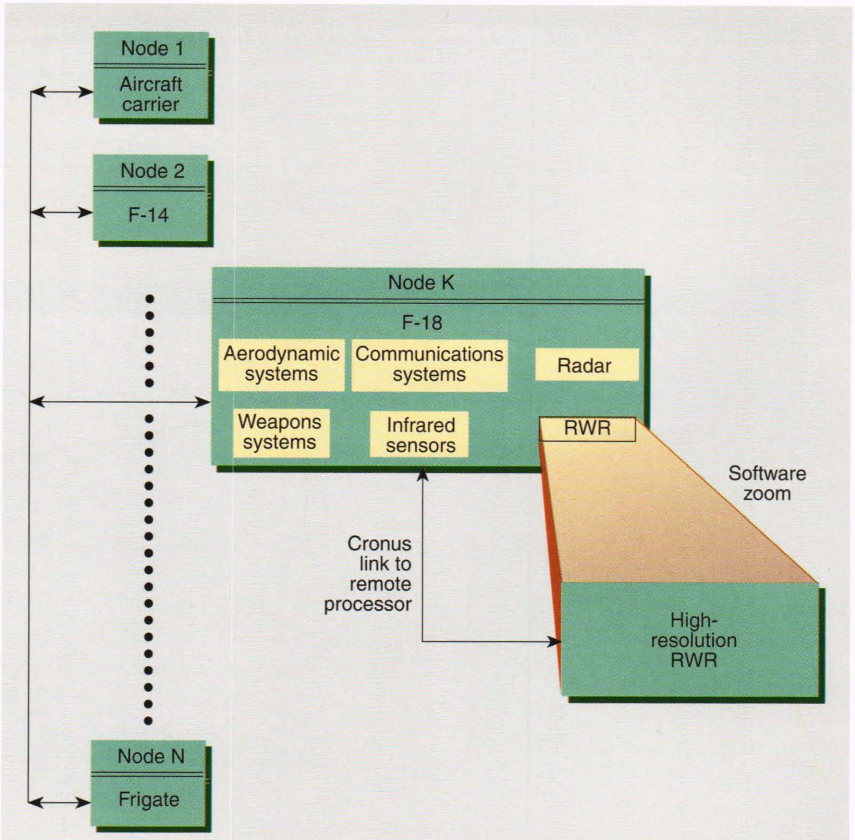


Figure 6. Fictitious application of the software zoom concept to a real-time exercise involving, among other participants, an aircraft carrier, a frigate, an F-14, and an F-18. Systems shown on the F-18 are part of the high-level model; the Cronus link is allowing the analyst to take a higher-resolution look at the radar warning receiver (RWR).

SUMMARY

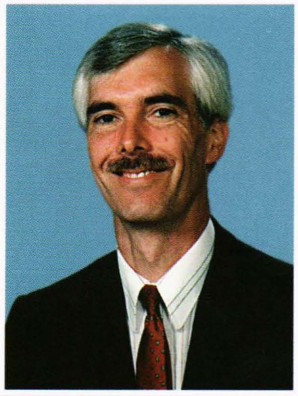
Relating the impact of low-level subsystem modifications to overall force effectiveness represents a major unsolved problem within the DoD acquisition community. Distributed computing environments can be used to link detailed software methodologies in existing legacy software to the simulation framework of higher-order models, offering a viable technical approach and potential solution to this problem. Although practical issues remain concerning the application of this technique within a structured analytical methodology, our experiment successfully demonstrated a point of departure from which a more complete methodology can be developed to address broad classes of potential applications in support of the DoD acquisition process.

REFERENCES

[1] Zeigler, B. P., *Theory of Modeling and Simulation*, John Wiley & Sons, New York, pp. 27–49 (1976).
[2] Zeigler, B. P., *Multifaceted Modeling and Discrete Event Simulation*, Academic Press, Orlando, FL (1984).
[3] Zeigler, B. P., and Oren, T. I., "Multifaceted, Multiparadigm Modeling Perspectives: Tools for the 90's," in *1986 Winter Simul. Conf. Proc.*, pp. 708–711 (1986).
[4] Department of the Air Force, *Air Force Electronic Combat Test Process Guide*, AF Pamphlet 58-5, Washington, DC (1992).
[5] Sisti, A. F., *Large-Scale Battlefield Simulation Using a Multi-Level Model Integration Methodology*, RL-TR-92-69, Rome Air Development Center, Rome, NY (1992).

[6] *Capabilities Assessment, Simulation and Evaluation System (CASES) System/Segment Design Document*, NOSC Technical Memorandum SD-295, Naval Ocean Systems Center, San Diego, CA (1992).
[7] Berets, J. C., Cherniack, N., and Sands, R. M., *Introduction to CRONUS, Rev. 3.0*, Report 6986, BBN Systems and Technologies, Cambridge, MA (1993).
[8] *SUPPRESSOR Release 5.2 User's Guide; Volume I: Installation and Use*, Avionics Directorate, WL/AAWA, Wright-Patterson Air Force Base, Dayton, OH (1992).
[9] *Software User's Manual for the Advanced Low Altitude Radar Model (ALARM91)*, Avionics Directorate, WL/AAWA, Wright-Patterson Air Force Base, Dayton, OH (1992).

THE AUTHOR



ROBERT R. LUTZ is a member of the Senior Professional Staff of APL's Naval Warfare Analysis Department. He received a B.S. degree in applied mathematics in 1978 and an M.S. degree in operations research in 1980, both from the State University of New York at Stony Brook. He has worked on various aspects of systems engineering, software engineering, and operations research. Since joining APL in 1992, Mr. Lutz has specialized in the design, development, and implementation of military computer simulations and modeling architectures. Mr. Lutz serves as a technical advisor to the Director of the Naval Modeling and Simulation Technical Support Office. His e-mail address is Robert.Lutz@jhuapl.edu.